**CICS Transaction Server for z/OS**
**バージョン 5 リリース 3**

# CICS での Java アプリケーション



SC43-2885-00

**CICS Transaction Server for z/OS**
**バージョン 5 リリース 3**

# CICS での Java アプリケーション



SC43-2885-00

**注記**

本書および本書で紹介する製品をご使用になる前に、 275 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM CICS Transaction Server for z/OS バージョン 5 リリース 3 (製品番号 5655-Y04) および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

**原典：** SC34-7417-00
CICS Transaction Server for z/OS
Version 5 Release 3
Java Applications in CICS

**発行：** 日本アイ・ビー・エム株式会社

**担当：** トランスレーション・サービス・センター

# 目次

# 前書き

本書には、プログラムを作成するユーザーが IBM® CICS® Transaction Server バージョン 5 リリース 3 のサービスを使用するためのプログラミング・インターフェースが記述されています。

『本書について』
本書は、CICS における Java™ アプリケーションとエンタープライズ Bean の作成と使用の方法について説明します。

『本書の対象読者』

『Knowledge Center 内のトピックの場所』
本書のトピックは、CICS Transaction Server for z/OS の Knowledge Center でも見つけることができます。 Knowledge Center では、コンテンツ・タイプを使用して情報の表示方法が構成されています。

## 本書について

本書は、CICS における Java アプリケーションとエンタープライズ Bean の作成と使用の方法について説明します。

## 本書の対象読者

本書は次の読者を対象にしています。

- CICS の経験がほとんどなく、Java プログラムの作成と実行に必要なだけの CICS の知識が必要な、経験のある Java アプリケーション・プログラマー。
- Java サポートに対する CICS 要件に関する知識が必要な、経験のある CICS ユーザーおよびシステム・プログラマー。

## Knowledge Center 内のトピックの場所

本書のトピックは、CICS Transaction Server for z/OS の Knowledge Center でも見つけることができます。 Knowledge Center では、コンテンツ・タイプを使用して情報の表示方法が構成されています。

Knowledge Center のコンテンツ・タイプは全体としてタスク指向になっています。例えば、アップグレード、構成、インストールなどです。他にも、参照、概要、およびシナリオまたはチュートリアル・ベースの情報などのコンテンツ・タイプがあります。以下のマッピングは、本書内のトピックと Knowledge Center のコンテンツ・タイプの間の関係を示しています。外部の Knowledge Center へのリンクもあります。

*表 1. Knowledge Center のコンテンツ・タイプへの PDF トピックのマッピング*: この表は、PDF のトピックと Knowledge Center のコンテンツ・タイプのトピックとの関係をリストしています。

| 本書のトピックのセット | Knowledge Center での場所 |
| --- | --- |
| • 1 ページの『第 1 章 Java の入門』<br>• 27 ページの『第 2 章 CICS 用の Java アプリケーションの開発』<br>• 129 ページの『第 3 章 Java サポートのセットアップ』<br>• 181 ページの『第 4 章 JVM サーバーへのアプリケーションのデプロイ』<br>• 191 ページの『第 5 章 Java アプリケーションの管理』<br>• 209 ページの『第 6 章 Java パフォーマンスの改善』<br>• 251 ページの『第 8 章 Java アプリケーションのトラブルシューティング』 | • 始めに<br>• Configuring<br>• Administering<br>• アプリケーションの開発<br>• デプロイ<br>• パフォーマンスの改善<br>• Troubleshooting and support |

# CICS Transaction Server for z/OS、バージョン 5 リリース 3 の変更点

このリリースに加えられた変更点に関する情報は、Knowledge Center の『*新機能*』または以下の資料を参照してください。

- *CICS Transaction Server for z/OS 新機能*
- *CICS Transaction Server for z/OS CICS TS バージョン 5.3 へのアップグレード*

リリース後に本文を技術的に変更した箇所は、その箇所の左側に縦線 (|) 引いて示しています。

# 第 1 章 Java の入門

企業で Java を使用する場合、CICS によって、CICS 領域の制御下にある Java 仮想マシン (JVM) で Java アプリケーションを開発し、実行するためのツールおよびランタイム環境が提供されます。JVM サーバーで実行できる Java ワークロードは、zEnterprise® Application Assist Processor (zAAP) で実行できます。

## Java アプリケーション開発

OSGi Service Platform に準拠する、再使用可能なモジュラー Java アプリケーションを作成できます。これらのアプリケーションは、CICS と他のプラットフォーム間での移植が容易であり、OSGi は、依存関係とバージョンの管理に細分性を提供します。

Java CICS (JCICS) API を使用すると、ファイルまたは一時記憶域キューからの読み取りなどの CICS サービスにアクセスするアプリケーションを作成できます。Java アプリケーションは、他の CICS アプリケーションにリンクでき、DB2® および IMS™ 内のデータにアクセスすることができます。 Java アプリケーションは、JVM サーバーで実行します。

Liberty プロファイルで提供されているさまざまな Web ツールを使用することにより、アプリケーションの Web プレゼンテーション層を作成できます。 CICS では、JSP ページと Web サーブレットを、同じ JVM サーバー上でアプリケーションとして実行できます。

## Axis2 JVM サーバーにおける Web サービス

異機種混合環境でサービス・プロバイダーとサービス・リクエスターを処理するための Java Web サービスを作成できます。 Java Web サービスは JVM サーバーで実行され、SOAP 処理は Liberty JVM サーバーの Apache CXF Web サービス・エンジン、または JVM サーバーの Apache Axis2 Web サービス・エンジンによって実行されます。また、標準の Java API とアノテーションを使用して、Java Web サービスの作成、データ変換の実行、XML の処理、または構造化データの処理を行えます。

## Java における CICS との接続

JCA (Java コネクター・アーキテクチャー) を使用することで、CICS Transaction Gateway を介して、外部の Java アプリケーションを CICS に接続できます。 JCA は、外部の Java 環境から CICS アプリケーションを呼び出すための関連テクノロジーです。この方法で呼び出される CICS アプリケーションは Java で、または他のサポートされている言語で実装できます。

2 ページの『CICS における Java サポート』
CICS は、CICS 領域の制御下にある Java 仮想マシン (JVM) で Java エンタープライズ・アプリケーションを開発し、実行するためのツールおよびランタイム環境を提供します。Java アプリケーションは、CICS サービスおよび他の言語で作成されたアプリケーションと対話できます。

16 ページの『OSGi に準拠する Java アプリケーション』
CICS には、JVM サーバーで OSGi 仕様に従う Java アプリケーションを実行するために、OSGi フレームワークの Equinox 実装環境が組み込まれています。

20 ページの『Liberty JVM サーバーでの Java アプリケーション』
CICS では、軽量の Java サーブレットおよび JavaServer Pages を実行できる Web コンテナーが提供されています。開発者は、Java サーブレットおよび JSP 仕様の豊富な機能を使用することにより、CICS のための最新式の Web アプリケーションを作成できます。 Web コンテナーは JVM サーバーの中で実行され、WebSphere® Application Server Liberty プロファイル・テクノロジーに基づいて作成されています。

22 ページの『Java Web サービス』
CICS には、Java Web サービスを実行するための Axis2 テクノロジーが組み込まれています。Axis2 は、Apache Foundation からのオープン・ソース Web サービス・エンジンであり、Java 環境で SOAP メッセージを処理するために CICS に提供されています。

**関連情報**:

JVM サーバーのセットアップ
JVM サーバー内で Java アプリケーション、Web アプリケーション、Axis2、または CICS セキュリティー・トークン・サービスを実行するには、CICS リソースをセットアップして、JVM にオプションを渡す JVM プロファイルを作成する必要があります。

## CICS における Java サポート

CICS は、CICS 領域の制御下にある Java 仮想マシン (JVM) で Java エンタープライズ・アプリケーションを開発し、実行するためのツールおよびランタイム環境を提供します。Java アプリケーションは、CICS サービスおよび他の言語で作成されたアプリケーションと対話できます。

z/OS® 上の Java は、Java アプリケーションを実行するための包括的なサポートを提供します。CICS では、IBM 64-bit SDK for z/OS, Java テクノロジー・エディション、バージョン 7、バージョン 7 リリース 1、またはバージョン 8 を使用します。 SDK には、Java API のフルセットおよび 1 組の開発ツールをサポートする Java ランタイム環境が含まれています。汎用プロセッサーの生産性を高めて z/OS Java テクノロジー・ベースのアプリケーションのコンピューティングにかかる全体的なコストを削減することができるように、特定の z Systems™ ハードウェアで特殊なプロセッサーを選択できます。 CICS の Java ワークロードなどの適格な Java ワークロードを実行するために、IBM zEnterprise Application Assist Processor (zAAP) および IBM z Systems Integrated Information Processor (zIIP) は追加のプロセッサー容量を提供できます。 Java Standard Edition Products on z/OS で、z/OS プラットフォーム上の Java に関する詳細情報を見つけ、64 ビット・バージョンの SDK をダウンロードすることができます。

CICS は Eclipse ベースのツール、および Java アプリケーション用の JVM サーバー・ランタイム環境を提供します。

**CICS Explorer® SDK**

CICS Explorer SDK は、Eclipse ベースの統合開発環境 (IDE) 用に自由にダウンロードできます。SDK は、OSGi Service Platform 仕様に従うアプリケーションの開発とデプロイをサポートします。OSGi Service Platform は、コンポーネント・モデルを使用してアプリケーションを開発し、それらのアプリケーションを OSGi バンドルとしてフレームワークにデプロイするためのメカニズムを提供します。*OSGi バンドル* は、アプリケーション・コンポーネントのデプロイメントの単位であり、バージョン管理情報、依存関係、およびアプリケーション・コードが入っています。OSGi の主な利点は、*OSGi サービス* と呼ばれる明確に定義されたインターフェースからのみアクセスされる再使用可能コンポーネントから、アプリケーションを作成できることです。また、Java アプリケーションのライフサイクルと依存関係をきめ細かく管理することもできます。

CICS Explorer SDK は、サポートされている任意のリリースの CICS 用に Java アプリケーションの開発をサポートします。 SDK には、CICS サービスにアクセスするためのクラス、および CICS 用のアプリケーション開発に取り掛かるためのサンプルから成る、Java CICS (JCICS) ライブラリーが組み込まれています。また、このツールを使用すると、既存の Java アプリケーションを OSGi に変換することもできます。

CICS Explorer SDK は、CICS にデプロイ可能な CICS バンドルへの Liberty アプリケーションのパッケージ化をサポートします。

**JVM サーバー**

JVM サーバーは、CICS における Java アプリケーション用の戦略的なランタイム環境です。JVM サーバーは、単一の JVM でさまざまな Java アプリケーションからの多数の並行要求を処理できます。 JVM サーバーを使用することにより、CICS 領域で Java アプリケーションを実行するために必要な JVM の数が減ります。 JVM サーバーを使用するには、Java アプリケーションはスレッド・セーフであり、OSGi 仕様に従う必要があります。JVM サーバーには、次のような利点があります。

- 適格な Java ワークロードを zAAP および zIIP プロセッサーで実行し、トランザクションのコストを削減できます。
- スレッド・セーフ Java プログラムや Web サービスなどの異なるタイプの作業を 1 つの JVM サーバーで実行できます。
- JVM サーバーを再始動することなく、OSGi フレームワークでアプリケーションのライフサイクルを管理できます。
- CICS とその他のプラットフォームとの間で、OSGi を使用してパッケージされた Java アプリケーションをより容易に移植できます。
- Liberty JVM サーバーの中に Web アプリケーションをデプロイできます。

4 ページの『OSGi Service Platform』
OSGi Service Platform は、コンポーネント・モデルを使用してアプリケーションを開発し、それらのアプリケーションを OSGi フレームワークにデプロイするためのメカニズムを提供します。OSGi アーキテクチャーは、Java アプリケーションの作成と管理に役立つ機能を備えた複数のレイヤーに分けられます。

6 ページの『JVM サーバー・ランタイム環境』
*JVM サーバー* は、単一の JVM でさまざまな Java アプリケーションに対する

多数の並行要求を処理できるランタイム環境です。 JVM サーバーを使用すると、OSGi フレームワークでスレッド・セーフ Java アプリケーションを実行し、Liberty プロファイルで Web アプリケーションを実行し、Axis2 Web サービス・エンジンで Web サービス要求を処理することができます。

9 ページの『JVM プロファイル』
JVM プロファイルは、JVM の特性を決定する Java ランチャー・オプションとシステム・プロパティーが入っているテキスト・ファイルです。任意の標準テキスト・エディターを使用して JVM プロファイルを編集することができます。

10 ページの『JVM の構造』
CICS で実行される JVM は、JVM プロファイルで定義される 1 組のクラスとクラスパスを使用し、64 ビット・ストレージを使用します。各 JVM が実行される Language Environment エンクレーブを調整すると、MVS ストレージを最も効率よく使用することができます。

13 ページの『CICS タスクとスレッドの管理』
CICS はオープン・トランザクション環境 (OTE) を使用して JVM サーバーの作業を実行します。各タスクは、JVM サーバーのスレッドとして実行され、1 つの T8 TCB を使用して接続されます。OSGi を使用する主な利点は、OSGi フレームワークのアプリケーションが ExecutorService を使用して、CICS で追加タスクを非同期に実行するスレッドを作成できることです。CICS では、特殊な手段によってランナウェイ・タスクが処理されます。

15 ページの『共用クラス・キャッシュ』
共用クラス・キャッシュでは、単一のキャッシュに保管されている Java アプリケーション・クラスを複数の JVM が共用するためのメカニズムが提供されます。 IBM SDK for z/OS は共用クラス・キャッシュをサポートします。クラス・キャッシュは、OSGi JVM サーバー、非 OSGi JVM サーバー (Apache Axis2 など) と共に使用できます。

## OSGi Service Platform

OSGi Service Platform は、コンポーネント・モデルを使用してアプリケーションを開発し、それらのアプリケーションを OSGi フレームワークにデプロイするためのメカニズムを提供します。OSGi アーキテクチャーは、Java アプリケーションの作成と管理に役立つ機能を備えた複数のレイヤーに分けられます。

OSGi フレームワークは、OSGi Service Platform 仕様の中核をなします。CICS は OSGi フレームワークの Equinox 実装環境を使用します。 OSGi フレームワークは、JVM サーバーの始動時に初期化されます。Java アプリケーションに OSGi を使用すると、次の主な利点があります。

- 新規 Java アプリケーション、および Java アプリケーションの新規バージョンを、稼働中の実動システムにデプロイできます。その際に JVM を再始動する必要はなく、その JVM にデプロイされている他の Java アプリケーションに影響を与えることはありません。
- Java アプリケーションの移植可能性が向上し、リエンジニアリングが容易になり、要件の変化への適応性が高まります。
- Plain Old Java Object (POJO) プログラミング・モデルに従って、動的なライフサイクルを持つ 1 組の OSGi バンドルとしてアプリケーションをデプロイすることを選択できます。

- アプリケーション・バンドルの依存関係とバージョンの管理が容易になります。

OSGi アーキテクチャーには、以下のレイヤーがあります。

- モジュール・レイヤー
- ライフサイクル・レイヤー
- サービス・レイヤー

## モジュール・レイヤー

デプロイメントの単位は OSGi バンドルです。モジュール・レイヤーでは、OSGi フレームワークがバンドルのモジュラー・アスペクトを処理します。OSGi フレームワークがこの処理を実行できるようにするメタデータは、バンドルのマニフェスト・ファイルで提供されます。

OSGi の主な利点の 1 つは、クラス・ローダー・モデルで、これは、マニフェスト・ファイルのメタデータを使用します。OSGi にはグローバル・クラスパスはありません。バンドルが OSGi フレームワークにインストールされると、それらのメタデータはモジュール・レイヤーによって処理され、宣言された外部依存関係は、インストールされた他のモジュールによって宣言されたエクスポートおよびバージョン情報に照らして調整されます。OSGi フレームワークは、すべての依存関係を解明し、バンドルごとに独立した必須のクラスパスを計算します。この方法により、以下の要件が満たされるようになるため、単純な Java クラス・ロードの短所が解決されます。

- 各バンドルは、それが明示的にエクスポートする Java パッケージからのみ可視である。
- 各バンドルが、明示的にそのパッケージ依存関係を宣言する。
- パッケージを特定のバージョンでエクスポートし、特定のバージョンで、または特定の範囲のバージョンからインポートできる。
- パッケージの複数のバージョンが異なるクライアントに対して同時に使用可能である。

## ライフサイクル・レイヤー

OSGi におけるバンドルのライフサイクル管理レイヤーは、JVM のライフサイクルとは無関係に、バンドルを動的にインストール、開始、停止、およびアンインストールすることを可能にします。ライフサイクル・レイヤーは、バンドルが、その依存関係がすべて解決される場合のみ開始することを確実にし、実行時の ClassNotFoundException 例外の発生を減らします。未解決の依存関係がある場合、OSGi フレームワークは問題を報告し、バンドルを開始しません。

各バンドルは、バンドル・マニフェストで識別されるバンドル・アクティベーター・クラスを提供できます。フレームワークはこのクラスにイベントの開始および停止を要求します。

## サービス・レイヤー

OSGi のサービス・レイヤーは、本来、非永続サービス・レジストリー・コンポーネントを使用してサービス指向アーキテクチャーをサポートします。バンドルはサービス・レジストリーにサービスを公開し、他のバンドルがそれらのサービスをサー

ビス・レジストリーからディスカバーできます。これらのサービスは、バンドル相互間のコラボレーションの 1 次的な手段です。OSGi サービスは、1 つ以上の Java インターフェース名でサービス・レジストリーに公開される Plain Old Java Object (POJO) であり、オプションのメタデータがカスタム・プロパティー (名前/値のペア) として保管されます。ディスカバーする側のバンドルは、インターフェース名によってサービス・レジストリーでサービスを検索することができ、カスタム・プロパティーに基づいて検索されるサービスを潜在的にフィルターに掛けることができます。

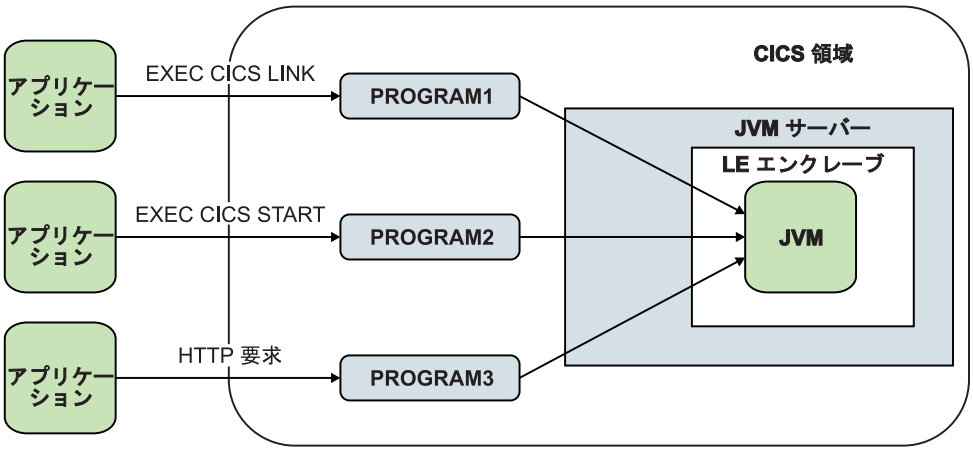
サービスは完全に動的であり、通常、それらのサービスを提供するバンドルと同じライフサイクルを持ちます。

## JVM サーバー・ランタイム環境

*JVM サーバー* は、単一の JVM でさまざまな Java アプリケーションに対する多数の並行要求を処理できるランタイム環境です。 JVM サーバーを使用すると、OSGi フレームワークでスレッド・セーフ Java アプリケーションを実行し、Liberty プロファイルで Web アプリケーションを実行し、Axis2 Web サービス・エンジンで Web サービス要求を処理することができます。

JVM サーバーは JVMSERVER リソースで表されます。JVMSERVER リソースを使用可能にすると、CICS は MVS™ にストレージを要求し、Language Environment® エンクレーブをセットアップし、そのエンクレーブで 64 ビット JVM を起動します。CICS は、JVMSERVER リソースで指定された JVM プロファイルを使用して、正しいオプションを持つ JVM を作成します。このプロファイルで、JVM オプションやシステム・プロパティーを指定したり、ネイティブ・ライブラリーを追加したりすることができます。例えば、Java アプリケーションから DB2 または WebSphere MQ にアクセスするためのネイティブ・ライブラリーを追加することができます。

JVM サーバーを使用する利点の 1 つは、同一 JVM でさまざまなアプリケーションに対して多数の要求を実行できることです。次の図では、3 つのアプリケーションが、別々のアクセス方式を使用して、CICS 領域内の 3 つの Java プログラムを同時に呼び出しています。各 Java プログラムは同じ JVM サーバーで実行されます。



## Java アプリケーション

JVM サーバーで Java アプリケーションを実行するには、その Java アプリケーションがスレッド・セーフであり、1 つの CICS バンドル内の 1 つ以上の OSGi バンドルとしてパッケージされなければなりません。JVM サーバーは、OSGi バンドルおよび、OSGi サービスを実行できる OSGi フレームワークを実装します。OSGi フレームワークは、サービスを登録し、バンドル相互間の依存関係とバージョンを管理します。OSGi は、フレームワーク内のすべてのクラスパス管理を処理するので、JVM サーバーの停止と再始動を行うことなく、Java アプリケーションを追加、更新、および除去できます。

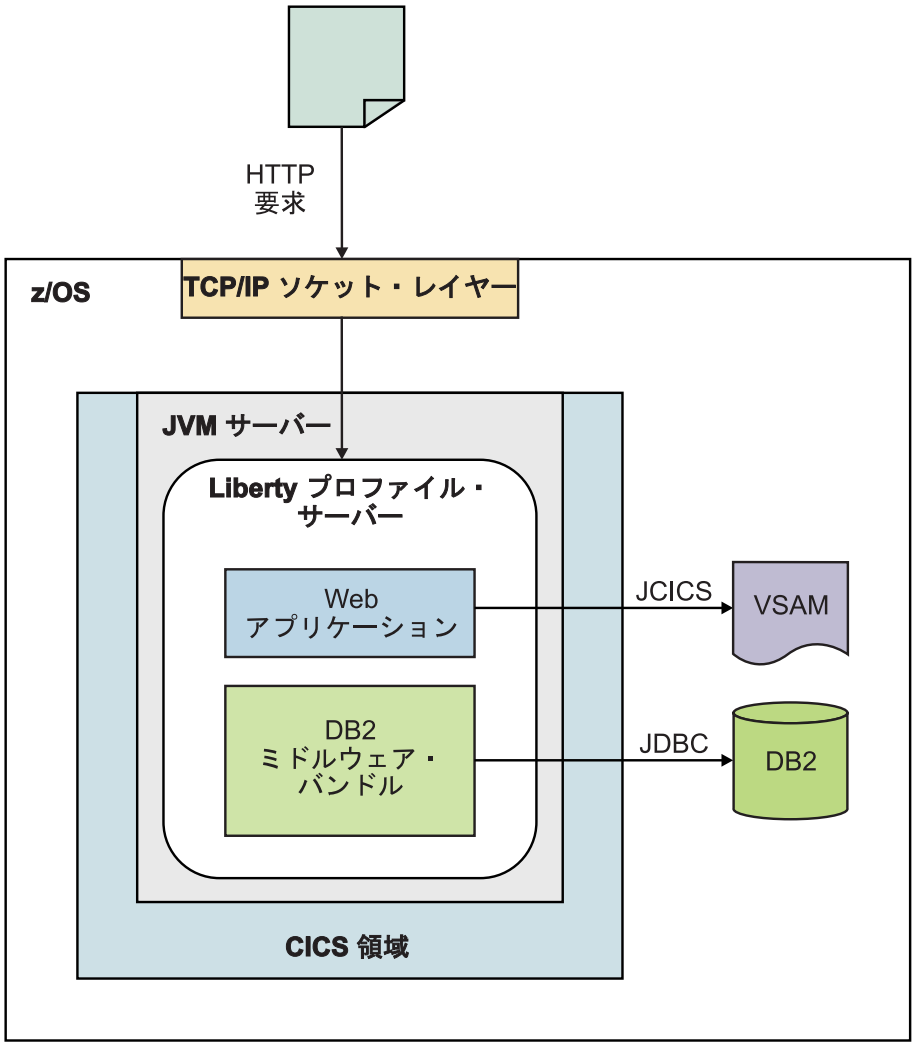
OSGi を使用してパッケージされる Java アプリケーションのデプロイメントの単位は、CICS バンドルです。 BUNDLE リソースは、アプリケーションを CICS に対して表現するもので、これを使用してアプリケーションのライフサイクルを管理することができます。CICS Explorer SDK は、CICS バンドル・プロジェクト内の OSGi バンドルを zFS にデプロイするためのサポートを提供します。

OSGi フレームワーク外部から Java アプリケーションにアクセスするには、PROGRAM リソースを使用して、アプリケーションが実行される JVM サーバー、および OSGi サービスの名前を識別してください。OSGi サービスは、CICS メイン・クラスを指します。

JVM サーバーにおける OSGi フレームワークの使用について詳しくは、 16 ページの『OSGi に準拠する Java アプリケーション』を参照してください。

## Java Web アプリケーション

JVM サーバーでは、OSGi フレームワークでの Java アプリケーションの実行に加えて、WebSphere Application Server Liberty プロファイルの実行もサポートされます。 Liberty プロファイルは Web アプリケーションを実行するための軽量のアプリケーション・サーバーです。 Web アプリケーションは JCICS を使用して CICS 内のリソースとサービスにアクセスし、さらに DB2 内のデータにアクセスすることもできます。 Liberty プロファイル・サーバーで実行されるアプリケーションは、CICS Web サポートを通してではなく、z/OS の TCP/IP ソケット・レイヤーを通してアクセスされます。




Java Web アプリケーションのデプロイメントは、開発者が Liberty サーバーのドロップイン・ディレクトリーに Web アーカイブ (WAR) ファイルまたはエンタープライズ・アーカイブ (EAR) ファイルを直接デプロイできる Liberty モデルに従って行うことも、CICS バンドルを作成する CICS アプリケーション・モデルを使用して行うこともできます。 CICS バンドルでは、ライフサイクル管理が提供され、OSGi バンドルや WAR ファイルなどの多数のコンポーネントを一緒に含む 1 つのアプリケーションをパッケージできます。

Web アプリケーションから OSGi バンドルにアクセスするには、Enterprise Bundle Archive (EBA) ファイルとしてアプリケーションをデプロイする必要があります。EBA を開発するには、Rational® Application Developer を使用するか、または Eclipse IDE、CICS Explorer SDK、および IBM WebSphere Application Server Developer Tools for Eclipse バージョン 8.5.5 を組み合わせて使用することができます。後者のツール・セットは自由に使用可能ですが、CICS Explorer SDK を除いて IBM サポートの対象外となります。

Liberty プロファイル・サーバーの使用について詳しくは、 20 ページの『Liberty JVM サーバーでの Java アプリケーション』を参照してください。

### Web サービス

JVM サーバーを使用して、Web サービス・リクエスターおよび Web サービス・プロバイダーのアプリケーションの SOAP 処理を実行できます。 Java に基づく SOAP エンジンである Axis2 をパイプラインで使用する場合、SOAP 処理は JVM サーバーで実行されます。 Web サービスに JVM サーバーを使用する利点は、作業を zAAP プロセッサーにオフロードできることです。

Web サービスに対する JVM サーバーの使用について詳しくは、 22 ページの『Java Web サービス』を参照してください。

## JVM プロファイル

JVM プロファイルは、JVM の特性を決定する Java ランチャー・オプションとシステム・プロパティーが入っているテキスト・ファイルです。任意の標準テキスト・エディターを使用して JVM プロファイルを編集することができます。

CICS が Java プログラムの実行要求を受け取ると、JVM プロファイルの名前が Java ランチャーに渡されます。 JVM プロファイルのオプションを使用して作成された JVM で、Java プログラムが実行されます。

CICS が使用する JVM プロファイルは、JVMPROFILEDIR システム初期設定パラメーターによって指定される z/OS UNIX システム・サービス・ディレクトリー内にあります。このディレクトリーには、CICS が JVM プロファイルを読み取るための適切な権限が必要です。

### サンプル JVM プロファイル

CICS には、Java 環境の構成に役立ついくつかのサンプル JVM プロファイルが含まれています。これらのプロファイルは、CICS のインストール処理時にカスタマイズされます。これらのファイルは、CICS でデフォルトとして使用されるか、システム・プログラムに使用されます。

JVM プロファイルは、Java 用の CICS ランチャーで使用されるオプションをリストします。CICS に固有のオプションもあれば、JVM ランタイム環境に標準のオプションもあります。例えば、JVM プロファイルは、ストレージ・ヒープの初期サイズや拡張できる程度を制御します。また、プロファイルは、JVM によって作成されるメッセージやダンプ出力の宛先を定義することもできます。JVM プロファイルは JVMSERVER リソース定義内の JVMPROFILE 属性で指定されます。 JVMSERVER 属性を参照してください。

これらのサンプルをコピーし、独自のアプリケーションに合わせてカスタマイズすることができます。 CICS で提供されるサンプル JVM プロファイルは、z/OS UNIX 上の `/usr/lpp/cicsts/cicsts53/JVMProfiles` ディレクトリーにあります。これらのサンプルをインストール・ディレクトリーから、**JVMPROFILEDIR** システム初期設定パラメーターで指定されたディレクトリーにコピーしてください。インストール場所にあるサンプル JVM プロファイルは、これらのファイルの変更を含む

APAR を適用すると、上書きされます。変更内容が失われないように、必ず、サンプルを別の場所にコピーしてから、独自のアプリケーション・クラスの追加またはオプションの変更を行ってください。

次の表は、各サンプル JVM プロファイルの主な特性をまとめています。

*表 2. CICS で提供されるサンプル JVM プロファイル*

| JVM プロファイル | 特性 |
|---|---|
| DFHJVMAX.jvmprofile | Axis2 JVM サーバー用に提供されるサンプル・プロファイル。この JVM プロファイルは JVMSERVER リソースで指定されます。CICS は DFHJVMAX.jvmprofile を使用して JVM サーバーを初期化します。 |
| DFHJVMST.jvmprofile | セキュリティー・トークン・サービスに関して JVM サーバー用に提供されているサンプル・プロファイル。この JVM プロファイルは JVMSERVER リソースで指定されます。CICS は DFHJVMST.jvmprofile を使用して JVM サーバーを初期化します。 |
| DFHOSGI.jvmprofile | OSGi JVM サーバー用に提供されるサンプル・プロファイル。この JVM プロファイルは JVMSERVER リソースで指定されます。CICS は DFHOSGI.jvmprofile を使用して JVM サーバーを初期化します。 |
| DFHWLP.jvmprofile | Liberty JVM サーバー用に提供されるサンプル・プロファイル。この JVM プロファイルは JVMSERVER リソースで指定されます。CICS は DFHWLP.jvmprofile を使用して Liberty JVM サーバーを初期化します。 |

## JVM の構造

CICS で実行される JVM は、JVM プロファイルで定義される 1 組のクラスとクラスパスを使用し、64 ビット・ストレージを使用します。各 JVM が実行される Language Environment エンクレーブを調整すると、MVS ストレージを最も効率よく使用することができます。

IBM 64-bit SDK for z/OS, Java テクノロジー・エディション について詳しくは、IBM SDK, Java Technology Edition バージョン 7 の z/OS ユーザーズ・ガイドまたはIBM SDK, Java Technology Edition バージョン 8 の z/OS ユーザーズ・ガイドを参照してください。

11 ページの『JVM におけるクラスおよびクラスパス』
CICS で実行される JVM では、さまざまなタイプのクラスまたはライブラリー・ファイルを使用できます。それには、原始クラス (システム・クラスと標準拡張クラス)、ネイティブ C DLL ライブラリー・ファイル、およびアプリケーション・クラスがあります。

12 ページの『JVM におけるストレージ・ヒープ』
ランタイム JVM ストレージは単一の 64 ビット・ストレージ・ヒープによって管理されます。

12 ページの『JVM が構成される場所』
JVM が必要な場合、JVM の CICS ランチャー・プログラムは MVS にストレージを要求し、Language Environment エンクレーブをセットアップし、その Language Environment エンクレーブで JVM を起動します。並行して実行される JVM 間の分離を確実にするために、各 JVM は独自の Language Environment エンクレーブ内で構成されます。

13 ページの『JVM および z/OS 共用ライブラリー領域』
共用ライブラリー領域は、アドレス・スペースがダイナミック・リンク・ライブラリー (DLL) ファイルを共用できるようにする z/OS 機能です。

## JVM におけるクラスおよびクラスパス

CICS で実行される JVM では、さまざまなタイプのクラスまたはライブラリー・ファイルを使用できます。それには、原始クラス (システム・クラスと標準拡張クラス)、ネイティブ C DLL ライブラリー・ファイル、およびアプリケーション・クラスがあります。

JVM はこれらの各コンポーネントの目的を認識し、コンポーネントのロード方法を決定し、保管先を決定します。 JVM のクラスパスは、JVM プロファイル内のオプションによって定義され、(オプションとして) JVM プロパティー・ファイルで参照されます。

- *原始クラス* は、JVM で基本サービスを提供する z/OS JVM コードです。原始クラスはシステム・クラスと標準拡張クラスに分類できます。
- z/OS UNIX では、*ネイティブ C ダイナミック・リンク・ライブラリー (DLL) ファイル* には拡張子 `.so` が付いています。 JVM の実行には何らかのライブラリーが必要であり、アプリケーション・コードまたはサービスによって追加のネイティブ・ライブラリーをロードすることができます。例えば、追加のネイティブ・ライブラリーには、DB2 JDBC ドライバーを使用する DLL ファイルを含めることができます。
- *アプリケーション・クラス* は、JVM で実行されるアプリケーション用のクラスであり、ユーザー作成アプリケーションに属するクラスを含みます。また、Java アプリケーション・クラスには、JCICS インターフェース・クラス、JDBC、JNDI などのリソースにアクセスするサービスを提供するための IBM 提供または他のベンダー提供のクラスも含まれます (JDBC と JNDI は CICS 用の標準 JVM セットアップに含まれていません)。クラス・キャッシュにロードされた Java アプリケーション・クラスは保持されるため、同じ JVM で実行される他のアプリケーションはこれらを再使用できます。

クラスまたはネイティブ・ライブラリーを指定できるクラスパスは、ライブラリー・パスと標準クラスパスです。

- *ライブラリー・パス* は、JVM が使用するネイティブ C ダイナミック・リンク・ライブラリー (DLL) ファイルを指定します。これには、アプリケーション・コードまたはサービスによってロードされる JVM および追加のネイティブ・ライブラリーの実行に必要なファイルが含まれます。各 DLL ファイルの 1 つのコピーのみがロードされ、すべての JVM がそのコピーを共用しますが、各 JVM には、その DLL 用に静的データ域の独自のコピーがあります。

  JVM の基本ライブラリー・パスは、**USSHOME** システム初期設定パラメーターと JVM プロファイルの **JAVA_HOME** オプションで指定されたディレクトリーを使用して自動的に作成されます。この基本ライブラリー・パスは、JVM プロファイルでは表示されません。このライブラリー・パスには、CICS が使用する JVM とネイティブ・ライブラリーを実行するのに必要なすべての DLL ファイルが含まれています。 **LIBPATH_SUFFIX** オプションまたは **LIBPATH_PREFIX** オプションを使用して、このライブラリー・パスを拡張できます。 **LIBPATH_SUFFIX** は、ライブラリー・パスの終わりに、IBM 提供のライブラリーの後に項目を追加します。

**LIBPATH_PREFIX** は、項目を先頭に追加し、同じ名前である場合は IBM 提供のライブラリーの代わりにロードされます。問題判別の目的でそれを実行することが必要になる場合もあります。

ライブラリー・パスに組み込むすべての DLL ファイルを、LP64 オプションを使用してコンパイルし、リンクしてください。基本ライブラリー・パスで提供される DLL ファイルおよび DB2 JDBC ドライバーなどのサービスで使用される DLL ファイルは、LP64 オプションを指定して作成されます。

- *標準クラスパス* を OSGi が有効な JVM サーバーで使用しないでください。OSGi フレームワークは、アプリケーションを含む OSGi バンドル内の情報に基づいて、アプリケーションのクラスパスを自動的に判別するからです。 CICS の Axis2 環境など、OSGi 用に構成されていない JVM サーバーで使用するために、標準クラスパスが保持されます。標準クラスパスが使用されている Axis2 の場合など、例外的なシナリオでは、クラスパス・エントリーにワイルドカード接尾部を使用して、特定のディレクトリー内のすべての JAR ファイルを指定できます。

  また、CICS は、**USSHOME** システム初期設定パラメーターで指定されたディレクトリーの `/lib` サブディレクトリーを使用して、JVM の基本クラスパスを自動的に作成します。このクラスパスには、CICS と JVM によって用意されている JAR ファイルが入ります。それは JVM プロファイルでは見られません。

システム・クラスと標準拡張クラス (原始クラス) は既に JVM のブート・クラスパスに含まれているので、これらをクラスパスに組み込む必要はありません。

## JVM におけるストレージ・ヒープ

ランタイム JVM ストレージは単一の 64 ビット・ストレージ・ヒープによって管理されます。

各 JVM のヒープは、JVM の Language Environment エンクレーブにある 64 ビット・ストレージから割り振られます。各ヒープのサイズは、JVM プロファイル内のオプションによって決まります。

単一のストレージ・ヒープはヒープ と呼ばれ、場合によってはガーベッジ・コレクション・ヒープ と呼ばれます。その初期ストレージ割り振りは、JVM プロファイルの **-Xms** オプションによって設定され、その最大サイズは **-Xmx** オプションによって設定されます。

ヒープのサイズを調整すると、JVM の最適なパフォーマンスを実現することができます。 219 ページの『JVM サーバー・ヒープとガーベッジ・コレクションの調整』を参照してください。

## JVM が構成される場所

JVM が必要な場合、JVM の CICS ランチャー・プログラムは MVS にストレージを要求し、Language Environment エンクレーブをセットアップし、その Language Environment エンクレーブで JVM を起動します。並行して実行される JVM 間の分離を確実にするために、各 JVM は独自の Language Environment エンクレーブ内で構成されます。

Language Environment エンクレーブは、Language Environment 事前初期設定モジュール CELQPIPI を使用して作成され、JVM は z/OS UNIX プロセスとして実行されます。したがって、JVM は、CICS Language Environment サービスではなく、MVS Language Environment サービスを使用します。JVM に使用されるストレージは、MVS Language Environment サービスの呼び出しによって取得された MVS 64 ビット・ストレージです。このストレージは、CICS アドレス・スペース内に置かれますが、CICS 動的ストレージ域 (DSA) には含まれません。

JVM の Language Environment エンクレーブは、その JVM のストレージ要件に応じて拡張することができます。Language Environment エンクレーブに CICS が使用する Language Environment ランタイム・オプションは、Language Environment エンクレーブのヒープ・ストレージの初期サイズおよび追加増分を制御します。

Language Environment エンクレーブに CICS が使用するランタイム・オプションを調整することができます。その結果、CICS がそのエンクレーブのために要求するストレージ量は、JVM プロファイルで指定されたストレージ量と可能なかぎり近くなります。したがって、MVS ストレージを最も効率よく使用することができます。ストレージの調整について詳しくは、220 ページの『JVM の Language Environment エンクレーブ・ストレージ』を参照してください。

### JVM および z/OS 共用ライブラリー領域

共用ライブラリー領域は、アドレス・スペースがダイナミック・リンク・ライブラリー (DLL) ファイルを共用できるようにする z/OS 機能です。

この機能により、 CICS 領域は JVM に必要な DLL を共用できるようになり、各領域が DLL を個別にロードする必要はなくなります。これにより、MVS が使用する実際のストレージの量、および領域へのファイルのロード所要時間を大幅に削減できます。

共用ライブラリー領域用に予約されているストレージは、最初の JVM が領域で開始されるときにそれぞれの CICS 領域に割り振られます。割り振られるストレージの量は、z/OS の **SHRLIBRGNSIZE** パラメーターによって制御されます。共用ライブラリー領域に割り振られるストレージ量の調整について詳しくは、228 ページの『z/OS 共用ライブラリー領域の調整』を参照してください。

## CICS タスクとスレッドの管理

CICS はオープン・トランザクション環境 (OTE) を使用して JVM サーバーの作業を実行します。各タスクは、JVM サーバーのスレッドとして実行され、1 つの T8 TCB を使用して接続されます。OSGi を使用する主な利点は、OSGi フレームワークのアプリケーションが ExecutorService を使用して、CICS で追加タスクを非同期に実行するスレッドを作成できることです。CICS では、特殊な手段によってランナウェイ・タスクが処理されます。

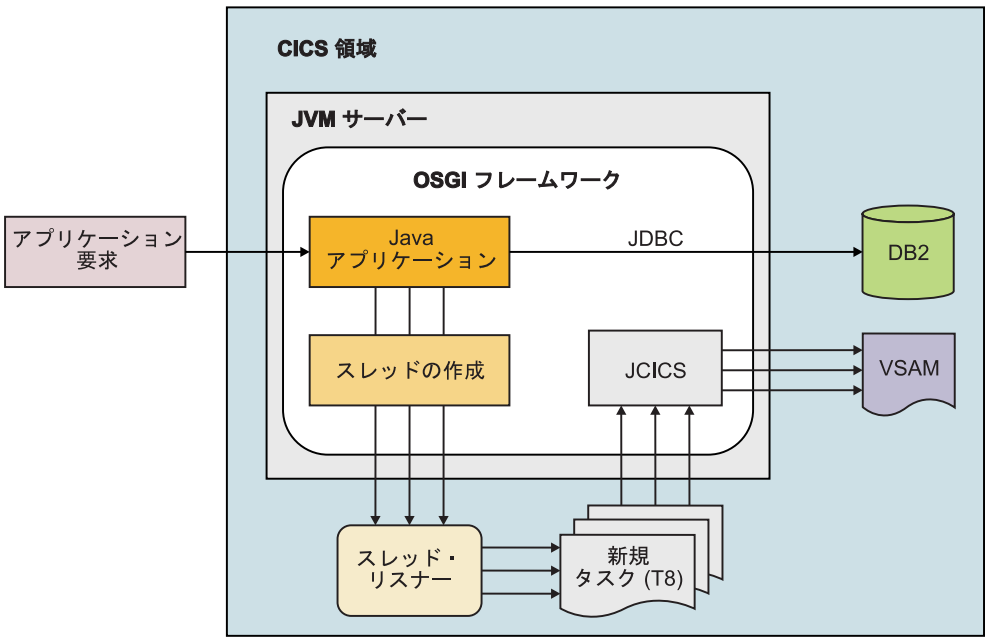
CICS で JVM サーバーを有効にすると、JVM サーバーは言語環境プロセス・スレッドで実行されます。このスレッドは TP TCB の子です。各 CICS タスクは、1 つの T8 TCB を使用して JVM のスレッドに接続されます。 JVMSERVER リソースの THREADLIMIT 属性を設定すると、JVM サーバーから使用可能な T8 TCB の数を制御できます。

JVM サーバー用に作成される T8 TCB は、仮想プールに存在し、同じ CICS 領域で実行される別の JVM サーバーで再利用することはできません。 1 つの CICS 領域内に存在できる T8 TCB の最大数は、すべての JVM サーバーを合わせて 2000 です。また、特定の 1 つの JVM サーバーでの最大数は 256 です。

## マルチスレッド・アプリケーション

OSGi フレームワークで実行される Java アプリケーションは、ExecutorService OSGi サービスを使用して CICS タスクを非同期に開始することもできます。 JVM サーバーは、始動時に ExecutorService を OSGi サービスとして登録します。 ExecutorService では、CICS によって提供される実装が自動的に使用されます。この実装では、JCICS API を使用して CICS サービスにアクセスできるスレッドが作成されます。この方式により、アプリケーションは、スレッドを作成するために、特定の JCICS API メソッドを使用する必要がなくなります。ただし、アプリケーションは CICSExecutorService を使用して、別個の CICS 対応スレッドの処理を実行することもできます。

JVM サーバーが有効な場合は、このサーバーによって CJSL トランザクションが開始され、JVM サーバー・リスナーという長期実行タスクが作成されます。このリスナーは、アプリケーションからの新しいスレッド要求を待機し、CJSA トランザクションを実行して、T8 TCB でディスパッチされる CICS タスクを作成します。この処理は、以下の図で示されています。




高度なシナリオでは、アプリケーションは OSGi サービスを使用して、多数のスレッドを非同期で実行できます。これらのスレッドはすべて JCICS を介して CICS サービスにアクセスでき、T8 TCB 下で実行されます。

### JVM サーバーの実行キー

Java プログラムは、正しい実行キーで実行される JVM を使用する必要があります。JVM サーバーは CICS キーで実行されます。JVM サーバーを使用するには、Java プログラムの PROGRAM リソースで、EXECKEY 属性が CICS に設定されなければなりません。 CICS は、T8 TCB を使用して JVM を実行し、CICS キーの MVS ストレージを取得します。

### ランナウェイ・タスク

CICS JVM サーバー・インフラストラクチャーでは、タスク・ランナウェイ検出メカニズムの使用がサポートされます。 JVM ベースのタスクに、有効なランナウェイ限界より長いプロセッサー制御が保持される場合、JVM の関連スレッドは AICA または同様の異常終了で強制終了されます。

スレッドの強制終了によって、JVM が不整合状態のままになる場合があります。Java プログラミング言語および JVM の設計では、スレッドがこのような方法で停止されることは許可されていません。例えば、強制終了されたスレッドによってロックが保持されていた場合、ファイナライザー・ブロックが実行されない可能性があり、リソースが割り振られたままで、共用ストレージが不整合状態のままになることがあります。アプリケーションとシステムの両方の状態が影響を受けることがあります。

ランナウェイ処理のためにスレッドが強制終了されると、CICS では自動的に JVM サーバーが再始動され、JVM が整合した状態にリカバリーされます。再始動には JVM サーバーの段階的停止も含まれ、その後すぐに再使用可能化されます。 JVM サーバーでは、この期間中、新規作業を処理できません。 CICS はメッセージ DFHSJ1009 を出して、再始動が実行されたことを報告します。

JVM サーバーで実行中のタスクのランナウェイ状態は、同じ JVM サーバーを使用する作業すべてにおいて一時的な可用性問題の原因になることがあります。このため、CICS では、構成されていたランナウェイ・タイムアウト値が 10 倍に乗算され (最大値 45 分まで) 変更されます。この新規の値が、有効ランナウェイ限界です。このようにランナウェイ限界を高くすると、効率の悪い (ただし機能はしている) アプリケーションのためにランナウェイ状態が検出される可能性が減ります。例えば、トランザクション定義で RUNAWAY=SYSTEM が指定されており、ICVR システム初期設定パラメーターが 5000 ミリ秒のデフォルト限界を示している場合、タスクが JVM サーバーで実行されるときの有効ランナウェイ値は 50000 ミリ秒です。

## 共用クラス・キャッシュ

共用クラス・キャッシュでは、単一のキャッシュに保管されている Java アプリケーション・クラスを複数の JVM が共用するためのメカニズムが提供されます。 IBM SDK for z/OS は共用クラス・キャッシュをサポートします。クラス・キャッシュは、OSGi JVM サーバー、非 OSGi JVM サーバー (Apache Axis2 など) と共に使用できます。

Java 7 (CICS ではない) が、JVM サーバーでのクラス・キャッシュ機能の使用をサポートします。したがって、CICS SPI コマンドや CEMT コマンドを使用して、JVM サーバー・クラス・キャッシュを有効にしたり無効にしたりすることはできま

せん。 JVM サーバー・クラス・キャッシュを有効にしたり無効にしたりするには、JVM コマンド行パラメーターを使用します。また、JVM コマンド行パラメーターを使用してクラス・キャッシュのサイズも設定します。

以下のコンポーネント (ファイル、オブジェクト、変数、およびコンパイル済みクラス) はクラス・キャッシュにはロードされません。

- JVM プロファイル・ライブラリー・パスで指定されたネイティブ C DLL ファイル。各 DLL ファイルの単一コピーがファイルへのアクセスを必要とするすべての JVM で使用されるため、これらはロードされません。
- アプリケーション・オブジェクトと変数。作業データが JVM に保管されるため、これらはロードされません。
- ジャストインタイム (JIT) コンパイル済みクラス。コンパイル・プロセスがワークロードによって異なる場合があるため、これらはロードされませんが、JVM には保管されます。

### キャッシュの有効化

デフォルトでは、クラス・キャッシュは JVM で無効になっています。クラス・キャッシュを有効にするには、JVM コマンド行パラメーターを使用します。例を以下に示します。

```
-Xshareclasses:name=cics.&APPLID;
```

### キャッシュ・サイズの指定

クラス・キャッシュのサイズを指定するには、JVM コマンド行パラメーターを使用します。例えば、以下のパラメーターはサイズを 20M に設定します。

```
-Xscmx20M
```

Java クラス・データ共用について詳しくは、JVM 間でのクラス・データの共用を参照してください。

## OSGi に準拠する Java アプリケーション

CICS には、JVM サーバーで OSGi 仕様に従う Java アプリケーションを実行するために、OSGi フレームワークの Equinox 実装環境が組み込まれています。

OSGi Service Platform 仕様は、4 ページの『OSGi Service Platform』 に記述されているように、動的なモジュラー Java アプリケーションを実行し、管理するためのフレームワークを提供します。JVM サーバーのデフォルト構成には、OSGi フレームワークの Equinox 実装環境が組み込まれています。JVM サーバーの OSGi フレームワークにデプロイされる Java アプリケーションは、OSGi を使用する利点、および CICS におけるアプリケーションの実行に固有のサービス品質が得られます。

次のいずれかの理由で Java アプリケーションを使用できます。

- トランザクションのコストを削減するために、zAAP で実行できる Java ワークロードを作成したい。
- 他のプラットフォームで OSGi を使用する Java アプリケーションを作成した経験があり、CICS で Java アプリケーションを作成したい。

- アプリケーションおよびそれらのアプリケーションが実行される JVM の可用性に影響を与えることなく、独立して再使用し、更新できる 1 組のモジュラー・コンポーネントとして、Java アプリケーションを提供したい。

OSGi に準拠する Java アプリケーションを効率よく開発し、デプロイし、管理するには、CICS Explorer SDK と CICS Explorer を使用する必要があります。

- CICS Explorer SDK は、既存の Eclipse 統合開発環境 (IDE) を拡張して、Java 開発者が CICS における Java アプリケーションを作成し、デプロイするのに役立つツールとサポートを提供します。既存の Java アプリケーションを OSGi バンドルに変換するには、このツールを使用します。
- CICS Explorer は、OSGi バンドルと OSGi サービス、およびそれらが実行される JVM サーバーのビューをシステム管理者に提供する、Eclipse ベースのシステム管理者ツールです。Java アプリケーションの使用可能化と使用不可化、フレームワーク内の OSGi バンドルとサービスの状況の確認、および JVM サーバーのパフォーマンスに関する予備統計の取得に、このツールを使用します。

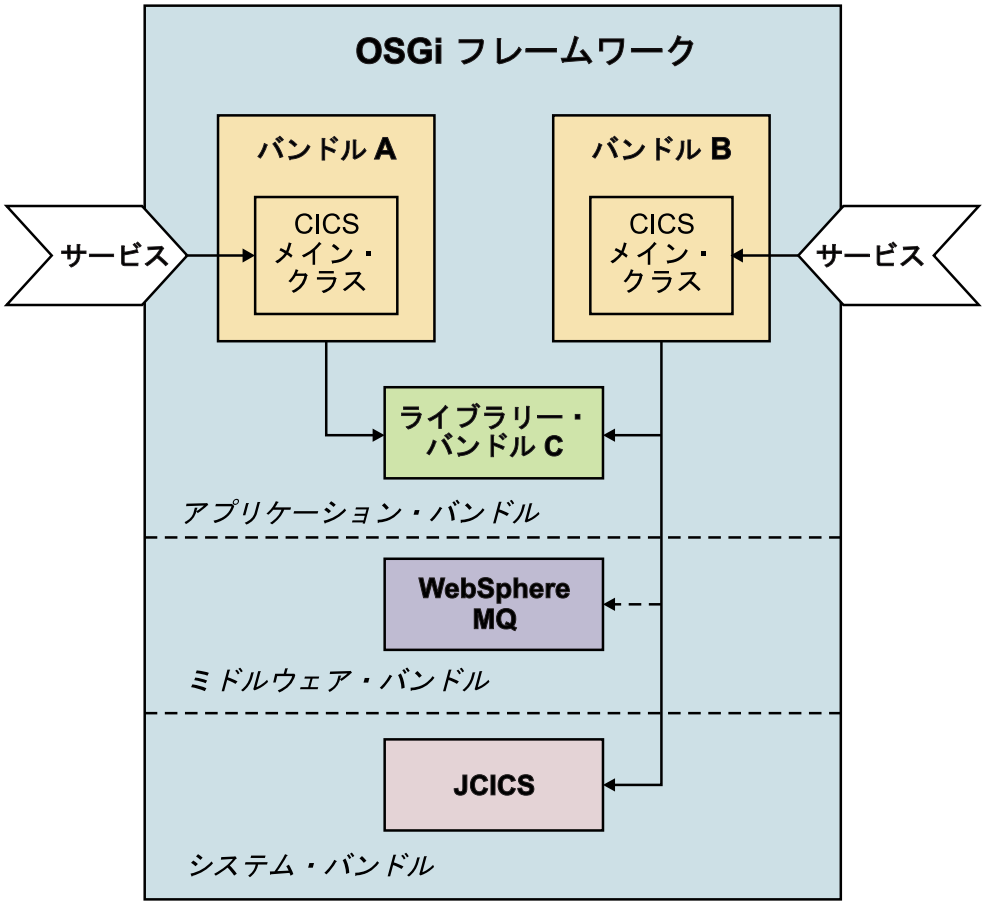
OSGi を使用する Java 開発者またはシステム管理者には、これらの自由に使用できるツールへのアクセス権限が必要です。

以下の例では、CICS において OSGi を使用する Java アプリケーションの実行方法を説明しています。

## 同一 JVM サーバー内の複数の Java アプリケーションの実行

JVM サーバーは、同一 JVM で複数の要求を同時に処理できます。したがって、同一アプリケーションを同時に複数回呼び出したり、同一 JVM サーバーで複数のアプリケーションを実行したりすることができます。

JVM サーバー間でアプリケーションを分ける方法が決定したら、OSGi モデルを使用して、複数のアプリケーションを 1 組の OSGi バンドルとしてコンポーネント化する方法を計画できます。また、アプリケーションにサービスを提供するために、フレームワークで必要な対応 OSGi バンドルを決定することも必要です。次の図に示されているように、OSGi フレームワークには、異なるタイプの OSGi バンドルを含むことができます。




**アプリケーション・バンドル**

アプリケーション・バンドルは、アプリケーション・コードを含む OSGi バンドルです。OSGi バンドルは、自己完結型であるか、フレームワーク内の他のバンドルに依存する場合があります。これらの依存関係はフレームワークによって管理され、未解決の依存関係がある OSGi バンドルはフレームワークで実行できません。CICS においてフレームワークの外部でアプリケーションにアクセスできるようにするために、OSGi バンドルは、CICS メイン・クラスを OSGi サービスとして宣言する必要があります。PROGRAM リソースが CICS メイン・クラスを指し示す場合、OSGi フレームワーク外部の他のアプリケーションは Java アプリケーションにアクセスできます。1 つ以上のアプリケーションの共通ライブラリーを含む OSGi バンドルがある場合、Java 開発者は、CICS メイン・クラスを宣言しないことを決定することがあります。この OSGi バンドルは、フレームワーク内の他の OSGi バンドルからのみ使用可能です。

Java アプリケーションのデプロイメント単位は、CICS バンドルです。CICS バンドルは、任意の数の OSGi バンドルを含むことができ、1 つ以上の JVM サーバーにデプロイできます。JVM サーバーの管理とは無関係に、アプリケーション・バンドルを追加、更新、および除去することができます。

**ミドルウェア・バンドル**

ミドルウェア・バンドルは、WebSphere MQ への接続などのシステム・サービスを実装するためのクラスを含む OSGi バンドルです。もう 1 つの例は、ネイティブ・コードを含み、OSGi フレームワークに 1 回だけロードする必要がある OSGi バンドルです。ミドルウェア・バンドルは、クラスを使用するアプリケーションではなく、JVM サーバーのライフサイクルで管理されます。ミドルウェア・バンドルは、JVM サーバーの JVM プロファイルで指定され、JVM サーバーの始動時に CICS によってロードされます。

**システム・バンドル**

システム・バンドルは、アプリケーションに主なサービスを提供するために CICS と OSGi フレームワーク間の対話を管理する OSGi バンドルです。主な例は、CICS サービスとリソースにアクセスできるようにする JCICS OSGi バンドルです。

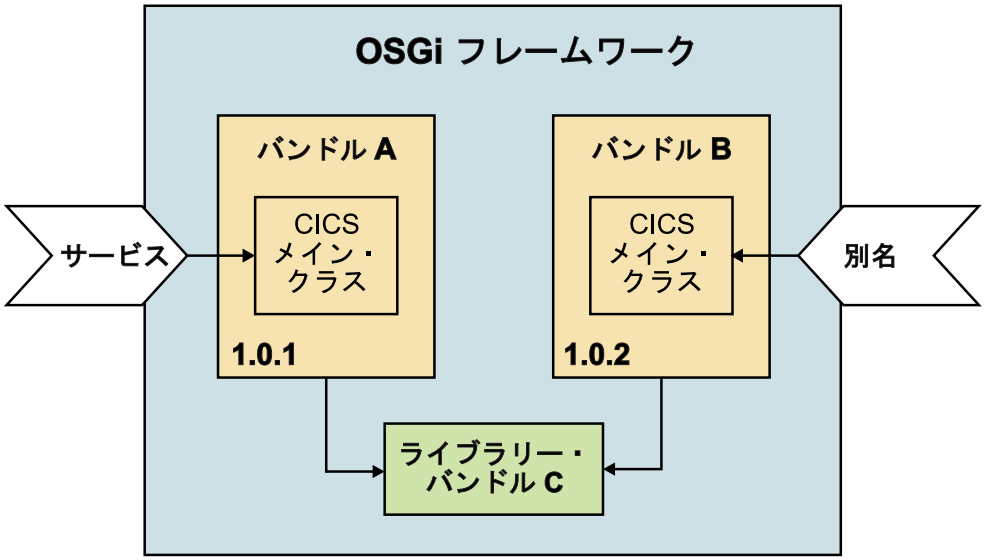
Java アプリケーションの管理をシンプルにするために、以下のベスト・プラクティスに従ってください。

- アプリケーションを構成する密結合 OSGi バンドルを同一 CICS バンドルにデプロイします。密結合バンドルは、OSGi サービスを使用することなく、相互にクラスを直接エクスポートします。これらの OSGi バンドルをまとめて 1 つの CICS バンドルにデプロイして、一緒に更新し、管理します。
- アプリケーション間に依存関係が生じないようにします。代わりに、共通ライブラリーを別個の OSGi バンドルで作成し、独自の CICS バンドルで管理します。アプリケーションとは別個にライブラリーを更新できます。
- バンドル間の依存関係を作成する場合は、バージョンを使用する OSGi のベスト・プラクティスに従ってください。バージョンの範囲を使用すると、アプリケーションは、依存するバンドルへの両立可能な更新を許容できます。
- JVM サーバーの命名規則をセットアップし、システム・プログラマーと Java 開発者間でこの規則を合意します。
- シングルトン OSGi バンドルの使用は避けます。他のバンドルが従属しているシングルトン・バンドルを破棄すると、従属バンドルで障害が起こる恐れがあります。

## JVM サーバーにおける同一 Java アプリケーションの複数のバージョンの実行

OSGi フレームワークは、フレームワーク内で OSGi バンドルの複数バージョンの実行をサポートします。したがって、アプリケーションの使用可能状態を中断することなく、アプリケーションに段階的に更新を導入できます。同じ OSGi サービスの複数の実装形態をフレームワーク内にインストールすることができますが、そのサービスを呼び出した時には、バージョン・プロパティーが最も高いサービスが使用されます。CICS では、バージョン・プロパティーは、基礎となる OSGi バンドルから推定されます。そのため、同じ Java アプリケーションの複数バージョンを同時に JVM サーバーで実行する必要があり、別々のバージョンの OSGi バンドルに同じ CICS メイン・クラスがある場合は、いずれかの CICS メイン・クラスの定義で別名を使用する必要があります。別名を CICS メイン・クラスの宣言で指定すると、特定のバージョンのアプリケーション用の OSGi サービスとして OSGi フレー

ムワークで登録されます。この別名を別の PROGRAM リソースで指定して、そのバージョンのアプリケーションを使用可能にします。




別名の定義について詳しくは、*CICS Explorer 製品ヘルプ* にある「*CICS Java 開発者ガイド*」内の『*プラグイン・プロジェクトのマニフェスト・ファイルの更新*』を参照してください。

**関連情報**:

OSGi アプリケーション用の JVM サーバーの構成
OSGi バンドルにパッケージされている Java アプリケーションをデプロイする場合、OSGi フレームワークを実行するように JVM サーバーを構成します。

JCICS サンプルの概要

## Liberty JVM サーバーでの Java アプリケーション

CICS では、軽量の Java サーブレットおよび JavaServer Pages を実行できる Web コンテナーが提供されています。開発者は、Java サーブレットおよび JSP 仕様の豊富な機能を使用することにより、CICS のための最新式の Web アプリケーションを作成できます。 Web コンテナーは JVM サーバーの中で実行され、WebSphere Application Server Liberty プロファイル・テクノロジーに基づいて作成されています。

Liberty プロファイルは、短時間で起動し、さまざまなプラットフォームで実行可能なアプリケーション開発用の軽量 Web コンテナーです。これは、Java 開発者がアプリケーションの開発とテストの時間を短縮できるよう最適化されており、それにより Web サーバーを構成および開始するために必要な労力を最小限に抑えることができます。 Java 開発者は、無料で利用できる Eclipse のさまざまなツールを使用することにより、アプリケーションと Web サーバーをまとめてパッケージ化して、デプロイメントを簡素化できます。利用できる Web サービス・サポートには、Java API for RESTful Web Services (JAX-RS) および Java API for XML Web

Services (JAX-WS) が含まれます。 Liberty プロファイルの詳細については、Liberty プロファイルの概要を参照してください。

Liberty プロファイル・テクノロジーは CICS と共にインストールされ、JVM サーバー内の Web コンテナーとして実行されます。 Liberty JVM サーバーは、Liberty プロファイルで使用可能な機能のサブセットをサポートしています。 OSGi アプリケーション、Java サーブレット、および JSP ページを実行することができます。サポートされる機能について詳しくは、『Liberty フィーチャー』を参照してください。

Liberty JVM サーバーとそれに関連するツールを使用する理由としては、以下のことが考えられます。

- 3270 画面を Web ブラウザーおよび RESTful クライアントで置き換えて、CICS アプリケーションの表示インターフェースを最新化する必要がある。
- Java 標準ベースの開発ツールを使用して、既存の他の CICS アプリケーションと共に Web クライアントをパッケージ化、併置、および管理する必要がある。
- WebSphere Application Server で既に Liberty プロファイル・アプリケーションを使用しており、それらを CICS で実行するために移植する必要がある。
- CICS において、既に Jetty または類似のサーブレット・エンジンを使用しており、Liberty プロファイルに基づく Web コンテナーにマイグレーションする必要がある。
- データ・ソース定義を使用して、Java から DB2 データベースにアクセスする必要がある (CICS DB2 接続の定義を参照)。
- Java Transaction API (JTA) を使用することにより、タイプ 4 JDBC データベース・ドライバーを介したリモート・リソース・マネージャーの更新と CICS リカバリー可能リソースの更新を調整する必要がある。
- JAX-RS を使用して、REST (Representational State Transfer) 原則に準拠するサービスを開発する必要がある。
- JAX-WS を使用して、標準のアノテーション・ベースのモデル・サポートにより、アプリケーションを開発する必要がある。
- JMS を介してセキュアなメッセージを送受信する Web アプリケーションを開発する必要がある。

**関連概念**:

『Configuring』の『Web アプリケーションのための Liberty JVM サーバーの構成』

Java Web アプリケーションの開発

84 ページの『Liberty フィーチャーの制約事項』
Liberty プロファイルの一部のフィーチャーには、CICS で使用する場合に既知の制約事項がいくつかあります。

**関連情報**:

Liberty プロファイル用の JVM サーバーの構成
Java サーブレットおよび JSP ページを使用する Java Web アプリケーションをデプロイする場合には、Web コンテナーを実行するように Liberty JVM サーバーを構成します。 Web コンテナーは、Liberty プロファイル・テクノロジーに基づきます。

# Java Web サービス

CICS には、Java Web サービスを実行するための Axis2 テクノロジーが組み込まれています。Axis2 は、Apache Foundation からのオープン・ソース Web サービス・エンジンであり、Java 環境で SOAP メッセージを処理するために CICS に提供されています。

Axis2 は、複数の Web サービス仕様をサポートする、Web サービス SOAP エンジンの Java の実装です。また、Axis2 で実行できる Java アプリケーションの作成方法を記述するプログラミング・モデルも提供します。Axis2 は、Java 環境で Web サービスを処理するために CICS で提供されているため、zAAP プロセッサーへの適格な Java 処理のオフロードをサポートします。

JVM サーバーは、既存の Web サービスを変更することなく、SOAP パイプラインでインバウンドおよびアウトバウンド SOAP メッセージを処理するための Axis2 の実行をサポートします。ただし、Java アプリケーションから Web サービスを作成し、同じ JVM サーバーで実行することもできます。JVM サーバーの Axis2 リポジトリーにアプリケーションをデプロイすることによって、Java アプリケーションと SOAP 処理の両方が、zEnterprise Application Assist Processor (zAAP) で実行できるようになります。

次のいずれかの理由で Java Web サービスを使用できます。

- 他のプラットフォームで Axis2 Web サービスの経験があり、CICS で Web サービスを作成したい。
- 標準の Java API を使用して、Axis2 に統合される Java データ・バインディングを作成したい。
- CICS Web サービス・アシスタントで処理するのが困難な複雑な WSDL 文書がある。

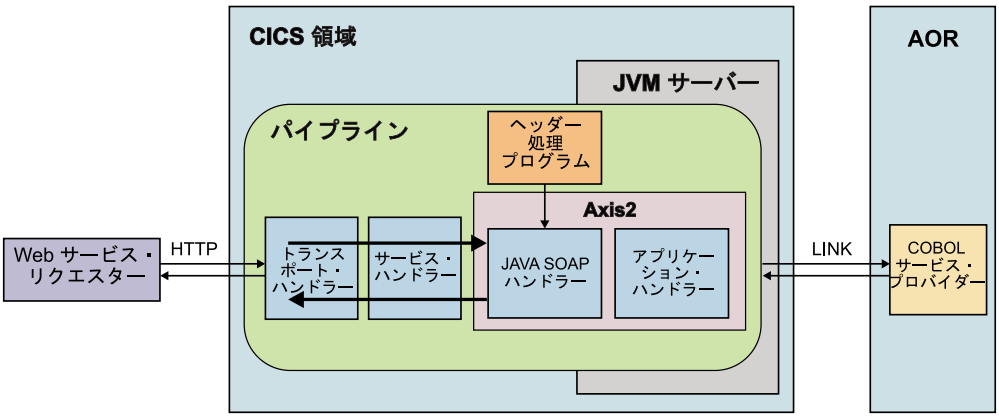
以下の例では、Web サービスで Java を使用する方法を説明しています。

## JVM サーバーにおける SOAP メッセージの処理

Web サービス・パイプラインで発生する大部分の SOAP 処理は、SOAP ハンドラーおよびアプリケーション・ハンドラーによって実行されます。オプションとして、JVM サーバーでこの SOAP 処理を実行し、zAAP を使用して作業を実行できます。 COBOL、C、C++、または PL/I で作成される Web サービス・アプリケーションを引き続き使用できます。

既存の Web サービスがある場合、JVM サーバーを使用するようにパイプラインの構成を更新できます。Web サービスを変更する必要はありません。パイプラインが SOAP ヘッダー処理プログラムを使用する場合、Axis2 プログラミング・モデルを使用することによって、Java でそのプログラムを再作成するのが最善の方法です。このヘッダー処理プログラムは、追加のデータ変換を行うことなく、Java オブジェクトを Axis2 と共用できます。例えば、COBOL のヘッダー処理プログラムがある場合、データが Java から COBOL に変換された後、再び戻す必要があります。これにより、SOAP 処理のパフォーマンスが低下する場合があります。

次の図に示されているシナリオは、Web サービス・プロバイダーである COBOL アプリケーションの例です。要求は、Java をサポートするように構成されているパイプラインで処理されます。SOAP ハンドラーおよびアプリケーション・ハンドラーは、Axis2 によって処理され、JVM サーバーで実行される Java プログラムです。アプリケーション・ハンドラーは、データを XML から COBOL に変換し、アプリケーションにリンクします。



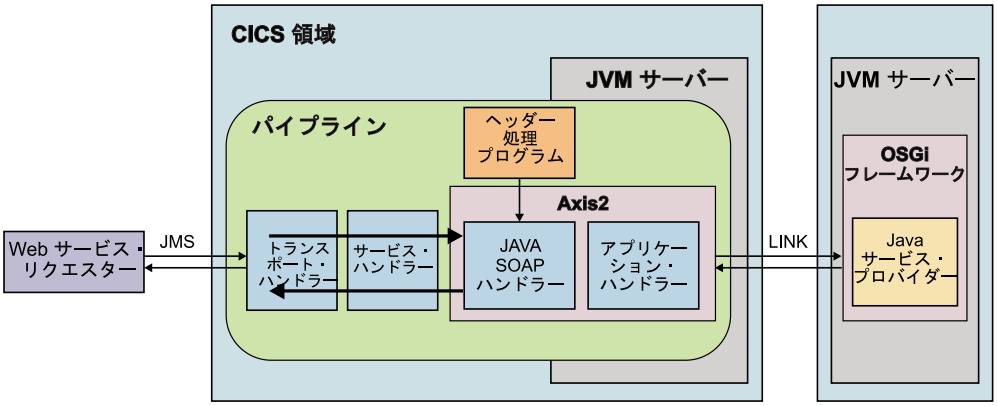

環境を計画する際には、JVM サーバーに 1 組の専用領域を使用するようにしてください。この例では、COBOL アプリケーションが実行されているアプリケーション専有領域 (AOR) は、JVM サーバーが実行される CICS 領域とは別です。ワークロード管理を使用すると、例えば **EXEC CICS LINK** ではアプリケーション・ハンドラーから、インバウンド要求では Web サービス・リクエスターからのように、ワークロードのバランスを取ることができます。

## CICS Web サービス・アシスタントからの出力を使用する Java アプリケーションの作成

言語構造を解釈し、CICS Web サービス・アシスタントによって生成されるデータ・バインディングを使用する Java アプリケーションを作成できます。Web サービス・アシスタントは、WSDL から言語構造を作成し、言語構造から WSDL を作成することができます。また、このアシスタントは、SOAP 処理時に XML とターゲット言語間でデータを変換する方法を記述する Web サービス・バインディングも作成します。

アシスタントを使用して言語構造を生成する場合は、JZOS または J2C を使用して言語構造を処理して、Java クラスを生成できます。これらのツールを使用すると、Java 開発者は他の CICS アプリケーションと対話できます。この例では、これらのツールを使用して、CICS が XML からデータを変換した後にインバウンド SOAP メッセージを処理できる Java アプリケーションを作成することができます。詳しくは、40 ページの『Java からの構造化データとの対話』を参照してください。

次の図に示されているシナリオは、Web サービス・プロバイダーである Java アプリケーションの例です。SOAP 処理は、JVM サーバーで Axis2 によって処理されます。アプリケーション・ハンドラーがリンクする Java アプリケーションは、1 つ以上の OSGi バンドルとしてパッケージされ、デプロイされ、JVM サーバーで実行されます。




この方法の利点は、データ・バインディングが Web サービス・アシスタントによって生成されたため、CICS では Web サービスは WEBSERVICE リソースによって表されることです。CICS で統計、リソース管理、およびその他の機能を使用して、Web サービスを管理できます。欠点は、Java 開発者が、慣れていないプログラミング言語の言語構造を処理しなければならないことです。

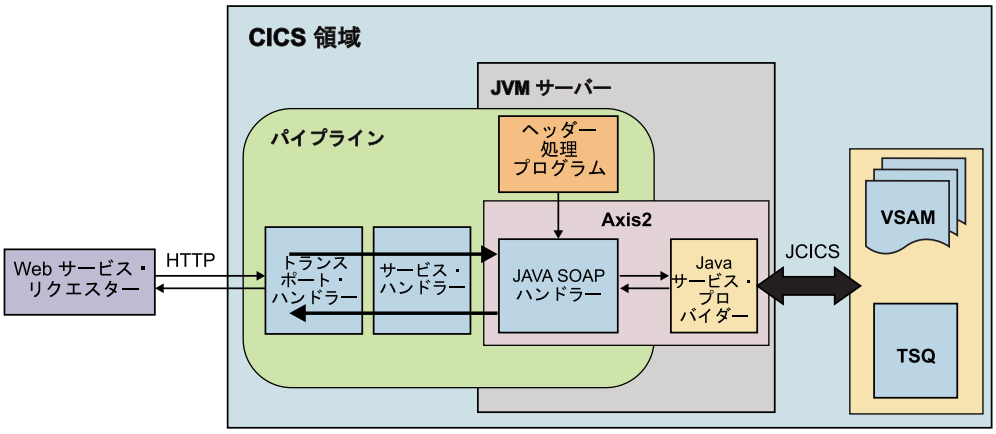
このタイプのアプリケーション用の環境を計画する場合は、アプリケーションを実行するために別個の JVM サーバーを使用してください。

- さまざまなワークロードに合わせて JVM サーバーをより効率よく管理し、調整することができます。
- インバウンド要求および **EXEC CICS LINK** でワークロード管理を使用して、ワークロードのバランスを取り、環境を拡大することができます。
- CICS における OSGi サポートを利用して、Java アプリケーションを管理できます。

## Java データ・バインディングを使用する Java アプリケーションの作成

SOAP メッセージの XML を生成し、解析する Java アプリケーションを作成できます。Java 7 API は、XML を処理するために標準の Java ライブラリーを提供します。例えば、Java Architecture for XML Binding (JAXB) を使用して、Java データ・バインディングを作成し、Java API for XML Web Services (JAX-WS) ライブラリーを使用して、XML を生成し、解析することができます。これらのライブラリーを使用すると、アプリケーションは、SOAP パイプライン処理と同じ JVM サーバーの Axis2 で実行できます。

次の図に示されているシナリオは、Web サービス・プロバイダーであり、JVM サーバー内の Axis2 SOAP エンジンによって処理される Java アプリケーションの例です。




Java アプリケーションは Java データ・バインディングを使用し、Java SOAP ハンドラーと対話するので、アプリケーション・ハンドラーはありません。この例では、Web サービス・リクエスターは HTTP を使用して CICS 領域に接続しますが、JMS も使用することができます。Java アプリケーションは JCICS を使用して CICS サービス (この例では VSAM ファイルと一時記憶域キュー) にアクセスします。

この方法の利点は、Java 開発者が使い慣れたテクノロジーを使用してアプリケーションを作成することです。また、Java 開発者は、Web サービス・アシスタントが処理できない複雑な WSDL 文書を使用して、バインディングを作成できます。ただし、この方法には、次のようないくつかの制限があります。

- このタイプのアプリケーションには WS-Security を使用できません。したがって、セキュリティーを使用したい場合は、SSL を使用して接続を保護してください。
- パイプライン処理でユーザー ID のコンテキスト切り替えが行われません。要求でユーザー ID を変更するには、URIMAP リソースを使用してください。
- Web サービス・アシスタントからの Web サービス・バインディングを使用しないため、WEBSERVICE リソースがありません。
- アプリケーションが Web サービス・リクエスターである場合、パイプライン処理はバイパスされます。したがって、パイプラインで使用可能なサービス品質が得られません。

CICS 領域にワークロード管理を実装する場合は、このタイプのワークロードの経路指定方法を計画する必要があります。Java アプリケーションは SOAP 処理と同じ JVM サーバーで実行されるため、CICS は経路指定を行う機会を提供しません。ただし、経路指定が必要な場合は、JAX-WS アプリケーションで他のプログラムに対して分散プログラム・リンクを実装できます。

# 第 2 章 CICS 用の Java アプリケーションの開発

CICS サービスを使用し、CICS 制御下で実行される Java アプリケーション・プログラムを作成できます。CICS Explorer SDK を使用すると、JCICS クラス・ライブラリーを使用して CICS リソースにアクセスし、他の言語で作成されるプログラムと対話するアプリケーションを開発できます。また、さまざまなプロトコルやテクノロジー (Web サービスや CICS Transaction Gateway など) を使用して、Java プログラムに接続することもできます。

CICS は、Java アプリケーションをサポートするためのツールやランタイム環境を提供します。CICS Explorer SDK は Eclipse ベースのツールであり、Java アプリケーションを開発し、CICS にデプロイするためのサポートを提供します。CICS リソースやサービスにアクセスするアプリケーションを開発するための JCICS クラス・ライブラリーが含まれています。例えば、VSAM ファイル、一時データ・キュー、および一時記憶にアクセスできます。 また、JCICS を使用して、他の言語 (COBOL や C など) で作成された CICS アプリケーションにリンクすることもできます。

CICS Explorer SDK で提供されるその他の機能は以下のとおりです。

- 特定のリリースの CICS でサポートされるクラスのみを使用することを確実にするためのターゲット環境を構成できます。SDK は、アプリケーション開発に適した正しいライブラリーをインポートします。
- OSGi バンドル、OSGi アプリケーション・プロジェクト (EBA ファイル)、Web 使用可能 OSGi バンドル・プロジェクト (WAB ファイル)、エンタープライズ・アーカイブ (EAR ファイル) および動的 Web プロジェクト (WAR ファイル) を、デプロイメント用に 1 つ以上の CICS バンドルにパッケージ化できます。ただし、WAR ファイルは OSGi バンドルにアクセスできないため、同じ CICS バンドル内で WAR ファイルと OSGi バンドルをパッケージ化することの利点はほとんどありません。 CICS は、サーブレットや JSP ページなどの Web コンポーネントが含まれたアプリケーションを実行する Web コンテナーを提供します。Web コンテナーは、WebSphere Application Server からの Liberty テクノロジーに基づいてビルドされています。
- CICS バンドルを 1 つのアプリケーション・プロジェクトにパッケージ化して、プラットフォームにデプロイすることができます。

SDK にはサンプル・セットが含まれています。CICS 用の Java アプリケーションの開発に慣れていないユーザーは、これを使用して開発に取り掛かることができます。

29 ページの『CICS について必要な知識』
CICS は、ユーザーが要求によってアプリケーションを実行するためのサービスを提供するトランザクション処理のサブシステムです。多数のユーザーが、同じファイルとプログラムを使用する同じアプリケーションを同時に実行する要求を実行依頼することが可能です。 CICS は、リソースの共用、データの保全性、実行の優先順位付けを管理する一方で、短い応答時間を維持します。

34 ページの『CICS Explorer SDK を使用したアプリケーションの開発』
CICS Explorer Software Development Kit (SDK) は、OSGi および Web プロジェクトのサポートを含め、Java アプリケーションを開発して CICS にデプロイするための環境を提供します。

38 ページの『JVM の状態の制御』
CICS で実行される Java アプリケーションの設計と開発を行うときには、アプリケーションが、JVM を不適切な状態のままにしたり、その状態を望ましくない方法で変更したりしないようにしてください。 JVM の状態の制御には CICS サービスが役立ちます。

40 ページの『Java からの構造化データとの対話』
多くの場合、CICS Java プログラムは、当初は他のプログラミング言語用に設計されたデータと対話します。例えば、Java プログラムは、COBOL コピーブックで定義された COMMAREA を使用して COBOL プログラムにリンクしたり、C++ ヘッダー・ファイルを使用してデータが定義される VSAM ファイルからレコードを読み取ったりすることができます。インポーターを使用すると、これらの形式の構造化データと対話することができます。

42 ページの『JCICS を使用した Java の開発』
CICS Java クラス・ライブラリー (JCICS) を使用して CICS サービスにアクセスする Java アプリケーションを作成できます。JCICS は、Java において、CICS でサポートされている他の言語 (COBOL など) に提供される **EXEC CICS** アプリケーション・プログラミング・インターフェースに相当するものです。

77 ページの『JCA ローカル ECI サポート』
JCA ローカル ECI リソース・アダプターを使用するよう構成されている Liberty JVM サーバーに、JCA ECI アプリケーションをデプロイすることができます。

77 ページの『Java アプリケーションからのデータへのアクセス』
DB2 および VSAM のデータのアクセスと更新を行うことができる Java アプリケーションを作成できます。または、他の言語のプログラムにリンクして、DB2、VSAM、および IMS にアクセスすることができます。

78 ページの『CICS 内の Java アプリケーションからの接続』
CICS 環境内の Java プログラムは、TCP/IP ソケットをオープンし、外部プロセスと通信することができます。Java プログラムをゲートウェイとして使用すると、他の言語の CICS プログラムからは使用できない可能性がある他のエンタープライズ・アプリケーションに接続することができます。例えば、リモート・サーブレットまたはデータベースと通信する Java プログラムを作成できます。

79 ページの『Liberty フィーチャー』
CICS は WebSphere Application Server Liberty プロファイルのフィーチャーをサポートすることで、JEE Web アプリケーションを Liberty JVM サーバーにデプロイできるようにします。

85 ページの『Java プログラムから DB2 データにアクセスするための JDBC および SQLJ の使用』
CICS で実行される Java プログラムでは、DB2 データベースに保持されているデータにアクセスするためにいくつかの方法を使用できます。

94 ページの『Java プログラムからの WebSphere MQ へのアクセス』
CICS で実行される Java プログラムは、WebSphere MQ classes for Java または WebSphere MQ classes for JMS を使用して WebSphere MQ にアクセスできます。

103 ページの『Liberty JVM サーバーで実行する Java アプリケーションの開発』
WebSphere Liberty プロファイルを使用する Java Web アプリケーションをデプロイする場合は、Web コンテナーを実行するように Liberty JVM サーバーを構成します。

128 ページの『OSGi フレームワークの使用』
CICS での OSGi フレームワークの使用。

## CICS について必要な知識

CICS は、ユーザーが要求によってアプリケーションを実行するためのサービスを提供するトランザクション処理のサブシステムです。多数のユーザーが、同じファイルとプログラムを使用する同じアプリケーションを同時に実行する要求を実行依頼することが可能です。 CICS は、リソースの共用、データの保全性、実行の優先順位付けを管理する一方で、短い応答時間を維持します。

CICS アプリケーションは、製品オーダーの処理や会社の給与計算の準備などの業務を連携して実行する、関連したプログラムの集合です。CICS アプリケーションは、CICS 制御下で実行され、CICS サービスとインターフェースを使用してプログラムとファイルにアクセスします。

CICS アプリケーションを実行するには、トランザクション 要求を実行依頼します。 CICS では、トランザクションという用語に特別な意味があります。CICS での意味と、業界でより一般的に使用される意味との違いについては、 30 ページの『CICS トランザクション』を参照してください。トランザクションの実行は、必要な機能を実装する 1 つ以上のアプリケーション・プログラムの実行で構成されます。

CICS 用の Java アプリケーションを開発するには、CICS プログラム、トランザクション、およびタスク間の関係を理解する必要があります。これらの用語は、CICS 資料全体で使用され、多くのプログラミング・コマンドで表示されます。また、ランタイム環境において CICS が Java アプリケーションを処理する方法も理解しておく必要があります。

30 ページの『CICS トランザクション』
トランザクションは、単一の要求によって開始される 1 つの処理です。

30 ページの『CICS タスク』
タスクは、トランザクションを実行する単一のインスタンスです。

31 ページの『CICS アプリケーション・プログラム』
Java プログラムでは、CICS 用の Java クラス・ライブラリー (JCICS) を使用して CICS サービスにアクセスし、他の言語で作成されたアプリケーション・プログラムにリンクすることができます。

31 ページの『CICS サービス』
Java プログラムは、JCICS プログラミング・インターフェースを介して、デー

タ管理サービス、通信サービス、作業単位サービス、プログラム・サービス、および診断サービスの各 CICS サービスにアクセスできます。

33 ページの『CICS の Java ランタイム環境』
CICS は、スレッド・セーフ Java アプリケーションを実行するための JVM サーバー環境を提供します。スレッド・セーフではないアプリケーションは、JVM サーバーを利用することはできません。

## CICS トランザクション

トランザクションは、単一の要求によって開始される 1 つの処理です。

要求は通常、ユーザーによって端末で行われます。ただし、Web ページから、リモート・ワークステーション・プログラムから、または別の CICS 領域のアプリケーションから行われる場合があります。もしくは、事前定義された時点に自動的にトリガーされる場合もあります。『Product overview』の『CICS web support concepts and structure』 および 『製品の概要』の『CICS 外部インターフェースの概要』 では、CICS トランザクションのさまざまな実行方法が記述されています。

単一トランザクションは、1 つ以上の*アプリケーション・プログラム* で構成されます。これらのアプリケーション・プログラムが実行されると、必要な処理を実行します。

ただし、CICS ではトランザクション という用語は、単一イベントと、同じタイプの他のすべてのトランザクションの両方を意味するのに使用されます。CICS に対し、それぞれのトランザクション・タイプを、TRANSACTION リソース定義を使用して記述します。この定義により、トランザクション・タイプに名前 (トランザクション ID、すなわち TRANSID) が指定され、実行される作業に関する複数の項目が CICS に指示されます。例えば、最初にどのプログラムを呼び出すか、トランザクションの実行全体でどの種類の認証が必要であるかなどです。

トランザクションを実行するには、その TRANSID を CICS に対して送信します。CICS は、TRANSACTION 定義に記録された情報を使用して正しい実行環境を確立し、最初のプログラムを開始します。

*トランザクション* という用語は、*リカバリー単位*、または CICS で*作業単位* と呼ばれるものを記述するために、IT 業界で広く使用されています。一般に、これはリカバリー可能な完全な論理オペレーションです。プログラムされたコマンド、またはシステム障害の発生によって、トランザクション全体をコミットまたはバックアウトすることができます。多くの場合、CICS トランザクションの有効範囲は単一の作業単位でもありますが、CICS 資料を読むときは、意味の違いを認識する必要があります。

## CICS タスク

タスクは、トランザクションを実行する単一のインスタンスです。

CICS では、このタスク という用語に特別の意味があります。CICS はトランザクションの実行要求を受け取ると、トランザクション・タイプの実行のこの *1* つのインスタンス に関連した新規タスクを開始します。すなわち、CICS タスクは、通常は特定のユーザーのために、データの独自の専用セットを使用した、トランザクションの 1 つの実行です。また、タスクを*スレッド* と見なすこともできます。タス

クは、優先順位と準備度にしたがって CICS によってディスパッチ されます。トランザクションが完了すると、タスクは終了します。

## CICS アプリケーション・プログラム

Java プログラムでは、CICS 用の Java クラス・ライブラリー (JCICS) を使用して CICS サービスにアクセスし、他の言語で作成されたアプリケーション・プログラムにリンクすることができます。

CICS アプリケーション・プログラムは COBOL、C、C++、Java、PL/I、またはアセンブラー言語で作成できます。大部分の処理ロジックは標準言語ステートメントで表されますが、CICS サービスを要求するには、アプリケーションは提供されたアプリケーション・プログラミング・インターフェースを使用します。
COBOL、C、C++、PL/I、またはアセンブラー・プログラムは、**EXEC CICS** アプリケーション・プログラミング・インターフェースまたは C++ クラス・ライブラリーを使用できます。Java プログラムは JCICS クラス・ライブラリーを使用します。
JCICS については、43 ページの『CICS 用 Java クラス・ライブラリー (JCICS)』で説明しています。

## CICS サービス

Java プログラムは、JCICS プログラミング・インターフェースを介して、データ管理サービス、通信サービス、作業単位サービス、プログラム・サービス、および診断サービスの各 CICS サービスにアクセスできます。

CICS サービス・マネージャーの名称には、通常は「管理」または「制御」という語が含まれています (例えば、「端末管理」、「プログラム制御」など)。これらの用語は、CICS 資料で幅広く使用されています。

### データ管理サービス

CICS が提供するデータ管理サービスは、次のとおりです。

- 仮想記憶アクセス方式 (VSAM) データ・セットにアクセスする際の、保全性のあるレコード・レベル共用。データのバックアウト (トランザクション障害またはシステム障害の場合)、または順方向リカバリー (メディア障害の場合) をサポートするために、CICS はアクティビティーをログに記録します。 CICS ファイル制御は、VSAM データを管理します。

  また、CICS は 2 つの専有ファイル構造も実装し、それらを操作するためのコマンドを提供します。

  **一時記憶**

  一時記憶 (TS) は、複数のトランザクションからデータを容易に使用可能にする手段です。データは、プログラムからの要求に応じて作成されるキューに保持されます。キューには順次にアクセスするか、または項目番号でアクセスすることができます。

  一時記憶域キューは、メインメモリーに常駐することも、ストレージ・デバイスに書き込むこともできます。

  一時記憶域キューは、名前付きのスクラッチパッドと見なすことができます。

**一時データ**

一時データ (TD) も複数のトランザクションから使用可能であり、キューに保持されます。ただし、TS キューとは異なり、TD キューは事前定義する必要があり、順次にしか読み取れません。各項目は、読み取られるとキューから除去されます。

一時データ・キューは、常にデータ・セットに書き込まれます。一時データ・キューは、特定数の項目が書き込まれると、特定トランザクションを開始するトリガーの役目をするように定義できます。 例えば、起動したトランザクションによってそのキューを処理できます。

- データベース製品とのインターフェースを使用した、他のデータベース (DB2 を含む) 内のデータへのアクセス。

## 通信サービス

CICS は、SNA と TCP/IP プロトコルを使用して、さまざまな端末 (ディスプレイ、プリンター、およびワークステーション) へのアクセスを可能にするコマンドを備えています。CICS 端末管理により、SNA ネットワークおよび TCP/IP ネットワークを管理できます。

拡張プログラム間通信機能 (APPC) コマンドを使用して、SNA プロトコルを使用してリモート・システム内の他のプログラムを開始し、通信するプログラムを作成できます。CICS APPC は、ピアツーピア分散アプリケーション・モデルを実装します。

次の CICS 専有の通信サービスが提供されています。

**機能シップ**

リモート CICS 領域で既存のものとして定義されるリソース (ファイル、キュー、およびプログラム) にアクセスするプログラム要求は、自動的に CICS によって専有領域に転送されます。

**分散プログラム・リンク (DPL)**

リモート CICS 領域で既存のものとして定義されるプログラムに対するプログラム・リンク要求は、自動的に専有領域に転送されます。CICS は、分散アプリケーションの保全性を維持するためのコマンドを提供します。

**非同期処理**

CICS は、プログラムが同じ CICS 領域またはリモート CICS 領域内の別のトランザクションを開始し、オプションとしてデータをそのトランザクションに渡すことを可能にするコマンドを提供します。新しいトランザクションは、新しいタスク内で独立してスケジュールされます。この機能は、他のソフトウェア製品によって提供される *fork* 操作に似ています。

**トランザクション・ルーティング**

リモート CICS 領域で既存のものとして定義されるトランザクションを実行する要求は、自動的に専有領域に転送されます。ユーザーへの応答は、要求を受け取った領域に返されます。

### 作業単位サービス

CICS がトランザクションを実行する新しいタスクを作成すると、新しい作業単位 (UOW) が自動的に開始されます (したがって、BEGIN コマンドは必要ないため、CICS はこのコマンドを提供しません)。CICS トランザクションは常にトランザクション内で実行されます。

CICS は、実行されたリカバリー可能な作業をコミットまたはロールバックするために SYNCPOINT コマンドを提供します。同期点が完了すると、CICS は自動的に別の作業単位を開始します。SYNCPOINT コマンドを発行せずにプログラムを終了すると、CICS は暗黙的な同期点を取り、トランザクションをコミットしようとします。

コミットの有効範囲には、リカバリー可能として定義されたすべての CICS リソース、および CICS によって提供されたインターフェースを使用してインタレストを登録した他のすべてのリソース・マネージャーが含まれます。

### プログラム・サービス

CICS は、プログラムが別のプログラムにリンクするか、制御を転送してから戻ることを可能にするコマンドを提供します。

### 診断サービス

CICS が提供するコマンドを使用して、プログラムをトレースし、ダンプを作成できます。

## CICS の Java ランタイム環境

CICS は、スレッド・セーフ Java アプリケーションを実行するための JVM サーバー環境を提供します。スレッド・セーフではないアプリケーションは、JVM サーバーを利用することはできません。

JVM サーバーは、単一の JVM で複数のタスクを実行できるランタイム環境です。この環境はそれぞれの Java タスクで必要な仮想ストレージの量を削減し、CICS が多数のタスクを同時に実行できるようにします。

CICS タスクは、同じ JVM サーバー・プロセス内のスレッドとして並列で実行されます。 JVM は、複数のアプリケーションを同時に実行している可能性のあるすべての CICS タスクにより共用されます。すべての静的データおよび静的クラスも共用されます。このため、CICS で JVM サーバーを使用するには、Java アプリケーションがスレッド・セーフである必要があります。各スレッドは T8 TCB で実行され、JCICS API を使用して CICS サービスにアクセスできます。

ご使用のアプリケーションで System.exit() メソッドを使用しないでください。このメソッドにより、JVM サーバーと CICS の両方がシャットダウンし、アプリケーションの状態と可用性に影響を与えます。

### マルチスレッド・アプリケーション

新しいスレッドを開始したり、スレッドを開始するライブラリーを呼び出すためにアプリケーション・コードを作成できます。アプリケーションにスレッドを作成す

る場合、OSGi レジストリーから汎用 ExecutorService を使用する方式が推奨されています。アプリケーションが JVM サーバーで実行されているとき、ExecutorService は自動的に CICS ExecutorService を使用して CICS スレッドを作成します。この方式により、アプリケーションを他の環境に移植することが容易になり、特定の JCICS API メソッドを使用する必要がなくなります。

しかし、CICS に固有のアプリケーションを作成している場合は、JCICS API 内の CICSExecutorService クラスを使用して、新しいスレッドを要求することができます。

どちらの方法を選択した場合にも、新しく作成されたスレッドは CICS タスクとして実行され、CICS サービスにアクセスできます。 JVM サーバーが使用不可にされると、CICS は、JVM で実行中の CICS タスクすべてが終了するまで待機します。ExecutorService クラスまたは CICSExecutorService クラスを使用することにより CICS は実行中のタスクを認識するので、アプリケーションの作業完了後に JVM サーバーをシャットダウンさせることが可能になります。

JCICS オブジェクトは、それらを作成したタスクでのみ使用できます。それらのオブジェクトをタスク間で共用しようとすると、予測不能な結果をもたらす可能性があります。

CICS ExecutorService の使用の詳細については、 47 ページの『スレッド』を参照してください。

### JVM サーバーの始動とシャットダウン

静的データは JVM サーバーで実行中のすべてのスレッドで共用されるため、静的データを初期化して JVM のシャットダウン時に適切な状態にするための OSGi バンドル・アクティベーター・クラスを作成できます。 JVM サーバーは、例えば、JVM の構成を変更したり、問題を修正したりするために、管理者が使用不可にするまで実行されます。バンドル・アクティベーター・クラスを提供することによって、ご使用のアプリケーションに状態が正しく設定されていることを確認できます。 CICS にはタイムアウトがあります。このタイムアウトは、JVM サーバーの始動または停止を続行するまでにこれらのクラスが完了するのを待機する時間を指定します。始動クラスおよび終了クラスで JCICS を使用することはできません。

## CICS Explorer SDK を使用したアプリケーションの開発

CICS Explorer Software Development Kit (SDK) は、OSGi および Web プロジェクトのサポートを含め、Java アプリケーションを開発して CICS にデプロイするための環境を提供します。

### このタスクについて

SDK を使用して、OSGi 仕様に準拠するように、新しいアプリケーションを作成したり、既存の Java アプリケーションを再パッケージ化したりできます。 OSGi Service Platform には、コンポーネント・モデルを使用してアプリケーションを開発し、それらのアプリケーションを OSGi バンドルとして 1 つのフレームワークにデプロイするためのメカニズムが用意されています。*OSGi バンドル* は、アプリケーションのデプロイメントの単位であり、バージョン管理情報、依存関係、およびア

プリケーション・コードが入っています。OSGi の主な利点は、*Java* パッケージ と呼ばれる明確に定義されたインターフェースを介してのみアクセスされる再使用可能コンポーネントから、アプリケーションを作成できることです。その後、*OSGi* サービス を使用して、Java パッケージにアクセスできます。また、Java アプリケーションのライフサイクルと依存関係をきめ細かく管理することもできます。 OSGi を使用したアプリケーションの開発については、OSGi Alliance Web サイトをご覧ください。

また、SDK を使用して、Java サーブレットと JSP ページが含まれる動的 Web プロジェクトおよび OSGi アプリケーション・プロジェクトを処理することもできます。 CICS サービスへのアクセス用に JCICS を使用する最新の Web レイヤーとビジネス・ロジックによるアプリケーションを作成することができます。別の OSGi バンドルからのコードに Web アプリケーションでアクセスする必要がある場合、OSGi アプリケーション・プロジェクト (EBA ファイル) としてそれをデプロイする必要があります。アプリケーション・マニフェストの中にもう一方の OSGi バンドルを含めるか、共通ライブラリーとして Liberty bundle_repository の中にもう一方のバンドルをインストールする必要があります。アプリケーションへの入り口点を提供し、それを URL として Web ブラウザーに公開するには、Web 使用可能 OSGi バンドル (WAB ファイル) を EBA ファイルに含める必要があります。

SDK を使用すると、サポートされている任意の CICS リリースで実行する Java アプリケーションを開発できます。異なるリリースの CICS は、別々のバージョンの Java をサポートします。また、JCICS API も、CICS のより多くの機能をサポートするように後のリリースで拡張されています。誤ったクラスを使用しないように、SDK はターゲット・プラットフォームをセットアップする機能を備えています。どのリリースの CICS 用に開発するかを定義すると、SDK は、使用できない Java クラスを自動的に非表示にします。

アプリケーションを開発し、デプロイするために以下の各ステップを実行する方法について詳しくは、SDK ヘルプを参照してください。

## 手順

1. Java 開発用のターゲット・プラットフォームをセットアップします。

   ターゲット・プラットフォームを使用すると、アプリケーション開発において、確実に、CICS のターゲット・リリースに適した Java クラスだけを使用することになります。

2. Java アプリケーション開発用の OSGi バンドル・プロジェクトまたはプラグイン・プロジェクトを作成します。
   a. プロジェクトのデフォルト・バージョンは `1.0.0.qualifier` です。「**バージョン**」フィールドで、「.qualifier」を使用する必要がない場合はバージョン番号の最後から「.qualifier」を削除します。または、日時タイムスタンプなどの意味のあるものに設定します。

   ベスト・プラクティスを使用して Java アプリケーションを開発します。例えば、OSGi バンドルどうしの従属関係を構成するには、Require-Bundle ではなく Import-Package/Export-Package を設定します。

3. CICS 用の Java アプリケーションの開発に慣れていない場合は、CICS Explorer SDK で提供されるサンプルを使用して開発に取り掛かることができます。 Java アプリケーションで JCICS を使用するには、com.ibm.cics.server パッケージをインポートする必要があります。
4. オプション: アプリケーション・プレゼンテーション層を開発するために、動的 Web アプリケーション (WAR) または Web 使用可能 OSGi バンドル・プロジェクト (WAB) を作成します。 動的 Web プロジェクト内でサーブレットや JSP ページを作成できます。また、WAR ファイルの場合、Liberty API バンドルにアクセスできるよう、ターゲット・プラットフォームを変更する必要もあります。詳細については、 104 ページの『開発環境のセットアップ』を参照してください。
5. 以下のようにして、配置用にアプリケーションをパッケージ化します。
   a. Web 使用可能 OSGi バンドル・プロジェクト (WAB) をデプロイする場合は、OSGi アプリケーション・プロジェクト (EBA) を作成します
   b. EBA、EAR ファイルまたは Web アプリケーション (WAR ファイル) を参照するための、1 つ以上の CICS バンドル・プロジェクトを作成します。 CICS バンドルは、CICS におけるアプリケーションのデプロイメント単位です。一緒に更新および管理される複数の Web アプリケーションを 1 つの CICS バンドル・プロジェクトに入れてください。アプリケーションの配置先の JVMSERVER リソースの名前を知っている必要があります。

      また、PROGRAM リソース、URIMAP リソース、および TRANSACTION リソースなどの CICS リソースのサブセットを CICS バンドル・プロジェクトに追加することもできます。これらのリソースは、アプリケーションの一部として動的にインストールされ、管理されます。
   c. オプション: アプリケーションを CICS プラットフォームにデプロイする場合は、CICS バンドルを参照するアプリケーション・プロジェクトを作成します。 アプリケーションにより、CICSplex 全体にわたってアプリケーションをデプロイおよび管理するための単一管理ポイントが CICS に提供されます。詳しくは、『アプリケーションの開発』の『デプロイ用アプリケーションのパッケージ化』を参照してください。
6. アプリケーション・プロジェクトまたは CICS バンドル・プロジェクトをエクスポートすることにより、Java アプリケーションを zFS にデプロイします。あるいは、配置用にソース・リポジトリーにプロジェクトを保管することもできます。

## タスクの結果

これで、CICS Explorer SDK を使用したアプリケーションの開発とエクスポートが正常に完了しました。

## 次のタスク

アプリケーションを JVM サーバーにインストールします。 CICS 内にリソースを作成する権限がない場合は、システム・プログラマーまたは管理者にアプリケーションの作成を依頼します。システム・プログラマーまたは管理者には、エクスポートしたバンドルがある場所とターゲット JVM サーバーの名前を伝える必要があります。詳しくは、 182 ページの『JVM サーバーにおける OSGi バンドルのデプロ

イ』を参照してください。サード・パーティーの OSGi バンドル (WebSphere MQ など) を Eclipse 開発環境に追加するために、ターゲット・プラットフォームを更新しなければならない場合もあります。『ターゲット・プラットフォームの更新』を参照してください。

『ターゲット・プラットフォームの更新』
サード・パーティーの Java クラスを Eclipse 開発環境のターゲット・プラットフォームに追加できます。

38 ページの『プロジェクト・ビルド・パスの更新』
プロジェクト・ビルド・パスの更新方法。

## ターゲット・プラットフォームの更新

サード・パーティーの Java クラスを Eclipse 開発環境のターゲット・プラットフォームに追加できます。

### 始める前に

サード・パーティーの Java クラスが入っている JAR ファイルを OSGi プラグインとして使用できることと、そのファイルがローカル・ワークステーションにコピーされていることを確認します。ターゲット・プラットフォーム「**CICS TS V5.3**」または「**CICS TS V5.3 with Liberty and PHP**」が既に構成されていることを確認します。

### このタスクについて

CICS Explorer Software Development Kit (SDK) には、CICS API や Web API を使用するために必要な Java クラスしか用意されていません。その他のインターフェースのサポートを追加するには、サード・パーティーの JAR が入っている OSGi プラグインを Eclipse ターゲット・プラットフォームに追加する必要があります。そうすれば、そのターゲット・プラットフォームを使用するすべてのアプリケーションで、エクスポートしたパッケージを使用できるようになります。

### 手順

1. Eclipse で「**ウィンドウ**」 > 「**設定**」 > 「**ターゲット・プラットフォーム**」をクリックします。
2. ターゲットの定義から「**CICS TS V5.3**」または「**CICS TS V5.3 with Liberty and PHP**」を選択します。
3. 「**編集**」をクリックし、「**ロケーション**」タブにある「**追加**」をクリックします。サード・パーティーのバンドル JAR が入っているディレクトリーを表示します。
4. 「**次へ**」をクリックします。 OSGi プラグインの内容が表示されます。
5. 「**終了**」をクリックし、再び「**終了**」をクリックしてから、「**OK**」をクリックします。

### タスクの結果

OSGi 環境を更新して、Java アプリケーション開発に必要なサード・パーティーの OSGi バンドルと CICS の OSGi バンドルを組み込むことができました。

### 次のタスク

Java アプリケーションを CICS JVM サーバーにデプロイして、サード・パーティーの JAR を、OSGi ミドルウェア・バンドルとして追加するか、Liberty 共用バンドル・リポジトリーに追加します。詳しくは、199 ページの『OSGi ミドルウェア・バンドルの更新』および 145 ページの『server.xml の手動調整』を参照してください。

## プロジェクト・ビルド・パスの更新

プロジェクト・ビルド・パスの更新方法。

### このタスクについて

動的 Web プロジェクトを使用する場合は、デプロイメント用の WAR ファイル・アーカイブが作成されます。この場合、Eclipse の OSGi フレームワークは使用されないため、WebSphere MQ JAR ファイルをプロジェクト・ビルド・パスに追加する必要があります。

### 手順

1. Eclipse で Web プロジェクトを選択し、**「ビルド・パス」** > **「ビルド・パスの構成」**を右クリックします。これにより、「Java のビルド・パス」ウィンドウが表示されます。
2. CICS および Liberty ライブラリーを追加し、**「ライブラリーの追加」** > **「Liberty JVM サーバー」** > **「次へ」** > **「終了」**をクリックします。
3. **「外部 JAR の追加」**をクリックし、前にダウンロードした WebSphere MQ JAR ファイルが配置されているディレクトリーにナビゲートします。アプリケーションで使用するインポートに応じて、以下の JAR ファイルを選択します。
   - com.ibm.mq.jar
   - com.ibm.mq.jmql.jar
   - com.ibm.mq.headers.jar

   **注:** ステップ 3 は、WebSphere MQ のみを使用する場合のオプションです。

### タスクの結果

プロジェクトのビルド・パスに、CICS と WebSphere MQ の両方の API を使用する Web アプリケーションを開発するための適切なインターフェースが含まれるようになりました。

---

## JVM の状態の制御

CICS で実行される Java アプリケーションの設計と開発を行うときには、アプリケーションが、JVM を不適切な状態のままにしたり、その状態を望ましくない方法で変更したりしないようにしてください。 JVM の状態の制御には CICS サービスが役立ちます。

## JVM の状態の保護

ご使用のアプリケーションにより JVM の状態が変わる場合は、そのアプリケーションも元の状態にリセットされることを確実にしてください。例えば、アプリケーションはデフォルトのタイム・ゾーンをリセットし、このタイム・ゾーンに基づいて計算を行うことができます。同じ JVM を使用するその他のアプリケーションは、新しいデフォルトのタイム・ゾーンを使用しますが、このタイム・ゾーンが適切でない場合があります。

Java アプリケーションは JVM サーバー・ランタイム環境で実行され、同じ JVM の別のスレッドでも実行されている可能性のある他のアプリケーションから分離されません。 JVM に対してアプリケーションが加える変更はすべて、その JVM で実行されている他のすべてのアプリケーションに影響を与えます。

ご使用のアプリケーションで System.exit() メソッドを使用しないでください。System.exit() メソッドを使用すると、JVM サーバーと CICS の両方がシャットダウンするため、アプリケーションの状態に影響を与える可能性があります。

## JVM における静的状態の制御

JVM を不要な状態のままにしないでください。状態は、JVM サーバーで実行されているすべてのアプリケーション間で共用されます。

変更可能なクラス・フィールドの状態に依存する場合、アプリケーションは独自の静的ストレージを再初期化しなければなりません。すべてのアプリケーション・クラスとシステム・クラスについて、静的変数の値は JVM 内で持続します。これには、アプリケーションによって明示的に使用されていなくてもアプリケーションに影響を与える可能性があるクラス、および静的イニシャライザーで使用されている値が含まれます。

大部分の場合、静的変数はストレージの再初期設定を避けるために使用され、静的変数が持続することを可能にすると、パフォーマンスを改善することができます。これらの変数の値のリセットがアプリケーションで必要な場合、アプリケーションは値自体をリセットする必要があります。アプリケーション設計の一部として意図的に含まれなかった、変更可能なクラス・フィールドと静的イニシャライザーの識別と除去を試みてください。

可能な場合は常に、クラス・フィールドを private および final として定義してください。native メソッドは final クラス・フィールドに書き込むことができ、private 以外のメソッドは、そのクラス・フィールドによって参照されるオブジェクトを取得でき、オブジェクトまたは配列の状態を変更できます。

情報をプログラム呼び出し間で持続させたい場合は、Java アプリケーションの設計の際に有利に状態を伝える機能を使用できます。静的状態と静的状態により参照されるオブジェクト・インスタンスは JVM 内で持続するので、同じ JVM の同じアプリケーションの将来の実行に役立つ可能性がある永続項目をアプリケーションが作成することは許容されます。

例えば、ある操作で DB2 情報を読み取って複合データ構造体を構成するとします。これは、コストのかかる操作であり、絶対必要な回数よりも多く繰り返したくありません。その複雑なデータ構造体をアプリケーションの静的ストレージに保管

すれば、同じ JVM 内でそのアプリケーションを後で実行したときにもアクセスすることができるので、不要な初期化を避けることができます。オブジェクトが静的ストレージ、つまり静的クラス・フィールドにアンカーされる場合、決してガーベッジ・コレクションの候補になりません。

JVM サーバーでは、システム・プログラマーが JVM サーバーを使用不可にするまで、すべてのアプリケーションの静的状態が持続します。OSGi バンドル・アクティベーター・クラスを提供すれば、JVM サーバーが再始動されてもオブジェクトの状態を維持することができます。これらのクラスには、JCICS 呼び出しを含むことはできません。

### 使用後の DB2 接続、ソケット、およびその他のタスク存続期間中のシステム・リソースのクローズ

Java アプリケーションは JVM サーバー・ランタイム環境で実行されるため、異なるアプリケーションから DB2 への複数の接続が可能です。したがって、DB2 でタスクが終了したら、接続をクローズすることがベスト・プラクティスですが、必須ではありません。これは、タスクが完了すると接続が削除されるからです。

アプリケーションのスレッドを開始して、java.net パッケージを使用したソケットの管理を行う場合、そのアプリケーションが接続を管理し、クローズする必要があります。 java.net クラスを使用して作成されたソケットは、CICS ソケット・ドメインではなく、JVM のネイティブ・ソケット機能を使用します。CICS は、これらのソケットを使用して実行される通信の管理もモニターも行うことはできません。

アプリケーションによって使用されるその他のタスク存続期間中のシステム・リソースにも同じことが当てはまります。使用後に解放されなければなりません。

### スレッド・セーフ問題があるかどうかについてのアプリケーションのテスト

JVM サーバー・ランタイム環境は非スレッド・セーフの Java アプリケーションの使用をサポートしないので、スレッド・セーフの Java アプリケーションを作成する必要があります。複数のスレッドで同時に使用されている場合でも、オブジェクトが常に有効な状態を維持していることを確認してください。 Java アプリケーションが、所定の JVM で最初に使用されるときに正しく機能するものの、それ以降の使用時に正しく動作しない場合、この問題はおそらくスレッド・セーフの問題が原因です。

## Java からの構造化データとの対話

多くの場合、CICS Java プログラムは、当初は他のプログラミング言語用に設計されたデータと対話します。例えば、Java プログラムは、COBOL コピーブックで定義された COMMAREA を使用して COBOL プログラムにリンクしたり、C++ ヘッダー・ファイルを使用してデータが定義される VSAM ファイルからレコードを読み取ったりすることができます。インポーターを使用すると、これらの形式の構造化データと対話することができます。

## JZOS および J2C を使用した、Java へのアプリケーション・データのインポート

CICS はコピーブック・インポーターをサポートするので、Java で他のプログラミング言語からの構造化データを使用することができます。サポートされているインポーターは、JZOS ツールと Rational で提供されています。 Rational ツールでは、Java EE コネクター・アーキテクチャー (JCA、別名 J2C) を使用します。

インポーターは、ソース・プログラムに含まれるデータ型をマップするので、アプリケーションはデータ構造内の個々のフィールドにアクセスできます。JZOS または Rational の J2C ツールを使用すると、データと対話して Java クラスを作成することができます。その結果、CICS で Java と他のプログラム間でデータを受け渡すことができます。

CICS は、以下のインポーターからの Java 成果物をサポートします。

- Rational Application Developer (RAD) および Rational Developer for z System z® における J2C ツールからのデータ・バインディング Bean
- IBM JZOS Batch Toolkit for z/OS SDK からのレコード

IBM Redbooks® 資料「Java Application Development for CICS」では、既存の COBOL アプリケーションを操作する Heritage Trader アプリケーションというサンプル・アプリケーションを使用します。以下のトピックに関する情報が提供されます。

- JZOS および J2C のインストール方法
- JCICS への COBOL アプリケーションのマイグレーション
- J2C 用の Java データ・バインディング・クラスの作成
- JZOS でのラッパー・クラスの生成
- JCICS API を使用した、Web、ファイル、および DB2 アクセスの実装例

### J2C 要件

エンタープライズ・アプリケーションの作成に使用できる Java EE コネクター成果物を作成することができます。 RAD J2C ウィザードは、COBOL およびその他のアプリケーション・プログラム・データ構造にマップするクラスまたは一連のクラスの作成に役立ちます。

Rational J2C インポーターを使用するには、Windows または Linux ワークステーションに RAD が必要です。

### JZOS 要件

IBM JZOS Batch Toolkit for z/OS SDK は、z/OS で Java バッチ機能を提供する 1 組のツールです。JZOS には、バッチ・ジョブまたは開始タスクとして Java アプリケーションを直接実行するためのランチャー、および Java アプリケーションから直接使用可能な従来の z/OS データと主なシステム・サービスにアクセスする 1 組の Java メソッドが含まれます。

JZOS は、COBOL コピーブックおよびアセンブラー DSECT からのレコード・クラスの自動生成をサポートします。

JZOS ダウンロードには、PDF 形式で「*JZOS COBOL Record Generator User's Guide*」および「*JZOS Assembler Record Generator User's Guide*」が入っています。

**IBM Redbooks**

Java Application Development for CICS

Java Connectors for CICS Featuring the Java EE Connector Architecture

Java Stand-alone Applications on z/OS Volume 2

**J2C 情報**

エンタープライズ情報システムへの接続 (Rational Application Developer for WebSphere ソフトウェア製品資料)

COBOL Importer の概要 (Rational Application Developer for WebSphere ソフトウェア製品資料)

CICS Transaction Gateway for z/OS

**JZOS 情報**

JZOS Java Launcher and Toolkit Overview

「IBM SDK for z/OS, Java Technology Edition Installation and User's Guide」の『JZOS Batch Launcher and Toolkit function』

## JCICS を使用した Java の開発

CICS Java クラス・ライブラリー (JCICS) を使用して CICS サービスにアクセスする Java アプリケーションを作成できます。JCICS は、Java において、CICS でサポートされている他の言語 (COBOL など) に提供される **EXEC CICS** アプリケーション・プログラミング・インターフェースに相当するものです。

JCICS を使用すると、CICS リソースにアクセスする Java アプリケーションを作成し、他の言語で作成されたプログラムと統合することができます。**EXEC CICS** API の大部分の機能がサポートされます。このライブラリーは、`com.ibm.cics.server.jar` ファイルで CICS および CICS Explorer SDK に提供されています。

`com.ibm.cics.server` パッケージのレベルと、使用する CICS バージョンの間のマッピングは、レベルによって異なります。マッピングは以下のとおりです。

- CICS TS 4.1: `com.ibm.cics.server 1.201.0`
- CICS TS 4.2: `com.ibm.cics.server 1.300.0`
- CICS TS 5.1: `com.ibm.cics.server 1.401.0`
- CICS TS 5.2: `com.ibm.cics.server 1.500.0`

43 ページの『CICS 用 Java クラス・ライブラリー (JCICS)』
JCICS では、**EXEC CICS** API コマンドのほとんどの機能がサポートされます。

48 ページの『データ・エンコード』
JVM は文字エンコードに CICS とは異なるコード・ページを使用できます。CICS では常に EBCDIC コード・ページを使用しなければなりませんが、JVM

では ASCII などの別のエンコード方式を使用できます。JCICS API を使用するアプリケーションを開発する際には、使用するエンコード方式が正しいものであることを確認する必要があります。

49 ページの『JCICS API サービスと例』
CICS は、Java アプリケーション用のさまざまな API とサービスをサポートしています。 EXEC CICS API を介して非 Java プログラムで使用可能なサービスの多くは、Java SDK で提供される標準の Java SE API とともに、JCICS API を介して Java プログラムで使用可能です。

75 ページの『JCICS の使用』
通常の Java クラスのように、JCICS ライブラリーからクラスを使用します。アプリケーションは必要なタイプの参照を宣言し、クラスの新しいインスタンスが `new` 演算子を使用して作成されます。

76 ページの『Java の制約事項』
Java アプリケーションを開発する際に、適用すべきさまざまな制約事項があります。制約事項に従わないと、アプリケーションが CICS で実行されているときに問題が発生する可能性があります。

## CICS 用 Java クラス・ライブラリー (JCICS)

JCICS では、**`EXEC CICS`** API コマンドのほとんどの機能がサポートされます。

JCICS クラスは、クラス定義から生成される Javadoc で詳しく説明されています。Javadoc は JCICS Class Reference で入手可能です。

44 ページの『JavaBeans』
JCICS の一部のクラスは JavaBeans として使用できます。つまり、Eclipse などのアプリケーション開発ツールでカスタマイズし、直列化し、JavaBeans API を使用して操作することができます。

44 ページの『ライブラリー構造』
各 JCICS ライブラリー・コンポーネントは、4 つのカテゴリー (インターフェース、クラス、例外、エラー) のいずれかに分類されます。

45 ページの『CICS リソース』
プログラムや一時記憶域キューなどの CICS リソースは、該当する Java クラスのインスタンスによって表され、リソースの名前などの各種プロパティーの値によって識別されます。

45 ページの『データを渡すための引数』
チャネルとコンテナーを使用するか、通信域 (COMMAREA) を使用して、プログラム間でデータを受け渡すことができます。

46 ページの『直列化可能クラス』
JCICS 直列化可能クラスのリスト。

46 ページの『Task.out および Task.err』
Java 関連の CICS タスクごとに、CICS は、標準出力および標準エラー・ストリームとして使用できる、2 つの Java `PrintWriters` クラスを自動的に作成します。標準出力と標準エラー・ストリームは、`out` と `err` と呼ばれる、`Task` クラス内のパブリック・フィールドです。

47 ページの『スレッド』
JVM サーバー環境では、OSGi フレームワークで実行されているアプリケーションは ExecutorService を使用して、CICS タスクで非同期的に実行するスレッドを作成できます。

## JavaBeans

JCICS の一部のクラスは JavaBeans として使用できます。つまり、Eclipse などのアプリケーション開発ツールでカスタマイズし、直列化し、JavaBeans API を使用して操作することができます。

以下の JavaBeans が JCICS で使用可能です。

- Program
- ESDS
- KSDS
- RRDS
- TDQ
- TSQ
- AttachInitiator
- EnterRequest

これらの Bean はイベントを定義しません。プロパティーとメソッドで構成されます。次の 3 つの方法のいずれかで実行時にインスタンス化することができます。

- クラス自体の new メソッドを呼び出す。この方法が優先されます。
- プロパティー値を手動で設定して、クラスの名前の Beans.instantiate() を呼び出す。
- プロパティー値を設計時に設定して、`.ser` ファイルの Beans.instantiate() を呼び出す。

最初の 2 つのオプションのどちらかを選択する場合、プロパティー値 (CICS リソースの名前を含めて) は、実行時に適切な set メソッドを呼び出すことによって設定されなければなりません。

## ライブラリー構造

各 JCICS ライブラリー・コンポーネントは、4 つのカテゴリー (インターフェース、クラス、例外、エラー) のいずれかに分類されます。

**インターフェース**
一部のインターフェースは、1 組の定数を定義するために提供されます。例えば、TerminalSendBits インターフェースは、`java.util.BitSet` の構成に使用できる 1 組の定数を提供します。

**クラス**
指定されたクラスは、大部分の JCICS 機能を提供します。`API` クラスは抽象クラスであり、ABEND と例外を除いて、CICS API の一部に対応する、すべてのクラスに共通する初期化を提供します。例えば、`Task` クラスは、CICS タスクに対応する 1 組のメソッドと変数を提供します。

### エラーと例外

Java 言語は、クラス Throwable のサブクラスとして例外とエラーの両方を定義します。JCICS は、Error のサブクラスとして CicsError を定義します。CicsError は、重大エラーに使用される他のすべての CICS エラー・クラスのスーパークラスです。

JCICS は、Exception のサブクラスとして CicsException を定義します。CicsException は、(CICS QIDERR 条件を表す、InvalidQueueIdException などの CicsConditionException クラスを含む) すべての CICS 例外クラスのスーパークラスです。

詳しくは、 54 ページの『エラー処理と異常終了』を参照してください。

## CICS リソース

プログラムや一時記憶域キューなどの CICS リソースは、該当する Java クラスのインスタンスによって表され、リソースの名前などの各種プロパティーの値によって識別されます。

CICS リソースは、CICS Explorer、CEDA トランザクション、または CICSPlex® SM WUI を使用して定義します。暗黙的なリモート・アクセスを使用するには、リモート・リソースを指すリソースをローカルで定義します。

CICS リソースの定義について詳しくは、 「*CICS Resource Definition Guide*」または「*CICSPlex System Manager 概念および計画*」マニュアルを参照してください。

## データを渡すための引数

チャネルとコンテナーを使用するか、通信域 (COMMAREA) を使用して、プログラム間でデータを受け渡すことができます。

COMMAREA を使用する場合、一度に渡すデータは 32 KB に制限されます。チャネルとコンテナーを使用する場合、プログラム間で 32 KB より多く渡すことができます。COMMAREA またはチャネル、およびその他のすべてのパラメーターは、引数として該当するメソッドに渡されます。

メソッドの多くは多重定義されています。すなわち、バージョンが異なれば、取る引数の数か、引数のタイプのどちらかが異なります。引数がないか、最小限の必須引数があるメソッドや、すべての引数があるメソッドがあります。例えば、Program クラスには、次のようにさまざまな link() メソッドが含まれています。

**link()**
このメソッドは、COMMAREA を使用してデータを受け渡したり、他のオプションを指定したりせず、単純なリンクを行います。

**link(com.ibm.cics.server.CommAreaHolder)**
このメソッドは、COMMAREA を使用してデータを受け渡すのみで、他のオプションを指定せず、単純なリンクを行います。

**link(com.ibm.cics.server.CommAreaHolder, int)**
このメソッドは、COMMAREA を使用してデータを受け渡し、DATALENGTH 値を使用して COMMAREA 内のデータの長さを指定することにより、分散リンクを行います。

**link(com.ibm.cics.server.Channel)**
このメソッドは、チャネルを使用して 1 つ以上のコンテナー内のデータを受け渡すことによって、リンクを行います。

**関連情報**:

55 ページの『チャネルとコンテナーの例』
コンテナー は、プログラム間で情報を渡すための、データの名前付きブロックです。コンテナーは、チャネル と呼ばれる集合にグループ化されます。この資料では、Java アプリケーションでチャネルとコンテナーを使用する方法について説明し、いくつかのサンプル・コードを示します。

## 直列化可能クラス

JCICS 直列化可能クラスのリスト。

- AddressResource
- AttachInitiator
- CommAreaHolder
- EnterRequest
- ESDS
- File
- KeyedFile
- KSDS
- NameResource
- Program
- RemotableResource
- Resource
- RRDS
- StartRequest
- SynchronizationResource
- SyncLevel
- TDQ
- TSQ
- TSQType

**関連資料**:

69 ページの『直列化サービス』
JCICS は、タスクによるリソースの使用をスケジュールできる CICS 直列化サービスをサポートします。

## Task.out および Task.err

Java 関連の CICS タスクごとに、CICS は、標準出力および標準エラー・ストリームとして使用できる、2 つの Java `PrintWriters` クラスを自動的に作成します。標準出力と標準エラー・ストリームは、`out` と `err` と呼ばれる、`Task` クラス内のパブリック・フィールドです。

CICS タスクが端末 (この場合、端末は基本機構 と呼ばれます) から駆動される場合、CICS は標準出力および標準エラー・ストリームをタスクの端末にマップします。

タスクに基本機構としての端末がない場合、標準出力および標準エラー・ストリームは System.out および System.err に送信されます。

## スレッド

JVM サーバー環境では、OSGi フレームワークで実行されているアプリケーションは ExecutorService を使用して、CICS タスクで非同期的に実行するスレッドを作成できます。

CICS には、Java ExecutorService インターフェースが実装されています。この実装では、JCICS API を使用して CICS サービスにアクセスできるスレッドが作成されます。 JVM サーバーは始動時に、OSGi サービスとして CICS ExecutorService を登録します。 Java Thread クラスではなくこのサービスを使用して、JCICS を使用できるタスクを作成してください。

CICS で提供される ExecutorService は、スレッドを作成するためにアプリケーションで使用できるように、OSGi フレームワークにおいて優先順位が高いものとして登録されます。アプリケーションは通常、サービスをフィルタリングして特定の実装を使用するのでない限り、最も高い優先順位の ExecutorService を使用します。

アプリケーションにスレッドを作成する場合、OSGi レジストリーから汎用 ExecutorService を使用する方式が推奨されています。アプリケーションが JVM サーバーで実行されているときに、OSGi レジストリーは自動的に CICS ExecutorService を使用して CICS スレッドを作成します。この方式により、アプリケーションは実装環境とは分離されるので、JCICS API メソッドを使用してスレッドを作成する必要がなくなります。

ただし、CICS に固有のアプリケーションを作成している場合は、JCICS API 内の CICSExecutorService クラスを使用して、新しいスレッドを要求できます。

### CICSExecutorService

このクラスは java.util.concurrent.ExecutorService インターフェースを実装します。CICSExecutorService クラスには runAsCICS() という静的メソッドが用意されています。このメソッドを使用して Runnable Java オブジェクトをサブミットして、新しい JCICS 対応スレッドで実行できます。 runAsCICS() メソッドは、OSGi レジストリー検索を実行し、アプリケーションの CICSExecutorService のインスタンスを取得するユーティリティー・メソッドです。

このクラスは Java ExecutorService インターフェースの実装として登録されるため、ExecutorService を要求するアプリケーションには、JVM サーバーでの実行時に CICSExecutorService のみが指定されます。

```
CICSExecutorService.runAsCICS(Runnable runnable)
```

### 制約事項

JCICS を使用できるスレッドを作成するには、execute() メソッドを使用する必要があります。 submit() メソッドを使用すると、アプリケーションは、JCICS を実行できない Java スレッドを取得します。

OSGi フレームワーク (Axis2 Java プログラムなど) で実行されていないアプリケーションの場合は、ExecutorService を使用できないため、初期アプリケーション・スレッドのみで JCICS にアクセスできます。さらに、以下のいずれかのアクションを行う前に、初期スレッド以外のすべてのスレッドが終了している必要があります。

- com.ibm.cics.server.Program クラスでの link メソッド
- com.ibm.cics.server.TerminalPrincipalFacility クラスでの setNextTransaction(String) メソッド
- com.ibm.cics.server.TerminalPrincipalFacility クラスでの setNextCOMMAREA(byte[]) メソッド
- com.ibm.cics.server.Task クラスでの commit() メソッド
- com.ibm.cics.server.Task クラスでの rollback() メソッド
- com.ibm.cics.server クラスから `AbendException` 例外を返す。

**関連資料**:

69 ページの『スレッドとタスクのサンプル』
CICSExecutorService を使用して、CICS 内でタスクを開始するスレッドを作成できます。このサービスを使用してスレッドを作成すると、JCICS API を使用して CICS サービスにアクセスできる CICS タスクが作成されます。

## データ・エンコード

JVM は文字エンコードに CICS とは異なるコード・ページを使用できます。CICS では常に EBCDIC コード・ページを使用しなければなりませんが、JVM では ASCII などの別のエンコード方式を使用できます。JCICS API を使用するアプリケーションを開発する際には、使用するエンコード方式が正しいものであることを確認する必要があります。

JCICS API は、基礎となっている JVM ではなく CICS 領域で指定されているコード・ページを使用します。したがって、JVM が異なるファイル・エンコード方式を使用する場合には、アプリケーションが異なるコード・ページを処理する必要が生じます。 CICS が使用しているコード・ページの判別に役立つように、CICS はいくつかの Java プロパティーを提供しています。

- **com.ibm.cics.jvmserver.supplied.ccsid** プロパティーは、CICS 領域に指定されているコード・ページを返します。デフォルトでは、JCICS API は文字エンコードに、このコード・ページを使用します。ただし、この値は JVM サーバー構成でオーバーライドできます。
- **com.ibm.cics.jvmserver.override.ccsid** プロパティーは、JVM プロファイル内のオーバーライド値を返します。JCICS API は文字エンコードに、CICS 領域で使用されているコード・ページではなく、この値で示されたコード・ページを使用します。
- **com.ibm.cics.jvmserver.local.ccsid** プロパティーは、JVM サーバーで JCICS API が文字エンコードに使用しているコード・ページを返します。

Java アプリケーションにこれらのプロパティーを設定して、JCICS のエンコード方式を変更することはできません。コード・ページを変更するには、システム管理者に、JVM プロファイルを更新して JVM システム・プロパティー **-Dcom.ibm.cics.jvmserver.override.ccsid** を追加するよう依頼する必要があります。

### エンコードの例

java.lang.String パラメーターを入力として受け入れる JCICS メソッドはいずれも、データが CICS に渡される前に、正しいコード・ページで自動的にエンコードされます。同様に、JCICS API から返されるすべての java.lang.String 値も、正しいコード・ページでエンコードされます。JCICS API は、ほとんどのクラスにヘルパー・メソッドを提供しています。これらのヘルパー・メソッドはアプリケーションのために、ストリングやデータを処理してコード・ページを判別し、設定します。

アプリケーションが String.getBytes() メソッドまたは new String(byte[] bytes) メソッドを使用する場合、そのアプリケーションは必ず正しいエンコード方式を使用する必要があります。これらのメソッドをアプリケーションで使用する場合、以下のように Java プロパティーを使用すれば、データを正しくエンコードすることができます。

```
String.getBytes(System.getProperty("com.ibm.cics.jvmserver.local.ccsid"))
String(bytes, System.getProperty("com.ibm.cics.jvmserver.local.ccsid"))
```

以下の例に、アプリケーションが COMMAREA からフィールドを読み取る場合の JCICS エンコードの使用方法を示します。

```
public static void main(CommAreaHolder ca)
{
 //Convert first 8 bytes of ca into a String using JCICS encoding
String str=new String(ca.getValue(), 0, 8, System.getProperty("com.ibm.cics.jvmserver.local.ccsid"));
}
```

## JCICS API サービスと例

CICS は、Java アプリケーション用のさまざまな API とサービスをサポートしています。 EXEC CICS API を介して非 Java プログラムで使用可能なサービスの多くは、Java SDK で提供される標準の Java SE API とともに、JCICS API を介して Java プログラムで使用可能です。

以下のトピックでは、JCICS サービス、および Java 例外処理との統合について詳しく説明しています。 Liberty JVM サーバーでは、他の JEE API を使用できます。詳しくは、 103 ページの『Liberty JVM サーバーで実行する Java アプリケーションの開発』を参照してください。

51 ページの『Java プログラムにおける CICS 例外処理』
CICS で発生する問題に対処するために、CICS ABEND および例外が Java 例外処理体系に組み込まれています。

53 ページの『Java Web アプリケーションにおける CICS 例外処理』
Liberty Web コンテナーが CICS アプリケーションで発生する問題に対処するために、CICS ABEND および例外が Java 例外処理体系に組み込まれています。Web アプリケーションで処理されない Java 例外は Web コンテナーによってキ

ャッチされ、サーブレット例外処理プロセスを起動します。この処理の一部として、コミットされていない CICS 作業単位は CICS によってロールバックされます。

54 ページの『エラー処理と異常終了』
Java プログラムから ABEND を開始するには、Task.abend() または Task.forceAbend() メソッドのいずれかを呼び出す必要があります。

55 ページの『APPC マップ式会話』
APPC 非マップ式会話は、JCICS API からサポートされません。

55 ページの『基本マッピング・サポート (BMS)』
基本マッピング・サポート (BMS) は、CICS プログラムと端末装置間のアプリケーション・プログラミング・インターフェースです。JCICS は、BMS アプリケーション・プログラミング・インターフェースの一部をサポートします。

55 ページの『チャネルとコンテナーの例』
コンテナー は、プログラム間で情報を渡すための、データの名前付きブロックです。コンテナーは、チャネル と呼ばれる集合にグループ化されます。この資料では、Java アプリケーションでチャネルとコンテナーを使用する方法について説明し、いくつかのサンプル・コードを示します。

59 ページの『診断サービス』
JCICS アプリケーション・プログラミング・インターフェースは、以下の CICS トレースおよびダンプ・コマンドをサポートします。

60 ページの『文書サービス』
このセクションでは、DOCUMENT アプリケーション・プログラミング・インターフェースにおけるコマンドの JCICS サポートについて説明します。

60 ページの『環境サービス』
CICS 環境サービスは、アプリケーション・プログラムに関連する CICS データ域、パラメーター、およびリソース属性にアクセスできるようにします。

63 ページの『ファイル・サービス』
JCICS は、CICS ファイルおよび索引のタイプごとに **EXEC CICS** API コマンドにマップされるクラスおよびメソッドを提供します。

66 ページの『HTTP サービスおよび TCP/IP サービス』
`HttpHeader`、`NameValueData`、および `FormField` クラスの getter は、HTTP ヘッダー、名前と値のペア、および該当する API コマンドのフォーム・フィールド値を戻します。

67 ページの『プログラム・サービス』
JCICS は、CICS プログラム制御コマンド LINK、RETURN、および INVOKE APPLICATION をサポートします。

68 ページの『スケジューリング・サービス』
JCICS は、CICS スケジューリング・サービスをサポートします。これにより、タスク用に保管されたデータを取り出し、インターバル制御要求を取り消し、指定された時間にタスクを開始することができます。

69 ページの『直列化サービス』
JCICS は、タスクによるリソースの使用をスケジュールできる CICS 直列化サービスをサポートします。

69 ページの『ストレージ・サービス』
CICS サービスを使用した明示的なストレージ管理 (**EXEC CICS** GETMAIN など)

はサポートされません。標準の Java ストレージ管理機能で、タスク専用ストレージのニーズを十分に満たすことができます。

69 ページの『スレッドとタスクのサンプル』
CICSExecutorService を使用して、CICS 内でタスクを開始するスレッドを作成できます。このサービスを使用してスレッドを作成すると、JCICS API を使用して CICS サービスにアクセスできる CICS タスクが作成されます。

70 ページの『一時記憶域キュー・サービス』
JCICS は、CICS 一時記憶コマンド DELETEQ TS、READQ TS、および WRITEQ TS をサポートします。

71 ページの『端末サービス』
JCICS は、以下の CICS 端末サービス・コマンドをサポートします。

72 ページの『データと XML の間の変換』
JCICS は、データから XML への変換とその逆の変換を実行するための API コマンドをサポートしています。それらのコマンドは、**EXEC CICS TRANSFORM DATATOXML** コマンドおよび **TRANSFORM XMLTODATA** コマンドと同等の機能を提供します。

73 ページの『一時データ・キュー・サービス』
JCICS は、CICS 一時データ・コマンド DELETEQ TD、READQ TD、および WRITEQ TD をサポートします。INTO オプションを除くすべてのオプションがサポートされます。

74 ページの『作業単位 (UOW) サービス』
JCICS は、CICS SYNCPOINT サービスをサポートします。

74 ページの『Web サービスの例』
JCICS は、アプリケーション内で Web サービスを操作するために使用できるすべての API コマンドをサポートします。

## Java プログラムにおける CICS 例外処理

CICS で発生する問題に対処するために、CICS ABEND および例外が Java 例外処理体系に組み込まれています。

通常の CICS ABEND はすべて単一の Java 例外 `AbendException` にマップされます。一方、CICS 条件はそれぞれ別の Java 例外にマップされます。これにより、Java の ABEND 処理モデルは他のプログラミング言語と同じようになります。つまり、すべての ABEND について単一のハンドラーに制御が与えられ、そのハンドラーは特定の ABEND を照会してから、処理内容を決定する必要があります。

条件を表す例外は、CICS 自体によってキャッチされると ABEND になります。

Java 例外処理は、他の言語の ABEND および条件処理に完全に統合されるため、ABEND は、言語に依存しない標準的な方法で、Java プログラムと非 Java プログラム間に波及することができます。条件は、ABEND にマップされてから、その条件の原因になったかその条件を検出したプログラムを終了します。

ただし、他のプログラミング言語の ABEND 処理モデルには、次のような複数の相違点があります。これらの相違点は、Java 例外処理体系の性質や、Java API の基礎となる一部のテクノロジーの実装に起因します。

- 他のプログラミング言語では処理できない ABEND を、Java プログラムでキャッチできます。これらの ABEND は通常、同期点処理時に発生します。これらの ABEND が Java アプリケーションを中断しないようにするために、チェックなし例外の拡張にマップされます。したがって、これらの ABEND の宣言もキャッチも必要ありません。
- プログラム終了などの複数の内部 CICS イベントも、Java 例外にマップされるので、Java アプリケーションでキャッチできます。また、正常な状態を中断しないように、これらのイベントはチェックなし例外の拡張にマップされ、キャッチも宣言も必要ありません。

例外の以下の 3 つのクラス階層が CICS に関連しています。

1. CicsError。java.lang.Error を拡張し、AbendError および UnknownCicsError のベースです。
2. CicsRuntimeException。java.lang.RuntimeException を拡張する一方で、以下によって拡張されます。

   **AbendCancelException**
   CICS ABEND CANCEL を表します。

   **AbendException**
   通常の CICS ABEND を表します。

   **EndOfProgramException**
   リンク先プログラムが通常どおりに終了したことを示します。
3. CicsException。java.lang.Exception を拡張し、サブクラスを持ちます。

   **CicsConditionException.**
   すべての CICS 条件の基本クラス。

『CICS のエラー処理コマンド』
Java で **EXEC CICS** のエラー処理コマンドをサポートする方式について説明します。

53 ページの『CICS 条件』
Java の条件処理モデルは、他の CICS プログラミング言語とは異なります。

### CICS のエラー処理コマンド:

Java で **EXEC CICS** のエラー処理コマンドをサポートする方式について説明します。

CICS の条件処理は、 51 ページの『Java プログラムにおける CICS 例外処理』で説明しているように、Java の例外処理体系に組み込まれています。以下のように、**EXEC CICS** の同等のコマンドが Java でサポートされています。

**HANDLE ABEND**
CICS でサポートされる任意の言語のプログラムによって生成された ABEND を処理するには、catch 節に AbendException を指定して、Java の try-catch ステートメントを使用します。

**HANDLE CONDITION**
PGMIDERR などの特定の条件を処理するには、適切な例外の名前を指定した catch 節を使用します。この場合は、InvalidProgramException です。あるいは、すべての CICS 条件をキャッチする場合は、CicsConditionException を指定した catch 節を使用します。

**IGNORE CONDITION**
このコマンドは、Java アプリケーションでは適切ではありません。

**POP HANDLE および PUSH HANDLE**
これらのコマンドは、Java アプリケーションでは適切ではありません。CICS ABEND および条件を表すために使用される Java 例外は、スコープ内の catch ブロックによってキャッチされます。

**CICS 条件:**

Java の条件処理モデルは、他の CICS プログラミング言語とは異なります。

COBOL では、条件ごとに条件処理ラベルを定義できます。その条件が CICS コマンドの処理中に発生する場合、制御はラベルに転送されます。

C および C++ では、条件に例外処理ラベルを定義することはできません。条件を検出するために、各 CICS コマンドの後に、EIB 内の RESP フィールドが検査されなければなりません。

Java では、CICS コマンドによって戻される条件はすべて、Java 例外にマップされます。すべての CICS コマンドを try-catch ブロックに組み込んで、条件ごとに特定の処理を行うか、特定の例外が適切でない場合は、単一の null catch 節を持つことができます。または、条件を波及させて、より大きいスコープで catch 節によって処理できるようにすることができます。

## Java Web アプリケーションにおける CICS 例外処理

Liberty Web コンテナーが CICS アプリケーションで発生する問題に対処するために、CICS ABEND および例外が Java 例外処理体系に組み込まれています。 Web アプリケーションで処理されない Java 例外は Web コンテナーによってキャッチされ、サーブレット例外処理プロセスを起動します。この処理の一部として、コミットされていない CICS 作業単位は CICS によってロールバックされます。

HttpServlet インターフェースを拡張するすべての Java Web アプリケーションは、HttpServlet インターフェースに定義されているように、`IOException` や `ServletException` 以外のすべてのチェック例外を処理する必要があります。チェック例外には、未処理の CICS 条件を表す `com.ibm.cics.server.CicsConditionException` のサブクラスがすべて含まれます。したがって、CICS 条件をキャッチするすべての例外処理コードは、作業単位をロールバックする必要があり、例に示されているように、Task オブジェクトで rollback() メソッドを使用して明示的に同期点ロールバックを呼び出すか、`AbendException` をスローするエラー条件をすべて識別する必要があります。

```
try
{
 TSQ tsqQ = new TSQ();
 tsqQ.setName("tsq1");
 tsqQ.writeString("input data");
 }  catch (IOErrorException e) {
          // Log error
          try
          {
          Task.getTask().rollback();
```

```
        } catch   (InvalidRequestException e1)      {
           throw new RuntimeException(e1);
        }
}
```

java.lang.RuntimeException のサブクラスであるチェックなし Java 例外は、Web アプリケーションを含むすべての Java アプリケーションによってスローされる可能性があり、`com.ibm.cics.server.AbendException` および `com.ibm.cics.server.AbendCancelException` を含みます。したがって、`AbendException` をスローするか、トランザクションの異常終了を処理しない Web アプリケーションは、サーブレット例外処理プロセスおよび関連する作業単位のロールバック処理を起動します。

Java Transaction API (JTA) を使用して作業単位をコミットする Web アプリケーションは、Liberty トランザクション・マネージャーの制御に従ってコミットされます。詳細については、109 ページの『Java Transaction API (JTA)』を参照してください。

## エラー処理と異常終了

Java プログラムから ABEND を開始するには、Task.abend() または Task.forceAbend() メソッドのいずれかを呼び出す必要があります。

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| abend()、forceAbend() | タスク | ABEND |

**ABEND**

Java プログラムから ABEND を開始するには、Task.abend() メソッドのいずれかを呼び出します。これにより、アベンド条件が CICS で設定され、AbendException がスローされます。AbendException が上位レベルのアプリケーション・オブジェクト内でキャッチされないか、呼び出し側プログラムに登録されている ABEND ハンドラー (ある場合) によって処理される場合、CICS はトランザクションを終了し、ロールバックします。

各種 abend() メソッドは次のとおりです。

- abend(String *abcode*)。ABEND コード *abcode* で ABEND が生じます。
- abend(String *abcode*, boolean *dump*)。ABEND コード *abcode* で ABEND が生じます。 **dump** パラメーターが false である場合、ダンプは取られません。
- abend()。ABEND コードもダンプもない ABEND が生じます。

**ABEND CANCEL**

処理できない ABEND を開始するには、Task.forceAbend() メソッドのいずれかを呼び出します。上記のように、これにより、AbendCancelException がスローされ、Java プログラムでキャッチされます。これを行う場合は、**ABEND_CANCEL** 処理を完了するために例外を再スローする必要があります。その結果、制御が CICS に戻ると、CICS はトランザクションを終了し、ロールバックします。通知目的の AbendCancelException のキャッチのみを行ってから、再スローしてください。

各種 forceAbend() メソッドは次のとおりです。

- forceAbend(String *abcode*)。ABEND コード *abcode* で **ABEND CANCEL** が生じます。

- forceAbend(String *abcode*, boolean *dump*)。ABEND コード *abcode* で **ABEND CANCEL** が生じます。**dump** パラメーターが false である場合、ダンプは取られません。
- forceAbend()。ABEND コードもダンプもない **ABEND CANCEL** が生じます。

## APPC マップ式会話

APPC 非マップ式会話は、JCICS API からサポートされません。

APPC マップ式会話:

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| initiate() | AttachInitiator | ALLOCATE、CONNECT PROCESS |
| converse() | Conversation | CONVERSE |
| get*() メソッド | Conversation | EXTRACT ATTRIBUTES |
| get*() メソッド | Conversation | EXTRACT PROCESS |
| free() | Conversation | FREE |
| issueAbend() | Conversation | ISSUE ABEND |
| issueConfirmation() | Conversation | ISSUE CONFIRMATION |
| issueError() | Conversation | ISSUE ERROR |
| issuePrepare() | Conversation | ISSUE PREPARE |
| issueSignal() | Conversation | ISSUE SIGNAL |
| receive() | Conversation | RECEIVE |
| send() | Conversation | SEND |
| flush() | Conversation | WAIT CONVID |

## 基本マッピング・サポート (BMS)

基本マッピング・サポート (BMS) は、CICS プログラムと端末装置間のアプリケーション・プログラミング・インターフェースです。JCICS は、BMS アプリケーション・プログラミング・インターフェースの一部をサポートします。

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| sendControl() | TerminalPrincipalFacility | SEND CONTROL |
| sendText() | TerminalPrincipalFacility | SEND TEXT |
| | サポートされていない | SEND MAP、RECEIVE MAP |

## チャネルとコンテナーの例

コンテナー は、プログラム間で情報を渡すための、データの名前付きブロックです。コンテナーは、チャネル と呼ばれる集合にグループ化されます。この資料では、Java アプリケーションでチャネルとコンテナーを使用する方法について説明し、いくつかのサンプル・コードを示します。

チャネルとコンテナーの概要、および非 Java アプリケーションでのチャネル使用の手引きについては、「アプリケーションの開発」の『チャネルを使用した拡張プログラム間データ転送』を参照してください。Java プログラムが既存の CICS アプリ

ケーション・データにアクセスできるようにするツールについては、 40 ページの『Java からの構造化データとの対話』を参照してください。

表 3 では、チャネルとコンテナーに対する JCICS サポートを実装するクラスとメソッドをリストしています。

*表 3. チャネルとコンテナーに対する JCICS サポート*

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| containerIterator() | Channel | STARTBROWSE CONTAINER |
| createContainer() | Channel | |
| delete() | Channel | DELETE CHANNEL |
| deleteContainer() | Channel | DELETE CONTAINER CHANNEL |
| getContainer() | Channel | |
| getContainerCount() | Channel | QUERY CHANNEL |
| getName() | Channel | |
| delete() | Container (コンテナー) | DELETE CONTAINER CHANNEL |
| get()、getLength() | Container (コンテナー) | GET CONTAINER CHANNEL [NODATA] |
| getName() | Container (コンテナー) | |
| put() | Container (コンテナー) | PUT CONTAINER CHANNEL |
| getOwner() | ContainerIterator | |
| hasNext() | ContainerIterator | |
| next() | ContainerIterator | GETNEXT CONTAINER BROWSETOKEN |
| remove() | ContainerIterator | |
| link() | Program | LINK |
| setNextChannel() | TerminalPrincipalFacility | RETURN CHANNEL |
| issue() | StartRequest | START CHANNEL |
| createChannel() | タスク | |
| getCurrentChannel() | タスク | ASSIGN CHANNEL |
| containerIterator() | タスク | STARTBROWSE CONTAINER |

CICS 条件 CHANNELERR の場合、結果として ChannelErrorException がスローされます。 CONTAINERERR CICS 条件の結果は ContainerErrorException になります。 CCSIDERR CICS 条件の結果は CCSIDErrorException になります。

57 ページの『JCICS におけるチャネルとコンテナーの作成』
チャネルを作成するには、Task クラスの createChannel() メソッドを使用します。

57 ページの『コンテナーへのデータの書き込み』
Container オブジェクトにデータを書き込むには、Container.put() メソッドを使用します。

58 ページの『別のプログラムまたはタスクへのチャネルの受け渡し』
プログラム・リンクでチャネルを渡すには、Program クラスの link() メソッドを使用します。

58 ページの『現行チャネルの受け取り』
プログラムが現行チャネルを明示的に受け取る必要はありません。ただし、プログラムは現行チャネルを現行タスクから取得することができます。

58 ページの『コンテナーからのデータの取得』
コンテナー内のデータをバイト配列に読み取るには、Container.get() メソッドを使用します。

58 ページの『現行チャネルのブラウズ』
チャネルが渡される JCICS プログラムは、そのチャネルを明示的に受け取ることなく、すべての Container オブジェクトにアクセスできます。

59 ページの『チャネルとコンテナーの例』
この例は、PAYR という名前の COBOL サーバー・プログラムを呼び出す、Payroll と呼ばれる Java クラスの抜粋を示しています。Payroll クラスは、JCICS の com.ibm.cics.server.Channel クラスと com.ibm.cics.server.Container クラスを使用して、チャネルとそのコンテナーを処理します。

**関連概念**:

45 ページの『データを渡すための引数』
チャネルとコンテナーを使用するか、通信域 (COMMAREA) を使用して、プログラム間でデータを受け渡すことができます。

**JCICS におけるチャネルとコンテナーの作成:**

チャネルを作成するには、Task クラスの createChannel() メソッドを使用します。

以下に例を示します。

```
Task t=Task.getTask();
Channel custData = t.createChannel("Customer_Data");
```

createChannel メソッドに提供されるストリングは、Channel オブジェクトが CICS に認識されている名前です (この名前は、CICS 命名規則に準拠するために、16 文字までスペースで埋め込まれます)。

チャネルに新しいコンテナーを作成するには、Channel の createContainer() メソッドを使用します。以下に例を示します。

```
Container custRec = custData.createContainer("Customer_Record");
```

createContainer() メソッドに提供されるストリングは、Container オブジェクトが CICS に認識されている名前です (この名前は、CICS 命名規則に準拠するために、必要に応じて 16 文字までスペースで埋め込まれます)。同じ名前のコンテナーがこのチャネルに既に存在する場合、ContainerErrorException がスローされます。

**コンテナーへのデータの書き込み:**

Container オブジェクトにデータを書き込むには、Container.put() メソッドを使用します。

Container オブジェクトにデータを書き込むには、Container.put() メソッドを使用します。データはバイト配列またはストリングとしてコンテナーに追加できます。以下に例を示します。

```
String custNo = "00054321";
byte[] custRecIn = custNo.getBytes();
custRec.put(custRecIn);
```

または

```
custRec.put("00054321");
```

**別のプログラムまたはタスクへのチャネルの受け渡し:**

プログラム・リンクでチャネルを渡すには、Program クラスの link() メソッドを使用します。

```
programX.link(custData);
```

プログラム戻り呼び出しで次のチャネルを設定するには、TerminalPrincipalFacility クラスの setNextChannel() メソッドを使用します。

```
terminalPF.setNextChannel(custData);
```

START 要求でチャネルを渡すには、StartRequest クラスの issue メソッドを使用します。

```
startrequest.issue(custData);
```

**現行チャネルの受け取り:**

プログラムが現行チャネルを明示的に受け取る必要はありません。ただし、プログラムは現行チャネルを現行タスクから取得することができます。

プログラムが現行タスクから現行チャネルを取得する場合、タスクはコンテナーを名前で取り出すことができます。

```
Task t = Task.getTask();
Channel custData = t.getCurrentChannel();
if (custData != null) {
    Container custRec = custData.getContainer("Customer_Record");
} else {
    System.out.println("There is no Current Channel");
}
```

**コンテナーからのデータの取得:**

コンテナー内のデータをバイト配列に読み取るには、Container.get() メソッドを使用します。

```
byte[] custInfo = custRec.get();
```

**現行チャネルのブラウズ:**

チャネルが渡される JCICS プログラムは、そのチャネルを明示的に受け取ることなく、すべての Container オブジェクトにアクセスできます。

これを行うには、ContainerIterator オブジェクトを使用します (ContainerIterator クラスは java.util.Iterator インターフェースを実装します)。Task オブジェクトが現行タスクからインスタンス化される場合、その containerIterator() メソッドは、現行チャネルの Iterator を戻し、現行チャネルがない場合は null を戻します。以下に例を示します。

```
Task t = Task.getTask();
ContainerIterator ci = t.containerIterator();
while (ci.hasNext()) {
    Container custData = ci.next();
    // Process the container...
}
```

**チャネルとコンテナーの例:**

この例は、PAYR という名前の COBOL サーバー・プログラムを呼び出す、Payroll と呼ばれる Java クラスの抜粋を示しています。Payroll クラスは、JCICS の com.ibm.cics.server.Channel クラスと com.ibm.cics.server.Container クラスを使用して、チャネルとそのコンテナーを処理します。

```
import com.ibm.cics.server.*;
public class Payroll {
     ...
     Task t=Task.getTask();

     // create the payroll_2004 channel
     Channel payroll_2004 = t.createChannel("payroll-2004");

     // create the employee container
     Container employee = payroll_2004.createContainer("employee");

     // put the employee name into the container
     employee.putString("John Doe");

     // create the wage container
     Container wage = payroll_2004.createContainer("wage");

     // put the wage into the container
     wage.putString("2000");

     // Link to the PAYROLL program, passing the payroll_2004 channel
     Program p = new Program();
     p.setName("PAYR");
     p.link(payroll_2004);

     // Get the status container which has been returned
     Container status = payroll_2004.getContainer("status");

     // Get the status information
     byte[] payrollStatus = status.get();
     ...
}
```

*図 1. JCICS の com.ibm.cics.server.Channel および com.ibm.cics.server.Container クラスを使用して、COBOL サーバー・プログラムにチャネルを渡す Java クラス*

## 診断サービス

JCICS アプリケーション・プログラミング・インターフェースは、以下の CICS トレースおよびダンプ・コマンドをサポートします。

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| | サポートされていない | DUMP |
| enterTrace() | EnterRequest | ENTER |

## 文書サービス

このセクションでは、DOCUMENT アプリケーション・プログラミング・インターフェースにおけるコマンドの JCICS サポートについて説明します。

`Document` クラスは **`EXEC CICS DOCUMENT`** API にマップされます。

クラス `Document` のデフォルトの引数なしコンストラクターは、CICS 内に新しい文書を作成します。コンストラクター `Document(byte[] docToken)` は、以前に作成された既存の文書の文書トークンを受け入れます。例えば、別のプログラムが文書を作成して、その文書トークンを COMMAREA またはコンテナー内の Java アプリケーションに渡すことができます。

`DocumentLocation` クラスのコンストラクターは、EXEC CICS DOCUMENT API の AT および TO キーワードにマップされます。

SymbolList クラスの setter と getter は、**`EXEC CICS DOCUMENT`** API の SYMBOLLIST、LENGTH、DELIMITER、および UNESCAPE キーワードにマップされます。

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| create*() | Document | DOCUMENT CREATE |
| append*() | Document | DOCUMENT INSERT |
| insert*() | Document | DOCUMENT INSERT |
| addSymbol() | Document | DOCUMENT SET |
| setSymbolList() | Document | DOCUMENT SET |
| retrieve*() | Document | DOCUMENT RETRIEVE |
| get*() | Document | DOCUMENT |

## 環境サービス

CICS 環境サービスは、アプリケーション・プログラムに関連する CICS データ域、パラメーター、およびリソース属性にアクセスできるようにします。

JCICS サポートに相当する **`EXEC CICS`** コマンドとオプションは次のとおりです。

- ADDRESS
- ASSIGN
- INQUIRE SYSTEM
- INQUIRE TASK
- INQUIRE TERMINAL/NETNAME

61 ページの『ADDRESS』
**`ADDRESS`** API コマンド・オプションには次のサポートが提供されます。

61 ページの『ASSIGN』
**`ASSIGN`** API コマンド・オプションには次のサポートが提供されます。

62 ページの『INQUIRE SYSTEM』
**`INQUIRE SYSTEM`** SPI オプションに対するサポートが提供されます。

62 ページの『INQUIRE TASK』
**`INQUIRE TASK`** API コマンド・オプションには次のサポートが提供されます。

63 ページの『INQUIRE TERMINAL および INQUIRE NETNAME』
**INQUIRE TERMINAL** および **INQUIRE NETNAME** SPI オプションには次のサポートが提供されます。

**ADDRESS:**

**ADDRESS** API コマンド・オプションには次のサポートが提供されます。

**EXEC CICS ADDRESS** コマンドの詳細については、『Reference』->『Application development』の『ADDRESS』を参照してください。

**ACEE** アクセス制御環境エレメント (ACEE) は、CICS ユーザーがサインオンするときに外部セキュリティー・マネージャーによって作成されます。このオプションは JCICS ではサポートされません。

**COMMAREA**
COMMAREA には、コマンドで渡されるユーザー・データが入っています。COMMAREA ポインターは、**CommAreaHolder** 引数によって、リンクされたプログラムに自動的に渡されます。詳しくは、45 ページの『データを渡すための引数』を参照してください。

**CWA** 共通作業域 (CWA) には、タスク間で共用可能なグローバル・ユーザー・データが入っています。CWA のコピーは、Region クラスの getCWA() メソッドを使用して取得できます。

**EIB** には、最後に実行された CICS コマンドに関する情報が入っています。EIB 値へのアクセスは、該当するオブジェクトのメソッドによって提供されます。その例を次に示します。

**eibtrnid**
Task クラスの getTransactionName() メソッドによって戻されます。

**eibaid** TerminalPrincipalFacility クラスの getAIDbyte() メソッドによって戻されます。

**eibcposn**
Cursor クラスの getRow() および getColumn() メソッドによって戻されます。

**TCTUA**
端末管理テーブル・ユーザー域 (TCTUA) には、CICS トランザクションを起動する端末 (基本機構) に関連したユーザー・データが入っています。この領域は、アプリケーション・プログラム間で情報を渡すのに使用されますが、関係するアプリケーション・プログラムに同じ端末が関連付けられている場合のみです。TCTUA の内容は、TerminalPrincipalFacility クラスの getTCTUA() メソッドを使用して取得できます。

**TWA** トランザクション作業域 (TWA) には、CICS タスクに関連したユーザー・データが入っています。この領域は、アプリケーション・プログラム間で情報を渡すのに使用されますが、同じタスク内にある場合のみです。TWA のコピーは、Task クラスの getTWA() メソッドを使用して取得できます。

**ASSIGN:**

**ASSIGN** API コマンド・オプションには次のサポートが提供されます。

このコマンドについて詳しくは、『Reference』->『Application development』の『ASSIGN』を参照してください。

| メソッド | JCICS クラス |
|---|---|
| getABCODE() | AbendException |
| getApplicationContext() | タスク |
| getAPPLID() | 領域 |
| getCurrentChannel() | タスク |
| getCWA() | 領域 |
| getName() | TerminalPrincipalFacility または ConversationPrincipalFacility |
| getFCI() | タスク |
| getNetName() | TerminalPrincipalFacility または ConversationPrincipalFacility |
| getPrinSysid() | TerminalPrincipalFacility または ConversationPrincipalFacility |
| getProgramName() | タスク |
| getQNAME() | タスク |
| getSTARTCODE() | タスク |
| getSysid() | 領域 |
| getTCTUA() | TerminalPrincipalFacility |
| getTERMCODE() | TerminalPrincipalFacility |
| getTWA() | タスク |
| getUSERID(), Task.getUSERID() | Task、TerminalPrincipalFacility または ConversationPrincipalFacility |

その他の ASSIGN オプションはサポートされません。

**INQUIRE SYSTEM:**

**INQUIRE SYSTEM** SPI オプションに対するサポートが提供されます。

| メソッド | JCICS クラス |
|---|---|
| getAPPLID() | 領域 |
| getSYSID() | 領域 |

その他の **INQUIRE SYSTEM** オプションはサポートされません。

**INQUIRE TASK:**

**INQUIRE TASK** API コマンド・オプションには次のサポートが提供されます。

| メソッド | JCICS クラス |
|---|---|
| getSTARTCODE() | タスク |
| getTransactionName() | タスク |
| getUSERID() | タスク |

**FACILITY**
タスクの基本機構で getName() メソッドを呼び出すことによって、タスクの基本機構の名前を見つけることができます。基本機構は、現行の Task オブジェクトで getPrincipalFacility() メソッドを呼び出すことによって見つけることができます。

**FACILITYTYPE**
Java instanceof 演算子を使用して、戻されたオブジェクト参照のクラスを確認することによって、機構のタイプを判別できます。

その他の INQUIRE TASK オプションはサポートされません。

**INQUIRE TERMINAL および INQUIRE NETNAME:**

**INQUIRE TERMINAL** および **INQUIRE NETNAME** SPI オプションには次のサポートが提供されます。

| メソッド | JCICS クラス |
|---|---|
| getUSERID() | Terminal、ConversationalPrincipalFacility |
| Terminal.getUser() | Terminal、ConversationalPrincipalFacility |

現行の Task オブジェクトで、またはタスクの基本機構を表すオブジェクトで getUSERID() メソッドを呼び出すことでも、USERID 値を見つけることができます。

その他の **INQUIRE TERMINAL** または **INQUIRE NETNAME** オプションはサポートされません。

## ファイル・サービス

JCICS は、CICS ファイルおよび索引のタイプごとに **EXEC CICS** API コマンドにマップされるクラスおよびメソッドを提供します。

Java プログラムが既存の CICS アプリケーション・データにアクセスできるようにするツールについては、「アプリケーションの開発」の『Java からの構造化データとの対話』を参照してください。

CICS は、次のタイプのファイルをサポートします。
- キー順データ・セット (KSDS)
- 入力順データ・セット (ESDS)
- 相対レコード・データ・セット (RRDS)

KSDS および ESDS ファイルには、代替 (または 2 次) 索引を備えることができます。CICS は、2 次索引から RRDS ファイルへのアクセスをサポートしません。2 次索引は、それ自体が別々の KSDS ファイルである、つまり別々の FD 項目を持つ場合と同じように CICS によって扱われます。

KSDS、ESDS (1 次索引)、および ESDS (2 次索引) ファイルのアクセスにはいくつかの相違点があります。すなわち、常に共通インターフェースを使用できるとは限りません。

レコードは、あらゆるタイプのファイルで読み取り、更新、削除、およびブラウズを行うことができます。ただし、ESDS ファイルからはレコードを削除できません。

データ・セットについて詳しくは、VSAM data sets: KSDS, ESDS, RRDS を参照してください。

データを読み取る Java コマンドは、**EXEC CICS** コマンドの SET オプションに相当するもののみをサポートします。戻されるデータは、CICS ストレージから Java オブジェクトに自動的にコピーされます。

ファイル制御に関連する Java インターフェースには、5 つのカテゴリーがあります。

**File** 他のファイル・クラスのスーパークラス。すべてのファイル・クラスに共通のメソッドが含まれます。

**KeyedFile**
1 次索引を使用してアクセスされる KSDS ファイル、2 次索引を使用してアクセスされる KSDS ファイル、および 2 次索引を使用してアクセスされる ESDS ファイルに共通のインターフェースが含まれます。

**KSDS** KSDS ファイルに固有のインターフェースが含まれます。

**ESDS** 相対バイト・アドレス (RBA、1 次索引) または拡張相対バイト・アドレス (XRBA) からアクセスされる ESDS ファイルに固有のインターフェースが含まれます。RBA ではなく、XRBA を使用するには、setXRBA(true) メソッドを発行します。

**RRDS** 相対レコード番号 (RRN、1 次索引) からアクセスされる RRDS ファイルに固有のインターフェースが含まれます。

ファイルごとに、作動可能な 2 つのオブジェクト、つまり、File オブジェクトと FileBrowse オブジェクトがあります。 File オブジェクトはファイル自体を表し、次の API オペレーションを実行するメソッドで使用できます。

- DELETE
- READ
- REWRITE
- UNLOCK
- WRITE
- STARTBR

File オブジェクトは、必要なファイル・クラスを明示的に開始するユーザー・アプリケーションによって作成されます。FileBrowse オブジェクトは、ファイル上のブラウズ・オペレーションを表します。特定のファイルに対して常に複数のアクティブ・ブラウズが可能であり、各ブラウズは REQID で区別されます。メソッドは、次の API オペレーションを実行するために FileBrowse オブジェクトに対してインスタンス化することができます。

- ENDBR
- READNEXT
- READPREV

- RESETBR

FileBrowse オブジェクトは、ユーザー・アプリケーションによって明示的にインスタンス化されません。STARTBR オペレーションを実行するメソッドによって作成され、ユーザー・クラスに戻されます。

以下の表では、JCICS クラスとメソッドが、CICS ファイルおよび索引のタイプごとに **EXEC CICS** API コマンドにどのようにマップされるかを示しています。これらの表では、JCICS クラスとメソッドは `class.method()` の形式で示されます。例えば、KeyedFile.read() は、KeyedFile クラスの read() メソッドを参照します。

最初の表では、キー付きファイルのクラスとメソッドを示しています。

*表 4. キー付きファイルのクラスとメソッド*

| **KSDS 1 次または 2 次索引のクラスとメソッド** | **ESDS 2 次索引のクラスとメソッド** | **CICS File API コマンド** |
|---|---|---|
| KeyedFile.read() | KeyedFile.read() | **READ** |
| KeyedFile.readForUpdate() | KeyedFile.readForUpdate() | **READ UPDATE** |
| KeyedFile.readGeneric() | KeyedFile.readGeneric() | **READ GENERIC** |
| KeyedFile.rewrite() | KeyedFile.rewrite() | **REWRITE** |
| KSDS.write() | KSDS.write() | **WRITE** |
| KSDS.delete() | | **DELETE** |
| KSDS.deleteGeneric() | | **DELETE GENERIC** |
| KeyedFile.unlock() | KeyedFile.unlock() | **UNLOCK** |
| KeyedFile.startBrowse() | KeyedFile.startBrowse() | **START BROWSE** |
| KeyedFile.startGenericBrowse() | KeyedFile.startGenericBrowse() | **START BROWSE GENERIC** |
| KeyedFileBrowse.next() | KeyedFileBrowse.next() | **READNEXT** |
| KeyedFileBrowse.previous() | KeyedFileBrowse.previous() | **READPREV** |
| KeyedFileBrowse.reset() | KeyedFileBrowse.reset() | **RESET BROWSE** |
| FileBrowse.end() | FileBrowse.end() | **END BROWSE** |

次の表は、キー付きでないファイルのクラスとメソッドを示しています。ESDS と RRDS は 1 次索引によってアクセスされます。

| **ESDS 1 次索引のクラスとメソッド** | **RRDS 1 次索引のクラスとメソッド** | **CICS File API コマンド** |
|---|---|---|
| ESDS.read() | RRDS.read() | **READ** |
| ESDS.readForUpdate() | RRDS.readForUpdate() | **READ UPDATE** |
| ESDS.rewrite() | RRDS.rewrite() | **REWRITE** |
| ESDS.write() | RRDS.write() | **WRITE** |
| | RRDS.delete() | **DELETE** |
| KeyedFile.unlock() | RRDS.unlock() | **UNLOCK** |
| ESDS.startBrowse() | RRDS.startBrowse() | **START BROWSE** |
| ESDS_Browse.next() | RRDS_Browse.next() | **READNEXT** |
| ESDS_Browse.previous() | RRDS_Browse.previous() | **READPREV** |

| ESDS 1 次索引のクラスとメソッド | RRDS 1 次索引のクラスとメソッド | CICS File API コマンド |
|---|---|---|
| ESDS_Browse.reset() | RRDS_Browse.reset() | **RESET BROWSE** |
| FileBrowse.end() | FileBrowse.end() | **END BROWSE** |
| ESDS.setXRBA() | | |

ファイルに書き込まれるデータは、Java バイト配列でなければなりません。

データはファイルから RecordHolder オブジェクトに読み取られます。ストレージは CICS によって提供され、プログラムの終わりに自動的に解放されます。

File メソッドには **KEYLENGTH** 値を指定する必要はありません。使用される長さは、渡されるキーの実際の長さです。FileBrowse オブジェクトが作成されると、startBrowse メソッドで指定されたキーの長さが入っています。この長さは、そのオブジェクトに対する以降のブラウズ要求で CICS に渡されます。

ブラウズ・オペレーションに **REQID** を指定する必要はありません。各ブラウズ・オブジェクトには、固有の REQID が含まれ、そのブラウズ・オブジェクトに対する以降のすべてのブラウズ要求に自動的に使用されます。

## HTTP サービスおよび TCP/IP サービス

`HttpHeader`、`NameValueData`、および `FormField` クラスの getter は、HTTP ヘッダー、名前と値のペア、および該当する API コマンドのフォーム・フィールド値を戻します。

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| get*() | CertificateInfo | EXTRACT CERTIFICATE / EXTRACT TCPIP |
| get*() | HttpRequest | EXTRACT WEB |
| getHeader() | HttpRequest | WEB READ HTTPHEADER |
| getFormField() | HttpRequest | WEB READ FORMFIELD |
| getContent() | HttpRequest | WEB RECEIVE |
| getQueryParm() | HttpRequest | WEB READ QUERYPARM |
| startBrowseHeader() | HttpRequest | WEB STARTBROWSE HTTPHEADER |
| getNextHeader() | HttpRequest | WEB READNEXT HTTPHEADER |
| endBrowseHeader() | HttpRequest | WEB ENDBROWSE HTTPHEADER |
| startBrowseFormField() | HttpRequest | WEB STARTBROWSE FORMFIELD |
| getNextFormField() | HttpRequest | WEB READNEXT FORMFIELD |
| endBrowseFormField() | HttpRequest | WEB ENDBROWSE FORMFIELD |
| startBrowseQueryParm() | HttpRequest | WEB STARTBROWSE QUERYPARM |
| getNextQueryParm() | HttpRequest | WEB READNEXT QUERYPARM |
| endBrowseQueryParm() | HttpRequest | WEB ENDBROWSE QUERYPARM |
| writeHeader() | HttpResponse | WEB WRITE |
| getDocument() | HttpResponse | WEB RETRIEVE |
| getCurrentDocument() | HttpResponse | WEB RETRIEVE |

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| sendDocument() | HttpResponse | WEB SEND |

**注:** HttpRequest オブジェクトを取得するには、メソッド get HttpRequestInstance() を使用してください。

CICS Web サポートによって処理される各着信 HTTP 要求には、HTTP ヘッダーが含まれています。要求で POST HTTP verb を使用する場合、文書データも含まれます。CICS Web サポートによって生成される各応答 HTTP 要求には、HTTP ヘッダーと文書データが含まれています。

これを処理するために、JCICS は次の Web および TCP/IP サービスを提供します。

**HTTP ヘッダー**
HttpRequest クラスを使用して HTTP ヘッダーを調べることができます。GET モードの HTTP では、クライアントが HTTP フォームに入力し、送信ボタンを選択した場合、照会ストリングが送信されます。

**SSL** CICS Web サポートは、TcpipRequest クラスを提供します。これは、要求を送信したクライアントに関する詳細情報と、SSL サポートに関する基本情報を取得するために、HttpRequest によって拡張されます。SSL 証明書が提供される場合、CertificateInfo クラスを使用して詳細に調べることができます。

**文書** 文書がサーバーに公開される (HTTP POST) 場合、CICS 文書として提供されます。HttpRequest クラスの getDocument() メソッドを呼び出すことによって、この文書にアクセスできます。既存文書の処理の詳細については、60 ページの『文書サービス』を参照してください。

要求から生じる HTTP クライアント Web コンテンツを提供するために、サーバー・プログラマーは、Document Services API を使用して CICS 文書を作成し、sendDocument() メソッドを呼び出す必要があります。

CICS Web サポートについて詳しくは、『Product overview』の『Internet, TCP/IP, and HTTP concepts』を参照してください。 JCICS Web クラスについて詳しくは、「*JCICS Class Reference*」を参照してください。

## プログラム・サービス

JCICS は、CICS プログラム制御コマンド LINK、RETURN、および INVOKE APPLICATION をサポートします。

Java プログラムが既存の CICS アプリケーション・データにアクセスできるようにするツールについては、「アプリケーションの開発」の『Java からの構造化データとの対話』を参照してください。

68 ページの表 5 には、CICS プログラム制御コマンドにマップされるメソッドと JCICS クラスがリストされています。

*表 5. メソッド、JCICS クラス、および CICS コマンド間の関係 :*

| EXEC CICS コマンド | JCICS クラス | JCICS メソッド |
|---|---|---|
| LINK | Program | link() |
| RETURN | TerminalPrincipalFacility | setNextTransaction()<br>、setNextCOMMAREA()<br>、setNextChannel() |
| INVOKE APPLICATION | Application | invoke() |

**LINK**

link() メソッドを使用して、CICS に対して定義される別のプログラムに制御を移動することができます。ターゲット・プログラムは、CICS でサポートされる任意の言語にすることができます。

**RETURN**

このコマンドの疑似会話型の側面のみがサポートされます。return に対する CICS 呼び出しを行う必要はありません。アプリケーションは通常どおり終了できます。疑似会話型機能は、TerminalPrincipalFacility クラスのメソッドでサポートされます。setNextTransaction() は、RETURN の TRANSID オプションの使用に相当します。setNextCOMMAREA() は、COMMAREA オプションの使用に相当します。一方、setNextChannel() は、CHANNEL オプションの使用に相当します。これらのメソッドは、プログラムの実行中にいつでも呼び出すことができ、プログラムが終了するときに有効になります。

**INVOKE**

いずれか 1 つのプログラム・エントリー・ポイントに対応する操作を指定することで、アプリケーションを呼び出すことができます。アプリケーション・エントリー・ポイント・プログラムの名前を知っておく必要はなく、プログラムがパブリック/プライベートのどちらであるかも無関係です。

**注:** 指定される COMMAREA の長さは、CICS の LENGTH 値として使用されます。 COMMAREA が任意の 2 つの CICS サーバー (製品/バージョン/リリースの任意の組み合わせ) 間で渡される場合、この値は 24 KB を超えてはなりません。この制限により、COMMAREA およびスペースがヘッダー用に許可されます。

## スケジューリング・サービス

JCICS は、CICS スケジューリング・サービスをサポートします。これにより、タスク用に保管されたデータを取り出し、インターバル制御要求を取り消し、指定された時間にタスクを開始することができます。

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| cancel() | StartRequest | CANCEL |
| retrieve() | タスク | RETRIEVE |
| issue() | StartRequest | START |

Task.retrieve() メソッドによって取り出される内容を定義するには、java.util.BitSet オブジェクトを使用します。com.ibm.cics.server.RetrieveBits クラスは、BitSet オブジェクトで設定できる次のビットを定義します。

- RetrieveBits.DATA

- RetrieveBits.RTRANSID
- RetrieveBits.RTERMID
- RetrieveBits.QUEUE

これらは、**EXEC CICS** RETRIEVE コマンドのオプションに対応します。

Task.retrieve() メソッドは、RetrieveBits の設定に応じて、単一の呼び出しで最大 4 つの情報を取り出します。DATA、RTRANSID、RTERMID および QUEUE データは、RetrievedData オブジェクトに置かれ、このオブジェクトは RetrievedDataHolder オブジェクトに保持されます。次の例では、データと transid を取り出します。

```
BitSet bs = new BitSet();
bs.set(RetrieveBits.DATA, true);
bs.set(RetrieveBits.RTRANSID, true);
RetrievedDataHolder rdh = new RetrievedDataHolder();
t.retrieve(bs, rdh);
byte[] inData = rdh.value.data;
String transid = rdh.value.transId;
```

## 直列化サービス

JCICS は、タスクによるリソースの使用をスケジュールできる CICS 直列化サービスをサポートします。

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| dequeue() | SynchronizationResource | DEQ |
| enqueue()、tryEnqueue() | SynchronizationResource | ENQ |

**関連資料**:

46 ページの『直列化可能クラス』
JCICS 直列化可能クラスのリスト。

## ストレージ・サービス

CICS サービスを使用した明示的なストレージ管理 (**EXEC CICS** GETMAIN など) はサポートされません。標準の Java ストレージ管理機能で、タスク専用ストレージのニーズを十分に満たすことができます。

タスク間のデータの共用は、CICS リソースを使用して行われなければなりません。

名前は一般的に、Java ストリングまたはバイト配列として表されます。これらが必要な長さであることを確認する必要があります。

## スレッドとタスクのサンプル

CICSExecutorService を使用して、CICS 内でタスクを開始するスレッドを作成できます。このサービスを使用してスレッドを作成すると、JCICS API を使用して CICS サービスにアクセスできる CICS タスクが作成されます。

CICS タスクを開始するスレッドの作成に関するオプションが 2 つあります。コードを移植可能にする場合は、OSGi フレームワーク内で ExecutorService を使用できます。 JVM サーバー内でアプリケーションを実行している場合は、スレッドの作成時に OSGi フレームワークは自動的に CICS 実装を使用して CICS タスクを開始します。 CICS 専用のアプリケーションを作成している場合は、JCICS を使用して直接 CICSExecutorService クラスを使用できます。

次の例では、スレッドを開始して一時ストレージ・キューを作成し、そのキューにデータを書き込む Java クラスを抜粋して示しています。

```
public class ExecutorTest2
{
    public static void main(String[] args)
    {
        // Inline the new Runnable class
        class CICSJob2 implements CICSTransactionRunnable
        {
            public void run()
            {
                // Create a temporary storage queue
                TSQ test_tsq = new TSQ();
                test_tsq.setType(TSQType.MAIN);

                // Set the TSQ name
                test_tsq.setName("TSQWRITE");

                // Write to the temporary storage queue
                // Use the CICS region local CCSID so it is readable
                String test_string = "Hello from a non CICS Thread - "+ threadId;
                try
                {
                    test_tsq.writeItem(test_string.getBytes(System.getProperty("com.ibm.cics.jvmserver.local.ccsid")));
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }

            @Override
            public String getTranid()
            {
          return "IJSA";
        }
    }

      // Create and run the new CICSJob Runnable
      Runnable task = new CICSJob2();
      CICSExecutorService.runAsCICS(task);
    }
}
```

**関連概念**:

47 ページの『スレッド』
JVM サーバー環境では、OSGi フレームワークで実行されているアプリケーションは ExecutorService を使用して、CICS タスクで非同期的に実行するスレッドを作成できます。

## 一時記憶域キュー・サービス

JCICS は、CICS 一時記憶コマンド DELETEQ TS、READQ TS、および WRITEQ TS をサポートします。

## JCICS メソッドと EXEC CICS コマンド間の対話

Java プログラムが既存の CICS アプリケーション・データにアクセスできるようにするツールについては、「アプリケーションの開発」の『Java からの構造化データとの対話』を参照してください。

71 ページの表 6 は、CICS 一時記憶コマンドにマップされるメソッドと JCICS クラスをリストしています。

*表 6. メソッド、JCICS クラスおよび CICS コマンド間の関係*

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| delete() | TSQ | DELETEQ TS |
| readItem()、readNextItem() | TSQ | READQ TS |
| writeItem()、rewriteItem()<br>writeItemConditional()<br>rewriteItemConditional() | TSQ | WRITEQ TS |

**DELETEQ TS**
TSQ クラスの delete() メソッドを使用して一時記憶域キュー (TSQ) を削除することができます。

**READQ TS**
CICS INTO オプションは Java プログラムではサポートされません。TSQ クラスの readItem() および readNextItem() メソッドを使用して、TSQ から特定の項目を読み取ることができます。これらのメソッドは、引数の 1 つとして ItemHolder オブジェクトを取ります。これには、バイト配列で読み取られるデータが含まれます。このバイト配列のストレージは、CICS によって作成され、プログラムの終わりにガーベッジ・コレクションされます。

**WRITEQ TS**
Java バイト配列で一時記憶域キューに書き込まれるデータを提供する必要があります。NOSPACE 条件が検出される場合、writeItem() および rewriteItem() メソッドは中断し、データをキューに書き込むためのスペースが使用可能になるまで待機します。writeItemConditional() および rewriteItemConditional() メソッドは、NOSPACE 条件の場合に中断しませんが、この条件を NoSpaceException としてアプリケーションに即時に戻します。

## 端末サービス

JCICS は、以下の CICS 端末サービス・コマンドをサポートします。

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| converse() | TerminalPrincipalFacility | CONVERSE |
| | サポートされていない | HANDLE AID |
| receive() | TerminalPrincipalFacility | RECEIVE |
| send() | TerminalPrincipalFacility | SEND |
| | サポートされていない | WAIT TERMINAL |

タスクに基本機能として端末が割り振られている場合、CICS は、標準出力と標準エラー・ストリームとして使用できる 2 つの Java PrintWriter コンポーネントを自動的に作成します。これらのコンポーネントはタスク端末にマップされます。 `out` および `err` という名前の 2 つのストリームは Task オブジェクトのパブリック・ファイルであり、System.out や System.err と同様に使用できます。

端末に送られるデータは、Java バイト配列で提供されなければなりません。データは、端末から DataHolder オブジェクトに読み取られます。 CICS では返されるデータ用のストレージが提供されます。これは、プログラムの終了時に割り振り解除されます。

## データと XML の間の変換

JCICS は、データから XML への変換とその逆の変換を実行するための API コマンドをサポートしています。それらのコマンドは、**EXEC CICS TRANSFORM DATATOXML** コマンドおよび **TRANSFORM XMLTODATA** コマンドと同等の機能を提供します。

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| SetName | XmlTransform | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |
| dataToXML | Transform | TRANSFORM DATATOXML |
| xmltoData | Transform | TRANSFORM XMLTODATA |
| setChannel | TransformInput | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |
| setDataContainer | TransformInput | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |
| setElementName | TransformInput | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |
| setElementNamespace | TransformInput | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |
| setNsContainer | TransformInput | TRANSFORM XMLTODATA |
| setTypeName | TransformInput | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |
| setTypeNamespace | TransformInput | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |
| setXmlContainer | TransformInput | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |
| setXmltransform | TransformInput | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |
| getElementName | TransformOutput | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |
| getElementNamespace | TransformOutput | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |
| getTypeName | TransformOutput | TRANSFORM DATATOXML<br>TRANSFORM XMLTODATA |

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| getTypeNamespace | TransformOutput | TRANSFORM DATATOXML<br><br>TRANSFORM XMLTODATA |

## 一時データ・キュー・サービス

JCICS は、CICS 一時データ・コマンド DELETEQ TD、READQ TD、および WRITEQ TD をサポートします。INTO オプションを除くすべてのオプションがサポートされます。

### JCICS メソッドと EXEC CICS コマンド間の対話

Java プログラムが既存の CICS アプリケーション・データにアクセスできるようにするツールについては、「アプリケーションの開発」の『Java からの構造化データとの対話』を参照してください。

表 7 は、CICS 一時データ・コマンドにマップされるメソッドと JCICS クラスをリストしています。

*表 7. メソッド、JCICS クラスおよび CICS コマンド間の関係*

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| delete() | TDQ | DELETEQ TD |
| readData()、readDataConditional() | TDQ | READQ TD |
| writeData() | TDQ | WRITEQ TD |

**DELETEQ TD**

TDQ クラスの delete() メソッドを使用して一時データ・キュー (TDQ) を削除することができます。

**READQ TD**

CICS INTO オプションは Java プログラムではサポートされません。TDQ クラスの readData() または readDataConditional() メソッドを使用して TDQ から読み取ることができます。これらのメソッドは、バイト配列で読み取られるデータが入っている DataHolder オブジェクトのインスタンスをパラメーターとして取ります。このバイト配列のストレージは、CICS によって作成され、プログラムの終わりにガーベッジ・コレクションされます。

readDataConditional() メソッドは、CICS NOSUSPEND ロジックを駆動します。QBUSY 条件が検出されると、QueueBusyException として即時にアプリケーションに戻されます。

readData() メソッドは、別のタスクによって使用中のレコードにアクセスしようとするときに、コミットされたレコードがそれ以上ない場合は中断します。

**WRITEQ TD**

Java バイト配列で TDQ に書き込まれるデータを提供する必要があります。

## 作業単位 (UOW) サービス

JCICS は、CICS SYNCPOINT サービスをサポートします。

*表 8. UOW サービスの JCICS と EXEC CICS コマンド間の関係*

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| commit()、rollback() | タスク | SYNCPOINT |

Liberty JVM サーバーでは、UOW 同期点処理を Java Transaction API (JTA) を使用して制御できます。詳しくは、 109 ページの『Java Transaction API (JTA)』を参照してください。

## Web サービスの例

JCICS は、アプリケーション内で Web サービスを操作するために使用できるすべての API コマンドをサポートします。

| メソッド | JCICS クラス | EXEC CICS コマンド |
|---|---|---|
| invoke() | WebService | INVOKE WEBSERVICE |
| create() | SoapFault | SOAPFAULT CREATE |
| addFaultString() | SoapFault | SOAPFAULT ADD FAULTSTRING |
| addSubCode() | SoapFault | SOAPFAULT ADD SUBCODESTR |
| delete() | SoapFault | SOAPFAULT DELETE |
| create() | WSAEpr | WSAEPR CREATE |
| delete() | WSAContext | WSACONTEXT DELETE |
| set*() | WSAContext | WSACONTEXT BUILD |
| get*() | WSAContext | WSACONTEXT GET |

次の例に、JCICS を使用して Web サービス要求を作成する方法を示します。

```
Channel requesterChannel = Task.getTask().createChannel("TestRequester");
                    Container appData = requesterChannel.createContainer("DFHWS-DATA");
                    byte[] exampleData = "ExampleData".getBytes();
                    appData.put(exampleData);

                    WebService requester = new WebService();
                    requester.setName("MyWebservice");
                    requester.invoke(requesterChannel, "myOperationName");

                    byte[] response = appData.get();
```

Web サービス要求の中で送受信されるアプリケーション・データを処理する際に、JZOS などのツールを使用して、構造化データを処理するためのクラスを自動的に生成できます。詳しくは、 40 ページの『Java からの構造化データとの対話』を参照してください。また、Java を使用して XML の生成とコンシュームを直接行うこともできます。

# JCICS の使用

通常の Java クラスのように、JCICS ライブラリーからクラスを使用します。アプリケーションは必要なタイプの参照を宣言し、クラスの新しいインスタンスが new 演算子を使用して作成されます。

## このタスクについて

基礎の CICS リソースの名前を指定するために、setName メソッドを使用して CICS リソースに名前を付けます。リソースを作成した後、標準の Java 構成体を使用してオブジェクトを操作することができます。宣言されたオブジェクトのメソッドを通常の方法で呼び出すことができます。クラスごとにサポートされるメソッドの詳細は、提供される Javadoc で入手可能です。

CICS Java プログラムでファイナライザーを使用しないでください。ファイナライザーを推奨しない理由の解説については、IBM SDK for z/OS, Java Technology Edition バージョン 7 (『Troubleshooting and support』セクション)を参照してください。

System.exit() 呼び出しを発行することによって CICS Java プログラムを終了しないでください。 Java アプリケーションが CICS で実行される場合、Java ラッパーと呼ばれる別の Java プログラムを使用して public static void main() メソッドが呼び出されます。ラッパーを使用する場合、CICS は、Java アプリケーションの環境を初期化し、さらに重要なことに、アプリケーションの存続中に使用されるすべてのプロセスをクリーンアップします。 JVM を終了すると、クリーンな戻りコード 0 が返されたとしても、このクリーンアップ・プロセスを実行できずにデータの不整合が生じる可能性があります。アプリケーションが JVM サーバーで実行中に System.exit() を使用すると、その JVM サーバーが終了し、CICS が即時に静止します。

## 手順

1. main メソッドを書き込みます。 CICS は、PROGRAM リソースの JVMCLASS 属性で指定されるクラスで、main(CommAreaHolder) の署名を使用するメソッドに制御を渡そうとします。このメソッドが見つからない場合、CICS は、main(String[]) メソッドを呼び出そうとします。
2. JCICS を使用してオブジェクトを作成するために、以下の手順を実行します。
   a. 参照を宣言します。
      ```
      TSQ tsq;
      ```
   b. new 演算子を使用してオブジェクトを作成します。
      ```
      tsq = new TSQ()
      ```
   c. setName メソッドを使用して、オブジェクトに名前を指定します。
      ```
      tsq.setName("JCICSTSQ");
      ```
3. オブジェクトを使用して、CICS と対話します。

## 例

この例では、TSQ オブジェクトを作成し、作成したばかりの一時記憶域キュー・オブジェクトで delete メソッドを呼び出し、キューが空の場合はスローされた例外をキャッチする方法を示しています。

```
// Define a package name for the program
package unit_test;

// Import the JCICS package
import com.ibm.cics.server.*;

// Declare a class for a CICS application
public class JCICSTSQ
{

    // The main method is called when the application runs
    public static void main(CommAreaHolder cah)
    {

         try
         {
             // Create and name a Temporary Storage queue object
             TSQ tsq = new TSQ();
             tsq.setName("JCICSTSQ");

             // Delete the queue if it exists
             try
             {
                   tsq.delete();
             }
             catch(InvalidQueueIdException e)
             {
                  // Absorb QIDERR
                  System.out.println("QIDERR ignored!");
             }

             // Write an item to the queue
             String transaction = Task.getTask().getTransactionName();
             String message = "Transaction name is - " + transaction;
             tsq.writeItem(message.getBytes());

         }
         catch(Throwable t)
         {
             System.out.println("Unexpected Throwable: " + t.toString());
         }

         // Return from the application
         return;
    }
}
```

## Java の制約事項

Java アプリケーションを開発する際に、適用すべきさまざまな制約事項があります。制約事項に従わないと、アプリケーションが CICS で実行されているときに問題が発生する可能性があります。

CICS で使用される Java アプリケーションには、以下の制約事項が適用されます。

- System.exit() メソッド: このメソッドを Java アプリケーションで使用することはできません。使用すると、アプリケーションが異常終了し、JVM サーバーおよび CICS がシャットダウンします。セキュリティー・ポリシーを使用して System.exit() のサポートを無効にしてください。関連情報については、 248 ページの『Java セキュリティー・マネージャーの有効化』を参照してください。
- JCICS API 呼び出し: これらの呼び出しを OSGi バンドルのアクティベーター・クラスで使用することはできません。

**注:** OSGi バンドル・アクティベーターを実行する Java スレッドは、JCICS 対応になりません。CICSExecutorService.runAsCICS() API を使用することで、アクティベーターから新規 JCICS 対応スレッドを開始することができます。詳しくは、スレッドとタスクのサンプルで説明しています。

- バンドル・アクティベーターで使用される start メソッドおよび stop メソッド: これらのメソッドは、適切な時間内に戻す必要があります。

## JCA ローカル ECI サポート

JCA ローカル ECI リソース・アダプターを使用するよう構成されている Liberty JVM サーバーに、JCA ECI アプリケーションをデプロイすることができます。

アプリケーションの開発について詳しくは、111 ページの『JCA プログラミング・インターフェースを使用したアプリケーションの開発』を参照してください。既存の CICS Transaction Gateway アプリケーションの移植の詳細については、122 ページの『JCA ECI アプリケーションを Liberty JVM サーバーに移植する』を参照してください。

CICS TS JCA ローカル ECI リソース・アダプターに備わっている JCA ECI プログラミング・インターフェースは、クラス定義から生成される Javadoc で詳しく記述されています。 Javadoc は、JCA ローカル ECI Javadoc 情報から入手できます。

アプリケーション開発に必要なライブラリーと OSGi バンドルは、CICS Explorer SDK に備わっています。

## Java アプリケーションからのデータへのアクセス

DB2 および VSAM のデータのアクセスと更新を行うことができる Java アプリケーションを作成できます。または、他の言語のプログラムにリンクして、DB2、VSAM、および IMS にアクセスすることができます。

CICS のデータにアクセスするための Java アプリケーションを作成する際に、次のいずれかの手法を使用できます。CICS リカバリー・マネージャーがデータ保全性を維持します。

### リレーショナル・データへのアクセス

次のいずれかの方法を使用して、DB2 のリレーショナル・データにアクセスするための Java アプリケーションを作成できます。

- 構造化照会言語 (SQL) コマンドを使用してデータにアクセスするプログラムにリンクする **JCICS LINK** コマンド。
- 適切なドライバーが使用可能な場合は、Java Data Base Connectivity (JDBC) または Structured Query Language for Java (SQLJ) 呼び出しを使用して、データに直接アクセスします。DB2 に適切な JDBC ドライバーが使用可能です。 JDBC および SQLJ アプリケーション・プログラミング・インターフェースの使用について詳しくは、『アプリケーションの開発』の『Java プログラムから DB2 データにアクセスするための JDBC および SQLJ の使用』を参照してください。

- 基礎のアクセス機構として JDBC または SQLJ を使用する JavaBeans。このような JavaBeans を開発するには、適切な Java 統合開発環境 (IDE) を使用できます。

### DL/I データへのアクセス

IMS の DL/I データにアクセスするには、Java アプリケーションで **JCICS LINK** コマンドを使用して、EXEC DLI コマンドを発行してデータにアクセスする中間プログラムにリンクする必要があります。

### VSAM データへのアクセス

VSAM データにアクセスするには、Java アプリケーションで次のいずれかの方法を使用できます。

- VSAM に直接アクセスする場合は、JCICS ファイル制御クラス。
- CICS ファイル制御コマンドを出してデータにアクセスするプログラムにリンクする **JCICS LINK** コマンド。

## CICS 内の Java アプリケーションからの接続

CICS 環境内の Java プログラムは、TCP/IP ソケットをオープンし、外部プロセスと通信することができます。Java プログラムをゲートウェイとして使用すると、他の言語の CICS プログラムからは使用できない可能性がある他のエンタープライズ・アプリケーションに接続することができます。例えば、リモート・サーブレットまたはデータベースと通信する Java プログラムを作成できます。

この接続が CICS に統合されて、分散トランザクションや ID 伝搬などのエンタープライズ・サービス品質を提供する場合があります。また、CICS によって提供される分散トランザクションやその他のサービスなしに接続を使用できる場合もあります。必要な接続のタイプによっては、CICS で本来はサポートされないエンタープライズ・アプリケーションとの接続を可能にするサード・パーティー・ベンダー製品が使用できる場合があります。

一般に、CICS 環境における JVM の機能は、バッチ・モード JVM とほぼ同じです。バッチ・モード JVM は、CICS 環境の外部ではスタンドアロン・プロセスとして実行され、通常は、UNIX システム・サービスのコマンド行から、または JCL ジョブで開始されます。バッチ・モード JVM で作動可能な大部分のアプリケーションは、同じ範囲で CICS における JVM でも実行できます。例えば、サード・パーティーの JDBC ドライバーを使用して IBM 以外のデータベースと通信するバッチ・モード Java アプリケーションを作成する場合、同じアプリケーションがおそらく、CICS における JVM で作動します。ベンダー提供のコード (IBM 以外の JDBC ドライバーなど) を CICS における JVM で使用したい場合は、ベンダーに問い合わせて、そのコードが CICS における JVM で実行されることをサポートするかどうかを判別してください。

CICS における Java アプリケーションの動作について詳しくは、 33 ページの『CICS の Java ランタイム環境』を参照してください。

CICS 環境における JVM で実行されるバッチ・モード・アプリケーションは、通常、CICS の機能を利用しません。例えば、CICS の Java プログラムが、サード・

パーティーの JDBC ドライバーを使用して IBM 以外のデータベース内のレコードを更新する場合、CICS はこのアクティビティーを認識せず、現行の CICS トランザクションに更新を組み込もうとしません。

## Liberty フィーチャー

CICS は WebSphere Application Server Liberty プロファイルのフィーチャーをサポートすることで、JEE Web アプリケーションを Liberty JVM サーバーにデプロイできるようにします。

*表 9. CICS Liberty JVM サーバーでサポートされる Liberty フィーチャー*

| フィーチャー | 名前 | 説明 |
|---|---|---|
| アプリケーション・セキュリティー | appSecurity-1.0 | サーバー・ランタイム環境とアプリケーションを保護するためのサポートを提供します。このフィーチャーは、appSecurity-2.0 に置き換えられました。 |
| アプリケーション・セキュリティー | appSecurity-2.0 | サーバー・ランタイム環境とアプリケーションを保護するためのサポートを提供します。 |
| Bean 検証 | beanValidation-1.0 | JavaBeans を検証するための、アノテーション・ベースのモデルを提供します。 |
| Blueprint | blueprint-1.0 | OSGi blueprint container 仕様を使用した OSGi アプリケーションをデプロイするためのサポートを有効にします。 |
| Contexts and Dependency Injection | cdi-1.0 | EJB や Managed Bean などのコンポーネントを、JSP や EJB といった他のコンポーネントに注入するためのメカニズムを提供します。 |
| EJB Lite | ejbLite-3.1 | EJB 3.1 仕様の EJB Lite サブセットに記載されている Enterprise JavaBeans のサポートを有効にします。 |
| Java Architecture for XML Binding | jaxb-2.2 | Java クラスと XML 表現をマップするためのサポートを提供します。 |
| Java API for RESTful Web Services | jaxrs-1.1 | Liberty プロファイルでの Java API for RESTful Web Services (JAX-RS) のサポートを提供します。 |
| Java API for XML Web Services | jaxws-2.2 | SOAP Web サービスのサポートを提供します。 |

表 9. *CICS Liberty JVM* サーバーでサポートされる *Liberty* フィーチャー *(続き)*

| フィーチャー | 名前 | 説明 |
|---|---|---|
| Java EE Connector Architecture | jca-1.6 | アプリケーションからエンタープライズ情報システム (EIS) にアクセスするためのリソース・アダプターの構成を有効にします。 |
| Java Database Connectivity | jdbc-4.0 | タイプ 4 ドライバーを使用してリモート・データベースにアクセスするアプリケーションのサポートを提供します。 |
| Java Naming and Directory Interface | jndi-1.0 | Liberty プロファイルのサーバー構成における、単一の Java Naming and Directory Interface (JNDI) エントリー定義のサポートを提供します。 |
| Java Persistence API | jpa-2.0 | Java Persistence API 2.0 仕様に記載されているアプリケーション管理およびコンテナー管理の JPA を使用したアプリケーションのサポートを有効にします。 |
| JavaServer Faces | jsf-2.0 | JavaServer Faces (JSF) フレームワークを使用する Web アプリケーションのサポートを提供します。 |
| JavaScript Object Notation | json-1.0 | Java 環境用の一連の JSON ハンドリング・クラスを提供する、JavaScript Object Notation (JSON4J) ライブラリーを利用できるようにします。 |
| JavaServer Pages | jsp-2.2 | サーブレットおよび JavaServer Pages (JSP) アプリケーションのサポートを有効にします。 |
| JMS メッセージ駆動型 Bean | jmsMdb-3.1 | このフィーチャーは、JMS メッセージ駆動型 Enterprise JavaBeans の使用を可能にします。 MDB は、Java EE コンポーネント内でのメッセージの非同期処理を可能にします。 |

表 9. CICS Liberty JVM サーバーでサポートされる Liberty フィーチャー (続き)

| フィーチャー | 名前 | 説明 |
|---|---|---|
| LDAP ユーザー・レジストリー | ldapRegistry-3.0 | LDAP サーバーをユーザー・レジストリーとして使用するためのサポートが有効になります。LDAP バージョン 3.0 をサポートする任意のサーバーを使用できます。 複数の LDAP レジストリーを構成してから統合して、単一の論理レジストリー・ビューを実現できます。 |
| ローカル JMX コネクター | localConnector-1.0 | JVM にビルドされたローカル JMX コネクターを使用して、サーバー内の JMX リソースにアクセスできるようにします。 |
| 管理 Bean | managedBeans-1.0 | コンテナーによって管理される、さまざまな種類の Java EE コンポーネントのための共通基盤を提供します。Managed Bean で利用できる共通サービスには、リソース・インジェクション、ライフサイクル管理、インターセプターの使用などがあります。 |
| MongoDB Java Driver | mongodb-2.0 | MongoDB Java Driver のサポートを提供し、サーバー構成でリモート・データベース・インスタンスを構成できるようにします。アプリケーションは、MongoDB API を介してそれらのデータベースと対話します。 |
| モニター | monitor-1.0 | JMX クライアントを使用した WebSphere Liberty プロファイル・ランタイム・コンポーネントのパフォーマンス・モニターを可能にします。 |
| OSGi コンソール | osgiConsole-1.0 | ランタイムのデバッグを支援する OSGi コンソールを使用できるようにします。 |

表 9. *CICS Liberty JVM* サーバーでサポートされる *Liberty* フィーチャー *(続き)*

| フィーチャー | 名前 | 説明 |
|---|---|---|
| OSGi Java Persistence API | osgi.jpa-1.0 | このフィーチャーは、OSGi 機能が組み込まれた、blueprint-1.0 フィーチャーおよび jpa-2.0 フィーチャーに置き換えられました。これらのフィーチャーのいずれかをサーバーに追加した場合は、OSGi バンドルがサポートされるため、手動で osgi.jpa-1.0 フィーチャーを追加する必要はありません。 |
| Rest JMX コネクター | restConnector-1.0 | REST ベースのコネクターによる JMX クライアントのリモート・アクセスを可能にします。SSL およびユーザー・セキュリティー構成が必要です。 |
| Java サーブレット | servlet-3.0 | Java サーブレット 3.0 仕様に書き込まれる HTTP サーブレットのサポートを提供します。 |
| データベース・セッション・パーシスタンス | sessionDatabase-1.0 | JDBC を使用したデータ・ソースへの HTTP セッションのパーシスタンスを有効にします。 |
| Secure Sockets Layer | ssl-1.0 | Secure Sockets Layer (SSL) 接続および SAF 鍵リングのサポートを提供します。 |
| Web アプリケーション・バンドル | wab-1.0 | エンタープライズ・バンドル (EBA) に含まれる Web アプリケーション・バンドル (WAB) のサポートを提供します。 |
| WebSphere Embedded Messaging JMS クライアント | wasJmsClient-1.1 | アプリケーションが JMS 1.1 API を使用して、WebSphere Application Server Liberty プロファイルでホストされるメッセージ・キューにアクセスできるようにします。 |
| セキュア WebSphere Embedded Messaging ランタイム | wasJmsSecurity-1.0 | WebSphere Embedded Messaging サーバーがクライアントからのアクセスを認証および許可できるようにします。 |

*表 9. CICS Liberty JVM サーバーでサポートされる Liberty フィーチャー (続き)*

| フィーチャー | 名前 | 説明 |
|---|---|---|
| WebSphere Embedded Messaging サーバー | wasJmsServer-1.0 | サーバー内で組み込みメッセージング・サーバーを使用できるようにします。アプリケーションは、wasJmsClient フィーチャーを使用して、このエンジン内でメッセージを操作できます。 |
| DynaCache | webCache-1.0 と distributedMap-1.0 | このフィーチャーは、Web 応答のローカル・キャッシングを可能にします。 これには distributedMap フィーチャーが含まれ、応答時間とスループットを改善するために、Web アプリケーション応答の自動キャッシングを行います。 |
| Java EE Web Profile 6.0 | webProfile-6.0 | このフィーチャーは、Java EE 6 Web プロファイルをサポートするために必要な Liberty フィーチャーの便利な組み合わせを提供します。 |

これらのフィーチャーの機能について詳しくは、WebSphere Application Server Liberty プロファイルの資料 (Liberty プロファイルの概要) を参照してください。CICS Liberty JVM サーバーでこれらのフィーチャーを使用する際の制限については、84 ページの『Liberty フィーチャーの制約事項』を参照してください。

CICS は、CICS JVM サーバー環境に固有のカスタマイズ・フィーチャーのセットも提供します。これらは、WebSphere Application Server Liberty プロファイル・フィーチャーと CICS サービス品質の統合をサポートします。

*表 10. CICS フィーチャー*

| フィーチャー | 名前 | 説明 |
|---|---|---|
| CICS コア | cicsts:core-1.0 | CICS コア・フィーチャーおよび Java Transaction API (JTA) 1.0 を提供します。 |
| CICS JCA ローカル ECI | cicsts:jcaLocalEci-1.0 | CICS プログラムを呼ぶための、ローカル用に最適化された JCA ECI リソース・アダプターを提供します。 |
| CICS JDBC サポート | cicsts:jdbc-1.0 | アプリケーションが JDBC を使用してローカル CICS DB2 データベースにアクセスするためのサポートを提供します。 |
| CICS Distributed Identity | cicsts:distributedIdentity-1.0 | 配布 ID のマッピングのサポートを提供します。 |

*表 10. CICS フィーチャー (続き)*

| フィーチャー | 名前 | 説明 |
|---|---|---|
| CICS Liberty セキュリティー | cicsts:security-1.0 | スレッド ID の伝搬を含め、Liberty セキュリティーと CICS セキュリティーの統合を提供します。 |
| z/OS Connect for CICS | cicsts:zosConnect-1.0 | z/OS Connect を CICS Liberty JVM サーバーと統合します。 |

『Liberty フィーチャーの制約事項』
Liberty プロファイルの一部のフィーチャーには、CICS で使用する場合に既知の制約事項がいくつかあります。

## Liberty フィーチャーの制約事項

Liberty プロファイルの一部のフィーチャーには、CICS で使用する場合に既知の制約事項がいくつかあります。

*表 11. 制約事項*

| フィーチャー | 名前 | 制約事項 |
|---|---|---|
| Blueprint | blueprint-1.0 | トランザクション属性「NotSupported」は、Liberty JVM サーバーではサポートされません。 |
| EJB Lite | ejbLite-3.1 | トランザクション属性「NotSupported」は、Liberty JVM サーバーではサポートされません。 |
| Java EE Connector Architecture | jca-1.6 | JCA でのインバウンド通信はサポートされません。 |
| Java Persistence API | jpa-2.0 | Java Persistence API は、タイプ 4 のデータベース・ドライバーでは機能しますが、タイプ 2 では機能しません。 |
| Java サーブレット 3.0 | servlet-3.0 | 非同期のサーブレットおよびフィルターの使用は、Liberty JVM サーバーではサポートされません。 |
| データベース・セッション・パーシスタンス | sessionDatabase-1.0 | データベース・セッション・パーシスタンスは、タイプ 4 のデータベース・ドライバーでは機能しますが、タイプ 2 では機能しません。 |

WebSphere Liberty プロファイルの制約事項について詳しくは、ランタイム環境での既知の制約事項を参照してください。

# Java プログラムから DB2 データにアクセスするための JDBC および SQLJ の使用

CICS で実行される Java プログラムでは、DB2 データベースに保持されているデータにアクセスするためにいくつかの方法を使用できます。

Java プログラムでは、以下のことが可能です。

- JCICS LINK コマンドを使用することにより、構造化照会言語 (SQL) コマンドを使用してデータにアクセスする CICS プログラムにリンクする。
- Java Data Base Connectivity (JDBC) または Structured Query Language for Java (SQLJ) アプリケーション・プログラミング・インターフェースを使用して、データに直接アクセスする。

  86 ページの『CICS DB2 環境で JDBC および SQLJ を機能させる』
  CICS 用の Java アプリケーションが行う JDBC 要求および SQLJ 要求は、DB2® の提供する JDBC タイプ 2 ドライバーか、JDBC タイプ 4 ドライバーで処理することができます。

  86 ページの『DB2 をサポートするための JVM サーバーの構成』
  JVM サーバーは、Java アプリケーション用のランタイム環境です。JDBC および SQLJ ベースのアプリケーションをサポートするように JVM サーバーを構成できます。

  87 ページの『CICS DB2 環境での JDBC および SQLJ のプログラミング』
  CICS 用の Java プログラムでは、JDBC アプリケーション・プログラミング・インターフェース (API) のプログラミング規則を順守する必要があります。この規則は、一般的な CICS プログラミング・モデルよりも厳格な場合があります。

  88 ページの『データベースへの接続の取得』
  SQL ステートメントを実行する前に、JDBC または SQLJ アプリケーションで、データベースへの接続または接続コンテキストを取得する必要があります。アプリケーションは、2 つの Java クラスのうちのいずれかを使用してターゲット・データ・ソースに接続します。

  91 ページの『データベースへの接続を閉じる』
  データベースへの接続を閉じると、JDBC と SQLJ のリソースが自動的に解放されます。通常、データベースへの接続はタスクの終了時に閉じられます。

  92 ページの『作業単位のコミット』
  作業単位をコミットするために、JDBC および SQLJ アプリケーションで、JDBC および SQLJ のコミットとロールバック・メソッド呼び出しを実行することができます。 DB2 JDBC ドライバーはこれらの呼び出しを JCICS コミットまたは JCICS ロールバック呼び出しに変換します。その結果、CICS 同期点が取得されます。

  94 ページの『JDBC 要求または SQLJ 要求の間に CICS が異常終了する』
  DB2 提供の JDBC ドライバーによって作成される EXEC SQL 要求の処理中に発行される CICS 異常終了は、Java 例外に変換されないため、CICS 用の Java アプリケーションではキャッチされません。 CICS トランザクションは異常終了して、最終同期点までロールバックされます。

# CICS DB2 環境で JDBC および SQLJ を機能させる

CICS 用の Java アプリケーションが行う JDBC 要求および SQLJ 要求は、DB2® の提供する JDBC タイプ 2 ドライバーか、JDBC タイプ 4 ドライバーで処理することができます。

## このタスクについて

CICS 環境では、DB2 提供の JDBC タイプ 2 ドライバーは CICS DB2 言語インターフェース (スタブ) DSNCLI とリンク・エディットされます。ドライバーは、JDBC または SQLJ 要求を EXEC SQL 相当物に変換します。 DB2 提供の JDBC ドライバーからの変換済み要求は、他のプログラム (COBOL プログラムなど) からの EXEC SQL 要求とまったく同じ方法で CICS DB2 接続機能に流れます。そのため、CICS DB2 用の Java プログラムと CICS DB2 用の他のプログラムとの間に操作上の相違はなく、RDO を使用して選択可能なすべてのカスタマイズおよび調整のオプションは CICS DB2 用の Java プログラムに適用されます。

DB2 と共に配布される DB2 Universal JDBC ドライバーを使用します。 DB2 で提供される JDBC ドライバーを使用するには、適切なレベルの JDBC ドライバーをサポートする DB2 サブシステムに CICS を接続する必要があります。タイプ 2 ドライバーのみが CICS DB2 接続リソースを使用します。

Liberty JVM サーバーのユーザーは、CICS DB2 接続リソースを使用しない DB2 JDBC タイプ 4 ドライバーを使用できます。

JDBC および SQLJ アプリケーション・プログラミング・インターフェースを使用する Java アプリケーションのコーディングとビルドの方法について詳しくは、ご使用のバージョンの DB2 に適用される「*DB2 for z/OS: Programming for Java*」を参照してください。

# DB2 をサポートするための JVM サーバーの構成

JVM サーバーは、Java アプリケーション用のランタイム環境です。JDBC および SQLJ ベースのアプリケーションをサポートするように JVM サーバーを構成できます。

## 始める前に

DB2 と組み合わせて JVM サーバーを使用するには、IBM Data Server Driver for JDBC and SQLJ の最新バージョンが必要です。必要な APAR について詳しくは、CICS Transaction Server for z/OS 詳細システム要件を参照してください。 DB2 SDSNLOD2 ライブラリーを CICS STEPLIB 連結に追加する必要もあります。

## 手順

DB2 に付属の JDBC ドライバーをアプリケーションで使用できるようにします。

1. 正しい JDBC ドライバー情報を使用して JVM サーバーをセットアップします。 Liberty JVM サーバーについては、『Configuring』の『Web アプリケーションのための Liberty JVM サーバーの構成』を参照してください。OSGi JVM サーバーについては、『Configuring』の『Configuring a JVM server for OSGi applications』を参照してください。

2. CICS 環境で JDBC ドライバーを実行するときは、システム・プロパティーを変更する必要が生じることがあります。JVM サーバーの JVM プロファイルで、JDBC ドライバーに関連するシステム・プロパティーを設定できます。 JDBC ドライバーで使用されるシステム・プロパティー・ファイルを指定する DB2 環境変数 **DB2SQLJPROPERTIES** は使用されません。 CICS 環境ではほとんどの DB2 JDBC ドライバー・システム・プロパティーが使用されません。DB2 JDBC ドライバー・プロパティーのリストと、CICS 環境で使用されないか別の意味を持つプロパティーについては、「*DB2 for z/OS: Programming for Java*」を参照してください。

### 次のタスク

Java 2 セキュリティー・ポリシー・メカニズムがアクティブな OSGi JVM サーバーで Java アプリケーションから JDBC または SQLJ を使用する場合は、『デプロイ』の『Java セキュリティー・マネージャーの有効化』を参照してください。

## CICS DB2 環境での JDBC および SQLJ のプログラミング

CICS 用の Java プログラムでは、JDBC アプリケーション・プログラミング・インターフェース (API) のプログラミング規則を順守する必要があります。この規則は、一般的な CICS プログラミング・モデルよりも厳格な場合があります。

IBM Data Server Driver for JDBC and SQLJ では、JDBC 4.0 以前のバージョンがサポートされます。

**注:** タイプ 2 のドライバーを使用する SQLJ は Liberty JVM サーバーでサポートされますが、タイプ 4 ドライバーは引き続きサポートされません。

JDBC API の詳細については、JDBC Web サイトを参照してください。JDBC および SQLJ の API を使用する Java アプリケーションの開発方法については、「DB2 for Linux UNIX and Windows 10.1.0」の『アプリケーションの実行』を参照してください。

『JVM サーバーへのシリアライズされた SQLJ プロファイルのデプロイ』
SQLJ は、Java アプリケーションの組み込み静的 SQL を使用できるようにするものです。JDBC アプリケーション・プログラミング・インターフェースを使用する代わりの方法として、シリアライズされた SQLJ プロファイルを OSGi JVM サーバーまたは Liberty JVM サーバーにデプロイできます。

### JVM サーバーへのシリアライズされた SQLJ プロファイルのデプロイ

SQLJ は、Java アプリケーションの組み込み静的 SQL を使用できるようにするものです。JDBC アプリケーション・プログラミング・インターフェースを使用する代わりの方法として、シリアライズされた SQLJ プロファイルを OSGi JVM サーバーまたは Liberty JVM サーバーにデプロイできます。

### 始める前に

SQLJ プログラム用のプログラム準備の手順に従って、SQLJ プログラムの準備を行う必要があります。

DBRM を作成したら、それらを DB2 プランまたはパッケージと、CICS に定義された DB2 プランを指定した DB2 Entry にバインドする必要があります。これに失敗すると、SQL -805 エラーになるか、JDBC を使用するように SQLJ がデフォルト設定されます。バンドルを作成するワークステーションに、シリアライズされたプロファイルのファイルをダウンロードします。コード・ページ変換の必要がないように、転送はバイナリー形式で行ってください。これにより、後で問題が発生することを防止できます。

次に、シリアライズされたプロファイルを配置する場所を選択する必要があります。次の 2 つのシナリオがあります。

### 手順

1. シナリオ 1 - シリアライズされたプロファイルを、デプロイしたバンドル内に保持します。
   a. OSGI バンドルの場合、シリアライズされたプロファイルをバンドルのルート・ディレクトリーまたはバンドル・クラス・ディレクトリーに保持することができます。Liberty バンドルの場合、他のクラスと共に bin ディレクトリーに配置する必要があります。
2. シナリオ 2 - プロファイルを外部化します。
   a. シリアライズされたプロファイルを、jar の中から USS ファイル・システム内のディレクトリー (`/usr/lpp/cicsts/dev/sqlj.profile.dir` など) に移動します。ファイルは、`/usr/lpp/cicsts/dev/sqlj.profile.dir/com/ibm/cics/test/sqlj/CurrentTimeStamp_SJProfile0.ser` のように、同じパッケージ構造のディレクトリーに配置する必要があります。
   b. バンドル・マニフェストに項目を追加します。

      ```
      Bundle-ClassPath: .,external:$sqlj.profile.dir$
      ```

      「external」という文字列はバンドルの外部の場所を表し、**$** で囲まれた文字列は、Java システム・プロパティー `sqlj.profile.dir` の値に置き換えられます。
   c. JVM プロファイルに次の例のような Java システム・プロパティーを追加します。

      ```
      -Dsqlj.profile.dir=/usr/lpp/cicsts/dev/sqlj.profile.dir
      ```

### タスクの結果

これで、シリアライズされた SQLJ プロファイルが正常にデプロイされました。

## データベースへの接続の取得

SQL ステートメントを実行する前に、JDBC または SQLJ アプリケーションで、データベースへの接続または接続コンテキストを取得する必要があります。アプリケーションは、2 つの Java クラスのうちのいずれかを使用してターゲット・データ・ソースに接続します。

### このタスクについて

2 つの Java クラスとは、以下の Java クラスです。

- DriverManager: このクラスは、データベース URL で指定されたデータベースにアプリケーションを接続します。
- DataSource: このインターフェースは、基礎データベースに関する詳細がアプリケーションから認識されないようにします。これにより、アプリケーションの移植性が高まるため、DriverManager よりも好まれます。 DataSource の実装は、Java プログラムの外部で作成され、Java Naming and Directory Interface (JNDI) の DataSource 名ルックアップを使用して取得されます。 DataSource インターフェースは Liberty JVM サーバーでのみ使用可能です。

OSGi JVM サーバー環境では、DriverManager しか使用できず、また、タイプ 2 のドライバーを使用してローカルの DB2 を使用する必要があります。

Liberty JVM サーバー環境では、DriverManager と DataSource のどちらのインターフェースも使用可能です。このため、cicsts:jdbc-1.0 の機能を使用してローカルの DB2 を使用することも、jdbc-4.0 の機能を使用してタイプ 4 のドライバーでリモート・データ・ソースを使用することもできます。jdbc-4.0 の機能を使用する場合は、JTA を使用して、データベースへの更新と CICS で行われた更新を調整する必要があります。

ローカル DB2 を使用する CICS DB2 環境では、既存の CICS DB2 セキュリティー・プロシージャーが代わりに使用されるため、ユーザー ID とパスワードを指定する必要はありません。

IBM Data Server Driver for JDBC and SQLJ は、JDBC 4 以前のサポートを提供します。

CICS Explorer SDK には、OSGi と Liberty JVM サーバーの両方の JDBC サンプルが用意されています。詳しくは、『アプリケーションの開発』の『サーブレット・サンプルの概要』を参照してください。

JDBC の DriverManager および DataSource インターフェースを使用して接続を獲得する方法、およびアプリケーションで使用できるサンプル・コードについて詳しくは、使用する DB2 バージョンに適した「DB2 for z/OS」の『Java 用プログラミング』を参照してください。

『データベースへの DriverManager 接続の取得』
DriverManager クラスを使用する場合、データベースへの接続は、JDBC ドライバーに渡されるデータベース URL (Uniform Resource Locator) によって識別されます。

90 ページの『データベースへのデータ・ソース接続の取得』
DataSource インターフェースを使用する場合、データベースへの接続は、Java Naming and Directory Interface (JNDI) の DataSource 名ルックアップを使用して取得されます。 JDBC DataSource 実装は、CICS JDBC によって提供されます。

91 ページの『接続可能数』
CICS 用の Java アプリケーションでは、一度に最大 1 つの JDBC 接続または SQLJ 接続のコンテキストを開くことができます。

## データベースへの DriverManager 接続の取得

DriverManager クラスを使用する場合、データベースへの接続は、JDBC ドライバーに渡されるデータベース URL (Uniform Resource Locator) によって識別されます。

### このタスクについて

OSGi JVM サーバーで DriverManager 接続を構成するには、JVM プロファイルの OSGI_BUNDLES および LIBPATH_SUFFIX に DB2 JDBC jar およびネイティブ DLL を指定します。

Liberty JVM サーバーでは、DriverManager 構成は CICS JDBC Liberty 機能によって提供されます。

JDBC ドライバーは次の 2 つのタイプの URL を認識します。

**デフォルト URL**

デフォルト URL には、DB2 サブシステムのロケーション名は含まれません。 DB2 for z/OS のデフォルト URL は次の 2 つの形式のうちのいずれかで指定できます。

```
jdbc:db2os390sqlj:
```
または
```
jdbc:default:connection
```

デフォルト URL を指定すると、アプリケーションに、CICS の接続先のローカル DB2 への接続が指定されます。ご使用のシステムで DB2 データ共用を使用している場合は、ローカル DB2 からシスプレックス内のすべてのデータにアクセスできます。

**明示的 URL**

明示的 URL には DB2 サブシステムのロケーション名が含まれます。 DB2 for z/OS の明示的 URL の基本構造は以下のとおりです。

```
jdbc:db2os390:<location-name>
```
または
```
jdbc:db2os390sqlj:<location-name>
```

通常、ロケーション名は、CICS の接続先のローカル DB2 の名前です。ただし、アクセスするリモート DB2 の名前を指定することができます。この場合、CICS はローカル DB2 をパススルーとして使用し、DB2 分散データ機能を使用してリモート DB2 にアクセスします。

CICS 環境ではデフォルト URL を使用することをお勧めします。明示的 URL を使用すると、同じアプリケーション・スイートで複数のプログラムが使用されている場合に不都合となる可能性のある特定のアクションが、接続のクローズ時に実行される場合があります。また、デフォルト URL を使用すれば、接続の動作が JDBC ドライバーのバージョンによって影響を受けることはありません。

接続を獲得するには、Java アプリケーションから、URL を指定して getConnection() メソッドを呼び出す必要があります。以下に例を示します。

```
Connection connection = DriverManager.getConnection("jdbc:default:connection");
```

## データベースへのデータ・ソース接続の取得

DataSource インターフェースを使用する場合、データベースへの接続は、Java Naming and Directory Interface (JNDI) の DataSource 名ルックアップを使用して取得されます。 JDBC DataSource 実装は、CICS JDBC によって提供されます。

### このタスクについて

データ・ソース接続は、CICS JDBC Liberty 機能を使用して Liberty JVM サーバーで構成します。

データ・ソースを構成した後、アプリケーションは、Liberty サーバー構成の cicsts_dataSource 要素に指定されている jndiName の値を使用して、JNDI ネーミング・サービスから該当する DataSource クラスのインスタンスを取得できます。そして、その DataSource オブジェクトの getConnection() メソッドを呼び出して接続を取得できます。以下に例を示します。

```
Context context = new InitialContext();
DataSource dataSource = (DataSource) context.lookup("jdbc/defaultCICSDataSource");

Connection connection = dataSource.getConnection();
```

### 接続可能数

CICS 用の Java アプリケーションでは、一度に最大 1 つの JDBC 接続または SQLJ 接続のコンテキストを開くことができます。

JDBC ではアプリケーションは同時に複数の接続を行うことができますが、CICS ではこれを行うことはできません。ただし、アプリケーションは既存の接続を閉じて、新しい DB2 ロケーションへの接続を開くことができます。

接続が開いているアプリケーションは、JDBC または SQLJ を使用する必要がある別のアプリケーションにリンクする前に、その接続を閉じる必要があります。アプリケーション・スイートの一部である Java プログラムでは、接続を閉じた場合の影響を考慮する必要があります。明示的 URL を使用している場合、接続を閉じると同期点が取られる可能性があるからです。デフォルト URL を使用する場合は、接続を閉じるときに同期点を取る必要はありません。これについて詳しくは、92 ページの『作業単位のコミット』を参照してください。

## データベースへの接続を閉じる

データベースへの接続を閉じると、JDBC と SQLJ のリソースが自動的に解放されます。通常、データベースへの接続はタスクの終了時に閉じられます。

パフォーマンス上の理由から、アプリケーションは、同じ JVM のユーザーが後で使用できるように JDBC 接続を開いたままにする場合があります。 JDBC 接続が開いたままになっている場合、アプリケーションは、JDBC リソースが時間の経過に伴いリークしないようにする必要があります。

アプリケーションで JDBC または SQLJ 接続を開いたままにする場合は、そのアプリケーションで以下を行う必要があります。

- JDBC と SQLJ のリリースが解放されていることの確認。
- 基礎となる DB2 接続の DB2 SIGNON 後のリカバリー。
- キャッシュされた接続が無効になった場合 (例えば、StaleConnection, SQLCODE=4499) の接続のリサイクル。

接続がキャッシュされている場合は、所定の数のトランザクション後に接続をリサイクルするロジックをアプリケーションに組み込むことをお勧めします。キャッシュされた接続のリサイクルはリソースの漏えいを防ぎます。

# 作業単位のコミット

作業単位をコミットするために、JDBC および SQLJ アプリケーションで、JDBC および SQLJ のコミットとロールバック・メソッド呼び出しを実行することができます。 DB2 JDBC ドライバーはこれらの呼び出しを JCICS コミットまたは JCICS ロールバック呼び出しに変換します。その結果、CICS 同期点が取得されます。

## このタスクについて

JDBC または SQLJ コミットでは、DB2 に対して行われた更新だけでなく、CICS 作業単位全体がコミットされます。 CICS では、残りの CICS 作業単位とは別個に JDBC 接続を使用して行われた作業のコミットはサポートされません。

JDBC または SQLJ アプリケーションでは JCICS コミットまたはロールバックを直接実行することもでき、その結果は JDBC または SQLJ のコミットまたはロールバック・メソッド呼び出しを実行した場合と同じになります。作業単位全体は、DB2 更新と CICS 制御リソースに対する更新の両方について、まとめてコミットまたはロールバックされます。

JDBC 接続で作業する場合、DB2 データベースへの接続のクローズ時に、同期点の取得と作業単位のコミットを避けられないことがあります。これは、以下のいずれかの場合に当てはまります。

- JDBC 接続の自動コミット・プロパティーを使用した。
- 明示的 URL を使用して DriverManager 接続を獲得した。

スタンドアロン・アプリケーションの場合、これらの規則が原因で問題が発生することはありません。 CICS では、接続のクローズ時だけでなく、タスクの終了時にも同期点が確実に取得されるためです。ただし、JDBC および SQLJ アプリケーション・プログラミング・インターフェースでは、作業単位ごとの複数のアプリケーション・プログラムの概念はサポートされません。アプリケーションを構成する複数のプログラムがある場合、1 つのプログラムが DB2 にアクセスしてから、単一の作業単位の過程で、同様に DB2 へのアクセスを行う別のプログラムを呼び出す可能性があります。これらのプログラムとして、JDBC または SQLJ を使用する Java プログラムを指定する場合は、DB2 への接続のクローズ時に作業単位がコミットされないようにする必要があります。そうしないと、アプリケーションは計画通りに作動しません。既存のアプリケーションのプログラムを、JDBC または SQLJ を使用する Java プログラムに置き換えて、プログラム間で同じ CICS-DB2 スレッドを共用する場合は、特にこの要件に注意する必要があります。この問題に対処するには、デフォルトの URL で DriverManager を使用するか、DataSource 接続を使用してください。

93 ページの『自動コミット』
JDBC アプリケーションでは、JDBC 接続の自動コミット・プロパティーを使用できます。自動コミット・プロパティーによって、DB2 の各更新後にコミットが行われます。このコミットは CICS コミットであり、結果として作業単位全体がコミットされます。

93 ページの『明示的 URL およびデフォルト URL での DriverManager の同期点の問題』
JDBC または SQLJ を使用する CICS 用の Java アプリケーションで、明示的

URL を使用して接続が獲得される場合、アプリケーションは、SYNCONRETURN 属性でリンクされている DPL サーバー・プログラムの環境に類似した環境で動作します。

## 自動コミット

JDBC アプリケーションでは、JDBC 接続の自動コミット・プロパティーを使用できます。自動コミット・プロパティーによって、DB2 の各更新後にコミットが行われます。このコミットは CICS コミットであり、結果として作業単位全体がコミットされます。

自動コミット・プロパティーを使用すると、接続が閉じられたときにもコミットが実行されます。これは、明示的 URL またはデフォルトの URL を使用して取得された DriverManager 接続の場合でも、タイプ 2 ドライバーのデータ・ソース接続の場合でも同じです。

CICS 環境での自動コミットの使用は推奨されません。そのため、DB2 JDBC ドライバーでは、CICS 環境での実行時のデフォルトは `autocommit(false)` に設定されます。非 CICS 環境でのデフォルトは `autocommit(true)` です。

## 明示的 URL およびデフォルト URL での DriverManager の同期点の問題

JDBC または SQLJ を使用する CICS 用の Java アプリケーションで、明示的 URL を使用して接続が獲得される場合、アプリケーションは、SYNCONRETURN 属性でリンクされている DPL サーバー・プログラムの環境に類似した環境で動作します。

JDBC または SQLJ を使用するアプリケーション・プログラムが明示的 URL 接続を閉じるときに、CICS が IBM Data Server Driver for JDBC and SQLJ を使用している場合、暗黙的な同期点は取られません。

ただし、明示的 URL 接続のクローズは、作業単位の境界にあるときにのみ成功します。したがって、アプリケーションでは、接続を閉じる前に、JDBC または SQLJ のコミット・メソッド呼び出しか JCICS コミットを発行して、同期点を取る必要があります。 (アプリケーションでは、確実に同期点を取るために autocommit(true) を使用できますが、このプロパティーの使用は CICS 環境では推奨されません。) アプリケーション・プログラムが明示的 URL 接続を閉じたときに、作業単位の終了となります。

明示的 URL の代わりに**デフォルト** URL を使用して接続を獲得するか、デフォルト URL 接続を提供するデータ・ソースを使用することによって、この制限に対処できます (88 ページの『データベースへの接続の取得』 を参照)。デフォルト URL が使用される場合、Java アプリケーションでは、作業単位の境界で接続を閉じる必要はなく、接続が閉じられるときに同期点は取られません (autocommit(true) が指定されていないことを前提とします)。

CICS 環境では、常に、デフォルト URL 接続を使用することをお勧めします。

## JDBC 要求または SQLJ 要求の間に CICS が異常終了する

DB2 提供の JDBC ドライバーによって作成される EXEC SQL 要求の処理中に発行される CICS 異常終了は、Java 例外に変換されないため、CICS 用の Java アプリケーションではキャッチされません。 CICS トランザクションは異常終了して、最終同期点までロールバックされます。

# Java プログラムからの WebSphere MQ へのアクセス

CICS で実行される Java プログラムは、WebSphere MQ classes for Java または WebSphere MQ classes for JMS を使用して WebSphere MQ にアクセスできます。

WebSphere MQ classes for Java は、ネイティブ WebSphere MQ API である Message Queue Interface (MQI) をカプセル化します。プロシージャー型言語での MQI の使用をすでに理解している場合は、その知識を Java 環境で活用することが可能です。これらのクラスでは、WebSphere MQ の C++ および .NET インターフェースと類似のオブジェクト・モデルが使用されています。また、JMS で利用できる機能にとどまらず、WebSphere MQ の完全な機能一式を活用できます。

WebSphere MQ classes for JMS は、メッセージング・システムとして WebSphere MQ 用の JMS インターフェースを実装しています。組織が WebSphere MQ を初めて使用するものの、JMS アプリケーション開発スキルは既に持っている場合、WebSphere MQ に付属する他の API のいずれかを使用するよりも、使い慣れた JMS API を使用して WebSphere MQ リソースにアクセスする方が容易であることに気付くでしょう。

CICS JVM サーバー環境で各 WebSphere MQ クラスを使用するにあたり、それぞれに固有のインストール要件と運用上の特性があります。WebSphere MQ classes for Java は、OSGI JVM サーバーと Liberty JVM サーバーの両方で使用できます。WebSphere MQ classes for JMS は、OSGI JVM サーバーでのみサポートされ、Liberty JVM サーバーではサポートされません。

『OSGi JVM サーバーでの WebSphere MQ classes for Java の使用』
OSGi JVM サーバーで実行される Java プログラムは、WebSphere MQ classes for Java を使用して WebSphere MQ にアクセスできます。

97 ページの『Liberty JVM サーバーでの WebSphere MQ classes for Java の使用』
Liberty JVM サーバーで実行される Java プログラムは、WebSphere MQ classes for Java を使用して WebSphere MQ にアクセスできます。

100 ページの『OSGi JVM サーバーでの WebSphere MQ classes for JMS の使用』
OSGi JVM サーバーで実行される Java プログラムは、WebSphere MQ classes for JMS を使用して WebSphere MQ にアクセスできます。

## OSGi JVM サーバーでの WebSphere MQ classes for Java の使用

OSGi JVM サーバーで実行される Java プログラムは、WebSphere MQ classes for Java を使用して WebSphere MQ にアクセスできます。

WebSphere MQ classes for Java の概要については、WebSphere MQ バージョン 8 製品資料内の『IBM MQ classes for Java の使用』を参照してください。

『WebSphere MQ classes for Java を CICS OSGi WebSphere MQ 環境で機能させる』
WebSphere MQ 製品が備える WebSphere MQ classes for Java には、CICS Java アプリケーションで使用するための Java 版の Message Queue Interface (MQI) が用意されています。

96 ページの『WebSphere MQ classes for Java をサポートするための OSGi JVM サーバーの構成』
JVM サーバーは、Java アプリケーション用のランタイム環境です。WebSphere MQ classes for Java を使用したアプリケーションをサポートするように OSGi JVM サーバーを構成することができます。

96 ページの『CICS WebSphere MQ 環境での WebSphere MQ classes for Java を使用したプログラミング』
WebSphere MQ classes for Java は、ネイティブ WebSphere MQ API である Message Queue Interface (MQI) をカプセル化しており、他のオブジェクト指向インターフェースと同じオブジェクト・モデルを使用しています。WebSphere MQ の完全な機能一式を活用できます。その中には、WebSphere MQ classes for JMS では利用できないものもあります。

97 ページの『WebSphere MQ 要求に関わる作業単位のコミット』
CICS JVM サーバー環境で WebSphere MQ classes for Java によって送受信されるメッセージは常に、CICS の作業単位 (UOW) と関連付けられます。

97 ページの『WebSphere MQ 要求を処理中の CICS の異常終了』
WebSphere MQ classes for Java を使用すると、結果として WebSphere MQ の MQI コマンドが発行されます。MQI コマンドの処理中に発生した CICS の異常終了は、Java 例外に変換されないため、CICS Java アプリケーションによってキャッチされません。

## WebSphere MQ classes for Java を CICS OSGi WebSphere MQ 環境で機能させる

WebSphere MQ 製品が備える WebSphere MQ classes for Java には、CICS Java アプリケーションで使用するための Java 版の Message Queue Interface (MQI) が用意されています。

CICS 環境では、WebSphere MQ の提供するクラスは、バインディング・モードでの接続のみを許可します。クライアント・モードでリモート・キュー・マネージャーへの接続を使用しようとすると、例外が発生します。

バインディング・モードでは、呼び出し要求が WebSphere MQ の MQI 呼び出しに変換され、既存の CICS-WebSphere MQ アダプターによって通常どおりに処理されます。変換された要求は、他のプログラム (COBOL プログラムなど) からの MQI 要求とまったく同じ方式で CICS-WebSphere MQ アダプターに到着します。そのため、WebSphere MQ にアクセスする Java プログラムと他のプログラムの間に、動作の違いはありません。

接続オプションについては、WebSphere MQ 製品資料内の『IBM MQ classes for Java の接続オプション』を参照してください。

## WebSphere MQ classes for Java をサポートするための OSGi JVM サーバーの構成

JVM サーバーは、Java アプリケーション用のランタイム環境です。WebSphere MQ classes for Java を使用したアプリケーションをサポートするように OSGi JVM サーバーを構成することができます。

### このタスクについて

WebSphere MQ classes for Java を使用したアプリケーションを OSGi JVM サーバーがサポートできるようにするには、WebSphere MQ for Java バンドルを、JVM サーバー内の OSGi フレームワークで実行されるミドルウェア・バンドルのセットに追加する必要があります。また、このフレームワークは、関連付けられたネイティブ・ライブラリーのセットにアクセスできなければなりません。

### 手順

1. WebSphere MQ classes for Java を JVM サーバーに OSGi ミドルウェア・バンドルとして追加します。 WebSphere MQ バージョン 8 以降でこれらのクラスを追加するには、OSGi JVM サーバーの JVM プロファイルに以下の行を含めます。

   ```
   OSGI_BUNDLES=<MQ_ROOT>/OSGi/com.ibm.mq.osgi.allclientprereqs_<VERSION>.jar,\
   <MQ_ROOT>/OSGi/com.ibm.mq.osgi.allclient_<VERSION>.jar
   ```

   WebSphere MQ for z/OS バージョン 7.1 の場合は、以下の行を含めます。

   ```
   OSGI_BUNDLES=<MQ_ROOT>/OSGi/com.ibm.mq.osgi.java_<VERSION>.jar
   ```

   ここで、

   - *MQ_ROOT* は、WebSphere MQ for z/OS Unix System Services のインストール済み環境の `java/lib/` ディレクトリーです (例: `/usr/lpp/V8R0M0/java/lib`)。
   - *VERSION* は、使用する WebSphere MQ classes for Java のバージョンです (例: `8.0.0.0`)。

2. WebSphere MQ classes for Java ネイティブ・ライブラリーが含まれるディレクトリーを、OSGi JVM サーバーの JVM プロファイル内の LIBPATH_SUFFIX オプションに追加します。 以下に例を示します。

   ```
   LIBPATH_SUFFIX=<MQ_ROOT>
   ```

   ここで、*MQ_ROOT* は、WebSphere MQ for z/OS Unix System Services のインストール済み環境の `java/lib/` ディレクトリーです (例: `/usr/lpp/V8R0M0/java/lib`)。

## CICS WebSphere MQ 環境での WebSphere MQ classes for Java を使用したプログラミング

WebSphere MQ classes for Java は、ネイティブ WebSphere MQ API である Message Queue Interface (MQI) をカプセル化しており、他のオブジェクト指向インターフェースと同じオブジェクト・モデルを使用しています。WebSphere MQ の完全な機能一式を活用できます。その中には、WebSphere MQ classes for JMS では利用できないものもあります。

CICS アプリケーションでの WebSphere MQ classes for Java の使用は、WebSphere MQ for z/OS 7.1 以降でサポートされます。

WebSphere MQ classes for Java について詳しくは、WebSphere MQ バージョン 8 製品資料内の『IBM MQ classes for Java の使用』を参照してください。

### WebSphere MQ 要求に関わる作業単位のコミット

CICS JVM サーバー環境で WebSphere MQ classes for Java によって送受信されるメッセージは常に、CICS の作業単位 (UOW) と関連付けられます。

その UOW を完了するには、`com.ibm.cics.server.Task` オブジェクトのコミット・メソッドまたはロールバック・メソッドを呼び出すしかありません。あるいは、CICS タスクが正常に終了すれば、UOW は暗黙的にコミットされます。MQQueueManager に対するトランザクション制御メソッドの使用は、サポートされません。

混合言語アプリケーションの場合、Java 以外のプログラムから発行された **EXEC CICS SYNCPOINT** コマンドは、Java プログラムによって WebSphere MQ に加えられた更新を含め、作業単位全体をコミットします。

### WebSphere MQ 要求を処理中の CICS の異常終了

WebSphere MQ classes for Java を使用すると、結果として WebSphere MQ の MQI コマンドが発行されます。MQI コマンドの処理中に発生した CICS の異常終了は、Java 例外に変換されないため、CICS Java アプリケーションによってキャッチされません。

この状況では、CICS トランザクションが異常終了し、最後の同期点までロールバックされます。

## Liberty JVM サーバーでの WebSphere MQ classes for Java の使用

Liberty JVM サーバーで実行される Java プログラムは、WebSphere MQ classes for Java を使用して WebSphere MQ にアクセスできます。

WebSphere MQ classes for Java の概要については、WebSphere MQ バージョン 8 製品資料内の『IBM MQ classes for Java の使用』を参照してください。

98 ページの『WebSphere MQ classes for Java を CICS Liberty WebSphere MQ 環境で機能させる』
WebSphere MQ 製品が備える WebSphere MQ classes for Java には、CICS Java アプリケーションで使用するための Java 版の Message Queue Interface (MQI) が用意されています。

98 ページの『WebSphere MQ classes for Java をサポートするための Liberty JVM サーバーの構成』
JVM サーバーは、Java アプリケーション用のランタイム環境です。WebSphere MQ classes for Java を使用したアプリケーションをサポートするように Liberty JVM サーバーを構成することができます。

99 ページの『CICS WebSphere MQ 環境での WebSphere MQ classes for Java を使用したプログラミング』

WebSphere MQ classes for Java は、ネイティブ WebSphere MQ API である Message Queue Interface (MQI) をカプセル化しており、他のオブジェクト指向インターフェースと同じオブジェクト・モデルを使用しています。WebSphere MQ の完全な機能一式を活用できます。その中には、WebSphere MQ classes for JMS では利用できないものもあります。

100 ページの『WebSphere MQ 要求に関わる作業単位のコミット』
CICS Liberty JVM サーバー環境で WebSphere MQ classes for Java によって送受信されるメッセージは常に、CICS の作業単位 (UOW) と関連付けられます。

100 ページの『WebSphere MQ 要求を処理中の CICS の異常終了』
WebSphere MQ classes for Java を使用すると、結果として WebSphere MQ の MQI コマンドが発行されます。MQI コマンドの処理中に発生した CICS の異常終了は、Java 例外に変換されないため、CICS Java アプリケーションによってキャッチされません。

## WebSphere MQ classes for Java を CICS Liberty WebSphere MQ 環境で機能させる

WebSphere MQ 製品が備える WebSphere MQ classes for Java には、CICS Java アプリケーションで使用するための Java 版の Message Queue Interface (MQI) が用意されています。

CICS 環境では、WebSphere MQ の提供するクラスは、バインディング・モードでの接続のみを許可します。クライアント・モードでリモート・キュー・マネージャーへの接続を使用しようとすると、例外が発生します。

バインディング・モードでは、呼び出し要求が WebSphere MQ の MQI 呼び出しに変換され、既存の CICS-WebSphere MQ アダプターによって通常どおりに処理されます。変換された要求は、他のプログラム (COBOL プログラムなど) からの MQI 要求とまったく同じ方式で CICS-WebSphere MQ アダプターに到着します。そのため、WebSphere MQ にアクセスする Java プログラムと他のプログラムの間に、動作の違いはありません。

接続オプションについては、WebSphere MQ 製品資料内の『IBM MQ classes for Java の接続オプション』を参照してください。

## WebSphere MQ classes for Java をサポートするための Liberty JVM サーバーの構成

JVM サーバーは、Java アプリケーション用のランタイム環境です。WebSphere MQ classes for Java を使用したアプリケーションをサポートするように Liberty JVM サーバーを構成することができます。

### このタスクについて

Web アーカイブ (WAR) ファイルまたはエンタープライズ・アプリケーション・アーカイブ (EAR) ファイルとしてデプロイされるアプリケーションの場合と、エンタープライズ・バンドル・アーカイブ (EBA) ファイルとしてデプロイされるアプリケーションの場合とでは、異なる構成が必要になります。

### 手順

1. アプリケーションを Web アーカイブ (WAR) ファイルまたはエンタープライズ・アプリケーション・アーカイブ (EAR) ファイルとして Liberty JVM サーバーにデプロイする場合は、WebSphere MQ の JAR ファイルを Liberty グローバル・ライブラリーに追加します。これにより、それらの JAR ファイルがすべての非 OSGi Web アプリケーション (WAR ファイルと EAR ファイルを含む) のクラスパスに追加されます。 このグローバル・ライブラリーは、Liberty JVM サーバーにインストールされたすべての非 OSGi アプリケーションで使用可能です。これは、以下のように Liberty の `server.xml` 構成ファイルで定義します。

   ```
   <fileset id="mqjars" dir="MQ_ROOT" includes="*.jar"/>
   <library id="global" filesetRef="mqjars" />
   ```

   ここで、*MQ_ROOT* は、WebSphere MQ for z/OS Unix System Services のインストール済み環境の `java/lib/` ディレクトリーです (例: `/usr/lpp/V8R0M0/java/lib`)。 これらのライブラリーは、Liberty JVM サーバーにデプロイされたエンタープライズ・バンドル・アーカイブ (EBA) ファイルでは使用できない点に注意してください。詳しくは、WebSphere Application Server for z/OS 8.5.5 製品資料内の『すべての Java EE アプリケーションのグローバル・ライブラリーの提供』を参照してください。

2. アプリケーションをエンタープライズ・バンドル・アーカイブ (EBA) ファイルとして Liberty JVM サーバーにデプロイする場合は、WebSphere MQ classes for Java を OSGi ミドルウェア・バンドルとして Liberty 共用バンドル・リポジトリーに追加します。 このバンドル・リポジトリーは、Liberty JVM で実行されるすべての OSGi アプリケーションで使用可能になる、OSGi バンドルのセットです。これは、Liberty の `server.xml` 構成ファイルで定義します。 WebSphere MQ バージョン 8 以降では、以下の定義を使用します。

   ```
   <fileset id="mqosgilib" dir="MQ_ROOT/OSGi"
        includes="com.ibm.mq.osgi.allclientprereqs_VERSION.jar
                  com.ibm.mq.osgi.allclient_VERSION.jar"/>
   <bundleRepository filesetRef="mqosgilib" />
   ```

   WebSphere MQ バージョン 7 の場合は、以下の定義を使用します。

   ```
   <fileset id="mqosgilib" dir="MQ_ROOT/OSGi" includes="com.ibm.mq.osgi.java_VERSION.jar"/>
   <bundleRepository filesetRef="mqosgilib" />
   ```

   ここで、

   - *MQ_ROOT* は、WebSphere MQ for z/OS Unix System Services のインストール済み環境の `java/lib/` ディレクトリーです (例: `/usr/lpp/V8R0M0/java/lib`)。
   - *VERSION* は、使用する WebSphere MQ classes for Java のバージョンです (例: `8.0.0.0`)。

## CICS WebSphere MQ 環境での WebSphere MQ classes for Java を使用したプログラミング

WebSphere MQ classes for Java は、ネイティブ WebSphere MQ API である Message Queue Interface (MQI) をカプセル化しており、他のオブジェクト指向インターフェースと同じオブジェクト・モデルを使用しています。WebSphere MQ の完全な機能一式を活用できます。その中には、WebSphere MQ classes for JMS では利用できないものもあります。

CICS アプリケーションでの WebSphere MQ classes for Java の使用は、WebSphere MQ for z/OS 7.1 以降でサポートされます。

WebSphere MQ classes for Java について詳しくは、WebSphere MQ バージョン 8 製品資料内の『IBM MQ classes for Java の使用』を参照してください。

### WebSphere MQ 要求に関わる作業単位のコミット

CICS Liberty JVM サーバー環境で WebSphere MQ classes for Java によって送受信されるメッセージは常に、CICS の作業単位 (UOW) と関連付けられます。

その UOW を完了するには、`com.ibm.cics.server.Task` オブジェクトのコミット・メソッドまたはロールバック・メソッドを呼び出すしかありません。あるいは、CICS タスクが正常に終了すれば、UOW は暗黙的にコミットされます。MQQueueManager に対するトランザクション制御メソッドの使用は、サポートされません。

混合言語アプリケーションの場合、Java 以外のプログラムから発行された **EXEC CICS SYNCPOINT** コマンドは、Java プログラムによって WebSphere MQ に加えられた更新を含め、作業単位全体をコミットします。

### WebSphere MQ 要求を処理中の CICS の異常終了

WebSphere MQ classes for Java を使用すると、結果として WebSphere MQ の MQI コマンドが発行されます。MQI コマンドの処理中に発生した CICS の異常終了は、Java 例外に変換されないため、CICS Java アプリケーションによってキャッチされません。

この状況では、CICS トランザクションが異常終了し、最後の同期点までロールバックされます。

## OSGi JVM サーバーでの WebSphere MQ classes for JMS の使用

OSGi JVM サーバーで実行される Java プログラムは、WebSphere MQ classes for JMS を使用して WebSphere MQ にアクセスできます。

WebSphere MQ classes for JMS の概要については、WebSphere MQ 製品資料内の『IBM MQ classes for JMS の使用』を参照してください。

101 ページの『WebSphere MQ classes for JMS を使用するための OSGi JVM サーバーの設定』
CICS Java アプリケーションが JMS 要求を行うと、WebSphere MQ の提供する MQ クラスによってその要求が処理されます。

101 ページの『JMS をサポートするための OSGi JVM サーバーの構成』
JVM サーバーは、Java アプリケーション用のランタイム環境です。JMS を使用したアプリケーションをサポートするように OSGi JVM サーバーを構成することができます。

102 ページの『CICS WebSphere MQ 環境での WebSphere MQ classes for JMS を使用したプログラミング』
CICS Java プログラムは、IBM WebSphere MQ classes for JMS で提供される各

種機能を呼び出すことができます。 IBM WebSphere MQ classes for JMS は、JMS 準拠の IBM WebSphere MQ オファリングの一部を成しています。

102 ページの『JMS アプリケーションの接続ファクトリーおよび宛先の作成と構成』
CICS Java プログラムでは、接続ファクトリーおよび宛先を作成して構成することができます。

103 ページの『JMS 環境での作業単位のコミット』
CICS JVM サーバー環境で WebSphere MQ classes for JMS によって送受信されるメッセージは常に、CICS の作業単位 (UOW) と関連付けられます。

103 ページの『JMS 要求を処理中の CICS の異常終了』
JMS 要求の結果として、WebSphere MQ の MQI コマンドが発行されます。MQI コマンドの処理中に発生した CICS の異常終了は、Java 例外に変換されないため、CICS Java アプリケーションによってキャッチされません。

## WebSphere MQ classes for JMS を使用するための OSGi JVM サーバーの設定

CICS Java アプリケーションが JMS 要求を行うと、WebSphere MQ の提供する MQ クラスによってその要求が処理されます。

CICS 環境では、WebSphere MQ の提供するクラスは、バインディング・モードでの接続のみを許可します。クライアント・モードでリモート・キュー・マネージャーへの接続を使用しようとすると、例外が発生します。

バインディング・モードでは、JMS 要求が WebSphere MQ の MQI 呼び出しに変換され、既存の CICS-WebSphere MQ アダプターによって通常どおりに処理されます。変換された要求は、他のプログラム (COBOL プログラムなど) からの MQI 要求とまったく同じ方式で CICS-WebSphere MQ アダプターに到着します。そのため、WebSphere MQ にアクセスする Java プログラムと他のプログラムの間に、動作の違いはありません。

特定のレベルの JMS を使用するには、適切なレベルの JMS をサポートする WebSphere MQ キュー・マネージャーに CICS を接続する必要があります。詳しくは、WebSphere MQ バージョン 8 製品資料内の『CICS OSGi JVM サーバー内での IBM MQ classes for JMS の使用』を参照してください。.

WebSphere MQ classes for JMS を使用した WebSphere MQ からのサポートは、WebSphere MQ for z/OS バージョン 7.1 およびバージョン 8 で利用できます。

- バージョン 7.1 とは、MQ APAR PI29770 (基礎はフィックスパック 7.1.0.6) 以降のフィックスパック・レベルが該当します。
- バージョン 8 とは、ベースの APAR PI28482 およびフィックスパック 8.0.0.2 以降のフィックスパック・レベルが該当します。

## JMS をサポートするための OSGi JVM サーバーの構成

JVM サーバーは、Java アプリケーション用のランタイム環境です。JMS を使用したアプリケーションをサポートするように OSGi JVM サーバーを構成することができます。

### このタスクについて

WebSphere MQ classes for JMS を使用したアプリケーションを OSGi JVM サーバーがサポートできるようにするには、WebSphere MQ for JMS バンドルを、JVM サーバー内の OSGi フレームワークで実行されるミドルウェア・バンドルのセットに追加する必要があります。また、このフレームワークは、関連付けられたネイティブ・ライブラリーのセットにアクセスできなければなりません。

説明は、WebSphere MQ バージョン 8 製品資料内の『JVM サーバー環境のセットアップ』を参照してください。

## CICS WebSphere MQ 環境での WebSphere MQ classes for JMS を使用したプログラミング

CICS Java プログラムは、IBM WebSphere MQ classes for JMS で提供される各種機能を呼び出すことができます。 IBM WebSphere MQ classes for JMS は、JMS 準拠の IBM WebSphere MQ オファリングの一部を成しています。

JMS 仕様の観点から、WebSphere MQ classes for JMS は、CICS JVM サーバーを、進行中の JTA トランザクションが常に存在する Java EE 準拠のアプリケーション・サーバーとして扱います。特に、WebSphere MQ classes for JMS は、それが EJB コンテナーで実行されていると見なします。そのため、CICS 環境での JMS API の使用には制約があります。詳しくは、WebSphere MQ バージョン 8 製品資料内の『JMS API の制約事項』を参照してください。.

クラシック (JMS 1.1) インターフェースおよび単純化 (JMS 2.0) インターフェースの使用がサポートされています。ただし、CICS が、適切なレベルの JMS をサポートできるレベルの WebSphere MQ キュー・マネージャーに接続されており、適切なバージョンの WebSphere MQ classes for JMS を使用していることが条件です。

**注:** WebSphere MQ for z/OS バージョン 7.1 は JMS 1.1 のみをサポートし、バージョン 8 は JMS 1.1 と JMS 2.0 の両方をサポートします。

## JMS アプリケーションの接続ファクトリーおよび宛先の作成と構成

CICS Java プログラムでは、接続ファクトリーおよび宛先を作成して構成することができます。

以下の方式で、接続ファクトリーおよび宛先の WebSphere MQ 実装を作成して構成することができます。

- JNDI を使用した管理対象オブジェクトの取得
- IBM JMS 拡張機能の使用
- WebSphere MQ JMS 拡張機能の使用

たいていの JMS ユーザーは、JNDI リポジトリーを使用して、事前構成された一式の接続ファクトリーおよび宛先を見つけます。CICS では、JNDI 実装は提供されていません。また、LDAP は OSGi 環境では使用できません。

使用可能なオプションの詳細について、また、バンドル・アダプターの start メソッドを使用して OSGi に初期コンテキスト・ファクトリーと WebSphere MQ オブジ

エクト・ファクトリーを登録する方法の例については、WebSphere MQ バージョン 8 製品資料内の『接続ファクトリーおよび宛先の作成および構成』を参照してください。.

CICS と WebSphere MQ QMGR の間の接続は、CICS アドレス・スペースのユーザー ID を使用して管理されます。キューへのリソース・アクセスは、トランザクション・ユーザー ID によって許可されます。そのため、接続ファクトリーでユーザー ID およびパスワードを指定することはサポートされていません。

### JMS 環境での作業単位のコミット

CICS JVM サーバー環境で WebSphere MQ classes for JMS によって送受信されるメッセージは常に、CICS の作業単位 (UOW) と関連付けられます。

その UOW を完了するには、`com.ibm.cics.server.Task` オブジェクトのコミット・メソッドまたはロールバック・メソッドを呼び出すしかありません。あるいは、CICS タスクが正常に終了すれば、UOW は暗黙的にコミットされます。 JMS API 要求を使用して作業単位をコミットまたはロールバックすることは、サポートされていません。

上記の規則には、非永続メッセージのサポートに関係する例外が 1 つあります。非永続メッセージは、ネイティブ MQI でもサポートされます。詳しくは、WebSphere MQ バージョン 8 製品資料内の『トランザクション動作』を参照してください。.

混合言語アプリケーションの場合、Java 以外のプログラムから発行された **EXEC CICS SYNCPOINT** コマンドは、Java プログラムによって WebSphere MQ に加えられた更新を含め、作業単位全体をコミットします。

### JMS 要求を処理中の CICS の異常終了

JMS 要求の結果として、WebSphere MQ の MQI コマンドが発行されます。 MQI コマンドの処理中に発生した CICS の異常終了は、Java 例外に変換されないため、CICS Java アプリケーションによってキャッチされません。

この状況では、CICS トランザクションが異常終了し、最後の同期点までロールバックされます。

## Liberty JVM サーバーで実行する Java アプリケーションの開発

WebSphere Liberty プロファイルを使用する Java Web アプリケーションをデプロイする場合は、Web コンテナーを実行するように Liberty JVM サーバーを構成します。

CICS Explorer SDK をダウンロードしてインストールする方法については、CICS Explorer(r) SDK のインストールを参照してください。

104 ページの『開発環境のセットアップ』
Liberty JVM サーバーで実行する Web アプリケーションを開発するには、CICS Explorer SDK に付属のサーブレットおよび JSP フィーチャーをインストールする必要があります。

106 ページの『Java Management Extensions API (JMX)』
Java Management Extensions API (JMX) は、リソースのモニターと管理に使用します。

107 ページの『Java Message Service (JMS) を使用するアプリケーションの開発』
Java Message Service (JMS) は、Java EE をベースにしたアプリケーション・コンポーネントでメッセージの作成、送信、受信、および読み取りを可能にする API です。WebSphere Liberty プロファイルの JMS サポートは、JMS リソース・アダプターのデプロイメントをサポートする一群の関連フィーチャーとして提供されます。

109 ページの『Java Transaction API (JTA)』
Java Transaction API (JTA) を使用すると、複数のリソース・マネージャーに対するトランザクション更新を調整することができます。

111 ページの『JCA プログラミング・インターフェースを使用したアプリケーションの開発』
JCA は、CICS などのエンタープライズ情報システムを JEE プラットフォームに接続します。

122 ページの『サーブレットおよび JSP アプリケーションの開発』
CICS アプリケーションに対する最新のインターフェースを提供するために、Web アプリケーション・テクノロジーを使用するプレゼンテーション層を開発できます。Eclipse ベースの Web 開発ツールは、これらのアプリケーションを作成するための開発プラットフォームを提供します。CICS Explorer SDK は、CICS での実行用にこれらをビルドし、パッケージ化し、デプロイするためのサポートを提供します。

126 ページの『Liberty JVM サーバー内で実行するよう Java Web アプリケーションをマイグレーションする』
ネットワークを介して CICS にアクセスする Liberty プロファイル・インスタンスの中で実行する Java Web アプリケーションがある場合、そのアプリケーションを Liberty JVM サーバーの中で実行すると、パフォーマンスを最適化できます。

127 ページの『Liberty Web サーバー・プラグイン』
Web サーバー・プラグインを使用することにより、サポートされる Web サーバーから 1 つ以上のアプリケーション・サーバーに HTTP 要求を転送することが可能になります。

# 開発環境のセットアップ

Liberty JVM サーバーで実行する Web アプリケーションを開発するには、CICS Explorer SDK に付属のサーブレットおよび JSP フィーチャーをインストールする必要があります。

## このタスクについて

Liberty プロファイル用の開発者ツールを使用することで、Liberty プロファイル・サーバーのインスタンスで実行できるサーブレットと JSP アプリケーションを開発できます。 Liberty プロファイルは、単純な Web アプリケーションを実行するために WebSphere Application Server で提供されるアプリケーション・サーバーのインスタンスです。構成、デプロイ、および開始を素早く行うことができます。

また、Web アプリケーションを Liberty JVM サーバー内にインストールすることによって、CICS 環境でサーブレットおよび JSP ページを実行できます。CICS Explorer SDK には、CICS 用の Java アプリケーションを開発してデプロイするためのツールが用意されています。これらのツールを開発者ツールと併用すれば、Web インターフェースを持つ Java アプリケーションを CICS にデプロイできます。

ターゲット・プラットフォームを CICS TS V5.*x* Runtime with Liberty に設定してください。そうしない場合、ワークプレースでサンプルを作成する際にコンパイル・エラーが発生する可能性があります。詳しくは、サーブレット・サンプルの作成を参照してください。

## 手順

1. CICS Explorer SDK は既にインストールしているものの、サーブレットおよび JSP フィーチャーをインストールしていない場合は、そのフィーチャーを CICS Explorer SDK ダウンロード・アーカイブからインストールできます。
   a. Eclipse IDE で、「**ヘルプ**」 > 「**新規ソフトウェアのインストール**」をクリックします。

      **注:** 「**必要なソフトウェアを検索するために、インストール時にすべての更新サイトに接続する**」というオプションを必ず選択してください。 Liberty プロファイル・フィーチャーには、他の Eclipse コンポーネントとの依存関係があるため、それらの依存部分を更新マネージャーでダウンロードしてインストールする必要があります。
   b. リストから CICS Explorer SDK ダウンロード・ファイルを選択します。「**Add**」をクリックします。「サイトの追加」ダイアログで、「**アーカイブ**」をクリックします。
   c. CICS Explorer SDK ダウンロード・ファイルを参照し、「**開く**」をクリックします。
   d. 「**IBM CICS Explorer**」を展開します。
   e. 「**IBM CICS SDK for Servlet and JSP のサポート (IBM CICS SDK for Servlet and JSP support)**」の横のチェック・ボックスを選択し、「**次へ**」をクリックします。
   f. ライセンス情報を読んで同意し、「**終了**」をクリックしてフィーチャーをインストールします。 Eclipse IDE にまだインストールされていない依存部分もインストールされます。
2. オプション: EBA をデプロイする場合、WebSphere Application Server Developer Tools for Eclipse をインストールする必要があります。Developer Tools を使用して、Web アプリケーションをローカルでテストすることもできます。Developer Tools のダウンロードおよびインストールについては、WebSphere Application Server Developer Tools for Eclipse のインストールを参照してください。

## タスクの結果

Eclipse IDE を再始動した後に、Liberty JVM サーバーでの実行に適した Web アプリケーションを開発し、デプロイして、テストするためのツールを使用できます。

### 次のタスク

Liberty JVM サーバーで実行する Web アプリケーションの開発を開始できます。開発を開始するために、CICS Explorer SDK で提供されるサンプルを使用することができます。使用開始について詳しくは、topics/gettingstarted_liberty.ditaを参照してください。

## Java Management Extensions API (JMX)

Java Management Extensions API (JMX) は、リソースのモニターと管理に使用します。

JMX は、広く受け入れられている実装を使用してアプリケーションの内部情報を公開することで、その実装に準拠するツール (JConsole など) で情報を取得して使用できるようにする、Java フレームワークおよび API です。この操作のために、Managed Bean (MBean) という、パブリック・コンストラクターを持つ非静的な Java クラスを使用します。この Bean の get および set メソッドは「属性」として公開されますが、それ以外のすべてのメソッドは「操作」として公開されます。

monitor-1.0 機能は、アプリケーションと Liberty JVM サーバーによってどのような処理が実行されているのかを正確に理解できるようにする MBeans のホストを提供します。ローカルにまたはリモート・マシンから Liberty JVM サーバーの JMX に接続して、これらの MBeans の属性と操作を表示できます。ローカルに接続するには、`server.xml` に `localConnector-1.0` 機能を追加して、同じ JVM サーバー内から接続できるようにする必要があります。`restConnector-1.0` 機能を `server.xml` に追加すると、RESTful インターフェースを介して接続できるようになり、JMX へのリモート・アクセスが可能になります。

### WebSphere MBeans を使用したアプリケーションのモニター

1. まず、MBeanServer の参照を取得する必要があります。以下の例では、MBean の登録先のサーバーを確認する 「findMBeanServer」メソッドを使用して、「JvmStats」MBean を検索します。

```
// Create an ObjectName object for the MBean that we're looking for.
ObjectName beanObjName = new ObjectName("WebSphere:type=JvmStats");

// Obtain the full list of MBeanServers.
java.util.List servers = MBeanServerFactory.findMBeanServer(null);
MBeanServer server = null;

// Iterate through our list of MBeanServers and attempt to find the one we want.
for (int i = 0; i < servers.size(); i++)
{
   // Check if the MBean domain matches what we're looking for.
   if (((MBeanServer)servers.get(i).isRegistered(beanObjName))
   {
     server = (MBeanServer)servers.get(i);
   }
}
```

2. 正しい MBeanServer オブジェクトの参照から、MBean の参照を取得し、その MBean が公開する属性のデータを取得できます。次の例は、「JvmStats」MBean の「UpTime」属性を検索します。

```
Object attributeObj = server.getAttribute(beanObjName, "UpTime");
System.out.println("UpTime of JVM is: " + attributeObj + ".");
```

### Liberty での JMX へのリモート接続

Liberty JVM サーバーで JMX にリモート接続するには、 SSL 接続と JEE ロール許可を使用する必要があります。それによって、クライアント・コードは JMXServiceURL を使用してリモート MBean の参照を取得します。

1. REST コネクターでアクセスされるすべての JMX MBeans は、単一の JEE ロール「管理者」によって保護されています。このロールへのアクセスを提供するには、server.xml を編集して、管理者ロールに認証済みユーザーを追加します。

   ```
   <administrator-role>
       <user>myuserid</user>
       <group>group1</group>
   </administrator-role>
   ```

   JEE ロールの使用について詳しくは、 240 ページの『JEE アプリケーション・ロール・セキュリティー』を参照してください。
2. リモートの RESTful な JMX クライアントは、Liberty JVM サーバーへのアクセスに SSL を使用する必要があります。Liberty JVM サーバー用の SSL サポートを構成する方法については、 242 ページの『Java 鍵ストアを使用した Liberty JVM サーバー用の SSL (TLS) の構成』のトピックを参照してください。加えて、JMX クライアントには、restConnector クライアント・サイド JAR ファイルへのアクセス、およびサーバーの署名証明書を含む SSL クライアント鍵ストアへのアクセスが必要です。 restConnector.jar は CICS WLP のインストール時に組み込まれ、&USSHOME;/wlp/clients にあります。
3. クライアント側のコードで、JMXServiceURL オブジェクトを作成する必要があります。これによって、リモート MBeanServerConnection オブジェクトの参照を取得できます。以下の例では、<host> と <httpsPort> がそれぞれサーバーで使用されるものに対応します。

```
JMXServiceURL url = new JMXServiceURL("service:jmx:rest://<host>:<httpsPort>/IBMJMXConnectorREST");
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, environment);
MBeanServerConnection mbsc = jmxConnector.getMBeanServerConnection();
```

4. 接続の取得に成功すると、MBeanServerConnection オブジェクトは MBeanServer オブジェクトからローカル接続したときと同じ機能およびメソッドのセットを提供します。

WebSphere で提供される MBeans について詳しくは、Liberty プロファイル: 提供されている MBean のリストを参照してください。

## Java Message Service (JMS) を使用するアプリケーションの開発

Java Message Service (JMS) は、Java EE をベースにしたアプリケーション・コンポーネントでメッセージの作成、送信、受信、および読み取りを可能にする API です。WebSphere Liberty プロファイルの JMS サポートは、JMS リソース・アダプターのデプロイメントをサポートする一群の関連フィーチャーとして提供されます。

JMS は、キュー、トピック、接続、および他のリソースがサーバー構成によって作成および管理される管理モードで実行できます。これには、JMS 接続ファクトリー、キュー、トピック、およびアクティベーション仕様の構成が含まれます。代わりに、すべてのリソースをアプリケーションの一部として手動で構成する非管理モ

ードで実行することもできます。WebSphere Liberty 組み込み JMS プロバイダーは管理されるため、すべてのリソースは server.xml 構成の一部としてセットアップされます。

## JMS 仕様

Liberty JVM サーバーでサポートされる JMS 仕様レベルは、以下のとおりです。

- JMS 1.1 サポート (jms1.1) では、1.1 仕様レベルの Java Message Service AP を使用することによって、リソース・アダプターがメッセージング・システムにアクセスするように構成できます。

## JMS クライアント

以下の WebSphere Liberty プロファイル機能によって、Liberty JVM サーバーでは各種 JMS クライアント・プロバイダーがサポートされています。

- WebSphere JMS メッセージング・クライアント (wasJmsClient-1.1)。JMS クライアント機能により、JMS 1.1 クライアント・アプリケーションは同じインスタンスまたはリモート・インスタンスの WebSphere Liberty 組み込み JMS プロバイダーからメッセージを送受信できます。
- JCA 1.6 仕様に準拠するそれ以外の JMS リソース・アダプターも、汎用 JCA リソース・アダプター・リンクを使用することによって Liberty で使用できます (WebSphere Application Server ベータ製品資料内の『Overview of JCA configuration elements』を参照)。

## JMS プロバイダー

CICS TS の Liberty プロファイルは、組み込み WebSphere メッセージング・エンジンおよび JCA 1.6 仕様に準拠するサード・パーティーの JMS リソース・アダプターの使用をサポートします。現時点で、IBMMQ メッセージング・システムへの JMS アクセスは、Liberty JVM サーバーではサポートされていません。各種 JMS プロバイダーをサポートする Liberty プロファイル機能は、以下のとおりです。

- WebSphere メッセージング・サーバー (wasJmsServer-1.0)。JMS サーバー機能によって、組み込み WebSphere メッセージング・エンジンを Liberty 内でホストすることが可能になったため、JMS サーバーを別個にインストールして構成する必要がありません (WebSphere Application Server ベータ製品資料内の『Enabling JMS messaging for a single Liberty profile server』を参照)。このサーバーは、CICS 内部のまたは z/OS や他の分散プラットフォームでホストされる WebSphere Liberty サーバーの、別個の Liberty インスタンスでホストすることもできます (WebSphere Application Server ベータ製品資料内の『Enabling JMS messaging between two Liberty profile servers』を参照)。WebSphere JMS メッセージング・クライアント・コンポーネントを、WebSphere Application Server で実行される SIBUS 経由で JMS と対話するように構成することもできます (WebSphere Application Server ベータ製品資料内の『Enabling interoperability between the Liberty profile and WebSphere Application Server full profile』を参照)。
- WebSphere メッセージング・セキュリティー (wasJmsSecurity-1.0)。この JMS セキュリティー機能は、組み込み WebSphere メッセージング・クライアントおよびサーバー・コンポーネントにセキュリティー・サポートを提供します。この JMS セキュリティー機能を cicsts:security-1.0 機能と併用して、組み込み WebSphere

メッセージング・サーバーに対する認証要求時に、セキュリティー・レジストリーからどのユーザーを選択して接続ファクトリーで使用するかを指定することができます。許可について詳しくは、WebSphere Application Server ベータ製品資料内の『Authorizing users to connect to the messaging engine』を参照してください。

## Java Transaction API (JTA)

Java Transaction API (JTA) を使用すると、複数のリソース・マネージャーに対するトランザクション更新を調整することができます。

Java Transaction API (JTA) を使用すると、CICS リソースおよび他のサード・パーティー製リソース・マネージャー (Liberty JVM サーバー内のタイプ 4 データベース・ドライバー接続など) に対するトランザクション更新を調整できます。このシナリオでは Liberty トランザクション・マネージャーがトランザクション・コーディネーターです。 CICS 作業単位は、CICS システム外部でトランザクションが開始したかのように従属しています。

CICS データ・ソースを使用したローカル DB2 データベースへのタイプ 2 ドライバー接続では、CICS DB2 接続を使用してアクセスします。 JTA を使用して、他の CICS リソースに対する更新に合わせて調整する必要はありません。

JTA の中で、複数のリソース・マネージャーに対する更新をカプセル化して調整するための UserTransaction オブジェクトを作成します。以下のコード・フラグメントは、UserTransaction を作成して使用する方法を示しています。

```
InitialContext ctx = new InitialContext();
UserTransaction tran =
        (UserTransaction)ctx.lookup("java:comp/UserTransaction");

DataSource ds = (DataSource)ctx.lookup("jdbc/SomeDB");
Connection con = ds.getConnection();

// Start the User Transaction
tran.begin();

// Perform updates to CICS resources via JCICS API and
// to database resources via JDBC/SQLJ APIs

if (allOk) {
  // Commit updates on both systems
  tran.commit();
} else {
  // Backout updates on both systems
  tran.rollback();
}
```

なお、CICS 作業単位の場合とは異なり、`begin()` メソッドを使って明示的に UserTransaction を開始する必要があることに注意してください。 `begin()` を呼び出すと、CICS は UserTransaction の開始前に行われたすべての更新をコミットします。 UserTransaction は `commit()` または `rollback()` のいずれかのメソッド呼び出しにより終了するか、Web アプリケーション終了時に Web コンテナーによって終了されます。 UserTransaction がアクティブ状態である間、プログラムは JCICS `Task commit()` メソッドや `rollback()` メソッドを呼び出すことができません。

JCICS の Task.commit() および Task.rollback() メソッドは JTA トランザクション・コンテキスト内では無効になります。どちらを試行した場合も、InvalidRequestException がスローされます。

Liberty のデフォルト動作では、未確定 JTA トランザクションのリカバリーを試みる前に最初の UserTransaction が作成されるのを待つことに注意してください。しかし、CICS は、Liberty JVM サーバーの初期化が完了するとすぐにトランザクション・リカバリーを開始します。JVM サーバーが使用不可としてインストールされている場合、リカバリーは使用可能に設定されたときに実行されます。

『EJB での JTA トランザクションの使用』
Liberty の Enterprise JavaBeans (EJB) で JTA トランザクションを使用する方法。

## EJB での JTA トランザクションの使用

Liberty の Enterprise JavaBeans (EJB) で JTA トランザクションを使用する方法。

### このタスクについて

EJB は、Liberty JVM サーバーによって管理される Java オブジェクトで、Java アプリケーションのモジュラー・アーキテクチャーを可能にします。 Liberty JVM サーバーは、EJB フル仕様の一部である軽量な EJB Lite 3.1 をサポートしており、セッション、ライフサイクル (インターセプター・メソッドによる)、トランザクション、およびセキュリティー管理を提供します。 EJB は、エンタープライズ・アプリケーション・プロジェクトで作成されたエンタープライズ・アーカイブ (EAR) ファイルを使用して Liberty サーバーにデプロイされます。エンタープライズ・アプリケーション・プロジェクトには、EJB プロジェクトと Web プロジェクトの両方を含めることができます。

EJB Lite は、`ejbLite-3.1` 機能を `server.xml` 構成ファイルに追加することによって有効になります。EJB はコンテナーにデプロイされます。このコンテナーはバックグラウンドで動作し、セッション管理、トランザクション、およびセキュリティーなどの面で一定の規準に準拠するようにします。

EJB は、コンテナー管理と Bean 管理の 2 種類のトランザクション管理をサポートします。コンテナー管理トランザクションは、Bean メソッドの呼び出しにトランザクション・コンテキストを提供し、 Java アノテーションまたはデプロイメント記述子ファイル `ejb-jar.xml` を使用して定義されます。 Bean 管理トランザクションは、Java Transaction API (JTA) を使用して直接制御されます。どちらの管理トランザクションにおいても、CICS UOW は JTAトランザクションの結果に従属します。コンテナー管理トランザクションには、次の 6 つの異なるトランザクション属性を指定することができます。

- Mandatory
- Required
- RequiresNew
- Supports
- NotSupported
- Never

JTA トランザクションは、J2EE 仕様に定義されている分散作業単位です。メソッドのトランザクション属性の設定によって、メソッドを実行する CICS タスクが独自の作業単位として実行されるのか、それともより広範囲の分散 JTA トランザクションの一部として実行されるのかが決まります。トランザクション属性および呼び出し側のアプリケーションが既に JTA トランザクション・コンテキストを持っているかどうかに応じて、呼び出された EJBメソッドのトランザクション・コンテキストがどのようになるかを、下の表に示します。

*表 12. EJB トランザクション・サポート*

| トランザクション属性 | JTA トランザクションが存在しない | JTA トランザクションが既に存在する |
|---|---|---|
| Mandatory | `javax.ejb.EJBTransactionRequiredException` をスローする | 既存の JTA トランザクションを継承する |
| Required | EJB コンテナーは新しい JTA トランザクションを作成する | 既存の JTA トランザクションを継承する |
| RequiresNew | EJB コンテナーは新しい JTA トランザクションを作成する | `javax.ejb.EJBException` をスローする |
| Supported | JTA トランザクションなしで続ける | 既存の JTA トランザクションを継承する |
| NotSupported | JTA トランザクションなしで続ける | JTA トランザクションを停止するが、CICS UOW は停止しない |
| Never | JTA トランザクションなしで続ける | `javax.ejb.EJBException` をスローする |

**注:**

- トランザクション属性「NotSupported」は、CICS Liberty JVM サーバーではサポートできません。「NotSupported」のマークが付けられているメソッドを呼び出すと、JTA トランザクションは停止されますが、CICS UOW は停止されません。このメソッドの呼び出し中に行われた CICS リソースへの変更は、元に戻すことができます。
- トランザクション属性「RequiresNew」は、CICS Liberty JVM サーバーでサポートされていますが、CICS UOW をネストできないという制限があります。既に JTA トランザクションが開始された状態で「RequiresNew」のマークが付けられているメソッドを呼び出そうとすると、例外がスローされます。

WLP の標準的な制限も適用されます。ランタイム環境での既知の制約事項を参照してください。

## JCA プログラミング・インターフェースを使用したアプリケーションの開発

JCA は、CICS などのエンタープライズ情報システムを JEE プラットフォームに接続します。

JCA は、JEE アプリケーション・サーバーによって提供されるセキュリティー資格情報管理、接続プーリング、およびトランザクション管理のサービスの品質をサポートします。JCA を使用すると、これらのサービスの品質がアプリケーションではなく JEE アプリケーション・サーバーによって管理されるようになります。このことは、プログラマーがビジネス・コードの作成に専念でき、サービスの品質を意識

する必要がないことを意味します。サービス品質の提供および構成のガイダンスについては、ご使用の JEE アプリケーション・サーバーの資料を参照してください。JCA は、共通クライアント・インターフェース (CCI) と呼ばれるプログラミング・インターフェースを定義します。このインターフェースにわずかな変更を加えるだけで、どのエンタープライズ情報システムとも通信できます。

## プログラミング・インターフェース・モデル

CCI を使用するアプリケーションは、あらゆるエンタープライズ情報システムのための共通の構造を持ちます。 JCA は、EIS への接続を表す接続および接続ファクトリーを定義します。これらの接続オブジェクトによって、JEE アプリケーション・サーバーは、リソース・アダプターのセキュリティー、トランザクション・コンテキスト、および接続プールを管理することができます。アプリケーションを開始するには、まず接続ファクトリーにアクセスする必要があります。そこから接続を取得することができます。この接続のプロパティーは、ConnectionSpec オブジェクトによりオーバーライドすることができます。接続が取得された後で、特定の要求を出すために接続から対話を作成することができます。接続の場合と同様に、対話も InteractionSpec クラスによって設定されるカスタム・プロパティーを持つことができます。対話を行うためには、execute() メソッドを呼び出し、record オブジェクトを使用してデータを保持します。以下に例を示します。

```
/* Obtain a ConnectionFactory cf */
Connection c = cf.getConnection(ConnectionSpec)
Interaction i = c.createInteraction()
InteractionSpec is = newInteractionSpec();
i.execute(spec, input, output)
```

JEE アプリケーション・サーバーを使用する場合、接続ファクトリーは、サーバーの管理インターフェースを使用して構成することで作成します。Liberty サーバーでは、これは `server.xml` 構成で定義します。接続ファクトリーを作成したら、エンタープライズ・アプリケーションは、JNDI (Java Naming Directory Interface) でそれを検索して、アクセスできます。このタイプの環境は管理対象環境と呼ばれ、JEE アプリケーション・サーバーが接続のサービス品質を管理することを可能にします。管理対象環境について詳しくは、JEE アプリケーション・サーバーの資料を参照してください。

## レコード・オブジェクト

レコード・オブジェクトは、EIS との間で受け渡しされるデータを表わすために使用されます。これらのレコードを生成するには、アプリケーション開発ツールを使用することをお勧めします。

## リソース・アダプターのサンプル

リソース・アダプターの基本的なサンプルをインストールし、そのリソース・アダプターが提供するリソースのインスタンスを構成することができます。WebSphere Application Server ベータ製品資料内の『Configuring and deploying a basic JCA Resource Adapter』を参照してください。

113 ページの『Common Client Interface』
JEE Connector Architecture の共通クライアント・インターフェース (Common Client Interface: CCI) は、開発者用の標準インターフェースを提供します。開発

者は、汎用プログラミング・スタイルを使用し、それぞれのエンタープライズ情報システム (EIS) に固有のリソース・アダプターを介して、任意の数の EIS と通信することができます。

『JCA ローカル ECI リソース・アダプターの使用』
JCA ローカル ECI リソース・アダプターは、CICS プログラムの起動用に最適化されたローカル ECI リソース・アダプターを提供します。

## Common Client Interface

JEE Connector Architecture の共通クライアント・インターフェース (Common Client Interface: CCI) は、開発者用の標準インターフェースを提供します。開発者は、汎用プログラミング・スタイルを使用し、それぞれのエンタープライズ情報システム (EIS) に固有のリソース・アダプターを介して、任意の数の EIS と通信することができます。

CCI は、Java Database Connectivity (JDBC) で使用されるクライアント・インターフェースをモデルとして非常によく似た形に設計されており、Connection (接続) と Interaction (対話) についての考え方は JDBC と同様です。

『汎用 CCI クラス』
汎用 CCI クラスでは、JEE アプリケーションが CICS などのエンタープライズ情報システムとの間でデータを送受信できる環境を定義します。

**汎用 CCI クラス:**

汎用 CCI クラスでは、JEE アプリケーションが CICS などのエンタープライズ情報システムとの間でデータを送受信できる環境を定義します。

JEE コンポーネントを開発しているときは、以下の作業を実行する必要があります。

1. ConnectionFactory オブジェクトを使用して、接続オブジェクトを作成する。
2. 接続オブジェクトを使用して、対話オブジェクトを作成する。
3. Interaction オブジェクトを使用して、エンタープライズ情報システムに対するコマンドを実行する。
4. 対話と接続を閉じる。

以下の例は、エンタープライズ情報システム上でコマンドを実行するために使用される JEE CCI インターフェースを示しています。

```
ConnectionFactory cf = <Lookup from JNDI namespace>
Connection conn = cf.getConnection();
Interaction interaction = conn.createInteraction();
interaction.execute(<Input output data>);
interaction.close();
conn.close();
```

## JCA ローカル ECI リソース・アダプターの使用

JCA ローカル ECI リソース・アダプターは、CICS プログラムの起動用に最適化されたローカル ECI リソース・アダプターを提供します。

JCA ローカル ECI リソース・アダプターを使用することで、CICS プログラムに接続し、COMMAREA またはチャネルとコンテナーのいずれかでデータを渡します。このリソース・アダプターは、CICS JCA ローカル ECI Liberty フィーチャーに備わっています。

**注:** JCA ローカル ECI リソース・アダプターと CICS Transaction Gateway ECI リソース・アダプターを同じ Liberty JVM サーバー内で使用することはできません。

表 1 に、CICS 用語に対応する JCA オブジェクトを示します。

*表 13. CICS 用語と対応する JCA オブジェクト*

| CICS 用語 | JCA オブジェクト: プロパティー |
|---|---|
| 異常終了コード | CICSTxnAbendException |
| COMMAREA | Record |
| Channel | ECIChannelRecord。 118 ページの『JCA ローカル ECI リソース・アダプターをチャネルおよびコンテナーと共に使用する』を参照してください。 |
| データ・タイプ BIT のコンテナー | byte[] |
| データ・タイプ CHAR のコンテナー | String |
| プログラム名 | ECIInteractionSpec:FunctionName |
| トランザクション | ECIInteractionSpec:TPNName |

詳細については、 77 ページの『JCA ローカル ECI サポート』を参照してください。

115 ページの『JCA ローカル ECI リソース・アダプターのトレースの有効化』
JCA ローカル ECI リソース・アダプターのトレースの仕組みを詳しく説明します。リソース・アダプターを使用するアプリケーションの問題を解決する際には、トレースを有効にすると便利な場合があります。

115 ページの『ローカル ECI リソース・アダプターを使用した CICS 内のプログラムへのリンク』
JCA ローカル ECI リソース・アダプターを使用して CICS 内のプログラムを実行するには、ECIInteraction クラスの execute() メソッドを使用します。

119 ページの『JCA ローカル ECI リソース・アダプターの構成』
JCA 仕様の定義にしたがって、接続ファクトリーを使用して JCA ローカル ECI リソース・アダプターを構成できます。

120 ページの『JCA による作業単位の管理』
CICS ローカル ECI リソース・アダプターを使用する際には、CICS Liberty JVM サーバーによってトランザクション管理機能が提供されます。

120 ページの『JCA ローカル ECI リソース・アダプターの制限』
CICS TG ECI リソース・アダプターで利用できる一部の API 呼び出しは、CICS TS JCA ローカル ECI リソース・アダプターによってサポートされていません。

**JCA ローカル ECI リソース・アダプターのトレースの有効化:**

JCA ローカル ECI リソース・アダプターのトレースの仕組みを詳しく説明します。リソース・アダプターを使用するアプリケーションの問題を解決する際には、トレースを有効にすると便利な場合があります。

- JCA ローカル ECI リソース・アダプターのトレースは、SJ ドメイン・トレース・レベル 4 (SJ = 4 または SJ = ALL) によって有効になります。
- リソース・アダプターからのトレースは、コンポーネント ID `com.ibm.cics.wlp.jca.local.eci.adapter` を使って zFS で JVM サーバー・トレース出力に含まれます。

**ローカル ECI リソース・アダプターを使用した CICS 内のプログラムへのリンク:**

JCA ローカル ECI リソース・アダプターを使用して CICS 内のプログラムを実行するには、ECIInteraction クラスの execute() メソッドを使用します。

**このタスクについて**

このタスクでは、JCA ローカル ECI リソース・アダプターを使用して CICS プログラムを実行する方法をアプリケーション開発者に紹介します。

**手順**

1. JNDI を使用して、`eis/defaultCICSConnectionFactory` という名前の ConnectionFactory オブジェクトを検索します。
2. ConnectionFactory から Connection オブジェクトを取得します。
3. Connection から Interaction オブジェクトを取得します。
4. ECIInteractionSpec オブジェクトを新規作成します。
5. ECIInteractionSpec で set メソッドを使用して、プログラム名や COMMAREA の長さなど、実行に関わるプロパティーを設定します。
6. 入力データを入れるレコード・オブジェクトを作成して (COMMAREA/チャネルのトピックを参照)、データを取り込みます。
7. 出力データを入れるレコード・オブジェクトを作成します。
8. Interaction で execute メソッドを呼び出して、ECIInteractionSpec と 2 つの Record オブジェクトを渡します。
9. 出力したレコードからデータを読み取ります。

```
package com.ibm.cics.server.examples.wlp;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import javax.annotation.Resource;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;
import javax.resource.cci.Record;
import javax.resource.cci.Streamable;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.connector2.cics.ECIInteractionSpec;

/**
 * Servlet implementation class JCAServlet
 */
@WebServlet("/JCAServlet")
public class JCAServlet extends HttpServlet
{
 private static final long serialVersionUID = 4283052088313275418L;

 // 1. Use JNDI to look up the connection factory
 @Resource(lookup = "eis/defaultCICSConnectionFactory")
 private ConnectionFactory cf;

 protected void doGet(HttpServletRequest request, HttpServletResponse response)
                      throws ServletException, IOException
 {
  try
  {
   // 2. Get the connection object from the connection factory
   Connection conn = cf.getConnection();

   // 3. Get an interaction object from the connection
   Interaction interaction = conn.createInteraction();

   // 4. Create a new ECIInteractionSpec
   ECIInteractionSpec is = new ECIInteractionSpec();

   // 5. Use the set methods on ECIInteractionSpec
      //    to set the properties of execution.
   //    Change these properties to suit the target program
   is.setCommareaLength(20);
   is.setFunctionName("PROGNAME");
   is.setInteractionVerb(ECIInteractionSpec.SYNC_SEND_RECEIVE);

   // 6. Create a record object to contain the input data and populate data
   //    Change the contents to suit the data required by the program
   RecordImpl in = new RecordImpl();
   byte[] commarea = "COMMAREA contents".getBytes();
   ByteArrayInputStream inStream = new ByteArrayInputStream(commarea);
   in.read(inStream);

   // 7. Create a record object to contain the output data
   RecordImpl out = new RecordImpl();

   // 8. Call the execute method on the interaction
   interaction.execute(is, in, out);

   // 9. Read the data from the output record
   ByteArrayOutputStream outStream = new ByteArrayOutputStream();
   out.write(outStream);
   commarea = outStream.toByteArray();
  }
  catch (Exception e)
  {
   // Handle any exceptions by wrapping them into an IOException
   throw new IOException(e);
  }
 }

 // A simple class which extends Record and Streamable representing a commarea.
 public class RecordImpl implements Streamable, Record
 {
```

```
private static final long serialVersionUID = -947604396867020977L;

private String contents = new String("");

@Override
public void read(InputStream is)
{
 try
 {
  int total = is.available();
  byte[] bytes = null;
  if (total > 0)
  {
   bytes = new byte[total];
   is.read(bytes);
  }
  // Convert the bytes to a string.
  contents = new String(bytes);
 }
 catch (Exception e)
 {
  // Log the exception
  e.printStackTrace();
 }
}

@Override
public void write(OutputStream os)
{
 try
 {
  // Output the string as bytes
  os.write(contents.getBytes());
 }
 catch (Exception e)
 {
  // Log the exception
  e.printStackTrace();
 }
}

@Override
public String getRecordName()
{
 // Required by Record, unused in this sample
 return "";
}
@Override
public void setRecordName(String newName)
{
 // Required by Record, unused in this sample
}
@Override
public void setRecordShortDescription(String newDesc)
{
 // Required by Record, unused in this sample
}
@Override
public String getRecordShortDescription()
{
 // Required by Record, unused in this sample
 return "";
}
@Override
public Object clone() throws CloneNotSupportedException
{
 // Required by Record, unused in this sample
```

```
      return super.clone();
    }
   }
  }
```

**タスクの結果**

ECI リソース・アダプターを使用した CICS 内のプログラムへのリンクの作成が成功しました。

『JCA ローカル ECI リソース・アダプターをチャネルおよびコンテナーと共に使用する』
JCA ローカル ECI リソース・アダプターと共にチャネルおよびコンテナーを使用するには、入出力レコードが ECIChannelRecord のインスタンスである必要があります。

119 ページの『JCA ローカル ECI リソース・アダプターを COMMAREA と共に使用する』
JCA ローカル ECI リソース・アダプターと共に COMMAREA を使用するには、入出力レコードが、`javax.resource.cci.Record` および `javax.resource.cci.Streamable` を実装するクラスのインスタンスである必要があります。

### *JCA ローカル ECI リソース・アダプターをチャネルおよびコンテナーと共に使用する:*

JCA ローカル ECI リソース・アダプターと共にチャネルおよびコンテナーを使用するには、入出力レコードが ECIChannelRecord のインスタンスである必要があります。

ECIChannelRecord が ECIInteraction の execute() メソッドに渡されると、そのメソッドは ECIChannelRecord そのものを使用してチャネルを作成し、ECIChannelRecord 内の項目をコンテナーに変換してから、それを CICS に渡します。

この例は、ECI ChannelRecord で put() メソッドおよび get() メソッドを使用して JCA ローカル・リソース・アダプター用の入出力レコードを作成する方法を示しています。

```
ECIChannelRecord in = new ECIChannelRecord("CHANNELNAME");
byte[] bitData = "Container with BIT data".getBytes();
String charData = "Container with CHAR data";
in.put("BITCONTAINER", bitData);
in.put("CHARCONTAINER", charData);
ECIChannelRecord out = new ECIChannelRecord("CHANNELNAME");

interaction.execute(is, in, out);

bitData = (byte[]) out.get("BITCONTAINER");
charData = (String) out.get("CHARCONTAINER");
```

入力の種類に応じて、BIT コンテナーおよび CHAR コンテナーが次のように作成されます。

- 入力データが byte[] 型である場合、または Streamable インターフェースを実装するオブジェクトである場合には、BIT コンテナーが作成されます。コード・ページ変換は行われません。

- 入力データが String 型である場合には、CHAR コンテナーが作成されます。ストリング・データは Unicode でエンコードされ、コンテナーのエンコード方式に変換されます。 **EXEC CICS GET CONTAINER** によってこのコンテナーから読み取られるデータは、『アプリケーションの開発』の『コンテナーを使用したコード・ページ変換』に従って変換されます。

ECIChannelRecord を作成するときに、名前は有効な CICS チャネル名でなければなりません。作成された後、getRecordName() メソッドがチャネルの名前を取得します。 ECIChannelRecord にコンテナーを追加するとき、コンテナー名は有効な CICS コンテナー名でなければなりません。作成された後、KeySet() メソッドがすべてのコンテナーの名前を取得します。

***JCA ローカル ECI リソース・アダプターを COMMAREA と共に使用する:***

JCA ローカル ECI リソース・アダプターと共に COMMAREA を使用するには、入出力レコードが、`javax.resource.cci.Record` および `javax.resource.cci.Streamable` を実装するクラスのインスタンスである必要があります。

この例は、Streamable インターフェースで read() メソッドおよび write() メソッドを使用してローカル ECI リソース・アダプター用の入出力レコードを作成する方法を示しています。

```
RecordImpl in = new RecordImpl();
byte[] commarea = "COMMAREA contents".getBytes();
ByteArrayInputStream inStream = new ByteArrayInputStream(commarea);
in.read(inStream);
RecordImpl out = new RecordImpl();

interaction.execute(is, in, out);

ByteArrayOutputStream outStream = new ByteArrayOutputStream();
out.write(outStream);
commarea = outStream.toByteArray();
```

出力レコードからバイト配列を取得するには、**java.io.ByteArrayOutputStream** オブジェクトを使って Streamable インターフェースで write メソッドを使用します。**ByteArrayOutputStream** 上の toByteArray() メソッドは、COMMAREA からの出力データをバイト配列形式で提供します。

特定の JEE コンポーネントの機能を増強するために、コンストラクターを使用してレコード内容を設定できるようにする Record インターフェースの実装を作成することもできます。この方法により、例で使用した **java.io.ByteArrayInputStream** の使用を避けることができます。

Rational Application Developer に備わっている J2C ツールを使用すると、COBOL コピーブックなどの特定のネイティブ言語構造から Record インターフェースの実装を作成でき、Java と Java 以外のデータ型の間のデータ・マーシャルも標準でサポートされます。

**JCA ローカル ECI リソース・アダプターの構成:**

JCA 仕様の定義にしたがって、接続ファクトリーを使用して JCA ローカル ECI リソース・アダプターを構成できます。

JCA ローカル ECI の使用を開始するには、server.xml の featureManager エレメントにフィーチャー cicsts:jcaLocalEci-1.0 を追加します。

```
<featureManager>
    <feature>cicsts:jcaLocalEci-1.0</feature>
</featureManager>
```

JCA ローカル ECI は、JNDI 名 eis/defaultCICSConnectionFactory にバインドされたデフォルトの接続ファクトリー defaultCICSConnectionFactory を提供します。オプションで、異なる JNDI 名が必要な場合は、次のようなプロパティー・サブエレメントを使用して追加の接続ファクトリーを構成することもできます。

```
<connectionFactory id="localEci" jndiName="eis/ECI">
    <properties.com.ibm.cics.wlp.jca.local.eci/>
</connectionFactory>
```

**JCA による作業単位の管理:**

CICS ローカル ECI リソース・アダプターを使用する際には、CICS Liberty JVM サーバーによってトランザクション管理機能が提供されます。

CICS ローカル ECI リソース・アダプターを使用した他の CICS プログラムの呼び出しは、CICS の作業単位 (UOW) 管理に統合されています。これにより、syncpoint コマンドまたは JTA トランザクションを介して UOW を制御することができます。

リモート CICS 領域でのプログラムの呼び出しを実行すると、ミラー・トランザクションを使用した DPL 呼び出しになります。 Java トランザクション・コンテキストが使用されている場合、このミラー・タスク UOW は呼び出し側 UOW によって調整されます。この場合、呼び出される側のプログラムは DPL コマンド・サブセットに制限されるため、syncpoint 呼び出しを発行できません。呼び出し側プログラムに JTA トランザクション・コンテキストが含まれない場合は、SYNCONRETURN オプションを使用してミラー・タスク UOW が呼び出されます。このシナリオでは、呼び出されるプログラムの UOW が呼び出し側プログラムによって調整されないため、呼び出されるプログラムは syncpoint コマンドを発行できます。

詳しくは、、 74 ページの『作業単位 (UOW) サービス』、および 109 ページの『Java Transaction API (JTA)』を参照してください。

**JCA ローカル ECI リソース・アダプターの制限:**

CICS TG ECI リソース・アダプターで利用できる一部の API 呼び出しは、CICS TS JCA ローカル ECI リソース・アダプターによってサポートされていません。

**制限されたメソッド**

以下の API 呼び出しは、CICS TS JCA ローカル ECI リソース・アダプターによってサポートされていません。

- ECIInteractionSpec クラスの setExecuteTimeout()、getExecuteTimeout()、setReplyLength()、getReplyLength()、setTranName()、getTranName() メソッド

- CICSConnectionSpec クラスの setPassword()、getPassword()、setUserid()、getUserid ()、addPropertyChangeListener()、 removePropertyChangeListener() 、firePropertyChange メソッド
- ECIConnection クラスの getLocalTransaction() メソッド
- CICSUserInputException
- コンストラクター ECIConnectionSpec(String username, String password)。ECIConnectionSpec() が代わりに追加されました。
- コンストラクター ECIInteractionSpec(int verb, int timeout, String prog, int commLen, int repLen)。ECIInteractionSpec(int verb, String prog, int commLen) が代わりに追加されました。
- ECIChannelRecord クラスの values() メソッド。

これらの呼び出しは、Web アプリケーションの開発に CICS Explorer SDK を使用している場合はサポートされません。既存の ECI JCA アプリケーションを Liberty JVM サーバーに移植可能にするために、これらのメソッドは引き続き機能しますが、トランザクション、タイムアウト、応答の長さ、およびトランザクション名の設定は影響を及ぼしません。ECIInteractionSpec.setTPNName() でトランザクション ID を設定すると、リモート・プログラム (DPL) にリンクするときに、指定されたトランザクションのみが使用されます。ローカル・プログラムへのリンクは、引き続き現在のトランザクションを使用します。

**非管理対象環境**

JCA ローカル ECI は、(server.xml の構成により作成された) 管理接続ファクトリーのみをサポートします。 ManagedConnectionFactory のインスタンスを使用して作成された非管理対象接続はサポートされません。 CICS TG ECI リソース・アダプターと CICS TS JCA ローカル ECI リソース・アダプターでは、例外処理が少し異なる可能性があります。CICS のエラーは、JCICS API の場合と同様に、ECI ローカル・リソース・アダプターに CicsException として伝搬されます。リソース・アダプターは、これらの例外を ResourceException でラップします。CICS の障害を特定できるように、CicsException が例外の原因として設定され、java.lang.Throwable の getCause() メソッドを使用してアクセスできます。

**非同期呼び出し**

非同期呼び出しは、JCA ローカル ECI リソース・アダプターでは完全な非同期ではありません。SYNC_SEND 対話 verb を使用した呼び出しは、プログラムが完了するまでブロックし、その後 SYNC_RECEIVE 対話 verb を使用して、同じ ECIInteraction を使い、後続の呼び出しによって結果を収集できます。

**サポートされない CICS TG の機能**

次の CICS Transaction Gateway の機能は、CICS TS ローカル ECI リソース・アダプターではサポートされません。

- CICS TG へのリモート接続
- ID 伝搬
- クロス・コンポーネント・トレース (XCT)
- ユーザー出口のモニタリング要求

- リソース・アダプターからのトレースは CICS TS によって制御され、CICSLogTraceLevels の使用はサポートされていません。

  『JCA ECI アプリケーションを Liberty JVM サーバーに移植する』
  JCA ローカル ECI リソース・アダプター・サポートを使用することで、JCA アプリケーションを簡単に Liberty JVM サーバーに移植できます。

***JCA ECI アプリケーションを Liberty JVM サーバーに移植する:***

JCA ローカル ECI リソース・アダプター・サポートを使用することで、JCA アプリケーションを簡単に Liberty JVM サーバーに移植できます。

**移植**

CICS Transaction Gateway ECI リソース・アダプターを使用する既存の JCA アプリケーションを、スタンドアロン JEE アプリケーション・サーバーから CICS Liberty JVM サーバーに移植するには、以下の手順を使用します。

1. フィーチャー `cicsts:jcalocalEci-1.0` を `server.xml` に追加します。
2. アプリケーションを CICS にデプロイします。Web アプリケーションを CICS バンドルで Liberty JVM サーバーにデプロイするを参照してください。
3. 接続ファクトリー・アプリケーションの JNDI 名を `eis/defaultCICSConnectionFactory` に変更するか、または、必要な JNDI 名を指定して追加の接続ファクトリーを `server.xml` に構成します。

ECI リソース・アダプターの制限されたフィーチャー (linkout) を使用するアプリケーションの場合は、アプリケーションのコードを変更してそれらのサポートされていないフィーチャーを削除する必要があります。

# サーブレットおよび JSP アプリケーションの開発

CICS アプリケーションに対する最新のインターフェースを提供するために、Web アプリケーション・テクノロジーを使用するプレゼンテーション層を開発できます。Eclipse ベースの Web 開発ツールは、これらのアプリケーションを作成するための開発プラットフォームを提供します。CICS Explorer SDK は、CICS での実行用にこれらをビルドし、パッケージ化し、デプロイするためのサポートを提供します。

## このタスクについて

Liberty サーバーにデプロイ可能な Web アプリケーション・プロジェクトには、動的 Web プロジェクト (WAR)、OSGi アプリケーション・プロジェクト (EBA)、およびエンタープライズ・アプリケーション・プロジェクト (EAR) という 3 つのタイプがあります。

WAR には、イメージや HTML ファイルのような静的リソースの他に、サーブレット、JSP ファイル、フィルター、関連メタデータなどの動的 Java EE リソースを含めることができます。

EBA は、WAB と OSGi バンドルを含めることができる Java アーカイブ・ファイルです。WAB は Web 使用可能 OSGi バンドルで、イメージや HTML ファイルのような静的リソースの他に、JSP サーブレットとファイル、フィルター、関連メタデータなどが含まれます。

EAR は、EBA が WAB と OSGi バンドルを編成するのと同じ方法で、WAR と EJB モジュールを 1 つのコンテナーに編成する方法です。

『動的 Web プロジェクトの作成』
Java アプリケーション用の Web プレゼンテーション層を開発する場合、動的 Web プロジェクトを作成できます。

124 ページの『OSGi アプリケーション・プロジェクトの作成』
OSGi アプリケーション・プロジェクト (EBA) とは、一連のバンドルをまとめたものです。アプリケーションは、さまざまな種類の OSGi バンドルで構成できます。

**関連タスク**:

topics/gettingstarted_liberty.dita
CICS Explorer SDK には、CICS の Liberty JVM サーバーで実行できるサーブレットと JSP ページの開発を開始する際に役立つサンプルが含まれています。

126 ページの『Liberty JVM サーバー内で実行するよう Java Web アプリケーションをマイグレーションする』
ネットワークを介して CICS にアクセスする Liberty プロファイル・インスタンスの中で実行する Java Web アプリケーションがある場合、そのアプリケーションを Liberty JVM サーバーの中で実行すると、パフォーマンスを最適化できます。

## 動的 Web プロジェクトの作成

Java アプリケーション用の Web プレゼンテーション層を開発する場合、動的 Web プロジェクトを作成できます。

### 始める前に

Eclipse IDE に Web 開発ツールがインストールされていることを確認します。詳しくは、 104 ページの『開発環境のセットアップ』を参照してください。

Liberty プロファイルによって追加され、CICS TS v5.1 に適用された APAR によって導入された制限により、WAR ファイルでデプロイされたサーブレットから OSGi バンドルにアクセスできません。この制限には、CICS バンドルによって直接インストールされた OSGi バンドルへのアクセスも含まれます。この制限を回避するには、EBA (OSGi アプリケーション・プロジェクト) としてアプリケーションをデプロイする必要があります。EBA は、Web と OSGi コンポーネントが対話できるコンテナーです。

### このタスクについて

CICS Explorer SDK ヘルプには、Web アプリケーションを開発してパッケージするための各ステップの実行方法についての詳細が説明されています。

### 手順

1. アプリケーション用に動的 Web プロジェクトを作成します。ビルド・パスを更新して Liberty のライブラリーを追加する必要があります。

a. 動的 Web プロジェクトを右クリックし、「**ビルド・パス**」 > 「**ビルド・パスの構成**」をクリックします。プロジェクトの「プロパティー」ダイアログが開きます。
b. 「Java のビルド・パス」で、「**ライブラリー**」タブをクリックします。
c. 「**ライブラリーの追加**」をクリックし、Liberty JVM サーバーのライブラリーを選択します。
d. 「**次へ**」をクリックして CICS のバージョンを選択し、「**終了**」をクリックしてライブラリーの追加を完了します。
e. 「**OK**」をクリックして、変更内容を保管します。

2. Web アプリケーションを開発します。JCICS API を使用して CICS サービスにアクセスし、JDBC を使用して DB2 にアクセスし、JMS を使用して WebSphere MQ にアクセスできます。 CICS Explorer SDK には、JCICS と JDBC を使用した Web コンポーネントの例が含まれています。
3. オプション: CICS セキュリティーでアプリケーションを保護する場合は、動的 Web プロジェクトに `web.xml` ファイルを作成して、CICS セキュリティー制約を組み込みます。CICS Explorer SDK にはこのファイルのテンプレートが組み込まれており、そこには CICS に関する適切な情報が含まれています。詳しくは、237 ページの『Liberty JVM サーバーでのユーザー認証』を参照してください。
4. アプリケーションをパッケージするために、1 つ以上の CICS バンドル・プロジェクトを作成します。CICS リソースの定義とインポートを追加します。各 CICS バンドルには、ID とバージョンが含まれており、変更を細かく管理できます。
5. オプション: URI からのインバウンド Web 要求をマップして特定のトランザクションの下で実行する場合は、URIMAP および TRANSACTION リソースを CICS バンドルに追加します。これらのリソースを定義しない場合、CJSA という名前の提供されたトランザクションの下ですべての処理が実行されます。これらのリソースは、動的にインストールされ、CICS の中のバンドルの一部として管理されます。

### タスクの結果

開発環境のセットアップ、動的 Web プロジェクトからの Web アプリケーションの作成、そのアプリケーションをデプロイするためのパッケージ化が終わりました。

### 次のタスク

アプリケーションをデプロイする用意ができたら、CICS バンドル・プロジェクトを zFS にエクスポートします。参照されたプロジェクトは、ビルドされ、zFS への転送に組み込まれます。または、Liberty デプロイメント・モデルに従って、アプリケーションを WAR としてエクスポートし、稼働中の Liberty JVM サーバーの dropins ディレクトリーにそれをデプロイすることもできます。

## OSGi アプリケーション・プロジェクトの作成

OSGi アプリケーション・プロジェクト (EBA) とは、一連のバンドルをまとめたものです。アプリケーションは、さまざまな種類の OSGi バンドルで構成できます。

## 始める前に

WebSphere Application Server Developer Tools が Eclipse IDE にインストールされていることを確認します。詳しくは、104 ページの『開発環境のセットアップ』を参照してください。

## このタスクについて

CICS Explorer SDK ヘルプには、Web アプリケーションを開発してパッケージするための各ステップの実行方法についての詳細が説明されています。

## 手順

1. 「**CICS TS V5.3 with Liberty and PHP**」テンプレートを使用して、Java 開発用のターゲット・プラットフォームをセットアップします。ターゲットが、Eclipse の現行インストールよりも新しいバージョンであるという警告が表示されることがありますが、この警告メッセージは無視してかまいません。
2. アプリケーション用に OSGi バンドル・プロジェクトを作成します。ターゲット・プラットフォームでパッケージが実質的に使用可能になるため、適切な Import ステートメントをバンドル・マニフェストに含める必要があります。Web 使用可能 OSGi バンドル・プロジェクトは、動的 Web プロジェクトに相当するバンドルです。 Web 使用可能 OSGi バンドル・プロジェクトを使用すると、OSGi アプリケーション・プロジェクト (エンタープライズ・バンドル・アーカイブ、つまり EBA ファイル) 内にアプリケーションをデプロイできます。Web 使用可能 OSGi バンドル・プロジェクト (WAB ファイル) と Web 使用可能でない OSGi バンドル・プロジェクトを OSGi アプリケーション・プロジェクト内で混合することができます。通常、Web 使用可能 OSGi バンドル・プロジェクトはアプリケーションのフロントエンドを実装し、(ビジネス・ロジックを含んでいる) 非 Web OSGi バンドルと対話します。
3. Web アプリケーションを開発します。JCICS API を使用して、CICS サービスにアクセスし、DB2 に接続できます。CICS Explorer SDK には、JCICS と DB2 を使用する、Web コンポーネントと OSGi バンドルの例が含まれます。ビジネスとプレゼンテーションのロジックを分離するために、JCICS を使用する OSGi バンドルを作成してください。また、OSGi バンドルの中でセマンティック・バージョン管理を使用して、アプリケーションのビジネス・ロジックの更新を管理できます。JDBC DriverManager インターフェースを介して DB2 を使用する WAB または OSGi バンドルごとに、`com.ibm.db2.jcc` の Import-Package ヘッダーをバンドル・マニフェストに含めます。これはデータ・ソース接続に必須ではありません。
4. オプション: Web アプリケーションのユーザーを認証したい場合は、Web プロジェクトに web.xml ファイルを作成して、セキュリティー制約を含めます。CICS Explorer SDK には、CICS 用の正しい情報が入った、このファイル用のテンプレートが含まれます。詳しくは、237 ページの『Liberty JVM サーバーでのユーザー認証』を参照してください。
5. OSGi バンドルを参照する OSGi アプリケーション・プロジェクトを作成します。

6. OSGi アプリケーション・プロジェクトを参照する CICS バンドル・プロジェクトを作成します。CICS リソースの定義とインポートを追加することもできます。各 CICS バンドルには、ID とバージョンが含まれており、変更を細かく管理できます。
7. オプション: URI からのインバウンド Web 要求をマップして特定のトランザクションの下で実行する場合は、URIMAP および TRANSACTION リソースを CICS バンドルに追加します。これらのリソースを定義しない場合、CJSA という名前の提供されたトランザクションの下ですべての処理が実行されます。これらのリソースは、動的にインストールされ、CICS の中のバンドルの一部として管理されます。

### タスクの結果

これで、開発環境をセットアップし、OSGi Web アプリケーションを作成し、それをデプロイメント用にパッケージしました。

### 次のタスク

アプリケーションをデプロイする用意ができたら、CICS バンドル・プロジェクトを zFS にエクスポートします。参照されたプロジェクトは、ビルドされ、zFS への転送に組み込まれます。または、開発デプロイメント・モデルに従って、アプリケーションを EBA ファイルとしてエクスポートし、稼働中の Liberty JVM サーバーの dropins ディレクトリーにそれをデプロイすることもできます。dropins を使用する場合は、セキュリティーおよび他のサービス品質は構成できないことに注意してください。

## Liberty JVM サーバー内で実行するよう Java Web アプリケーションをマイグレーションする

ネットワークを介して CICS にアクセスする Liberty プロファイル・インスタンスの中で実行する Java Web アプリケーションがある場合、そのアプリケーションを Liberty JVM サーバーの中で実行すると、パフォーマンスを最適化できます。

### このタスクについて

CICS は、Liberty プロファイルで使用可能な機能のサブセットをサポートしています。すべてのコア Web コンテナー API (JSP やサーブレットなど) が、多くの共通 Java EE API および機能とともにサポートされます。サポートされる機能のリストについては、79 ページの『Liberty フィーチャー』を参照してください。

アプリケーションでセキュリティーを使用する場合は、Liberty セキュリティー機能を引き続き使用できますが、追加アクションなしで、CICS タスクがトランザクション CJSA 下で実行され、CICS での URIMAP マッチングが使用不可になり、すべてのリソース・アクセスが CICS のデフォルト・ユーザー ID で実行される可能性があります。Liberty で決定された同じユーザー ID で CICS タスクを実行できるようにして、セキュリティー・ソリューションをより効率的に CICS と統合する場合は、239 ページの『ユーザーに Liberty JVM サーバー内のアプリケーションを実行する権限を与える』を参照してください。

### 手順

1. アプリケーションが CICS とデータを受け渡しするときに正しい JCICS エンコードが使用されることを確認して、JCICS API を使用して CICS サービスに直接アクセスするようにアプリケーションを更新します。 エンコードの詳細については、48 ページの『データ・エンコード』を参照してください。
2. 基本認証に CICS セキュリティーを使用する場合は、動的 Web プロジェクトの `web.xml` ファイルのセキュリティー制約を更新して、認証に CICS ロールを使用するようにします。

```
<auth-constraint>
     <description>All authenticated users of my application</description>
     <role-name>cicsAllAuthenticated</role-name>
   </auth-constraint>
```

3. CICS バンドルに WAR (動的 Web プロジェクト) ファイル、EBA (OSGi アプリケーション・プロジェクト) ファイル、または EAR (エンタープライズ・アーカイブ) ファイルとしてアプリケーションをパッケージします。 CICS バンドルは、アプリケーションのデプロイメントの単位です。バンドルの中のすべての CICS リソースは、動的にインストールされ、一緒に管理されます。一緒に管理するアプリケーション・コンポーネントの CICS バンドル・プロジェクトを作成します。
4. CICS バンドル・プロジェクトを zFS にデプロイし、CICS バンドルを Liberty JVM サーバーにインストールします。

### タスクの結果

アプリケーションは、JVM サーバーで実行中です。

**関連タスク**:

122 ページの『サーブレットおよび JSP アプリケーションの開発』
CICS アプリケーションに対する最新のインターフェースを提供するために、Web アプリケーション・テクノロジーを使用するプレゼンテーション層を開発できます。Eclipse ベースの Web 開発ツールは、これらのアプリケーションを作成するための開発プラットフォームを提供します。CICS Explorer SDK は、CICS での実行用にこれらをビルドし、パッケージ化し、デプロイするためのサポートを提供します。

## Liberty Web サーバー・プラグイン

Web サーバー・プラグインを使用することにより、サポートされる Web サーバーから 1 つ以上のアプリケーション・サーバーに HTTP 要求を転送することが可能になります。

Web サーバー・プラグインを使用する主な理由が 3 つあります。

- 静的コンテンツを提供する Web サーバーの統合を可能にします。
- HTTPS 使用時に Web サーバーの SSL エンドポイントの終了を可能にします。
- 一群の Liberty サーバー間で HTTP 要求のロード・バランシングとフェイルオーバーを可能にします。

Liberty サーバーで `plugin-cfg.xml` ファイルを生成し、そのファイルを Web サーバーのホスト・マシンにコピーすることによって、Web サーバー・プラグインは構成されます。プラグインはインバウンド要求を受け取り、その要求をこのファイル

に含まれている構成データと照らしてチェックし、着信 HTTP 要求を構成済み Liberty サーバーの URI とホストに転送します。

Liberty プロファイル・サーバーを利用して `plugin-cfg.xml` を生成する手順では `generatePluginConfig` 操作を使用します。この操作は Liberty が提供する `com.ibm.ws.jmx.mbeans.generatePluginConfig` MBean によって公開されます。この JMX MBean をリモートから呼び出すには、 IBM Java SDK で提供されている JConsole ユーティリティーと Liberty サーバーの `restConnector-1.0` 機能を組み合わせて使用するか、または、MBean の必要な操作を呼び出すカスタム JMX アプリケーションを作成します。CICS Liberty サーバーでの JMX の使用について詳しくは、106 ページの『Java Management Extensions API (JMX)』を参照してください。

Web サーバー・プラグインのセットアップの詳細な情報は、WAS Knowledge Center にあります。WebSphere Application Server for z/OS 8.5.5 製品資料内の『Liberty プロファイルの Web サーバー・プラグインの構成』を参照してください。

## OSGi フレームワークの使用

CICS での OSGi フレームワークの使用。

### シングルトン・バンドル

xxxxx.

### OSGi フラグメント

特有のマニフェスト・ヘッダーを持つ Java アーカイブ・ファイルであり、それらのヘッダーにより、このファイルを 1 つまたは複数の指定されたホスト・バンドルに付加することが可能になります。

### Bundle-Activator

`Bundle-Activator` ヘッダーをバンドル `MANIFEST.MF` ファイルに追加して、OSGi フレームワークに対するバンドル・アクティベーター・クラスを指定します。

### OSGi レジストリー

xxxxx.

**注:** ベスト・プラクティスとして、依存関係を表すには Require-Bundle ではなく Import-Package を使用してください。`MANIFEST.MF` でインポートを定義する必要があります。

# 第 3 章 Java サポートのセットアップ

基本的なセットアップ・タスクを実行して、CICS 領域で Java をサポートし、Java アプリケーションを実行するように JVM サーバーを構成します。

## 始める前に

CICS に必要な Java コンポーネントは、製品のインストール時にセットアップされます。『インストール』の『Java コンポーネントのインストール検査』の情報を使用して、Java コンポーネントが正しくインストールされていることを確認する必要があります。

## このタスクについて

CICS は z/OS UNIX のファイルを使用して JVM を開始します。 CICS 領域が正しい zFS ディレクトリーを使用するように構成されており、それらのディレクトリーに正しい権限があることを確認する必要があります。 CICS を構成して zFS をセットアップした後、Java アプリケーションを実行するように JVM サーバーを構成できます。

## 手順

1. JVMPROFILEDIR システム初期設定パラメーターを、CICS 領域で使用される JVM プロファイルを保管する先の z/OS UNIX 内の適切なディレクトリーに設定します。 詳しくは、 130 ページの『JVM プロファイルのロケーションの設定』を参照してください。
2. CICS 領域には、必ず Java アプリケーションを実行できる十分なメモリーを持たせるようにします。 詳しくは、 132 ページの『Java のメモリー制限の設定』を参照してください。
3. JVM の作成に必要な JVM プロファイル、ディレクトリー、およびファイルを含めて、z/OS UNIX に保持されているリソースにアクセスする許可を CICS 領域に付与します。 詳しくは、 132 ページの『z/OS UNIX ディレクトリーおよびファイルに対するアクセス権の CICS 領域への付与』を参照してください。
4. JVM サーバーをセットアップします。 さまざまなワークロードを実行するように JVM サーバーを構成できます。詳しくは、 135 ページの『JVM サーバーのセットアップ』を参照してください。
5. オプション: 安全でない可能性のあるアクションを Java アプリケーションで実行しないように保護するには、Java セキュリティー・マネージャーを有効にします。 詳しくは、 248 ページの『Java セキュリティー・マネージャーの有効化』を参照してください。

## タスクの結果

Java をサポートする CICS 領域をセットアップし、Java アプリケーションを実行する JVM サーバーを作成しました。

### 次のタスク

既存の Java アプリケーションをアップグレードする場合は、アップグレード のガイダンスに従ってください。 JVM サーバーで Java アプリケーションの実行を開始するには、 181 ページの『第 4 章 JVM サーバーへのアプリケーションのデプロイ』を参照してください。

『JVM プロファイルのロケーションの設定』
CICS は、**JVMPROFILEDIR** システム初期設定パラメーターで指定された z/OS UNIX ディレクトリーから、JVM プロファイルをロードします。**JVMPROFILEDIR** パラメーターの値を新規ロケーションに変更し、提供されたサンプル JVM プロファイルをこのディレクトリーにコピーする必要があります。その結果、それらのプロファイルを使用してインストールを検証できます。

132 ページの『Java のメモリー制限の設定』
Java アプリケーションには、他の言語で作成されたプログラムよりも多くのメモリーが必要です。Java アプリケーションを実行するために使用できる、十分なストレージとメモリーが CICS および Java にあることを確認する必要があります。

132 ページの『z/OS UNIX ディレクトリーおよびファイルに対するアクセス権の CICS 領域への付与』
CICS では、z/OS UNIX 内のディレクトリーとファイルへのアクセスが必要です。各 CICS 領域には、インストール時に z/OS UNIX ユーザー ID (UID) が割り当てられます。これらの領域は、z/OS UNIX グループ ID (GID) を割り当てられた RACF® グループに接続されます。この UID および GID を使用して、CICS 領域が z/OS UNIX 内のディレクトリーおよびファイルにアクセスする権限を付与します。

135 ページの『JVM サーバーのセットアップ』
JVM サーバー内で Java アプリケーション、Web アプリケーション、Axis2、または CICS セキュリティー・トークン・サービスを実行するには、CICS リソースをセットアップして、JVM にオプションを渡す JVM プロファイルを作成する必要があります。

## JVM プロファイルのロケーションの設定

CICS は、**JVMPROFILEDIR** システム初期設定パラメーターで指定された z/OS UNIX ディレクトリーから、JVM プロファイルをロードします。**JVMPROFILEDIR** パラメーターの値を新規ロケーションに変更し、提供されたサンプル JVM プロファイルをこのディレクトリーにコピーする必要があります。その結果、それらのプロファイルを使用してインストールを検証できます。

### 始める前に

USSHOME システム初期設定パラメーターは、z/OS UNIX に CICS ファイル用のルート・ディレクトリーを指定する必要があります。

### このタスクについて

CICS 提供のサンプル JVM プロファイルは、CICS インストール処理時にシステムに合わせてカスタマイズされるので、これらのプロファイルを即時に使用してイン

ストールを検証できます。独自の Java アプリケーション用にこれらのファイルのコピーをカスタマイズすることができます。

JVM プロファイルでの使用に適している設定は、CICS のリリースごとに異なる可能性があるので、問題判別を容易にするために、すべてのプロファイルのベースとして CICS 提供のサンプルを使用してください。アップグレード情報を確認して、JVM プロファイルの新規オプションまたは変更されたオプションを見つけてください。

## 手順

1. JVMPROFILEDIR システム初期設定パラメーターを、CICS 領域で使用される JVM プロファイルを保管する先の z/OS UNIX のロケーションに設定します。指定する値の長さは 240 文字までにすることができます。

   **JVMPROFILEDIR** システム初期設定パラメーターに指定された設定は、`/usr/lpp/cicsts/cicsts53/JVMProfiles` です。これは、サンプル JVM プロファイルのインストール場所です。このディレクトリーは、カスタマイズされた JVM プロファイルを安全に保管できる場所ではありません。プログラムの保守時にサンプル JVM プロファイルが上書きされると、変更内容が失われる危険があるためです。したがって、必ず **JVMPROFILEDIR** を変更して、JVM プロファイルを保管できる別の z/OS UNIX ディレクトリーを指定する必要があります。JVM プロファイルをカスタマイズする必要があるユーザーに該当する許可を付与できるディレクトリーを選択してください。
2. 提供されるサンプル JVM プロファイルを、そのインストール場所から z/OS UNIX ディレクトリーにコピーします。

   CICS をインストールすると、サンプル JVM プロファイルが zFS ディレクトリーに置かれます。このディレクトリーは、DFHISTAR インストール・ジョブの **USSDIR** パラメーターで指定されます。デフォルトのインストール・ディレクトリーは `/usr/lpp/cicsts/cicsts53/JVMProfiles` です。

## タスクの結果

サンプル JVM プロファイルが zFS ディレクトリーにコピーされ、そのディレクトリーを使用するように CICS が構成されました。サンプル JVM プロファイルにはデフォルト値が含まれているため、それをすぐに使用して JVM サーバーをセットアップできます。

## 次のタスク

132 ページの『Java のメモリー制限の設定』で説明されているように、Java アプリケーションを実行するために十分なメモリーが CICS および Java にあることを確認してください。また、Java のインストール先であり、Java アプリケーションのデプロイ先である z/OS UNIX ディレクトリーへのアクセス権限が CICS 領域にあることを確認する必要もあります。詳しくは、132 ページの『z/OS UNIX ディレクトリーおよびファイルに対するアクセス権の CICS 領域への付与』を参照してください。

# Java のメモリー制限の設定

Java アプリケーションには、他の言語で作成されたプログラムよりも多くのメモリーが必要です。Java アプリケーションを実行するために使用できる、十分なストレージとメモリーが CICS および Java にあることを確認する必要があります。

## このタスクについて

Java は、16 MB 境界より下のストレージ、31 ビット・ストレージ、および 64 ビット・ストレージを使用します。JVM ヒープに必要なストレージは、CICS DSA ではなく、MVS 内の CICS 領域ストレージから取得されます。

## 手順

1. z/OS **MEMLIMIT** パラメーターが適切な値に設定されていることを確認します。このパラメーターは、CICS アドレス・スペースが使用できる 64 ビット・ストレージの量を制限します。CICS は 64 ビット・バージョンの Java を使用するので、**MEMLIMIT** が、CICS 領域で 64 ビット・ストレージを使用するこの機能やその他の機能の両方に十分な大きい値に設定されていることを確認する必要があります。

   次のトピックを参照してください。

   - 217 ページの『JVM サーバーのストレージ所要量の計算』
   - 「パフォーマンスの改善」の『MEMLIMIT の見積もり、確認、および設定』
2. 開始ジョブ・ストリームの **REGION** パラメーターに、Java の実行に十分な大きさの値が指定されていることを確認します。 それぞれの JVM に、アプリケーション (ジャストインタイム・コンパイル・コードを含む) を実行するための 16 MB 境界より下のストレージ、およびパラメーターを CICS に渡すための作業用ストレージが必要です。

# z/OS UNIX ディレクトリーおよびファイルに対するアクセス権の CICS 領域への付与

CICS では、z/OS UNIX 内のディレクトリーとファイルへのアクセスが必要です。各 CICS 領域には、インストール時に z/OS UNIX ユーザー ID (UID) が割り当てられます。これらの領域は、z/OS UNIX グループ ID (GID) を割り当てられた RACF グループに接続されます。この UID および GID を使用して、CICS 領域が z/OS UNIX 内のディレクトリーおよびファイルにアクセスする権限を付与します。

## 始める前に

自分が z/OS UNIX のスーパーユーザーであるか、ディレクトリーおよびファイルの所有者であることを確認してください。ディレクトリーおよびファイルの所有者は、最初は、製品をインストールしたシステム・プログラマーの UID として設定されます。ディレクトリーおよびファイルの所有者は、インストール時に GID が割り当てられた RACF グループに接続されなければなりません。所有者は、この RACF グループをデフォルト・グループ (DFLTGRP) として持つか、補足グループの 1 つとして RACF グループに接続することができます。

## このタスクについて

z/OS UNIX システム・サービスでは、個々の CICS 領域が単一の UNIX ユーザーとして扱われます。 z/OS UNIX ディレクトリーおよびファイルにアクセスするユーザー権限をさまざまな方法で付与することができます。例えば、CICS 領域の接続先の RACF グループに対して、ディレクトリーまたはファイルに関する適切なグループ権限を付与できます。このオプションは、実稼働環境に最適な場合があり、以下の手順で説明しています。

## 手順

1. CICS 領域からのアクセスが必要となる z/OS UNIX 内のディレクトリーおよびファイルを識別します。

| JVM サーバー・オプション | デフォルト・ディレクトリー | 権限 | 説明 |
|---|---|---|---|
| JAVA_HOME | /usr/lpp/java/J7.0_64/bin | 読み取りおよび実行 | IBM 64-bit SDK for z/OS, Java テクノロジー・エディション ディレクトリー |
| USSHOME | /usr/lpp/cicsts/cicsts53 | 読み取りおよび実行 | z/OS UNIX 上の CICS ファイルのインストール・ディレクトリー。このディレクトリー内のファイルには、サンプル・プロファイルと CICS 提供の JAR ファイルが含まれます。 |
| WORK_DIR | /u/*CICS region userid* | 読み取り、書き込み、および実行 | CICS 領域の作業ディレクトリー。このディレクトリーには、JVM からの入力、出力、およびメッセージが入っています。 |
| JVMPROFILEDIR | *USSHOME*/JVMProfiles/ | 読み取りおよび実行 | **JVMPROFILEDIR** システム初期設定パラメーターで指定された、CICS 領域の JVM プロファイルが入っているディレクトリー。 |
| WLP_USER_DIR | WORK_DIR/APPLID/JVMSERVER/wlp/usr/ | 読み取りおよび実行 | Liberty JVM サーバーの構成ファイルを格納するディレクトリーを指定します。 |
| WLP_OUTPUT_DIR | WLP_USER_DIR/servers | 読み取り、書き込み、および実行 | Liberty JVM サーバーの出力ディレクトリーを指定します。 |

**注:** Liberty JVM サーバーの自動構成を使用する場合は、CICS が server.xml に書き込みを実行できる必要があるため、WLP_USER_DIR に追加の x 許可 (読み取り、書き込み、実行) が必要になります。

2. ディレクトリーおよびファイルをリストして、権限を表示します。 開始したいディレクトリーに移動し、次の UNIX コマンドを発行します。

```
ls -la
```

現行ディレクトリーが CICSHT## のホーム・ディレクトリーであるときに、このコマンドが z/OS UNIX システム・サービスのシェル環境で発行されると、次の例のようなリストが表示される場合があります。

```
/u/cicsht##:>ls -la
total 256
drwxr-xr-x   2 CICSHT## CICSTS53     8192 Mar 15  2008 .
```

```
drwx------   4 CICSHT## CICSTS53     8192 Jul  4 16:14 ..
-rw-------   1 CICSHT## CICSTS53     2976 Dec  5  2010 Snap0001.trc
-rw-r--r--   1 CICSHT## CICSTS53     1626 Jul 16 11:15 dfhjvmerr
-rw-r--r--   1 CICSHT## CICSTS53        0 Mar 15  2010 dfhjvmin
-rw-r--r--   1 CICSHT## CICSTS53      458 Oct  9 14:28 dfhjvmout
/u/cicsht##:>
```

3. グループ権限を使用してアクセス権限を付与する場合は、各ディレクトリーおよびファイルのグループ権限が、CICS がリソースに対して必要とするアクセス・レベルを認可するものであることを確認します。 r、w、x、および - という文字を使用して 3 組の権限が指定されます。これらの文字は、「読み取り」、「書き込み」、「実行」、および「なし」を表し、コマンド行の左側の列の 2 文字目から表示されます。最初の組は所有者権限、2 番目の組はグループ権限、3 番目の組はその他の権限です。 前述の例では、所有者には dfhjvmerr、dfhjvmin、および dfhjvmout に対する読み取りおよび書き込み権限がありますが、グループおよびその他すべてには、読み取り権限しかありません。
4. 特定のリソースのグループ権限を変更したい場合は、 UNIX コマンド chmod を使用します。 次の例では、指定されたディレクトリーおよびそのサブディレクトリーとファイルのグループ権限を、読み取り、書き込み、および実行に設定します。 -R は、すべてのサブディレクトリーおよびファイルに権限を再帰的に適用します。

   ```
   chmod -R g=rwx directory
   ```

   次の例では、指定されたファイルのグループ権限を読み取りおよび実行に設定します。

   ```
   chmod g+rx filename
   ```

   次の例では、指定された 2 つのファイルでグループに対する書き込み権限をオフにします。

   ```
   chmod g-w filename filename
   ```

   上記のすべての例で、g はグループ権限を指定します。その他に訂正したい権限がある場合、 u はユーザー (所有者) 権限を指定し、o はその他の権限を指定します。
5. CICS 領域から z/OS UNIX にアクセスするために選択された RACF グループに対して、リソースごとにグループ権限を割り当てます。個々のディレクトリーとそのサブディレクトリー、およびそこに含まれるファイルに対して、グループ権限を割り当てる必要があります。 次の UNIX コマンドを入力します。

   ```
   chgrp -R GID directory
   ```

   *GID* は RACF グループの GID の数値であり、*directory* は、CICS 領域の権限を付与する対象となるディレクトリーの絶対パスです。 例えば、/usr/lpp/cicsts/cicsts53 ディレクトリーにグループ権限を割り当てるには、次のコマンドを使用します。

   ```
   chgrp -R GID /usr/lpp/cicsts/cicsts53
   ```

   CICS 領域のユーザー ID は RACF グループに接続されるため、CICS 領域は、これらすべてのディレクトリーおよびファイルに関する適切な権限を持つことになります。

### タスクの結果

これで、CICS には、Java アプリケーションを実行するために z/OS UNIX 内のディレクトリーとファイルにアクセスする適切な権限があることが確実になりました。

ファイルの移動または新規ファイルの作成などにより、セットアップ中の CICS 機能を変更した場合は、新規のファイルまたは移動したファイルに関する CICS 領域のアクセス権限が保持されるように、この手順を忘れずに繰り返してください。

### 次のタスク

サンプルのプログラムおよびプロファイルを使用して、Java サポートが正しくセットアップされたことを確認します。

## JVM サーバーのセットアップ

JVM サーバー内で Java アプリケーション、Web アプリケーション、Axis2、または CICS セキュリティー・トークン・サービスを実行するには、CICS リソースをセットアップして、JVM にオプションを渡す JVM プロファイルを作成する必要があります。

### このタスクについて

JVM サーバーは、単一の JVM でさまざまな Java アプリケーションの複数の並行要求を処理できます。 JVMSERVER リソースは、CICS における JVM サーバーを表します。このリソースによって定義される JVM プロファイルは、JVM の構成オプション、Language Environment エンクレーブに値を提供するプログラム、およびスレッドの限度を指定します。 JVM サーバーは、異なるタイプのワークロードを実行できます。JVM サーバーの用途ごとに、次のように異なる JVM プロファイルが用意されています。

- OSGi バンドルとしてパッケージされているアプリケーションを実行するには、DFHOSGI.jvmprofile を使用して JVM サーバーを構成します。このプロファイルには、JVM サーバーで OSGi フレームワークを実行するためのオプションが含まれています。
- JSP ページとサーブレットを含むアプリケーションを実行するには、DFHWLP.jvmprofile を使用して JVM サーバーを構成します。このプロファイルには、Liberty プロファイル・テクノロジーに基づく Web コンテナーを実行するためのオプションが含まれています。 Web コンテナーには OSGi フレームワークも含まれるため、OSGi バンドルとしてパッケージされているアプリケーションを実行することができます。
- Axis2 SOAP エンジンを使用して Web サービスの SOAP 処理を実行するには、DFHJVMAX.jvmprofile を使用して JVM サーバーを構成します。このプロファイルには、JVM サーバーで Axis2 を実行するためのオプションが含まれています。
- CICS セキュリティー・トークン・サービス (STS) を実行するには、DFHJVMST.jvmprofile を使用して JVM サーバーを構成します。このプロファイルには STS を実行するためのオプションが含まれています。

プロファイルに加える変更はすべて、そのプロファイルを使用するすべての JVM サーバーに適用されます。各プロファイルをカスタマイズする際に、変更内容が、JVM サーバーを使用するすべての Java アプリケーションに適切であることを確認してください。

CICS オンライン・リソース定義を使用して JVM サーバーや JVM プロファイルを構成することができます。あるいは、CICS Explorer を使用して CICS バンドル内に JVMSERVER リソースと JVM プロファイルを定義し、パッケージ化することもできます。詳しくは、CICS Explorer 製品資料内の『Working with bundles』を参照してください。

『OSGi JVM サーバーの構成』
OSGi バンドルにパッケージされている Java アプリケーションをデプロイする場合、OSGi フレームワークを実行するように JVM サーバーを構成します。

139 ページの『Liberty JVM サーバーの構成』
Java サーブレットおよび JSP ページを使用する Java Web アプリケーションをデプロイする場合には、Web コンテナーを実行するように Liberty JVM サーバーを構成します。 Web コンテナーは、Liberty プロファイル・テクノロジーに基づきます。

153 ページの『Axis2 用の JVM サーバーの構成』
Java Web サービスを実行するかまたは SOAP 要求をパイプラインで処理する場合、Axis2 を実行するように JVM サーバーを構成します。

156 ページの『CICS セキュリティー・トークン・サービスのための JVM サーバーの構成』
SAML トークンを検証して処理する場合は、CICS セキュリティー・トークン・サービスを実行するように JVM サーバーを構成します。

157 ページの『JVM プロファイルの検証およびプロパティー』
JVM プロファイルには、開始時に JVM に渡される一連のオプションとシステム・プロパティーが含まれています。一部の JVM プロファイル・オプションは CICS 環境に固有であり、他の環境の JVM には使用されません。 CICS は、JVM サーバーの始動時に JVM プロファイルが正しくコーディングされていることを検証します。

### タスクの結果

JVM サーバーが構成され、Java ワークロードを実行する準備ができました。

### 次のタスク

Java 環境のためのセキュリティーを構成します。 Java アプリケーションのデプロイやインストールのための適切なアクセス権をアプリケーション開発者に付与し、アプリケーション・ユーザーが CICS で Java プログラムやトランザクションを実行するための権限を与えます。詳しくは、『Java アプリケーションのセキュリティー』を参照してください。

## OSGi JVM サーバーの構成

OSGi バンドルにパッケージされている Java アプリケーションをデプロイする場合、OSGi フレームワークを実行するように JVM サーバーを構成します。

## このタスクについて

JVM サーバーには、自動的にクラス・ロードを処理する OSGi フレームワークが含まれているため、標準クラスパス・オプションを JVM プロファイルに追加することはできません。提供されているサンプルの DFHOSGI.jvmprofile は OSGi JVM サーバーに適しています。このタスクでは、このサンプル・プロファイルから OSGi アプリケーション用の JVM サーバーを定義する方法を示します。

JVM サーバーは、CICS オンライン・リソース定義を使用して定義することも、CICS Explorer の CICS バンドル内で定義することもできます。

## 手順

1. JVM サーバー用の JVMSERVER リソースを作成します。
   a. JVM サーバー用の JVM プロファイルの名前を指定します。 JVMSERVER の JVMPROFILE 属性では、1 文字から 8 文字の名前を指定します。この名前は、JVM サーバーの構成オプションを保持するファイルである JVM プロファイルの接頭部に使用されます。ここでは接尾部として .jvmprofile を指定する必要はありません。
   b. JVM サーバーのスレッド限度を指定します。 JVMSERVER の THREADLIMIT 属性では、JVM サーバーの Language Environment エンクレーブで許可されているスレッドの最大数を指定します。スレッド数は、JVM サーバーで実行するワークロードによって異なります。始めにデフォルト値を受け入れ、後で環境を調整できます。 1 つの JVM サーバーで最大 256 個のスレッドを設定できます。
2. JVM プロファイルを作成して、JVM サーバーの構成オプションを定義します。ベースとして、サンプル・プロファイル DFHOSGI.jvmprofile を使用することができます。このプロファイルには、JVM サーバーを開始するために適したオプションのサブセットが含まれています。 JVM プロファイルのすべてのオプションと値については、 157 ページの『JVM プロファイルの検証およびプロパティー』で説明されています。 158 ページの『JVM プロファイルのコーディング規則』のコーディング規則 (プロファイル名のコーディング規則を含む) に従ってください。
   a. JVM プロファイルの場所を設定します。 JVM プロファイルは、システム初期設定パラメーター JVMPROFILEDIR で指定するディレクトリー内になければなりません。詳しくは、 130 ページの『JVM プロファイルのロケーションの設定』を参照してください。
   b. サンプル・プロファイルを次のように変更します。
      - JAVA_HOME を IBM Java SDK のインストール先に設定します。
      - WORK_DIR を、JVM サーバーからのメッセージ、トレース、および出力の適当な宛先ディレクトリーに設定します。
      - TZ を、JVM サーバーからのメッセージに対するタイム・スタンプのタイム・ゾーンを指定するように設定します。英国の例は TZ=GMT0BST,M3.5.0,M10.4.0 です。
   c. 変更内容を JVM プロファイルに保管します。 z/OS UNIX システム・サービスのファイル・システムでは、JVM プロファイルを EBCDIC として保管する必要があります。
3. JVMSERVER リソースをインストールして使用可能にします。

## タスクの結果

CICS は Language Environment エンクレーブを作成し、JVM プロファイルから JVM サーバーにオプションを渡します。 JVM サーバーが始動し、OSGi フレームワークはすべての OSGi ミドルウェア・バンドルを解決します。 JVM サーバーが始動を正常に完了すると、JVMSERVER リソースが ENABLED 状態でインストールされます。

エラーが発生する場合 (CICS が JVM プロファイルの検出も読み取りもできない場合など)、JVM サーバーは始動できません。 JVMSERVER リソースは DISABLED 状態でインストールされ、CICS はシステム・ログにエラー・メッセージを発行します。詳しくは、 251 ページの『第 8 章 Java アプリケーションのトラブルシューティング』を参照してください。

## 次のタスク

- 257 ページの『JVM stdout、stderr、JVMTRACE、およびダンプ出力の場所の制御』の説明に従って、JVM ログのロケーションを構成します。
- 182 ページの『JVM サーバーにおける OSGi バンドルのデプロイ』の説明に従って、JVM サーバーの OSGi フレームワークにアプリケーションの OSGi バンドルをインストールします。
- DB2 や WebSphere MQ など、ネイティブ C ダイナミック・リンク・ライブラリー (DLL) ファイルを含むディレクトリーを指定します。これらのディレクトリーは、JVM プロファイルの LIBPATH_SUFFIX オプションで指定します。
- OSGi フレームワークで実行するミドルウェア・バンドルを指定します。ミドルウェア・バンドルは、WebSphere MQ および DB2 への接続などの共用サービスを実装するための Java クラスを格納する OSGi バンドルの 1 つのタイプです。これらのバンドルは、JVM プロファイルの OSGI_BUNDLES オプションで指定します。

  『JVM プロファイルの例』
  OSGI アプリケーション用の JVM プロファイルの例を示します。

## JVM プロファイルの例

OSGI アプリケーション用の JVM プロファイルの例を示します。

次の例では、DB2 バージョン 11 および JDBC 4.0 OSGi ミドルウェア・バンドルを使用する OSGi フレームワークを開始するように構成された JVM プロファイルを抜粋して示しています。

```
#*********************************************************************
#
#                          Required parameters
#                          -------------------
#
# When using a JVM server, the set of CICS options that are supported
JAVA_HOME=/usr/lpp/java/J7.0_64
WORK_DIR=.
LIBPATH_SUFFIX=/usr/lpp/db2v11/jdbc/lib
...
#*********************************************************************
#
#                     JVM server specific parameters
#                     ------------------------------
#
```

```
OSGI_BUNDLES=/usr/lpp/db2v11/jdbc/classes/db2jcc4.jar,\
             /usr/lpp/db2v11/jdbc/classes/db2jcc_license_cisuz.jar
OSGI_FRAMEWORK_TIMEOUT=60
#
#*********************************************************************
#
#                              JVM options
#                              -----------
# The following option sets the Garbage collection Policy.
#
-Xgcpolicy:gencon
#
#*********************************************************************
#
#                 Setting user JVM system properties
#                 ----------------------------------
#
# -Dcom.ibm.cics.some.property=some_value
#
#*********************************************************************
#
#              Unix System Services Environment Variables
#              ------------------------------------------
#
JAVA_DUMP_OPTS="ONANYSIGNAL(JAVADUMP,SYSDUMP),ONINTERRUPT(NONE)"
#
#
```

## Liberty JVM サーバーの構成

Java サーブレットおよび JSP ページを使用する Java Web アプリケーションをデプロイする場合には、Web コンテナーを実行するように Liberty JVM サーバーを構成します。 Web コンテナーは、Liberty プロファイル・テクノロジーに基づきます。

### このタスクについて

Liberty プロファイル・テクノロジーは CICS と共にインストールされ、JVM サーバー内の Web コンテナーとして実行されます。 Liberty JVM サーバーは、Liberty プロファイルで使用可能な機能のサブセットをサポートしています。

Liberty JVM サーバーの構成方法には次の 2 つがあります。

- 自動構成。 CICS は Liberty プロファイル用の構成ファイル `server.xml` を、CICS インストール・ディレクトリーに用意されているテンプレートから自動的に作成および更新します。自動構成では、Liberty プロファイルの最小構成値セットを使用して迅速に開始することができるため、開発環境で役立つ場合があります。自動構成を有効にするには、JVM システム・プロパティーの **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure** プロパティーを `true` に設定します。 CICS バンドル内で JVM サーバーを定義する場合は、このオプションを設定します。
- 手動構成。これはデフォルト設定です。構成ファイルとすべての値を提供します。実稼働環境では、一般に、自動構成に使用される CICS 領域のユーザー ID は `server.xml` に対する書き込み権限を持たないため、手動構成が適切です。

JVM サーバーは、CICS オンライン・リソース定義を使用するか、CICS バンドルに含めることにより定義できます。

## 手順

1. JVM サーバー用の JVMSERVER リソースを作成します。また、バンドルで JVMSERVER リソースを作成することもできます。詳細については、『Administering』の『Artifacts that can be deployed in bundles』を参照してください。
   a. JVM サーバー用の JVM プロファイルの名前を指定します。 JVMSERVER の JVMPROFILE 属性では、1 文字から 8 文字の名前を指定します。この名前は、JVM サーバーの構成オプションを保持するファイルである JVM プロファイルの接頭部に使用されます。ここでは接尾部として `.jvmprofile` を指定する必要はありません。
   b. JVM サーバーのスレッド限度を指定します。 JVMSERVER の THREADLIMIT 属性では、JVM サーバーの Language Environment エンクレーブで許可されているスレッドの最大数を指定します。スレッド数は、JVM サーバーで実行するワークロードによって異なります。始めにデフォルト値を受け入れ、後で環境を調整できます。 1 つの JVM サーバーで最大 256 個のスレッドを設定できます。
2. JVM プロファイルを作成して、JVM サーバーの構成オプションを定義します。ベースとして、サンプル・プロファイル `DFHWLP.jvmprofile` を使用することができます。 このプロファイルには、JVM サーバーを開始するために適したオプションのサブセットが含まれています。 JVM プロファイルのすべてのオプションと値については、 157 ページの『JVM プロファイルの検証およびプロパティー』で説明されています。 158 ページの『JVM プロファイルのコーディング規則』のコーディング規則 (プロファイル名のコーディング規則を含む) に従ってください。
   a. JVM プロファイルの場所を設定します。 JVM プロファイルは、システム初期設定パラメーター **JVMPROFILEDIR** で指定するディレクトリー内になければなりません。詳しくは、 130 ページの『JVM プロファイルのロケーションの設定』を参照してください。
   b. サンプル・プロファイルを次のように変更します。
      - **JAVA_HOME** を IBM Java SDK のインストール先に設定します。
      - **WORK_DIR** を、JVM サーバーからのメッセージ、トレース、および出力の適当な宛先ディレクトリーに設定します。
      - **WLP_INSTALL_DIR** を `&USSHOME;/wlp` に設定します。
      - TZ を、JVM サーバーからのメッセージに対するタイム・スタンプのタイム・ゾーンを指定するように設定します。英国の例は TZ=GMT0BST,M3.5.0,M10.4.0 です。
   c. 変更内容を JVM プロファイルに保管します。 JVM プロファイルは、UNIX システム・サービス上で EBCDIC ファイル・エンコードで保管する必要があり、ファイルの接尾部は `.jvmprofile` でなければなりません。
3. Liberty プロファイルのサーバー構成を作成します。自動構成を使用する場合、これは JVM サーバーのセットアップ時に実行されます。 Liberty JVM サーバーの自動構成を使用可能にするには、以下の JVM システム・プロパティーを JVM プロファイルに設定します。
   a. **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true** を設定して自動構成を使用可能にします。

b. **-Dcom.ibm.cics.jvmserver.wlp.server.http.port** を使用して HTTP サーバーのリスナー・ポートを設定します。

c. **-Dcom.ibm.cics.jvmserver.wlp.server.host** を使用して Liberty HTTP サーバーが listen するホストを設定します。

d. **-Dcom.ibm.cics.jvmserver.wlp.server.name** を使用して Liberty プロファイル・サーバーの名前を設定します。

代わりに、Liberty プロファイル・ディレクトリー構造および server.xml 構成ファイルを手動で構成することもできます。これがデフォルトの設定です。これは、構成ファイルを慎重に管理する必要がある実稼働環境に適しています。詳しくは、143 ページの『構成ディレクトリー構造の作成』および server.xml の手動調整を参照してください。 CICS バンドルに JVM サーバーを定義する場合は、server.xml 構成ファイルを JVM プロファイルと一緒に CICS バンドルに組み込むことはできないため、自動構成を使用する必要があります。

4. JVMSERVER リソースをインストールして使用可能にします。

## タスクの結果

CICS は Language Environment エンクレーブを作成し、JVM プロファイルから JVM サーバーにオプションを渡します。 JVM サーバーが始動して Liberty サーバーを初期化すると、Liberty HTTP リスナーの実行が開始されます。 JVM サーバーが始動を正常に完了すると、JVMSERVER リソースが ENABLED 状態でインストールされます。

エラーが発生する場合 (CICS が JVM プロファイルの検出も読み取りもできない場合など)、JVM サーバーは初期化できません。 JVM サーバーは DISABLED 状態でインストールされ、CICS はシステム・ログにエラー・メッセージを発行します。解決するには、Liberty JVM サーバーおよび Java Web アプリケーションのトラブルシューティングを参照してください。 JVM サーバー内で Liberty プロファイルが正常に開始されたことを確認する場合は、zFS 上の WLP_USER_DIR 出力ディレクトリーにある message.log ファイルを参照してください。

**注:** JVM サーバーを有効にすると Liberty プロファイル・サーバーが開始し、JVM サーバーを無効にすると Liberty プロファイル・サーバーが停止します。JVM サーバーで実行される Liberty プロファイル・サーバーを開始または停止するために Liberty プロファイルの bin/server スクリプトを使用しないでください。

**注:** JVM サーバーによって示されている現在のスレッド数は、ワークロードが実行中でない場合でも、正の値を返します。これは、効率性のためにスレッドが Liberty 内にプールされているためです。この値は、内部アルゴリズムによって自然に増減することがあります。

## 次のタスク

- URL http://<server>:<port>/com.ibm.cics.wlp.defaultapp/ を使用して CICS デフォルト Web アプリケーション CICSDefaultApp を実行し、Liberty JVM サーバーが開始されていることを確認します。 CICSDefaultApp の構成の詳細については、CICS デフォルト Web アプリケーションの構成を参照してください。

- WebSphere MQ など、ネイティブ C ダイナミック・リンク・ライブラリー (DLL) ファイルを含むディレクトリーを指定します。 IBM またはベンダーによって提供されるミドルウェアおよびツールによっては、DLL ファイルをライブラリー・パスに追加する必要があります。
- JVM プロファイルの `WLP` で始まるオプションをリセットして、Liberty JVM サーバーの環境変数をオーバーライドします。
- CICS 領域で複数の JVM サーバーを実行する場合は、追加の各 Liberty JVM サーバーの JVM プロファイルに WLP_ZOS_PLATFORM=FALSE を追加します。許可される「セキュリティーが有効な」Liberty JVM サーバーは CICS 領域ごとに 1 つのみです。これは、CICS が認証や許可に Liberty の z/OS プラットフォーム拡張を使用するためです。ただし、Liberty のこれらの z/OS プラットフォームの側面は、アドレス・スペース内の単一の Liberty サーバーに制限されます。デフォルトでは、Liberty JVM サーバーは z/OS プラットフォーム拡張を開始しますが、セキュリティーが正常に有効になるのは、CICS 領域内の最初の Liberty JVM サーバーのみです。
- セキュリティーのサポートを追加します。 233 ページの『Liberty JVM サーバーに関するセキュリティーの構成』を参照してください。
- 184 ページの『CICS バンドル内の Web アプリケーションの Liberty JVM サーバーへのデプロイ』の説明に従って、Web アプリケーション (EAR ファイル、WAR ファイルおよび EBA ファイル) をインストールします。
- 一般的な Liberty のセットアップについては、Liberty プロファイルの概要、WebSphere Application Server for z/OS 8.5.5 製品資料内の『Liberty プロファイルの概要』を参照してください。

143 ページの『JVM プロファイルの例』
Liberty サーバー用の JVM プロファイルの例を示します。

143 ページの『構成ディレクトリー構造の作成』
JVM サーバーの zFS 内に構成ディレクトリー構造を作成します。

144 ページの『CICS デフォルト Web アプリケーションの構成』
CICS Liberty デフォルト Web アプリケーションは、Liberty サーバーが実行中であることを確認します。また、デフォルト Web アプリケーションは、サーバー構成の情報も提供します。ユーザーは JVM プロファイル、JVM サーバー・ログ、Liberty の `server.xml`、およびメッセージ・ログを、FileViewer サーブレットを使用して参照できます。

145 ページの『server.xml の手動調整』
自動構成を使用しない場合は、実際の環境に合わせるために、server.xml ファイルを以下のように変更してください。

148 ページの『Liberty 用のデフォルトの CICS DB2 JDBC タイプ 2 ドライバー・データ・ソースの作成』
自動構成プロパティーを使用してデフォルトの CICS DB2 JDBC タイプ 2 ドライバー・データ・ソースを作成します。

149 ページの『Liberty 用の CICS DB2 JDBC タイプ 2 ドライバー・データ・ソースの手動構成』
CICS の Liberty プロファイル・ユーザーは、JDBC タイプ 2 ドライバー・データ・ソース定義を使用して Java アプリケーションから DB2 データベースにアクセスできます。

151 ページの『Liberty 用の DB2 JDBC タイプ 4 ドライバー・データ・ソースの手動構成』
CICS の Liberty プロファイル・ユーザーは、Liberty から提供されるデータ・ソース定義を使用し、JDBC タイプ 4 ドライバーを介して Java アプリケーションから DB2 データベースにアクセスできます。

**関連概念**:

『Securing』の『Liberty JVM サーバーのセキュリティーの構成 (Configuring security for a Liberty JVM server)』

**関連情報**:

## JVM プロファイルの例

Liberty サーバー用の JVM プロファイルの例を示します。

次の例では、必要な構成ファイルとディレクトリー構造を自動的に作成するように構成された JVM プロファイルを抜粋して示しています。 DB2 バージョン 11 が使用されています。

```
#*********************************************************************
# JVM profile: DFHWLP
#
JAVA_HOME=/usr/lpp/java/J7.0_64
WORK_DIR=.
#*********************************************************************
# JVM server parameters
#
OSGI_FRAMEWORK_TIMEOUT=60
#*********************************************************************
# Liberty JVM server
#
-Dcom.ibm.ws.logging.console.log.level=OFF
-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true
-Dcom.ibm.cics.jvmserver.wlp.server.http.port=12345
-Dcom.ibm.cics.jvmserver.wlp.server.host=*
-Dcom.ibm.cics.jvmserver.wlp.jdbc.driver.location=/usr/lpp/db2v11/jdbc
-Dfile.encoding=ISO-8859-1
WLP_INSTALL_DIR=&USSHOME;/wlp
WLP_USER_DIR=./&APPLID;/&JVMSERVER;
#*********************************************************************
# JVM options
-Xgcpolicy:gencon
-Xms128M-Xmx256M
-Xmso128K
#*********************************************************************
# Unix System Services Environment Variables
TZ=CET-1CEST,M3.5.0,M10.5.0
```

## 構成ディレクトリー構造の作成

JVM サーバーの zFS 内に構成ディレクトリー構造を作成します。

### 手順

1. JVM サーバーの zFS に構成ディレクトリー構造を作成します。 JVM サーバーの構成ファイルは、作業ディレクトリー内の *WLP_USER_DIR*/servers/*server_name* ディレクトリーになければなりません。ここで、*WLP_USER_DIR* は WLP_USER_DIR オプションの値で、*server_name* は **com.ibm.cics.jvmserver.wlp.server.name** プロパティーの値です。*server_name*

プロパティーには、常に先頭に -D が付きます。これらのサーバー・オプションについて詳しくは、JVM サーバー・オプションを参照してください。

2. *server_name* ディレクトリーに Liberty プロファイル構成を作成します。 少なくとも、server.xml ファイルを作成する必要があります。Liberty プロファイルのインストール・ディレクトリーに wlp/templates/servers/defaultServer/server.xml として提供されているテンプレートをベースにできます。このファイルは ISO-8859-1 または UTF-8 のファイル・エンコード方式で保管する必要があります。
3. ご使用のシステムに合わせて、server.xml ファイルを編集します。<httpEndpoint> のホスト名とポート番号を更新します。JVM サーバーにおける server.xml の構成について詳しくは、『 145 ページの『server.xml の手動調整』』を参照してください。セキュリティーを使用する場合は、『Liberty JVM サーバーに関するセキュリティーの構成』を参照してください。

## CICS デフォルト Web アプリケーションの構成

CICS Liberty デフォルト Web アプリケーションは、Liberty サーバーが実行中であることを確認します。また、デフォルト Web アプリケーションは、サーバー構成の情報も提供します。ユーザーは JVM プロファイル、JVM サーバー・ログ、Liberty の server.xml、およびメッセージ・ログを、FileViewer サーブレットを使用して参照できます。

### 始める前に

アプリケーション・セキュリティーを有効にしないと、デフォルト Web アプリケーションのフルアクセスがすべてのユーザーに付与されます。自動構成を有効にし、CICS セキュリティー (sec=yes) で実行している場合、または server.xml を手動で構成して cicsts:security-1.0 機能を追加した場合、アプリケーションを実行するにはユーザー ID に追加のアクセス権が必要です。デフォルト・サーブレットと基本的な情報にアクセスするには、**ユーザー**のロールでなければなりません。FileViewer サーブレットにアクセスするには、**管理者**のロールでなければなりません。

### 手順

1. SAF 権限を使用し、server.xml に <safAuthorization .../> エレメントが含まれている場合、次のプロファイルを作成する必要があります。
   a. デフォルト・サーブレットにアクセスするには、次の例を使用します。

```
RDEFINE EJBROLE BBGZDFLT.com.ibm.cics.wlp.defaultapp.User UACC(NONE)
PERMIT BBGZDFLT.com.ibm.cics.wlp.defaultapp.User CLASS(EJBROLE) ID(WLPSVRS) ACCESS(READ)
SETROPTS RACLIST(EJBROLE) REFRESH
```

   b. FileViewer サーブレットにアクセスするには、次の例を使用します。

```
RDEFINE EJBROLE BBGZDFLT.com.ibm.cics.wlp.defaultapp.Administrator UACC(NONE)
PERMIT BBGZDFLT.com.ibm.cics.wlp.defaultapp.Administrator CLASS(EJBROLE) ID(WLPSVRS) ACCESS(READ)
SETROPTS RACLIST(EJBROLE) REFRESH
```

2. また、SAF 権限が構成されていない場合は、デフォルトの JEE ロールベースのアクセス権限が使用されます。
   - CICS は、次の構成に示すように、デフォルトの権限定義を提供します。デフォルト・サーブレットへのアクセスは、特殊サブジェクト **ALL_AUTHENTICATED_USERS** への**ユーザー**・ロールを通じて付与されます。つま

り、すべてのユーザーが認証されます。デフォルトで、CICS は FileViewer サーブレットへのアクセスを提供しません。

```
<authorization-roles id="com.ibm.cics.wlp.defaultapp">
    <security-role name="User">
         <special-subject type="ALL_AUTHENTICATED_USERS"/>
    </security-role>
</authorization-roles>
```

- ただし、次の例のように許可エレメントを追加することで、デフォルトの JEE ロールの情報を server.xml でカスタマイズできます。この例は、user2 を管理者ロールに追加し、FileViewer サーブレットへのアクセス権限を付与することで、デフォルトの構成を拡張します。

```
<authorization-roles id="com.ibm.cics.wlp.defaultapp">
    <security-role name="User">
        <user name="user1"/>
        <user name="user2"/>
   </security-role>
    <security-role name="Administrator">
        <user name="user2"/>
    </security-role>
</authorization-roles>
```

この場合、user1 はデフォルト・サーブレットにアクセスできますが、FileViewer サーブレットにはアクセスできず、user2 はデフォルト・サーブレットと FileViewer サーブレットにアクセスできます。

### タスクの結果

CICS デフォルト Web アプリケーションの構成が正常に実行されました。

**関連情報**:

『Configuring』の『Web アプリケーションのための Liberty JVM サーバーの構成』

## server.xml の手動調整

自動構成を使用しない場合は、実際の環境に合わせるために、server.xml ファイルを以下のように変更してください。

### HTTP エンドポイントの構成

アプリケーションへの Web アクセスが必要な場合は、適切なホスト名とポート番号を使って httpEndpoint 属性を更新します。例えば、

```
<httpEndpoint host="winmvs2c.example.com" httpPort="28216" httpsPort="28217"
 id="defaultHttpEndpoint"/>
```

他の場所 (例えば CICS の TCPIPSERVICE) で使用されていないポート番号を使用してください。

HTTPS は、SSL が構成されている場合にのみ使用可能です (242 ページの『Java 鍵ストアを使用した Liberty JVM サーバー用の SSL (TLS) の構成』を参照)。

詳しくは、WebSphere Application Server for z/OS 8.5.5 製品資料内の『Liberty プロファイルの概要』を参照してください。

### 機能の追加

以下の機能を <featureManager> の機能リストに追加します。

- CICS 機能 `cicsts:core-1.0` は、CICS システム OSGi バンドルを Liberty フレームワークにインストールします。JVM サーバーが始動するには、この機能が必要です。 SAF レジストリーも定義する必要があります。
- CICS セキュリティー機能 `icsts:security-1.0` は、CICS Liberty セキュリティーに必要な CICS システム OSGi バンドルを Liberty フレームワークにインストールします。 CICS 外部セキュリティーが有効 (SIT で **SEC**=YES) になっており、Liberty サーバーのセキュリティーが必要な場合には、この機能が必要です。
- `jsp-2.2` 機能は、サーブレットおよび JavaServer Pages (JSP) アプリケーションのサポートを有効にします。この機能は、CICS バンドルとしてインストールされる Web サポート付き OSGi バンドル・プロジェクトを含む OSGi アプリケーション・プロジェクトおよび動的 Web プロジェクト (WAR ファイル) で必要とされます。
- `wab-1.0` 機能は、エンタープライズ・バンドル (EBA) の内部にある Web アーカイブ・バンドル (WAB) のサポートを有効にします。この機能は、CICS バンドルとしてインストールされる Web サポート付き OSGi バンドル・プロジェクトを含む OSGi アプリケーション・プロジェクトで必要とされます。
- `cicsts:jdbc-1.0` 機能は、JDBC DriverManager または DataSource インターフェースを介してアプリケーションが DB2 にアクセスできるようにします。

例:

```
<featureManager>
    <feature>cicsts:core-1.0</feature>
    <feature>cicsts:security-1.0</feature>
    <feature>jsp-2.2</feature>
    <feature>wab-1.0</feature>
    <feature>cicsts:jdbc-1.0</feature>
</featureManager>
```

サポートされる機能について詳しくは、を参照してください。

## CICS バンドル・デプロイ・アプリケーション

CICS バンドルを使用する Liberty アプリケーションをデプロイするには、以下の項目を server.xml ファイルに含める必要があります。

```
<include location="${server.output.dir}/installedApps.xml"/>
```

インクルードされるファイルは、CICS バンドル・デプロイ・アプリケーションを定義するために使用されます。

## バンドル・リポジトリー

共通 OSGi バンドルを共用するには、それらを 1 つのディレクトリーに入れて、bundleRepository エレメントでそのディレクトリーを参照します。以下に例を示します。

```
<bundleRepository>
    <fileset dir="directory_path" include="*.jar"/>
</bundleRepository>
```

### グローバル・ライブラリー

Web アプリケーション間で共通の JAR ファイルを共用するには、それらを 1 つのディレクトリーに入れて、グローバル・ライブラリー定義でそのディレクトリーを参照します。

```
<library id="global">
    <fileset dir="directory_path" include="*.jar"/>
</library>
```

グローバル・ライブラリーは EBA 内の OSGi アプリケーションでは使用できません。その場合は、バンドル・リポジトリーを使用する必要があります。

### Liberty サーバー・アプリケーションおよび構成更新のモニター

Liberty JVM サーバーは server.xml ファイルをスキャンして更新があるかどうか調べます。デフォルトでは 500 ミリ秒ごとにスキャンします。この値を変更するには、次のような項目を追加します。

```
<config monitorInterval="5s" updateTrigger="polled"/>
```

さらに dropins ディレクトリーもスキャンして、アプリケーションの追加、更新、または削除を検出します。 CICS バンドルの中に Web アプリケーションをインストールする場合、次のように dropins ディレクトリーを無効にしてください。

```
<applicationMonitor updateTrigger="disabled" dropins="dropins"
dropinsEnabled="false" pollingRate="5s"/>
```

### JTA トランザクション・ログ

Liberty トランザクション・マネージャーは、Java Transaction API (JTA) を使用するときに、リカバリー可能ログ・ファイルを zFS ファイリング・システムに保管します。トランザクション・ログのデフォルトの場所は ${WLP_USER_DIR}/tranlog/ です。この場所は、

```
<transaction transactionLogDirectory="/u/cics/CICSPRD/DFHWLP/tranlog/"/>
```
のように transaction エレメントを server.xml に追加することでオーバーライドできます。

### CICS デフォルト Web アプリケーション

CICS デフォルト Web アプリケーション (CICSDefaultApp) は、Liberty JVM サーバーが開始したことを確認する構成サービスです。このアプリケーションを使用可能にするには、 <cicsts_defaultApp deployed="true" /> を server.xml に追加します。URL http://<server>:<port>/com.ibm.cics.wlp.defaultapp/ を使用して、アプリケーションを実行します。

### System Authorization Facility (SAF) レジストリー

新しい配布 ID 機能を使用している場合を除き、次の例を使用して、System Authorization Facility (SAF) レジストリーを定義する必要があります。

```
<safRegistry id="saf"/>
```

詳細については、WebSphere Application Server for z/OS 8.5.5 製品資料内の『Liberty プロファイル: server.xml ファイルの構成エレメント』を参照してください。

### CICS JTA の統合

Liberty JTA トランザクションが存在する場合、CICS は、デフォルトで Liberty JTA トランザクションに統合されています。server.xml 構成エレメントを使用して、この統合を使用不可にできます。

```
<cicsts_jta integration="true|false"/>
```

**関連タスク**:

143 ページの『構成ディレクトリー構造の作成』
JVM サーバーの zFS 内に構成ディレクトリー構造を作成します。

## Liberty 用のデフォルトの CICS DB2 JDBC タイプ 2 ドライバー・データ・ソースの作成

自動構成プロパティーを使用してデフォルトの CICS DB2 JDBC タイプ 2 ドライバー・データ・ソースを作成します。

### 始める前に

DB2 に接続するよう CICS 領域を構成する必要があります。詳しくは、『Configuring』の『Defining the CICS DB2 connection』を参照してください。

### このタスクについて

JVM プロファイルの自動構成プロパティーを使用して、Liberty の server.xml に CICS DB2 JDBC タイプ 2 ドライバー・データ・ソース・リソースを作成できます。デフォルトの構成は、CICS Explorer SDK の CICS Java DataBase Connectivity (JDBC) サーブレットおよび JSP のサンプルで使用できます。 JNDI 名は `jdbc/defaultCICSDataSource` です。

### 手順

1. JVM プロファイルに **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true** を設定して自動構成を有効にします。
2. JVM プロファイルで **com.ibm.cics.jvmserver.wlp.jdbc.driver.location** に DB2 JDBC ライブラリーの場所を設定 (**-Dcom.ibm.cics.jvmserver.wlp.jdbc.driver.location=/usr/lpp/db2v11/jdbc** など) して、デフォルトの CICS DB2 JDBC タイプ 2 ドライバー・データ・ソースの自動構成を有効にします。
3. JVMSERVER リソースをインストールして使用可能にします。

### タスクの結果

デフォルトの CICS DB2 JDBC タイプ 2 ドライバー・データ・ソースが、Liberty サーバー構成ファイル `server.xml` に追加されました。

```
<cicsts_dataSource id="defaultCICSDataSource" jndiName="jdbc/defaultCICSDataSource"/>
<library id="defaultCICSDb2Library">
    <fileset dir="/usr/lpp/db2v11/jdbc/classes" includes="db2jcc4.jar db2jcc_license_cisuz.jar"/>
    <fileset dir="/usr/lpp/db2v11/jdbc/lib" includes="libdb2jcct2zos4_64.so"/>
</library>
```

**関連概念**:

『Securing』の『Liberty JVM サーバーのセキュリティーの構成 (Configuring security for a Liberty JVM server)』

**関連タスク**:

151 ページの『Liberty 用の DB2 JDBC タイプ 4 ドライバー・データ・ソースの手動構成』

CICS の Liberty プロファイル・ユーザーは、Liberty から提供されるデータ・ソース定義を使用し、JDBC タイプ 4 ドライバーを介して Java アプリケーションから DB2 データベースにアクセスできます。

## Liberty 用の CICS DB2 JDBC タイプ 2 ドライバー・データ・ソースの手動構成

CICS の Liberty プロファイル・ユーザーは、JDBC タイプ 2 ドライバー・データ・ソース定義を使用して Java アプリケーションから DB2 データベースにアクセスできます。

### 始める前に

DB2 に接続するよう CICS 領域を構成する必要があります。詳しくは、『Configuring』の『Defining the CICS DB2 connection』を参照してください。

### このタスクについて

このタスクでは、 JDBC タイプ 2 ドライバー接続を有効にするために server.xml 構成ファイルで必要なエレメントを定義する方法を説明します。これにより、ローカル DB2 データベースへの接続を確立できます。

**注:**

JVM プロファイルで **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure** を true に設定し、**-Dcom.ibm.cics.jvmserver.wlp.jdbc.driver.location** を設定してから JVM サーバーを使用可能にすると、server.xml ファイル内の既存の JDBC 構成のサンプルがデフォルトの構成に置き換えられ、ユーザーによる更新内容は失われます。ユーザーによる更新内容が失われないようにするには、cicsts_dataSource 属性を allowautoconfigure="false" に設定します。

### 手順

1. **cicsts:jdbc-1.0** 機能を featureManager エレメントに追加します。これで、後で使用する server.xml ファイルの **cicsts_jdbcDriver** エレメントと **cicsts_dataSource** エレメントを使用できるようになります。

```
<featureManager>
    <feature>cicsts:jdbc-1.0</feature>
</featureManager>
```

2. **cicsts_jdbcDriver** エレメントを追加します。これで、**java.sql.DriverManager** または **javax.sql.DataSource** を使用して、JDBC タイプ 2 ドライバー・データ・ソースへのアクセスをサポートできるようになります。**cicsts_jdbcDriver** エレメントは、JDBC ドライバー・コンポーネント (DB2 JDBC jar およびネイティブ dll ファイル) のロード元ライブラリーを指定するライブラリー定義を参照する必要があります。標準的な定義は次のようになります。

```
<cicsts_jdbcDriver libraryRef="defaultCICSDb2Library"/>
<library id="defaultCICSDb2Library">
    <fileset dir="/usr/lpp/db2v11/jdbc/classes" includes="db2jcc4.jar db2jcc_license_cisuz.jar"/>
    <fileset dir="/usr/lpp/db2v11/jdbc/lib" includes="libdb2jcct2zos4_64.so"/>
</library>
```

**注:** 必要な **cicsts_jdbcDriver** エレメントは 1 つだけです。複数のエレメントが指定されている場合は、server.xml ファイルの最後のエレメントのみが使用され、その他のエレメントは無視されます。

3. **java.sql.DriverManager** サポートのみが必要な場合は、前のステップで十分です。データ・ソース定義を使用して DB2 接続にアクセスするには、アクセスを定義するデータ・ソースごとに **cicsts_dataSource** エレメントが必要になります。 **cicsts_dataSource** は jndiName 属性を指定します。これで、そのデータ・ソースへの接続を確立する際にアプリケーション・プログラムによって使用される参照である JNDI 名が定義されます。定義は次のようになります。

```
<cicsts_dataSource id="defaultCICSDataSource" jndiName="jdbc/defaultCICSDataSource"/>
```

**注:** 使用されるデータ・ソース・クラスは、**javax.sql.DataSource** を実装する **com.ibm.db2.jcc.DB2SimpleDataSource** です。

4. オプション: **properties.db2.jcc** エレメントを使用して、**cicsts_dataSource** のプロパティーを設定できます。次の例は、その方法を示しています。

```
<cicsts_dataSource id="defaultCICSDataSource" jndiName="jdbc/defaultCICSDataSource">
    <properties.db2.jcc currentSchema="DB2USER" fullyMaterializeLobData="true" />
</cicsts_dataSource>
```

**properties.db2.jcc** エレメントで許可されるすべてのプロパティーが、JDBC タイプ 2 ドライバー・データ・ソースに適しているわけではありません。不適切なプロパティーは無視されます。

**注:** 認識されないプロパティーは無視されます。次のプロパティーは、CICS JDBC タイプ 2 ドライバー・サポートでは適切ではありません。指定した場合は、これらも無視され、警告メッセージが出ます:
driverType、serverName、portNumber、user、password、databaseName。

## タスクの結果

Liberty サーバーは、始動時に、JDBC タイプ 2 ドライバー接続経由での DB2 データベースへのアクセスが許可されるように構成されます。

**注:** CICS データ・ソースおよびそのコンポーネントの動的更新はサポートされていません。Liberty サーバーの稼働中に構成を更新すると、DB2 アプリケーションの障害が生じる場合があります。変更をアクティブにするために、サーバーをリサイクルする必要があります。

**関連概念**:

『Securing』の『Liberty JVM サーバーのセキュリティーの構成 (Configuring security for a Liberty JVM server)』

**関連タスク**:

『Liberty 用の DB2 JDBC タイプ 4 ドライバー・データ・ソースの手動構成』
CICS の Liberty プロファイル・ユーザーは、Liberty から提供されるデータ・ソース定義を使用し、JDBC タイプ 4 ドライバーを介して Java アプリケーションから DB2 データベースにアクセスできます。

## Liberty 用の DB2 JDBC タイプ 4 ドライバー・データ・ソースの手動構成

CICS の Liberty プロファイル・ユーザーは、Liberty から提供されるデータ・ソース定義を使用し、JDBC タイプ 4 ドライバーを介して Java アプリケーションから DB2 データベースにアクセスできます。

### 始める前に

Liberty DB2 JDBC タイプ 4 ドライバー・データ・ソースは CICS DB2 接続リソースを使用しませんが、APAR PI18798 と PI18799 を適用していない場合は、DB2 SDSNLOAD および SDSNLOD2 ライブラリーを CICS STEPLIB 連結に追加する必要があります。

### このタスクについて

このタスクでは、JDBC タイプ 4 ドライバー接続を有効にするために `server.xml` 構成ファイルに必要なエレメントを手動で定義する方法を説明します。これにより、リモート DB2 データベースへの接続を確立できます。 JDBC タイプ 4 ドライバーを使用して DB2 データベースに行われる更新では CICS DB2 接続リソースが使用されないため、それらの更新は CICS の作業単位には含まれません。ただし、JTA ユーザー・トランザクション内で行われた更新は除きます。 109 ページの『Java Transaction API (JTA)』を参照してください。

### 手順

1. **jdbc-4.0** 機能を featureManager エレメントに追加します。これにより、`server.xml` ファイルで後で使用する **dataSource** および **jdbcDriver** エレメントが有効になります。

```
<featureManager>
    <feature>jdbc-4.0</feature>
</featureManager>
```

2. **dataSource** および **jdbcDriver** エレメントを追加します。 **dataSource** エレメントは、JDBC ドライバーのコンポーネント (DB2 JDBC jar およびネイティブ DLL ファイル) のロード元ライブラリーを示すライブラリー定義を参照する必要があります。標準的な定義は次のようになります。

```
<dataSource jndiName="jdbc/defaultCICSDataSource">
   <jdbcDriver libraryRef="db2Lib"/>
    <properties.db2.jcc driverType="4"
      serverName="winmvs2c.hursley.ibm.com"
        portNumber="41100"
        databaseName="DSNV11P2"
        user="DBUSER"
        password="{xor}Lz4sLCgwLTs="/>
</dataSource>
```

```
<library id="db2Lib">
    <fileset dir="/usr/lpp/db2v11/jdbc/classes" includes="db2jcc4.jar
        db2jcc_license_cisuz.jar" />
    </library>
```

APAR PI18798 と PI18799 を適用していない場合は、DB2 ネイティブ・ライブラリーのファイル・セット・エントリーをライブラリー構成に追加する必要があります。例えば、次のようにします。

```
<fileset dir="/usr/lpp/db2v11/jdbc/lib" />
```

**dataSource** は **jndiName** 属性を指定します。これで、そのデータ・ソースへの接続を確立する際にアプリケーション・プログラムによって使用される参照である JNDI 名が定義されます。必要なプロパティーが以下のように **properties.db2.jcc** エレメントに設定されます。

**driverType**
説明: データベース・ドライバーのタイプ。ピュア Java ドライバーを使用する場合は、4 に設定する必要があります。

デフォルト値: 4

必須: いいえ

データ型: int

**serverName**
説明 : データベースが稼働しているサーバーのホスト名。これは、DB2 DISPLAY DDF コマンドの SQL DOMAIN 値です。

デフォルト値: localhost

必須: いいえ

データ型: ストリング

**portNumber**
説明: データベース接続を取得するポート。これは、DB2 DISPLAY DDF コマンドの TCPPORT 値です。

デフォルト値: 50000

必須: いいえ

データ型: int

**databaseName**
説明: データ・ソースの名前を指定します。これは、DB2 DISPLAY DDF コマンドの LOCATION 値です。

必須 : はい

データ型: ストリング

**user** 説明: データベースへの接続に使用するユーザー ID。

必須 : はい

データ型: ストリング

**password**
説明: データベースへの接続に使用するユーザー ID のパスワード。値は、平文形式またはエンコード形式で保管することができます。パスワ

ードはエンコードすることをお勧めします。エンコードするには、encode オプションを指定して securityUtility ツールを使用します。WebSphere Application Server for z/OS 8.5.5 製品資料内の『Liberty プロファイル: securityUtility コマンド』を参照してください。

必須 : はい

データ型: ストリング

### タスクの結果

Liberty サーバーは、始動時に、JDBC タイプ 4 ドライバー接続経由での DB2 データベースへのアクセスが許可されるように構成されます。詳しくは、WebSphere Application Server for z/OS 8.5.5 製品資料内の『Liberty プロファイル: server.xml ファイルの構成エレメント』を参照してください。

**関連概念**:

『Securing』の『Liberty JVM サーバーのセキュリティーの構成 (Configuring security for a Liberty JVM server)』

**関連タスク**:

148 ページの『Liberty 用のデフォルトの CICS DB2 JDBC タイプ 2 ドライバー・データ・ソースの作成』
自動構成プロパティーを使用してデフォルトの CICS DB2 JDBC タイプ 2 ドライバー・データ・ソースを作成します。

149 ページの『Liberty 用の CICS DB2 JDBC タイプ 2 ドライバー・データ・ソースの手動構成』
CICS の Liberty プロファイル・ユーザーは、JDBC タイプ 2 ドライバー・データ・ソース定義を使用して Java アプリケーションから DB2 データベースにアクセスできます。

## Axis2 用の JVM サーバーの構成

Java Web サービスを実行するかまたは SOAP 要求をパイプラインで処理する場合、Axis2 を実行するように JVM サーバーを構成します。

### このタスクについて

Axis2 は、Web サービス要求をプロバイダーおよびリクエスターのパイプラインで処理できる Java SOAP エンジンです。 Axis2 を実行するように JVM サーバーを構成している場合、CICS は要求された JAR ファイルを自動的にクラスパスに追加します。

JVM サーバーは、CICS オンライン・リソース定義を使用するか、CICS バンドルに含めることにより定義できます。

### 手順

1. JVM サーバー用の JVMSERVER リソースを作成します。
   a. JVM サーバー用の JVM プロファイルの名前を指定します。 JVMSERVER の JVMPROFILE 属性では、1 文字から 8 文字の名前を指定します。この名

前は、JVM サーバーの構成オプションを保持するファイルである JVM プロファイルの接頭部に使用されます。ここでは接尾部として .jvmprofile を指定する必要はありません。

b. JVM サーバーのスレッド限度を指定します。 JVMSERVER の THREADLIMIT 属性では、JVM サーバーの Language Environment エンクレーブで許可されているスレッドの最大数を指定します。必要なスレッド数は、JVM サーバーで実行したいワークロードによって異なります。始めにデフォルト値を受け入れてから、環境を調整できます。1 つの JVM サーバーで最大 256 個のスレッドを設定できます。

2. JVM プロファイルを作成して、JVM サーバーの構成オプションを定義します。ベースとして、サンプル・プロファイル DFHJVMAX.jvmprofile を使用することができます。このプロファイルには、JVM サーバーを開始するために適したオプションのサブセットが含まれています。 JVM プロファイルのすべてのオプションと値については、157 ページの『JVM プロファイルの検証およびプロパティー』で説明されています。 158 ページの『JVM プロファイルのコーディング規則』のコーディング規則 (プロファイル名のコーディング規則を含む) に従ってください。

a. JVM プロファイルの場所を設定します。 JVM プロファイルは、システム初期設定パラメーター JVMPROFILEDIR で指定するディレクトリー内になければなりません。詳しくは、130 ページの『JVM プロファイルのロケーションの設定』を参照してください。

b. サンプル・プロファイルを次のように変更します。

- JAVA_HOME を IBM Java SDK のインストール先に設定します。
- JAVA_PIPELINE を Axis2 を実行するように設定します。
- CLASSPATH_SUFFIX を、Java で記述されている Axis2 アプリケーションおよび SOAP ハンドラーのクラスを指定するように設定します。
- WORK_DIR を、JVM サーバーからのメッセージ、トレース、および出力の適当な宛先ディレクトリーに設定します。
- TZ を、JVM サーバーからのメッセージに対するタイム・スタンプのタイム・ゾーンを指定するように設定します。英国の例は TZ=GMT0BST,M3.5.0,M10.4.0 です。

c. 変更内容を JVM プロファイルに保管します。 z/OS UNIX システム・サービスのファイル・システムでは、JVM プロファイルを EBCDIC として保管する必要があります。

3. JVMSERVER リソースをインストールして使用可能にします。

## タスクの結果

CICS は Language Environment エンクレーブを作成し、JVM プロファイルから JVM サーバーにオプションを渡します。 JVM サーバーが始動し、Axis2 JAR ファイルがロードされます。 JVM サーバーが始動を正常に完了すると、JVMSERVER リソースが ENABLED 状態でインストールされます。

エラーが発生する場合 (CICS が JVM プロファイルの検出も読み取りもできない場合など)、JVM サーバーは始動できません。 JVMSERVER リソースは DISABLED 状態でインストールされ、CICS はシステム・ログにエラー・メッセージを発行しま

す。詳しくは、251 ページの『第 8 章 Java アプリケーションのトラブルシューティング』を参照してください。

## 次のタスク

- DB2 や WebSphere MQ など、ネイティブ C ダイナミック・リンク・ライブラリー (DLL) ファイルを含むディレクトリーを指定します。これらのディレクトリーは、JVM プロファイルの LIBPATH_SUFFIX オプションで指定します。
- 『アプリケーションの開発』の『Web サービスでの Java の使用』の説明に従って、JVM サーバーで Web サービス要求を実行するように CICS を構成します。

  『JVM プロファイルの例』
  Axis2 を開始するように構成された JVM プロファイルの例。

## JVM プロファイルの例

Axis2 を開始するように構成された JVM プロファイルの例。

次の例では、Axis2 を開始するように構成された JVM プロファイルを抜粋して示しています。

```
#*********************************************************************
#
#                         Required parameters
#                         -------------------
#
# When using a JVM server, the set of CICS options that are supported
JAVA_HOME=/usr/lpp/java/J7.0_64
WORK_DIR=.
LIBPATH_SUFFIX=/usr/lpp/db2910/lib
...
#*********************************************************************
#
#                     JVM server specific parameters
#                     ------------------------------
#
JAVA_PIPELINE=YES
#
#*********************************************************************
#
#                               JVM options
#                               -----------
# The following option sets the Garbage collection Policy.
#
-Xgcpolicy:gencon
#
#*********************************************************************
#
#                 Setting user JVM system properties
#                 ----------------------------------
#
# -Dcom.ibm.cics.some.property=some_value
#
#*********************************************************************
#
#              Unix System Services Environment Variables
#              ------------------------------------------
#
JAVA_DUMP_OPTS="ONANYSIGNAL(JAVADUMP,SYSDUMP),ONINTERRUPT(NONE)"
#
#
```

# CICS セキュリティー・トークン・サービスのための JVM サーバーの構成

SAML トークンを検証して処理する場合は、CICS セキュリティー・トークン・サービスを実行するように JVM サーバーを構成します。

## このタスクについて

提供されるサンプルの DFHJVMST.jvmprofile は、CICS セキュリティー・トークン・サービスを実行する JVM サーバーに適しています。

JVM サーバーは、CICS オンライン・リソース定義を使用するか、CICS バンドルに含めることにより定義できます。 CICS Explorer を使用して CICS バンドル内のリソースを作成および編集する方法について詳しくは、CICS Explorer 製品資料内の『Working with bundles』を参照してください。

## 手順

JVM サーバー用の JVMSERVER リソースを作成します。

1. JVM サーバー用の JVM プロファイルの名前を指定します。 JVMPROFILE 属性では、1 文字から 8 文字の名前を指定します。この名前は、JVM サーバーの構成オプションを保持するファイルである JVM プロファイルの接頭部に使用されます。接尾部として .jvmprofile を指定する必要はありません。
2. JVM サーバーのスレッド限度を指定します。 スレッド数は、JVM サーバーで実行するワークロードによって異なります。始めにデフォルト値を受け入れてから、後で環境を調整できます。 1 つの JVM サーバーで最大 256 個のスレッドを設定できます。
3. JVM プロファイルを作成して、JVM サーバーの構成オプションを定義します。JVM プロファイルは、システム初期設定パラメーター JVMPROFILEDIR で指定するディレクトリー内になければなりません。ベースとして、サンプル・プロファイル DFHJVMST.jvmprofile を使用することができます。このプロファイルには、JVM サーバーを開始するために適したオプションのサブセットが含まれています。 DFHJVMST.jvmprofile は、インストール・ディレクトリーから JVMPROFILEDIR で指定するディレクトリーにコピーすることも、CICS Explorer で選択してターゲット・ディレクトリーに保管することもできます。

   JVM プロファイルのすべてのオプションと値については、157 ページの『JVM プロファイルの検証およびプロパティー』で説明されています。 158 ページの『JVM プロファイルのコーディング規則』のコーディング規則に従ってください。

   サンプル・プロファイルを次のように変更します。

   - JAVA_HOME を IBM Java SDK のインストール先に設定します。
   - WORK_DIR を、JVM サーバーからのメッセージ、トレース、および出力の適当な宛先ディレクトリーに設定します。
   - SECURITY_TOKEN_SERVICE を YES に設定します。
   - TZ を、JVM サーバーからのメッセージに対するタイム・スタンプのタイム・ゾーンを指定するように設定します。英国の例は TZ=GMT0BST,M3.5.0,M10.4.0 です。

4. 変更内容を JVM プロファイルに保管します。 USS ファイル・システムでは、JVM プロファイルを EBCDIC として保管する必要があります。

### タスクの結果

JVMSERVER リソースをインストールして使用可能にすると、CICS は Language Environment エンクレーブを作成し、JVM プロファイルから JVM サーバーにオプションを渡します。 JVM が始動し、OSGi フレームワークはすべての OSGi ミドルウェア・バンドルを解決します。 JVM サーバーが始動を正常に完了すると、JVMSERVER リソースが ENABLED 状態でインストールされます。

エラーが発生する場合、例えば、CICS が JVM プロファイルの検出も読み取りもできない場合、JVM サーバーは初期化できません。 JVMSERVER リソースは DISABLED 状態でインストールされ、CICS はエラー・メッセージを発行します。 251 ページの『第 8 章 Java アプリケーションのトラブルシューティング』を参照してください。

### 次のタスク

JVM サーバーをさらにカスタマイズできます。例を以下に示します。

- DB2 や WebSphere MQ など、ネイティブ C ダイナミック・リンク・ライブラリー (DLL) ファイルを含むディレクトリーを指定します。これらのディレクトリーは LIBPATH_SUFFIX オプションで指定します。
- 詳しくは、Configuring the CICS Security Token Serviceを参照してください。

## JVM プロファイルの検証およびプロパティー

JVM プロファイルには、開始時に JVM に渡される一連のオプションとシステム・プロパティーが含まれています。一部の JVM プロファイル・オプションは CICS 環境に固有であり、他の環境の JVM には使用されません。 CICS は、JVM サーバーの始動時に JVM プロファイルが正しくコーディングされていることを検証します。

JVM オプションについては、 161 ページの『CICS 環境における JVM のオプション』に説明があります。 CICS には、CICS でサポートされる JVM サーバー構成ごとのサンプル・プロファイルが用意されています。これらのサンプル・プロファイルには、最も一般的な JVM オプションのデフォルト値があります。サンプル・プロファイルは zFS の `/usr/lpp/cicsts/cicsts53/JVMProfiles/`に保管されています。

また、JVM プロファイルで z/OS UNIX システム・サービス環境変数を指定することもできます。有効な JVM オプションではない名前と値のペアは、z/OS UNIX システム・サービス環境変数として扱われ、エクスポートされます。 JVM プロファイルで指定される z/OS UNIX システム・サービス環境変数は、そのプロファイルで作成される JVM にのみ適用されます。

環境変数の例として、Liberty プロファイルの `WLP_INSTALL_DIR` 変数や、JVM のタイム・ゾーンを変更するための `TZ` 変数があります。

Java クラス・ライブラリーには、JVM プロファイルに設定できる他のシステム・プロパティーが含まれています。例えば、アプリケーションに独自のシステム・プロ

パティーが含まれている場合があります。 IBM Java 資料は、主要な Java 情報源です。 JVM システム・プロパティーについて詳しくは、IBM SDK, Java Technology Edition バージョン 7 の z/OS ユーザーズ・ガイドを参照してください。

『JVM プロファイルのコーディング規則』
任意の標準テキスト・エディターを使用して JVM プロファイルを編集することができます。JVM プロファイルをコーディングする際には、以下の規則に従ってください。

160 ページの『JVM プロファイル・オプションの検証』
CICS は、JVM プロファイルで指定されたかぎとなるオプションに関して JVM の開始時に必ず幾つかの検査を実行します。これらの検査は、JVM セットアップにおける問題の早期検出を可能にします。

161 ページの『CICS 環境における JVM のオプション』
JVM プロファイル内のオプションは JVM サーバーを始動するために CICS によって使用されます。一部のオプションは CICS に固有ですが、環境変数と Java システム・プロパティーを指定することもできます。

## JVM プロファイルのコーディング規則

任意の標準テキスト・エディターを使用して JVM プロファイルを編集することができます。JVM プロファイルをコーディングする際には、以下の規則に従ってください。

**JVM プロファイルの名前**

- 名前の最大長は 8 文字です。
- 名前は、z/OS UNIX システム・サービス内のファイルに有効な任意の名前にすることができます。 DFH で始まる名前を使用しないでください。これらの文字は CICS が使用するために予約されているからです。
- JVM プロファイルは UNIX ファイルであるため、大文字小文字が重要です。 CICS で名前を指定する場合は、z/OS UNIX ファイル名にあるのと同じ大文字と小文字の組み合わせを使用して名前を入力する必要があります。
- ファイル・システムでは JVM プロファイルのファイル拡張子を `.jvmprofile` にする必要があります。ファイル拡張子は小文字で設定され、これを変更してはなりません。

**ディレクトリー**

JVM プロファイルでディレクトリーの値を指定する場合は、引用符を使用しないでください。

**CEDA**

CEDA パネルは、端末の UCTRAN 設定にかかわらず、JVMPROFILE フィールドでの大文字小文字混合入力を受け入れます。ただし、コマンド行から CEDA を使用する場合、または別の CICS トランザクションを使用する場合は、大文字小文字混合で JVM プロファイルの名前を入力する必要があります。使用する端末が、大文字変換が抑止された状態で正しく構成されていることを確実にしてください。提供される CEOT トランザクションを使用して、現行セッションに対してのみ、独自の端末の英大文字変換状況 (UCTRAN) を変更することができます。

### 大/小文字の区別

パラメーター・キーワードおよびオペランドはすべて、大/小文字の区別があり、 161 ページの『CICS 環境における JVM のオプション』および 172 ページの『JVM システム・プロパティー』に示されているとおり正確に指定する必要があります。

### クラスパス分離文字

CLASSPATH_SUFFIX などのクラスパス・オプションで指定するディレクトリー・パスを分離するには、: (コロン) 文字を使用してください。

### 継続

JVM オプションの場合、値は、テキスト・ファイル内の行の終わりで区切られます。入力または編集しようとする値が、エディターのウィンドウには長すぎる場合は、スクロールしないですむように改行することができます。次の行に継続するには、次の例のように、現在行の終わりに円記号 (¥) 文字とブランクの継続文字を付けます。

```
CLASSPATH_SUFFIX=/u/example/pathToJarOrZipFile/jarfile.jar:¥
/u/example/pathToRootDirectoryForClasses
```

同じ行に複数の JVM オプションを置くことはできません。

### コメント

コメントを追加するか、オプションを削除する代わりにコメント化するには、コメントの各行の先頭に # 記号を付けます。ファイルが JVM ランチャーによって読み取られるときに、コメント行は無視されます。

ブランク行も無視されます。オプション間、またはオプションのグループ間の分離文字としてブランク行を使用できます。

プロファイル構文解析コードは、UNIX のようなシェル処理に基づいてインライン・コメントを削除するため、サンプル JVM プロファイルの文書処理が改善されます。インライン・コメントは以下のように定義されます。

- コメントの先頭に # 記号が付いている。
- 前に 1 つ以上のスペース (またはタブ) がある。
- 引用符付きテキストには含まれない。

*表 14. インライン・コメントの例*

| コード | 結果 |
|---|---|
| `MYVAR=myValue  # Comment` | MYVAR=myValue |
| `MYVAR=#myValue  # Comment` | MYVAR=#myValue |
| `MYVAR=myValue  "# Quoted comment" # Comment` | MYVAR=myValue "# Quoted comment" |

### 文字のエスケープ・シーケンス

160 ページの表 15 に示されているエスケープ・シーケンスをコーディングできます。

*表 15. エスケープ・シーケンス*

| エスケープ・シーケンス | 文字の値 |
|---|---|
| ¥b | バックスペース |
| ¥t | 水平タブ |
| ¥n | 改行 |
| ¥r | 復帰 |
| \" | 二重引用符 |
| \' | 単一引用符 |
| \\ | 円記号 (¥) |
| ¥*xxx* | 8 進値 *xxx* に相当する文字。ここで、*xxx* は 000 から 377 の値です。 |
| ¥u*xxxx* | *xxxx* をエンコードする Unicode 文字。ここで、*xxxx* は 1 桁から 4 桁の 16 進数字です (詳しくは、注を参照してください)。 |

**注:** Unicode ¥u エスケープは、他のエスケープ・タイプとは異なります。Unicode エスケープ・シーケンスは、表 15 で記述されている他のエスケープ・シーケンスより前に処理されます。Unicode エスケープは、非 Unicode システムで表示できない文字を表す代替方法です。ただし、文字エスケープは特殊文字を表すことができますが、それらの文字は通常どおりに解釈されません。

**オプションの複数インスタンス**

同じオプションの複数のインスタンスが JVM プロファイルに含まれている場合、最後に検出されるオプションの値が使用され、それより前の値は無視されます。

**ストレージ・サイズ**

JVM プロファイルでストレージ関連のオプションを指定する場合、ストレージ・サイズを 1024 バイトの倍数で指定してください。文字 K は KB を、文字 M は MB を、および文字 G は GB をそれぞれ表します。例えば、ヒープの初期サイズとして 6 291 456 バイトを指定するには、以下のいずれかの方法で **-Xms** をコーディングします。

```
-Xms6144K
-Xms6M
```

## JVM プロファイル・オプションの検証

CICS は、JVM プロファイルで指定されたかぎとなるオプションに関して JVM の開始時に必ず幾つかの検査を実行します。これらの検査は、JVM セットアップにおける問題の早期検出を可能にします。

CICS は、以下の JVM プロファイル・オプションに関して検査を行います。

**CLASSPATH_PREFIX、CLASSPATH_SUFFIX**
これらのオプションのいずれかが JVM プロファイルにある場合は、JVM サーバーは OSGi フレームワークなしで始動します。

**JAVA_HOME**
CICS は、このディレクトリーの以下の点を検査します。

- ディレクトリーが z/OS UNIX に存在すること。
- CICS が、ディレクトリーにアクセスするために少なくとも読み取り権限を持っていること。
- JDK_INSTALL_OK ファイルがディレクトリー内にあること。これは、この場所でIBM 64-bit SDK for z/OS, Java テクノロジー・エディション 7 ファイルのインストールが完了したことを示します。
- JDK_INSTALL_OK ファイルの Java リリース番号が、CICS によってサポートされているバージョンであること。

何らかの問題が見つかると、CICS はエラー・メッセージを発行して JVM は開始されません。

**OSGI_BUNDLES**
JVM サーバー・プロファイルの場合、CICS は、指定された JAR ファイルが OSGi バンドルであることを検査します。また、CICS は、ミドルウェア・バンドルが正しく区切られ、正しい分離文字があることも検査します。

**LIBPATH、CLASSPATH は、使用すべきでないクラスパス・オプションです。**
JVM プロファイルにこれらのオプションが 1 つ以上ある場合、JVM の開始時に警告メッセージが出されます。JVM プロファイルでこれらのオプションを使用しないでください。メッセージには、代わりに使用する正しいオプションが表示されます。

**CICS_HOME、TMPREFIX、TMSUFFIX は、使用すべきでないクラスパス・オプションです。**
これらのオプションは無視されます。

## CICS 環境における JVM のオプション

JVM プロファイル内のオプションは JVM サーバーを始動するために CICS によって使用されます。一部のオプションは CICS に固有ですが、環境変数と Java システム・プロパティーを指定することもできます。

### コーディング規則

JVM オプションを指定する際には、コーディング規則に従っていることを確認してください。詳しくは、158 ページの『JVM プロファイルのコーディング規則』を参照してください。

### フォーマット

オプションの形式は、変わる場合があります。

- JVM プロファイル内のオプションでは、キーワードと値が等号 (=) によって区切られる形式 (例: `JAVA_PIPELINE=TRUE`) またはハイフンで始まる形式 (例: `-Xmx16M`) が使用されます。
- キーワードと値のペアは、`JAVA_PIPELINE=TRUE` などの CICS 変数であるか、あるいは CICS オプションとして認識されない場合には z/OS UNIX System Services の環境変数として扱われ、エクスポートされます。
- JVM プロファイル内の `-D` で始まるオプションは JVM システム・プロパティーです。 `-X` で始まるオプションは JVM コマンド行オプションとして扱われま

す。 - で始まるオプションは、置換シンボルが展開された後に JVM に渡されます。詳しくは、 172 ページの『JVM システム・プロパティー』を参照してください。

『JVM オプションに使用するシンボル』
JVM プロファイルに指定する変数または JVM サーバー・プロパティーでは、置換シンボルを使用できます。これらのシンボルの値は JVM サーバーの始動時に決定されるため、多くの JVM サーバーおよび CICS 領域で共通のプロファイルを使用できます。

163 ページの『JVM サーバー・オプション』
JVM サーバー・オプション、およびその説明と JVM サーバーのさまざまな用途への適用可能性。

171 ページの『JVM コマンド行オプション』
JVM コマンド・ライン・オプション、およびその説明。

172 ページの『JVM システム・プロパティー』
JVM システム・プロパティーは、JVM とそのランタイム環境に固有の構成情報を提供します。 JVM システム・プロパティーを指定するには、それらを JVM プロファイルに追加します。実行時に、CICS はプロパティーを JVM プロファイルから読み取り、JVM に渡します。

178 ページの『時間帯の設定』
TZ 環境変数はシステムの「ローカル」時間を指定します。この変数を JVM プロファイルに追加することで、JVM サーバーのローカル時間を設定できます。

**JVM オプションに使用するシンボル:**

JVM プロファイルに指定する変数または JVM サーバー・プロパティーでは、置換シンボルを使用できます。これらのシンボルの値は JVM サーバーの始動時に決定されるため、多くの JVM サーバーおよび CICS 領域で共通のプロファイルを使用できます。

以下のシンボルがサポートされています。

**&APPLID;**
このシンボルを使用すると、実行時に CICS 領域のアプリケーション ID に置換されます。この方法ですべての領域に同じプロファイルを使用でき、同時に領域固有の作業ディレクトリーまたは出力宛先を用いることができます。APPLID は常に大文字です。

**&CONFIGROOT;**
このシンボルを使用すると、JVM プロファイルが入っているディレクトリーの絶対パスが実行時に置換されます。 CICS バンドル内で定義される JVM サーバーの場合、JVM プロファイルはデフォルトでバンドルのルート・ディレクトリーの中に置かれます。他の方法で定義される JVM サーバーの場合、JVM プロファイルは JVMPROFILEDIR システム初期設定パラメーターで指定されたディレクトリーの中に置かれます。

**&DATE;**
このシンボルを使用すると、シンボルは実行時に *Dyymmdd* 形式の現在日付で置き換えられます。

**&JVMSERVER;**
このシンボルを使用する際、JVMSERVER リソースの名前は実行時に置換されます。JVM サーバーごとに固有の出力またはダンプ・ファイルを作成する場合に、このシンボルを使用します。

**&TIME;**
このシンボルを使用すると、シンボルは実行時に *Thhmmss* 形式の JVM 開始時刻で置き換えられます。

**&USSHOME;**
このシンボルを使用すると、シンボルは USSHOME システム初期設定パラメーターの値に置き換えられます。このシンボルを指定することで、CICS が Java と Liberty プロファイル用にライブラリーを提供する、z/OS UNIX のホーム・ディレクトリーを自動的に取得できます。

**JVM サーバー・オプション:**

JVM サーバー・オプション、およびその説明と JVM サーバーのさまざまな用途への適用可能性。

**さまざまな用途の JVM サーバーに対するオプションの使用可否**

JVM サーバーの用途に応じて、さまざまなオプションを利用可能です。次の表に、JVM サーバーの特定の用途に対して、各オプションが必須であるか、サポート対象だが任意指定であるか、それともサポート対象外であるかを示します。

*表 16. オプションとJVM サーバーのさまざまな用途*

| オプション | OSGi | Liberty | Axis2 | STS |
|---|---|---|---|---|
| CLASSPATH_PREFIX | サポートされていない | サポートされていない | サポートされる | サポートされる |
| CLASSPATH_SUFFIX | サポートされていない | サポートされていない | サポートされる | サポートされる |
| IDENTITY_PREFIX | サポートされる | サポートされる | サポートされる | サポートされる |
| DISPLAY_JAVA_VERSION | サポートされる | サポートされる | サポートされる | サポートされる |
| JAVA_DUMP_TDUMP_PATTERN | サポートされる | サポートされる | サポートされる | サポートされる |
| JAVA_HOME | 必須 | 必須 | 必須 | 必須 |
| JAVA_PIPELINE | サポートされていない | サポートされていない | 必須 | サポートされていない |
| JNDI_REGISTRATION | サポートされる | サポートされていない | サポートされる | サポートされる |
| JVMTRACE | サポートされる | サポートされる | サポートされる | サポートされる |
| LIBPATH_PREFIX | サポート対象 - IBM サービス担当員の指示があった場合にのみ使用 | サポート対象 - IBM サービス担当員の指示があった場合にのみ使用 | サポート対象 - IBM サービス担当員の指示があった場合にのみ使用 | サポート対象 - IBM サービス担当員の指示があった場合にのみ使用 |
| LIBPATH_SUFFIX | サポートされる | サポートされる | サポートされる | サポートされる |
| LOG_FILES_MAX | サポートされる | サポート対象 | サポート対象 | サポート対象 |
| LOG_PATH_COMPATIBILITY | サポートされる | サポートされる | サポートされる | サポートされる |

*表 16. オプションとJVM サーバーのさまざまな用途 (続き)*

| オプション | OSGi | Liberty | Axis2 | STS |
|---|---|---|---|---|
| OSGI_BUNDLES | サポートされる | サポートされていない | サポートされていない | サポートされていない |
| OSGI_FRAMEWORK_TIMEOUT | サポートされる | サポートされる | サポートされていない | サポートされていない |
| PRINT_JVM_OPTIONS | サポートされる | サポートされる | サポートされる | サポートされる |
| SECURITY_TOKEN_SERVICE | サポートされていない | サポートされていない | サポートされていない | 必須 |
| STDERR | サポートされる | サポートされる | サポートされる | サポートされる |
| STDIN | サポートされる | サポートされる | サポートされる | サポートされる |
| STDOUT | サポートされる | サポートされる | サポートされる | サポートされる |
| USEROUTPUTCLASS | サポートされる | サポートされていない | サポートされる | サポートされる |
| WLP_INSTALL_DIR | サポートされていない | 必須 | サポートされていない | サポートされていない |
| WLP_OUTPUT_DIR | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| WLP_USER_DIR | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| WLP_ZOS_PLATFORM | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| WORK_DIR | サポートされる | サポートされる | サポートされる | サポートされる |
| WSDL_VALIDATOR | サポートされる | サポートされていない | サポートされる | サポートされる |

**オプションの説明**

デフォルト値 (該当する場合) とは、オプションが指定されていない場合に CICS で使用される値のことです。一部またはすべてのサンプル JVM プロファイルで、デフォルト値とは異なる値が指定される場合があります。

**注:** これまでの資料に記載されていた YES|NO もまだ使用可能ですが、推奨される構文は TRUE|FALSE です。

**CLASSPATH_PREFIX, CLASSPATH_SUFFIX=***class_pathnames*

これらのオプションを使用して、OSGi 非対応の JVM によって検索されるディレクトリー・パス、Java アーカイブ・ファイル、圧縮ファイルを指定します。例えば、Java Web サービスに使用されます。 OSGi フレームワークは自動的にクラス・ロードを処理するので、OSGi フレームワークを使用する場合はクラスパスを設定しないでください。これらのオプションを使用して、Axis2 の標準クラスパスを指定する場合、Axis2 エンジンを開始するために、JAVA_PIPELINE=TRUE を指定する必要もあります。

CLASSPATH_PREFIX は、クラスパス・エントリーを標準クラスパスの先頭に追加し、CLASSPATH_SUFFIX は標準クラスパスの末尾に追加します。複数行にまたがってエントリーを指定するには、継続行の最後に \ (バックスラッシュ) を使用します。

CLASSPATH_PREFIX オプションは注意して使用してください。CLASSPATH_PREFIX のクラスは、CICS と Java ランタイムによって提供される、同じ名前のクラスより優先されるため、誤ったクラスがロードされる可能性があります。

CICS は、**USSHOME** システム初期設定パラメーターと JVM プロファイルの JAVA_HOME オプションで指定されたディレクトリーの /lib サブディレクトリーを使用して、JVM の基本クラスパスを作成します。この基本クラスパスには、CICS および JVM によって提供される Java アーカイブ・ファイルが含まれます。それは JVM プロファイルでは見られません。 JVM プロファイルのクラスパスでこれらのファイルを再度指定することはありません。

オプション CLASSPATH_PREFIX または CLASSPATH_SUFFIX を使用して複数の項目を指定する場合は、コンマではなくコロンを使用して項目を区切ってください。

**DISPLAY_JAVA_VERSION={TRUE|FALSE}**
このオプションを TRUE に設定すると、アプリケーションによって JVM が始動されるたびに、CICS は、使用される IBM Software Developer Kit for z/OS、Java Technology Edition のバージョンとビルドを示すメッセージ DFHSJ0901 を MSGUSER ログに書き込みます。

**IDENTITY_PREFIX={TRUE|FALSE}**
JVM サーバー出力の発信元を示すために、JES に転送されるすべての STDOUT および STDERR 項目は、接頭辞として JVM サーバー名の文字列を付けて書き込まれます。これは、複数の JVM サーバーで 1 つの JES 宛先を共用する場合に便利です。この動作を無効にするには、IDENTITY_PREFIX=FALSE を設定します。この設定により、接頭辞文字列が使用不可になります。

**JAVA_DUMP_TDUMP_PATTERN=**
JVM からのトランザクション・ダンプ (TDUMP) に使用されるファイル名を指定する z/OS UNIX システム・サービス環境変数。 Java TDUMP は、JVM 異常終了の場合にデータ・セット宛先に書き込まれます。

**JAVA_HOME=/usr/lpp/java/*javadir*/**
z/OS UNIX で IBM 64-bit SDK for z/OS, Java テクノロジー・エディション のインストール場所を指定します。この場所には、Java サポートに必要なサブディレクトリーと Java アーカイブ・ファイルが入ります。

提供されたサンプル JVM プロファイルには、DFHISTAR CICS インストール・ジョブで **JAVADIR** パラメーターによって生成されたパスが含まれています。**JAVADIR** パラメーターのデフォルトは、java/J7.0_64/ であり、これは、IBM 64-bit SDK for z/OS, Java テクノロジー・エディション のデフォルトのインストール場所です。この値では、JVM プロファイルの JAVA_HOME 設定値は /usr/lpp/java/J7.0_64/ になります。

**JAVA_PIPELINE={TRUE|FALSE}**
Java 標準 SOAP パイプラインでの Web サービス処理を JVM サーバーがサポートできるように、必要な Java アーカイブ・ファイルをクラスパスに追加します。デフォルト値は FALSE です。この値を設定すると、JVM サーバーは、

OSGi ではなく、Axis2 をサポートするように構成されます。CLASSPATH オプションを使用して、クラスパスに JAR ファイルをさらに追加することができます。

**注:** オプション JAVA_PIPELINE=TRUE と SECURITY_TOKEN_SERVICE=TRUE を同時に指定することはできません。

**JNDI_REGISTRATION={TRUE|FALSE}**
Java アプリケーションによる JNDI の使用をサポートするために、JNDI 登録 JAR ファイルを JVM ランタイム環境に自動的に追加することを指定します。Liberty JVM サーバーに関しては、このオプションは無視されます。JNDI_REGISTRATION=FALSE を設定すると、これらのファイルの自動追加をオプトアウトできます。この機能が不要な場合は、オプトアウトすることで、新しい JAR ファイルとの競合の可能性を排除し、JVM のフットプリントを小さく維持して不要なクラス・ロードを回避できます。

**JVMTRACE={{&APPLID;.&JVMSERVER;.}Dyyyymmdd.Thhmmss.dfhjvmtrc|*file_name*|JOBLOG|//DD:data_definition}**
JVM サーバーの稼働中に Java トレースを書き込む z/OS UNIX ファイルまたは JES DD の名前を指定します。このオプションに値を設定しない場合、CICS によって自動的に、JVM サーバーごとに固有のトレース・ファイルが作成されます。LOG_PATH_COMPATABILITY=FALSE を指定すると、ファイルは WORK_DIR/applid/jvmserver ディレクトリーに置かれます。LOG_PATH_COMPATABILITY=TRUE を指定すると、ファイルは WORK_DIR ディレクトリーに置かれます。JVMTRACE がデフォルトのままである場合、LOG_PATH_COMPATABILITY 値は JVMTRACE に影響を与えます。CICS は APPLID と JVMSERVER シンボル、および JVM サーバーが始動した日時のタイムスタンプを使用します。JES の DD に転送するには、構文 //DD:data_definition を使用して、JES のデータ定義名を指定します。このオプションを JOBLOG に設定すると、JVMTRACE は SYSPRINT 宛先に転送されます。

**LIBPATH_PREFIX, LIBPATH_SUFFIX=***pathnames*
JVM によって使用される、z/OS UNIX で拡張子 .so を持つネイティブ C ダイナミック・リンク・ライブラリー (DLL) ファイルを検索するときの対象となるディレクトリー・パスを指定します。これには、アプリケーション・コードまたはサービスによってロードされる JVM および追加のネイティブ・ライブラリーを実行するのに必要なファイルが含まれます。

JVM の基本ライブラリー・パスは、**USSHOME** システム初期設定パラメーターと JVM プロファイルの JAVA_HOME オプションで指定されたディレクトリーを使用して自動的に作成されます。この基本ライブラリー・パスは、JVM プロファイルでは表示されません。このライブラリー・パスには、CICS が使用する JVM とネイティブ・ライブラリーを実行するのに必要なすべての DLL ファイルが含まれています。

LIBPATH_SUFFIX オプションを使用すると、このライブラリー・パスを拡張できます。このオプションはディレクトリーを、ライブラリー・パスの最後、基本ライブラリー・パスの後に追加します。このオプションは、アプリケーションで使用する追加のネイティブ・ライブラリーが含まれるディレクトリーを指定するのに使用します。また、このオプションは、CICS の標準 JVM セットアップに含まれていないサービスで使用するディレクトリーを指定するのに使用します。例

えば、追加のネイティブ・ライブラリーには、DB2 JDBC ドライバーを使用するのに必要な DLL ファイルを含めることができます。

LIBPATH_PREFIX オプションは、ライブラリー・パスの先頭で、基本ライブラリー・パスの前にディレクトリーを追加します。このオプションは注意して使用してください。 指定されたディレクトリー内の DLL ファイルが、基本ライブラリー・パス上の DLL ファイルと同じ名前である場合は、提供されたファイルの代わりにロードされます。

オプション LIBPATH_PREFIX または LIBPATH_SUFFIX を使って複数の項目を指定するときには、コンマではなくコロンを使ってそれらを区切ってください。

アプリケーションで使用するためにライブラリー・パス上にある DLL ファイルをコンパイルし、XPLink オプションを指定してリンクする必要があります。XPLink オプションを指定してコンパイルおよびリンクすると、最適なパフォーマンスが得られます。基本ライブラリー・パスで提供される DLL ファイルおよび DB2 JDBC ドライバーなどのサービスで使用される DLL ファイルは、XPLink オプションを指定して作成されます。

**LOG_FILES_MAX={0|*number*}**
システム上に保持する古いログ・ファイルの数を指定します。デフォルト設定の 0 では、すべての古いバージョンのログ・ファイルが保持されます。この値を変更して、ファイル・システム上に保持する古いログ・ファイルの数を指定できます。

**LOG_PATH_COMPATABILITY=TRUE** の場合、**LOG_FILES_MAX** は無視されます。

If **STDOUT**、 **STDERR**、および **JVMTRACE** がデフォルト・スキームを使用している場合、またはカスタマイズされている場合は、&DATE;.&TIME; パターンを含めると、各ログ・タイプの最新の nn のみがシステムに保持されます。カスタマイズに出力を固有にする変数が含まれていない場合は、ファイルが追加され、削除の要求はありません。出力変数がカスタマイズされ、DD:// または JOBLOG に出力するよう経路指定されている場合、クリーンアップは適用されません。特殊値 0 は、削除しないことを意味します。

**LOG_PATH_COMPATABILITY={TRUE|FALSE}**
この動作のデフォルト値は LOG_PATH_COMPATABILITY=FALSE です。この場合は、統合されたログ出力動作になります。この新しい動作では、**JVMSERVER** ログ・ファイルは、**JVMSERVER** の既存のサブコンポーネント (OSGi フレームワークや Liberty サーバーなど) によって使用されるものと同じ出力ディレクトリー構造に配置されます。以前のリリースの動作に戻すには、このパラメーターを LOG_PATH_COMPATABILITY=TRUE に設定します。すると、**JVMSERVER** ログ・ディレクトリーは元の場所に作成されます。

**OSGI_BUNDLES=*pathnames***
OSGi JVM サーバーの OSGi フレームワークで有効なミドルウェア・バンドルのディレクトリー・パスを指定します。これらの OSGi バンドルには、WebSphere MQ または DB2 への接続などのシステム機能をフレームワークに実装するためのクラスが含まれています。複数の OSGi バンドルを指定する場合は、コンマを使用してそれらのバンドルを分離してください。

**OSGI_FRAMEWORK_TIMEOUT={60|*number*}**
OSGi フレームワークの初期設定またはシャットダウンがタイムアウトになるまでに CICS が待機する秒数を指定します。1 秒から 60000 秒までの範囲で値を

設定できます。デフォルト値は 60 秒です。 OSGi フレームワークが開始するのに、指定された秒数よりも長くかかる場合、JVM サーバーは初期設定できず、DFHSJ0215 メッセージが CICS によって発行されます。 zFS の JVM サーバー・ログ・ファイルにもエラー・メッセージが書き込まれます。OSGi フレームワークがシャットダウンするのに、指定された秒数よりも長くかかる場合、JVM サーバーは正常にシャットダウンできません。

**PRINT_JVM_OPTIONS={TRUE|FALSE}**
このオプションを TRUE に設定すると、JVM が始動するたびに、始動時の JVM に渡されるオプションも SYSPRINT に出力されます。 JVM がプロファイル内でこのオプションを指定して開始されるたびに、出力が生成されます。このオプションを使用すると、JVM プロファイルでは見えない、特定の JVM プロファイルのクラスパス (CICS によって作成される基本ライブラリー・パスと基本クラスパスを含む) の内容を確認できます。

**SECURITY_TOKEN_SERVICE={TRUE|FALSE}**
このオプションを TRUE に設定すると、JVM サーバーはセキュリティー・トークンを使用できます。このオプションを FALSE に設定すると、JVM サーバーのセキュリティー・トークン・サービスのサポートは無効になります。

**注:** オプション SECURITY_TOKEN_SERVICE=TRUE と JAVA_PIPELINE=TRUE を同時に指定することはできません。

**STDERR={{&APPLID;.&JVMSERVER;.}Dyyyymmdd.Thhmmss.dfhjvmerr|*file_name*|JOBLOG|//DD:data_definition}**
stderr ストリームをリダイレクトする z/OS UNIX ファイルまたは JES DD の名前を指定します。LOG_PATH_COMPATABILITY=FALSE を指定すると、ファイルは WORK_DIR/applid/jvmserver ディレクトリーに置かれます。LOG_PATH_COMPATABILITY=TRUE を指定すると、ファイルは WORK_DIR ディレクトリーに置かれます。STDERR がデフォルトのままである場合、LOG_PATH_COMPATABILITY 値は STDERR に影響を与えます。ファイルが存在する場合には、そのファイルの末尾に出力が追加されます。このオプションに値を設定しない場合、CICS によって自動的に、JVM サーバーごとに固有の出力ファイルが作成されます。CICS は APPLID と JVMSERVER シンボル、および JVM サーバーが始動した日時のタイムスタンプを使用します。JVM サーバーごとに固有の出力ファイルを作成するには、サンプルの JVM プロファイルに示されているように JVMSERVER シンボルと APPLID シンボルをファイル名に使用します。JES の DD に転送するには、構文 //DD:data_definition を使用して、JES のデータ定義名を指定します。このオプションを JOBLOG に設定すると、stderr は SYSOUT 宛先に転送されます。

JVM プロファイルで USEROUTPUTCLASS オプションを指定すると、このオプションで指定された Java クラスが、代わりに System.err 要求を処理します。USEROUTPUTCLASS オプションで指定されたクラスが目的の宛先にデータを書き込めない場合、STDERR オプションで指定された z/OS UNIX ファイルが引き続き使用されることがあります。提供されているサンプル・クラス com.ibm.cics.samples.SJMergedStream を使用する場合には、これが当てはまります。またこのファイルは、USEROUTPUTCLASS オプションで指定されたクラスによって、他の何らかの理由のために出力がそのファイルに送信される場合にも使用できます。

**STDIN=***file_name*
stdin ストリームの読み取り元の z/OS UNIX ファイルの名前を指定します。このオプションに値が指定されない限り、CICS はこのファイルを作成しません。

**STDOUT={{&APPLID;.&JVMSERVER;.}Dyyyymmdd.Thhmmss.dfhjvmout|***file_name***|JOBLOG|//DD:data_definition}**
stdout ストリームをリダイレクトする z/OS UNIX ファイルまたは JES DD の名前を指定します。LOG_PATH_COMPATABILITY=FALSE を指定すると、ファイルは WORK_DIR/applid/jvmserver ディレクトリーに置かれます。LOG_PATH_COMPATABILITY=TRUE を指定すると、ファイルは WORK_DIR ディレクトリーに置かれます。STDOUT がデフォルトのままである場合、LOG_PATH_COMPATABILITY 値は STDOUT にのみ影響を与えます。

ファイルが存在する場合には、そのファイルの末尾に出力が追加されます。このオプションに値を設定しない場合、CICS によって自動的に、JVM サーバーごとに固有の出力ファイルが作成されます。CICS は &APPLID; および &JVMSERVER; シンボルと、JVM サーバーが始動した日時を示すタイム・スタンプを使用します。JES の DD に転送するには、構文 //DD:data_definition を使用して、JES のデータ定義名を指定します。このオプションを JOBLOG に設定すると、stdout ストリームは SYSPRINT 宛先に転送されます。

JVM プロファイルで USEROUTPUTCLASS オプションを指定すると、このオプションで指定された Java クラスが、代わりに System.out 要求を処理します。USEROUTPUTCLASS オプションで指定されたクラスが目的の宛先にデータを書き込めない場合、STDOUT オプションで指定された z/OS UNIX ファイルが引き続き使用される可能性があります。例えば、サンプル・クラス com.ibm.cics.samples.SJMergedStream を使用する場合です。またこのファイルは、USEROUTPUTCLASS オプションで指定されたクラスによって、他の何らかの理由のために出力がそのファイルに送信される場合にも使用できます。

**USEROUTPUTCLASS=*classname***
JVM からの出力および JVM 内部からのメッセージを代行受信する Java クラスの完全修飾名を指定します。この Java クラスを使って JVM からの出力とメッセージをリダイレクトし、出力レコードにタイム・スタンプとヘッダーを追加することができます。これは Liberty ではサポートされていません。Java クラスがデータを目的の宛先に書き込めない場合、STDOUT および STDERR オプションで指定されたファイルが引き続き使用される場合があります。

USEROUTPUTCLASS オプションを指定すると、JVM のパフォーマンスに悪影響が出ます。実稼働環境で最高のパフォーマンスを発揮するには、このオプションは使用しないでください。ただし、このオプションは、JVM 出力を識別可能な宛先に送信できるので、同じ CICS 領域を使用するアプリケーション開発者に便利な場合があります。

このクラスおよび提供されたサンプルについて詳しくは、 257 ページの『JVM stdout、stderr、JVMTRACE、およびダンプ出力の場所の制御』を参照してください。

**WLP_INSTALL_DIR={&USSHOME;/wlp}**
Liberty プロファイル・テクノロジーのインストール・ディレクトリーを指定します。 Liberty プロファイルは、CICS の z/OS UNIX ホームにある wlp というサブディレクトリーにインストールされます。デフォルトのインストール・デ

ィレクトリーは /usr/lpp/cicsts/cicsts53/wlp です。常に &USSHOME; シンボルを使用して、正しいファイル・パスを設定し、wlp ディレクトリーを追加します。

この環境変数は、Liberty JVM サーバーを始動する場合に必要です。この環境変数を設定すると、Liberty JVM サーバーを構成するための他の環境変数およびシステム・プロパティーも指定できます。環境変数の接頭部は WLP です。また、システム・プロパティーについては、172 ページの『JVM システム・プロパティー』に説明されています。

**WLP_OUTPUT_DIR=$WLP_USER_DIR/servers**
Liberty プロファイルの出力ファイルを格納するディレクトリーを指定します。デフォルトでは、Liberty プロファイルは、サーバーにちなんだ名前を持つディレクトリー内に、そのサーバーに関するログ、作業域、構成ファイル、アプリケーションを格納します。

この環境変数はオプションです。これを指定しない場合、CICS はデフォルトで *$WORK_DIR/&APPLID;/&JVMSERVER;*/wlp/usr/servers を設定し、シンボルをランタイム値に置き換えます。

この環境変数が設定されている場合、出力ログと作業域は $WLP_OUTPUT_DIR/*server_name* に保管されます。

**WLP_USER_DIR={&APPLID;/&JVMSERVER;/wlp/usr/|*directory_path*}**
Liberty JVM サーバーの構成ファイルを格納するディレクトリーを指定します。この環境変数はオプションです。これを指定しない場合、CICS は作業ディレクトリー内の *&APPLID;/&JVMSERVER;*/wlp/usr/ を使用し、シンボルをランタイム値に置き換えます。構成ファイルは servers/*server_name* に書き込まれます。

**WLP_ZOS_PLATFORM={TRUE|FALSE}**
Liberty JVM サーバーでの z/OS プラットフォーム拡張を無効にします。これにより cicsts:security-1.0 機能と cicsts:distributedIdentity-1.0 機能が使用されなくなり、同じ領域内で複数の Liberty JVM サーバーを開始できるようになります。

**WORK_DIR={.|/tmp|*directory_name*}**
JVMSERVER に関連するアクティビティー用に CICS 領域が使用する z/OS UNIX 上の作業ディレクトリーを指定します。 CICS JVMSERVER は、このディレクトリーを構成および出力の手段として使用します。提供された JVM プロファイルではピリオド (.) が定義されています。これは、CICS 領域ユーザー ID のホーム・ディレクトリーが作業ディレクトリーとして使われることを示します。このディレクトリーは、CICS インストール時に作成されます。このディレクトリーが存在しないか、WORK_DIR が省略されている場合には、z/OS UNIX ディレクトリー名として /tmp が使用されます。

作業ディレクトリーの絶対パスまたは相対パスを指定できます。作業ディレクトリーの相対パスは、CICS 領域ユーザー ID のホーム・ディレクトリーから見た相対パスです。ホーム・ディレクトリーを Java に関連した活動の作業ディレクトリーとして使用しない場合や、CICS 領域が z/OS ユーザー ID (UID) を共用しているために同じホーム・ディレクトリーを持っている場合には、CICS 領域ごとに異なる作業ディレクトリーを作成できます。このために、&APPLID; シンボルを使用するディレクトリー名を指定します。CICS がこのシンボルを実際の CICS 領域アプリケーション ID に置換します。したがって、すべての CICS

領域で JVM プロファイルのセットを共用する場合でも、領域ごとに固有の作業ディレクトリーを持つことができます。例えば、次のように指定するとします。

```
WORK_DIR=/u/&APPLID;/javaoutput
```

その JVM プロファイルを使用する各 CICS 領域は、それぞれ独自の作業ディレクトリーを持ちます。該当するディレクトリーが z/OS UNIX 上に作成されていること、およびそれに対する読み取り/書き込み/実行アクセス権限が CICS 領域に与えられていることを確認してください。

また、作業ディレクトリーの固定名を指定することもできます。この場合は、適切なディレクトリーが z/OS UNIX に作成され、アクセス権限が正しい CICS 領域に付与されていることを確認する必要もあります。作業ディレクトリーに固定名を使用する場合は、JVM プロファイルを共用する CICS 領域内のすべての JVM サーバーからの出力ファイルが、そのディレクトリーに作成されます。ご使用の出力ファイルに固定のファイル名を使用する場合には、こうした CICS 領域内のすべての JVM サーバーからの出力は同じ z/OS UNIX ファイルに追加されます。同じファイルに追加されないようにするには、JVMSERVER シンボルと APPLID シンボルを使用して、JVM サーバーごとに固有の出力およびダンプ・ファイルを生成します。

**USSHOME** システム初期設定パラメーターで定義された CICS ファイルのホーム・ディレクトリーである、z/OS UNIX 上の CICS ディレクトリーで、作業ディレクトリーを定義しないでください。

**WSDL_VALIDATOR={TRUE|FALSE}**
SOAP 要求と応答をそれぞれの定義およびスキーマに照らして検証する操作を可能にします。 Liberty JVM サーバーに関しては、このオプションは無視されます。詳しくは、SOAP メッセージの検証を参照してください。
WSDL_VALIDATOR=FALSE を設定することで、このオプションをオフにすることができます。オプトアウトすることで、新しい JAR ファイルとの競合の可能性を排除し、ストレージの浪費や、始動速度の低下を回避できます。

**JVM コマンド行オプション:**

JVM コマンド・ライン・オプション、およびその説明。

**コマンド・ライン・オプションのリスト**

**注:** このリストは完全なものではありません。IBM JVM 専用の便利なオプションのリストです。-D を含むオプションは標準のオプションで、-X を含むオプションはベンダー指定のオプションです。

**-agentlib**
JVM でデバッグ・サポートを使用可能にするかどうかを指定します。

詳しくは、 265 ページの『Java アプリケーションのデバッグ』を参照してください。Java Platform Debugger Architecture (JPDA) について詳しくは、Oracle Technology Network Java の Web サイトを参照してください。

**-Xms**
ヒープの初期サイズを指定します。ストレージ・サイズは 1024 バイトの倍数で指定します。文字 K は KB を、文字 M は MB を、および文字 G は GB を

それぞれ表します。例えば、ヒープの初期サイズとして 6,291,456 バイトを指定するには、以下のいずれかの方法で **-Xms** をコーディングします。

```
-Xms6144K
-Xms6M
```

*size* を KB 数または MB 数として指定します。詳しくは、IBM SDK, Java Technology Edition 7.0.0 の『JVM コマンド行オプション』を参照してください。

**-Xmx**
ヒープの最大サイズを指定します。なお、この固定ストレージ量は、JVM 初期設定時に JVM によって割り振られます。

*size* を KB 数または MB 数として指定します。

**-Xscmx**
共用クラス・キャッシュのサイズを指定します。最小サイズは 4 KB です。最大サイズとデフォルト・サイズはプラットフォームによって異なります。

*size* を KB 数または MB 数として指定します。詳しくは、IBM SDK, Java Technology Edition 7.0.0 の『JVM コマンド行オプション』を参照してください。

**-Xshareclasses**
共用クラス・キャッシュでクラス・データ共用を使用可能にする場合に、このオプションを指定します。JVM は既存のキャッシュに接続するか、キャッシュが存在しない場合は作成します。**-Xshareclasses** オプションにサブオプションを追加すると、複数のキャッシュを持つことができ、正しいキャッシュを指定することができます。詳しくは、IBM SDK, Java Technology Edition 7.0.0 の『JVM 間でのクラス・データの共用』を参照してください。

**-XX:[+|-]EnableCPUMonitor**
JVM サーバーで動作している場合、これはデフォルトで **-XX:-EnableCPUMonitor** に設定されます。ただし、JMX CPU モニタリングの拡張機能を使用する場合は、 **-XX:+EnableCPUMonitor** に設定する必要があります。このオプションを使用可能にすると、CPU の使用量が増加します。

**-Xmso** JVM オプションとデフォルト値については、IBM SDK, Java Technology Edition 7.0.0 の『JVM コマンド行オプション』を参照してください。

**JVM システム・プロパティー:**

JVM システム・プロパティーは、JVM とそのランタイム環境に固有の構成情報を提供します。 JVM システム・プロパティーを指定するには、それらを JVM プロファイルに追加します。実行時に、CICS はプロパティーを JVM プロファイルから読み取り、JVM に渡します。

**プロパティーの接頭部**

システム・プロパティーは -D 接頭部を使用して設定する必要があります。例えば、**com.ibm.cics** の場合の正しい構文は **-Dcom.ibm.cics** となります。

**com.ibm.cics** は、このプロパティーが CICS 環境の IBM JVM に固有のものであることを示します。

**com.ibm** は、より広い範囲で使用される一般的な JVM プロパティーであることを示します。

**java.ibm** も、より広い範囲で使用される一般的な JVM プロパティーであることを示します。

一般的なプロパティーについては、157 ページの『JVM プロファイルの検証およびプロパティー』を参照してください。

### プロパティーのコーディング規則

プロパティーは一連のコーディング規則に従って指定する必要があります。規則について詳しくは、158 ページの『JVM プロファイルのコーディング規則』を参照してください。

### さまざまな用途の JVM サーバーに対するプロパティーの使用可否

JVM サーバーの用途に応じて、さまざまなプロパティーを利用可能です。次の表に、JVM サーバーの特定の用途に対して、各オプションが必須であるか、任意指定であるか、それとも使用不可であるかを示します。表に示しているプロパティーのいくつかは「読み取り専用」であることに注意してください。「読み取り専用」プロパティーを変更すると、ランタイム環境で障害が起きる可能性があります。このような READ-ONLY プロパティーについては、表の後で詳しく説明します。

*表 17. JVM サーバーの用途別オプション*

| システム・プロパティー | OSGi | Liberty | Axis2 | STS |
|---|---|---|---|---|
| com.ibm.cics.jvmserver.applid | サポートされる | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.configroot | サポートされる | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.controller.timeout | サポートされる | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.local.ccsid | サポートされる | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.name | サポートされる | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.override.ccsid | サポートされる | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.supplied.ccsid | サポートされる | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.threadjoin.timeout | サポートされる | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.trace.filename | サポートされる | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.trace.format | サポートされる | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.trigger.timeout | サポートされる | サポートされる | サポートされていない | サポートされていない |

*表 17. JVM サーバーの用途別オプション (続き)*

| システム・プロパティー | OSGi | Liberty | Axis2 | STS |
|---|---|---|---|---|
| com.ibm.cics.jvmserver.wlp.args | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.wlp.autoconfigure | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.wlp.defaultapp | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.wlp.jdbc.driver.location | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.wlp.jta.integration | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.wlp.optimize.static.resources | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.wlp.optimize.static.resources.extra | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.wlp.server.host | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.wlp.server.http.port | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.wlp.server.https.port | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.jvmserver.wlp.server.name | サポートされていない | サポートされる | サポートされていない | サポートされていない |
| com.ibm.cics.sts.config | サポートされていない | サポートされていない | サポートされていない | サポートされる |
| com.ibm.ws.logging.console.log.level | サポートされていない | サポートされる | サポートされる | サポートされていない |
| console.encoding | サポートされる | サポートされる | サポートされる | サポートされる |
| file.encoding | サポートされる | サポートされる | サポートされる | サポートされる |
| java.security.manager | サポートされる | サポートされていない | サポートされる | サポートされる |
| java.security.policy | サポートされる | サポートされていない | サポートされる | サポートされる |
| org.osgi.framework.system.packages.extra | サポートされる | サポートされる | サポートされていない | サポートされていない |

**プロパティーの説明 - 読み取り専用**

**com.ibm.cics.jvmserver.applid=**
CICS 領域アプリケーション ID (APPLID)。これは読み取り専用プロパティーです。アプリケーションで使用するためにこの値を表示できますが、変更することはできません。

**com.ibm.cics.jvmserver.configroot=***zFS_directory*
構成ファイル (例えば JVM サーバーの JVM プロファイル) が存在する zFS の場所。これは読み取り専用プロパティーです。アプリケーションで使用するためにこの値を表示できますが、変更することはできません。

**com.ibm.cics.jvmserver.local.ccsid=**
JCICS API が使用される場合のファイル・エンコード用のコード・ページを指定します。これは読み取り専用プロパティーです。アプリケーションで使用するためにこの値を表示できますが、変更することはできません。

**com.ibm.cics.jvmserver.name=**
JVM サーバーの名前。これは読み取り専用プロパティーです。アプリケーションで使用するためにこの値を表示できますが、変更することはできません。

**com.ibm.cics.jvmserver.supplied.ccsid=**
ローカル領域のデフォルト CCSID。これは読み取り専用プロパティーです。アプリケーションで使用するためにこの値を表示できますが、変更することはできません。

**com.ibm.cics.jvmserver.trace.filename=**
JVM サーバー・トレース・ファイルの名前。これは読み取り専用プロパティーです。アプリケーションで使用するためにこの値を表示できますが、変更することはできません。

**プロパティーの説明 - 変更可能**

**com.ibm.cics.jvmserver.controller.timeout={***time***|90000ms}**
このプロパティーは、IBM サービス担当者の指示があった場合にのみ使用してください。これは適宜、変更される場合があります。

**com.ibm.cics.jvmserver.override.ccsid=**
このプロパティーは上級ユーザー向けです。これは JCICS API が使用されるときのファイル・エンコード用のコード・ページをオーバーライドします。デフォルトでは、JCICS は **LOCALCCSID** システム初期設定パラメーターの値をファイルのエンコード方式として使用します。この値をオーバーライドすることを選択した場合には、このプロパティーでコード・ページを設定します。 EBCDIC コード・ページを使用してください。新しいコード・ページにアプリケーションが整合していることを確認する必要があります。そうでない場合、エラーが発生する可能性があります。有効な CCSID の詳細については、「Reference」の『LOCALCCSID システム初期設定パラメーター』-> System definitionを参照してください。

**com.ibm.cics.jvmserver.threadjoin.timeout={***time***|30000ms}**
スレッドを待機している要求がサービスのキューに入れられているときのタイムアウト値を制御します。

**com.ibm.cics.jvmserver.trace.format={FULL|SHORT|ABBREV}**
トレースの形式を制御します。目的に応じてトレース形式を変更できますが、診断情報を IBM サービス担当者に送るときには FULL に設定する必要があります。

**com.ibm.cics.jvmserver.trigger.timeout={***time***|500ms}**
このプロパティーは、IBM サービス担当者の指示があった場合にのみ使用してください。これは適宜、変更される場合があります。

**com.ibm.cics.jvmserver.wlp.args=**
このプロパティーは、IBM サービス担当者の指示があった場合にのみ使用してください。

**com.ibm.cics.jvmserver.wlp.autoconfigure={false|true}**
CICS が、Liberty JVM サーバーの server.xml ファイルを自動的に作成および更新するかどうかを指定します。このプロパティーを true に設定すると、CICS は、zFS にディレクトリー構造と構成ファイルを作成します。また、他の Java プロパティーの値 (例えば HTTP ポート番号) を提供した場合、CICS は server.xml ファイルを更新します。

**com.ibm.cics.jvmserver.wlp.defaultapp={false|true}**
サーバーが適切にインストールされて始動したことを確認するために使用できるデフォルト・アプリケーションをインストールするよう CICS に指示します。サーバーは、**com.ibm.cics.jvmserver.wlp.autoconfigure=true** の場合にのみ、このプロパティーの値で更新されます。

**com.ibm.cics.jvmserver.wlp.jdbc.driver.location=**
DB2 JDBC ドライバーが含まれる zFS 内のディレクトリーの場所を指定します。この場所には、DB2 JDBC ドライバーの classes および lib ディレクトリーが含まれている必要があります。自動構成プロパティー **com.ibm.cics.jvmserver.wlp.autoconfigure=true** を設定した場合、JVM サーバーを有効にすると、server.xml 内の既存のサンプル構成がデフォルトの構成に置き換えられ、ユーザーによる更新内容は失われます。

**com.ibm.cics.jvmserver.wlp.jta.integration={false|true}**
Liberty JTA トランザクションが存在する場合に、CICS JTA の Liberty との統合を使用可能にします。サーバーは、**com.ibm.cics.jvmserver.wlp.autoconfigure=true** の場合にのみ、このプロパティーの値で更新されます。

**com.ibm.cics.jvmserver.wlp.optimize.static.resources={true|false}**
静的リソース最適化を使用することにより、CICS はより少ない数のトランザクションを使って要求を処理できます。静的と認識されるファイルの種類は、.css、.gif、.ico、.jpg、.jpeg、.js、および .png です。

**com.ibm.cics.jvmserver.wlp.optimize.static.resources.extra=**
**com.ibm.cics.jvmserver.wlp.optimize.static.resources=true** である場合、最適化用に追加の静的リソースからなるカスタム・リストを提供できます。項目をコンマで区切ってピリオドで始める必要があります (例えば .css, .gif, .ico)。

**com.ibm.cics.jvmserver.wlp.server.host={*|*hostname*|*IP_address*}**
Web アプリケーションにアクセスするための HTTP 要求のホストの名前または IP アドレス (IPv4 または IPv6 形式) を指定します。 Liberty JVM サーバーは、デフォルト値として * を使用します。この値は CICS での Web アプリケーションの実行には適していない可能性があるため、このプロパティーを使用して別の値を指定するか、server.xml ファイルを更新します。このプロパティーはオプションであり、**com.ibm.cics.jvmserver.wlp.autoconfigure=true** である場合にのみ使用されます。

**com.ibm.cics.jvmserver.wlp.server.http.port={9080|*port_number*}**
Java Web アプリケーションに対する HTTP 要求を受け入れるためのポートを

指定します。 CICS では、Liberty プロファイルで提供されるデフォルト値を使用します。 Liberty JVM サーバーは TCPIPSERVICE リソースを使用しないので、z/OS システムでポート番号が空いているか、共用されていることを確認してください。このプロパティーはオプションであり、**com.ibm.cics.jvmserver.wlp.autoconfigure=true** である場合にのみ使用されます。

**com.ibm.cics.jvmserver.wlp.server.https.port={9443|*port_number*}**
Java Web アプリケーションに対する HTTPS 要求を受け入れるためのポートを指定します。 CICS では、Liberty プロファイルで提供されるデフォルト値を使用します。 Liberty JVM サーバーは TCPIPSERVICE リソースを使用しないので、z/OS システムでポート番号が空いているか、共用されていることを確認してください。このプロパティーはオプションであり、**com.ibm.cics.jvmserver.wlp.autoconfigure=true** である場合にのみ使用されます。

**com.ibm.cics.jvmserver.wlp.server.name={defaultServer|*server_name*}**
Liberty プロファイル・サーバーの名前を指定します。このプロパティーのデフォルトは `defaultServer` です。このプロパティーはオプションであり、zFS システム上の Liberty サーバー構成と出力ファイルおよびディレクトリーの場所に影響するため、指定する必要はありません。

**com.ibm.cics.sts.config=***path*
STS 構成ファイルの場所と名前。

**com.ibm.ws.logging.console.log.level={INFO|AUDIT|WARNING|ERROR|OFF}**
どんなメッセージが Liberty によって JVM サーバー STDOUT ファイルに書き込まれるかを制御します。このプロパティーの設定にかかわらず、Liberty コンソール・メッセージは Liberty `messages.log` ファイルにも書き込まれます。

**console.encoding=**
JVM サーバー出力ファイルのエンコード方式を指定します。

**file.encoding=**
JVM で文字を読み書きする場合のコード・ページを指定します。デフォルトでは、z/OS 上の JVM は EBCDIC コード・ページ IBM1047 (または cp1047) を使用します。

- OSGi 用に構成されているプロファイルには、JVM でサポートされているどのコード・ページでも指定できます。 JCICS は文字エンコードに CICS 領域のローカル CCSID を使用するので、CICS はどのコード・ページでも許容します。
- Liberty JVM サーバー用に構成されているプロファイルには、ISO-8859-1 がデフォルト値として提供されます。UTF-8 も使用できます。その他のコード・ページはファイルのエンコード用にはサポートされていません。
- Axis2 用に構成されているプロファイルには、EBCDIC コード・ページを指定しなければなりません。

**java.security.manager={default| "" | |*other_security_manager*}**
JVM に使用可能にする Java セキュリティー・マネージャーを指定します。デフォルトの Java セキュリティー・マネージャーを使用可能にするには、次のいずれかの形式でこのシステム・プロパティーを組み込みます。

- `java.security.manager=default`

- `java.security.manager=""`
- `java.security.manager=`

これらのステートメントはすべて、デフォルトのセキュリティー・マネージャーを使用可能にします。 **java.security.manager** システム・プロパティーを JVM プロファイルに組み込まない場合、JVM は Java セキュリティーが無効な状態で実行されます。JVM の Java セキュリティーを使用不可にするには、このシステム・プロパティーをコメント化します。

**java.security.policy=**

JVM のセキュリティー・ポリシーを判別するためにセキュリティー・マネージャーで使われる、追加のポリシー・ファイルの場所を記述します。デフォルトのポリシー・ファイルは、`/usr/lpp/java/J7.0_64/lib/security/java.policy` で JVM に提供されます。ここで、`java/J7.0_64` サブディレクトリー名は、IBM 64-bit SDK for z/OS, Java テクノロジー・エディション をインストールする際のデフォルト値です。デフォルトのセキュリティー・マネージャーは常にこのデフォルト・ポリシー・ファイルを使って JVM のセキュリティー・ポリシーを判別します。 **java.security.policy** システム・プロパティーを使用すると、デフォルト・ポリシー・ファイルに加えて、セキュリティー・マネージャーで考慮される任意のポリシー・ファイルを指定することができます。

Java セキュリティーがアクティブ状態のときに CICS Java アプリケーションを正常に実行できるようにするには、アプリケーション実行に必要な権限を CICS に与える追加のポリシー・ファイルを少なくとも 1 つ指定してください。

Java セキュリティーの有効化については、248 ページの『Java セキュリティー・マネージャーの有効化』を参照してください。

**org.osgi.framework.system.packages.extra=**

Liberty を含む OSGi 対応 JVM サーバーに固有のオプションで、JRE とカスタム Java パッケージの拡張機能を OSGi フレームワークを通して公開して、以後のバンドル・インポート解決に利用できるようにします。JVM ベンダーは各種拡張機能を JRE に提供できます。IBM JVM サーバーでは、このオプションは、CICS が IBM JRE から公開するように選択したパッケージのセットを含むように補強されています。追加のパッケージが必要になった場合は、このプロパティーを設定して追加のパッケージを定義できます。詳しくは、OSGi Alliance Specification を参照してください。

**時間帯の設定:**

TZ 環境変数はシステムの「ローカル」時間を指定します。この変数を JVM プロファイルに追加することで、JVM サーバーのローカル時間を設定できます。

JVM サーバーの時間帯を設定する際には、以下の点に注意する必要があります。

- JVM プロファイルの TZ 変数が GMT を基準にしたローカル MVS システムのオフセットと一致するようにします。
- TZ 変数を設定しないと、システムによってデフォルトで UTC に設定されます。
- 時間帯のカスタマイズはサポートされていないため、カスタマイズすると UTC にフェイルオーバーするか、または時間帯が混在した状態で JVMTRACE ファイルに出力されます。

- LOCALTIME を時間帯ストリングとして表示すると、構成に不整合が生じます。この不整合は、ローカル MVS 時間と設定する TZ の間、またはローカル MVS 時間と JVM プロファイルのデフォルト設定の間に生じる可能性があります。各項目は正しくても、時間帯が混在した状態で出力されます。
- POSIX TZ の短縮形を使用することで、入力ミスを防ぐこともできます。参考に、POSIX TZ の短縮形と完全形の例を示します。
  - TZ=GMT0BST (短縮形)
  - TZ=GMT0BST,M3.5.0,M10.5.0 (完全形)

**関連資料**:

143 ページの『JVM プロファイルの例』
Liberty サーバー用の JVM プロファイルの例を示します。

# 第 4 章 JVM サーバーへのアプリケーションのデプロイ

Java アプリケーションを JVM サーバーにデプロイするには、そのアプリケーションを正常にインストールして実行するために適切にパッケージ化する必要があります。アプリケーションのパッケージ化とデプロイには、CICS Explorer SDK を使用できます。

JVM サーバーで実行するすべての Java アプリケーションは (Web アーカイブ (WAR) ファイルとエンタープライズ・アーカイブ (EAR) ファイルにあるアプリケーションを除く)、スレッド・セーフで、OSGi 仕様に準拠している必要があります。デプロイする前に、使用するアプリケーションがこの要件を満たしていることを確認してください。Java アプリケーションをデプロイするには、以下のようないくつかのオプションがあります。

- OSGi フレームワークを実行する JVM サーバーの中に、アプリケーションの OSGi バンドルを含む 1 つ以上の CICS バンドルをデプロイする。
- Liberty JVM サーバーの中に、1 つ以上の WAR ファイルを含む 1 つ以上の CICS バンドルをデプロイする。
- Liberty JVM サーバーの中に、Enterprise Bundle Archive (EBA) ファイルを含む 1 つ以上の CICS バンドルをデプロイする。
- EAR ファイルを含む 1 つ以上の CICS バンドルを Liberty JVM サーバーにデプロイする。
- プラットフォームに、CICS バンドルと OSGi バンドルで構成されるアプリケーション・バンドルをデプロイする。

182 ページの『JVM サーバーにおける OSGi バンドルのデプロイ』
JVM サーバーに Java アプリケーションを配置するには、ターゲット JVM サーバーの OSGi フレームワークにそのアプリケーションの OSGi バンドルをインストールする必要があります。

184 ページの『CICS バンドル内の Web アプリケーションの Liberty JVM サーバーへのデプロイ』
Liberty JVM サーバーに、CICS バンドルとしてパッケージされている Java Web アプリケーションをデプロイすることができます。

186 ページの『Liberty JVM サーバーへの Web アプリケーションの直接デプロイ』
Web アプリケーションをデプロイするには、事前に定義した「dropins」ディレクトリーにアプリケーションを直接ドロップするか、アプリケーションを server.xml に手動で定義します。

187 ページの『Liberty JVM サーバーへのミドルウェアのデプロイ』
ミドルウェアは、DLL ファイル、JAR ファイル、OSGi バンドルのどの形式で提供されるかに応じてデプロイします。

188 ページの『JVM サーバー内の Java アプリケーションの呼び出し』
JVM サーバーで稼働している Java アプリケーションを呼び出す方法は多数あります。使用される方式は JVM サーバーの特性によって異なります。

# JVM サーバーにおける OSGi バンドルのデプロイ

JVM サーバーに Java アプリケーションを配置するには、ターゲット JVM サーバーの OSGi フレームワークにそのアプリケーションの OSGi バンドルをインストールする必要があります。

## 始める前に

アプリケーションの OSGi バンドルを含む CICS バンドルは、zFS にデプロイされる必要があります。ターゲット JVM サーバーが CICS 領域で有効になっている必要があります。

## このタスクについて

CICS バンドルには、1 つ以上の OSGi バンドルを含むことができます。 CICS バンドルは配置の単位であるため、すべての OSGi バンドルは、BUNDLE リソースの一部として一緒に管理されます。また、OSGi フレームワークは、依存関係とバージョン管理方式の管理を含めて、OSGi バンドルのライフサイクルを管理します。

1 つの Java アプリケーション・コンポーネントを構成するすべての OSGi バンドルを、必ず同じ CICS バンドル内にデプロイしてください。 OSGi バンドル相互間に依存関係がある場合は、それらを同じ CICS バンドルに配置してください。 CICS BUNDLE リソースをインストールすると、CICS によって OSGi バンドル間の依存関係がすべて解決されます。

共通コードのライブラリーを含む OSGi バンドルへの依存関係がある場合、そのライブラリー用に 1 つの別個の CICS バンドルを作成してください。この場合、そのライブラリーを含む CICS BUNDLE リソースを最初にインストールすることが重要です。 Java アプリケーションをインストールしてから、その Java アプリケーションが依存する CICS バンドルをインストールすると、OSGi フレームワークは、その Java アプリケーションの依存関係を解決できません。

OSGi バンドルを含む CICS バンドルを Liberty JVM サーバーの中にインストールしようと試みないでください。このような構成はサポートされていません。その代わりに、OSGi バンドルを Web アプリケーションと一緒にエンタープライズ・バンドル・アーカイブ (EBA) にパッケージ化できます。または WebSphere Liberty プロファイル・バンドル・リポジトリーを使用して、Liberty JVM サーバー内のすべての Web アプリケーションに対して OSGi バンドルを使用可能にすることもできます。

## 手順

1. zFS 内のバンドルのディレクトリーを指定する BUNDLE リソースを作成します。
   a. CICS SM パースペクティブで、CICS Explorer メニュー・バーの「**定義**」 > 「**バンドル定義**」をクリックして、「バンドル定義」ビューを開きます。
   b. そのビュー内の任意の場所を右クリックし、「**New**」をクリックして「New Bundle Definition」ウィザードを開きます。 そのウィザードのフィールドに、BUNDLE リソースの詳細を入力してください。
   c. BUNDLE リソースをインストールします。 リソースを Enabled 状態または Disabled 状態のどちらかでインストールできます。

- DISABLED 状態でリソースをインストールすると、CICS は OSGi バンドルをフレームワークにインストールし、依存関係を解決しますが、バンドルを開始しようとしません。
- ENABLED 状態でリソースをインストールすると、CICS は OSGi バンドルをインストールし、依存関係を解決し、OSGi バンドルを開始します。OSGi バンドルに遅延バンドルのアクティベーターが含まれている場合、OSGi フレームワークは、別の OSGi バンドルによって最初に呼び出されるまでそのバンドルを開始しようとしません。

2. オプション: BUNDLE リソースがまだ ENABLED 状態でない場合、そのリソースを使用可能にして、フレームワークで OSGi バンドルを開始します。
3. CICS Explorer メニュー・バーの「**Operations**」 > 「**Bundles**」をクリックして、「Bundles」ビューを開きます。BUNDLE リソースの状態を確認します。
   - BUNDLE リソースが ENABLED 状態である場合、CICS はバンドル内のすべてのリソースを正常にインストールできました。
   - BUNDLE リソースが DISABLED 状態である場合、CICS はバンドル内の 1 つ以上のリソースをインストールできませんでした。

   BUNDLE リソースが ENABLED 状態でインストールできなかった場合、BUNDLE リソースのバンドル・パーツを確認してください。いずれかのバンドル・パーツが UNUSABLE 状態である場合、CICS は OSGi バンドルを作成できませんでした。通常、この状態は、zFS で CICS バンドルに問題があることを示します。その BUNDLE リソースを破棄し、問題を修正してから、BUNDLE リソースを再度インストールする必要があります。
4. CICS Explorer メニュー・バーで「**Operations (操作)**」 > 「**Java**」 > 「**OSGi Bundles (OSGi バンドル)**」をクリックして、「OSGi Bundles (OSGi バンドル)」ビューを開きます。 OSGi フレームワークにインストールされた OSGi バンドルおよびサービスの状態を確認します。
   - OSGi バンドルが STARTING 状態である場合、バンドル・アクティベーターが呼び出されましたが、まだ戻っていません。OSGi バンドルに遅延活動化ポリシーがある場合、OSGi フレームワークで呼び出されるまで、そのバンドルはこの状態のままです。
   - OSGi バンドルと OSGi サービスがアクティブである場合、Java アプリケーションは作動可能です。
   - OSGi サービスが非アクティブである場合、CICS は、その名前を持つ OSGi サービスが OSGi フレームワークに既に存在することを検出した可能性があります。
   - BUNDLE リソースを使用不可にすると、OSGi バンドルは RESOLVED 状態に移ります。
   - OSGi バンドルが INSTALLED 状態である場合、OSGi バンドル内の依存関係を解決できなかったために、このバンドルは開始されなかったか開始に失敗しました。

## タスクの結果

BUNDLE が使用可能になり、OSGi バンドルが OSGi フレームワークに正常にインストールされ、すべての OSGi サービスがアクティブです。OSGi バンドルは、フレームワーク内の他のバンドルから使用可能です。

### 次のタスク

OSGi フレームワークの外部にある他の CICS アプリケーションから Java アプリケーションを使用可能にすることができます。それには、188 ページの『JVM サーバー内の Java アプリケーションの呼び出し』の説明に従ってください。アプリケーションを更新または削除する場合は、191 ページの『第 5 章 Java アプリケーションの管理』を参照してください。

## CICS バンドル内の Web アプリケーションの Liberty JVM サーバーへのデプロイ

Liberty JVM サーバーに、CICS バンドルとしてパッケージされている Java Web アプリケーションをデプロイすることができます。

### 始める前に

WAR ファイル、EAR ファイルまたは EBA ファイルのいずれの形式の Java Web アプリケーションも、zFS に CICS バンドルとしてデプロイする必要があります。ターゲット JVM サーバーが CICS 領域で有効になっている必要があります。

Java アプリケーションの作成および再パッケージ化に関する一般情報については、34 ページの『CICS Explorer SDK を使用したアプリケーションの開発』を参照してください。

共通コードのライブラリーを含む OSGi バンドルに依存している場合は、そのバンドルを Liberty バンドル・リポジトリーにインストールします ( 182 ページの『JVM サーバーにおける OSGi バンドルのデプロイ』を参照)。

### このタスクについて

CICS アプリケーション・モデルでは、Java アプリケーション・コンポーネントが CICS バンドルにパッケージされて zFS にデプロイされます。 CICS バンドルをインストールすることによって、アプリケーション・コンポーネントのライフサイクルを管理できます。 Java Web アプリケーションには、アプリケーションのプレゼンテーション層とビジネス・ロジックを提供する 1 つ以上の WAR ファイル、あるいは、EBA ファイルにエクスポートされた 1 つの OSGi アプリケーション・プロジェクト (その中にはプレゼンテーション層を提供する 1 つの Web 使用可能 OSGi バンドル・プロジェクトと、ビジネス・ロジックを提供する追加の OSGi バンドル・セットが含まれています) 、あるいは、プレゼンテーション層とビジネス・ロジックを提供する 1 つ以上の WAR ファイルが含まれるエンタープライズ・アーカイブ (EAR) ファイルが含まれます。

### 手順

1. zFS 内のバンドルのディレクトリーを指定する BUNDLE リソースを作成します。
   a. CICS SM パースペクティブで、CICS Explorer メニュー・バーの「**定義**」 > 「**バンドル定義**」をクリックして、「バンドル定義」ビューを開きます。

b. そのビュー内の任意の場所を右クリックし、「**New**」をクリックして「New Bundle Definition」ウィザードを開きます。 そのウィザードのフィールドに、BUNDLE リソースの詳細を入力してください。

c. BUNDLE リソースをインストールします。 以下に示すように、使用可能または使用不可の状態でリソースをインストールできます。

- DISABLED 状態でリソースをインストールすると、CICS は Web アプリケーションを Liberty サーバーにインストールしようとしません。
- ENABLED 状態でリソースをインストールすると、CICS は、Web アプリケーション (WAR ファイル、EAR ファイル、EBA ファイル) を `${server.output.dir}/installedApps` ディレクトリーにインストールし、<application> 項目を `${server.output.dir}/installedApps.xml` に追加します。

2. オプション: BUNDLE リソースがまだ ENABLED 状態でない場合は、そのリソースを有効にして、Liberty サーバーで Web アプリケーションを開始します。

3. CICS Explorer メニュー・バーの「**Operations**」 > 「**Bundles**」をクリックして、「Bundles」ビューを開きます。BUNDLE リソースの状態を確認します。

- BUNDLE リソースが ENABLED 状態である場合、CICS はバンドル内のすべてのリソースを正常にインストールしました。
- BUNDLE リソースが DISABLED 状態である場合、CICS はバンドル内の 1 つ以上のリソースをインストールできませんでした。

BUNDLE リソースが ENABLED 状態でインストールできなかった場合、BUNDLE リソースのバンドル・パーツを確認してください。いずれかのバンドル・パーツが UNUSABLE 状態である場合、問題の原因を説明するメッセージが発行されます。例えば、この状態は、zFS で CICS バンドルに問題がある、あるいは関連付けられている JVMSERVER リソースが利用不可であることを示します。その BUNDLE リソースを破棄し、報告された問題を解決してから、BUNDLE リソースを再度インストールする必要があります。

4. オプション: アプリケーション・トランザクションで Web アプリケーション要求を実行するために、URIMAP および TRANSACTION リソースを作成できます。 アプリケーションに対するセキュリティーを制御する場合は、URI マップの定義が役に立ちます。 URI を特定のトランザクションにマップして、トランザクション・セキュリティーを使用できるからです。通常、これらのリソースは CICS バンドルの一部として作成され、アプリケーションによって管理されます。しかし、それらのリソースを別に定義することが望ましい場合は、そうすることもできます。

a. PROGRAM 属性を DFHSJTHP に設定するアプリケーション用に TRANSACTION リソースを作成します。 この CICS プログラムは、Liberty JVM サーバーに対するインバウンド Web 要求のセキュリティー検査を処理します。何らかのリモート属性を設定しても、それらは CICS によって無視されます。トランザクションは常にローカル CICS 領域と接続されている必要があるからです。

b. JVMSERVER のタイプが USAGE である URIMAP リソースを作成します。TRANSACTION 属性をアプリケーション・トランザクションに設定し、スキームを HTTP または HTTPS に設定します。 **USERID** 属性を使用してユーザー ID を設定することもできます。アプリケーションのセキュリティー認証

メカニズムを使用する場合、この値は無視されます。認証が行われず、URI マップにユーザー ID が設定されていない場合、デフォルトの CICS のユーザー ID で処理は実行されます。

### タスクの結果

CICS リソースが有効になり、Web アプリケーションが Liberty JVM サーバーに正常にインストールされます。

### 次のタスク

Web クライアントを介して Java アプリケーションの使用可能性をテストすることができます。アプリケーションの更新または削除については、Java アプリケーションの管理を参照してください。

## Liberty JVM サーバーへの Web アプリケーションの直接デプロイ

Web アプリケーションをデプロイするには、事前に定義した「dropins」ディレクトリーにアプリケーションを直接ドロップするか、アプリケーションを server.xml に手動で定義します。

### 始める前に

JVM サーバーが、Liberty プロファイル・テクノロジーを使用するように構成されていて、かつ CICS 領域で使用可能にされている必要があります。

### このタスクについて

Liberty プロファイル・デプロイメント・モデルでは、Web コンポーネントをアプリケーション・サーバーにデプロイしてそれを実行する迅速な方法が提供されます。このモデルに従うには、Web アーカイブ (WAR)、エンタープライズ・バンドル・アーカイブ (EBA) (または エンタープライズ・アーカイブ (EAR)) ファイルとして、Web アプリケーションを Liberty JVM サーバーの `dropins` ディレクトリーにデプロイします。このモデルを使用すると、CICS は JVM サーバーで実行されている Web アプリケーションを認識しません。したがって、同じアプリケーションを両方のモデルを使用して同じ JVM サーバーにデプロイしないでください。これを行うと、エラーが発生する可能性があります。この方法でデプロイされたアプリケーションは、セキュリティーなどの追加のサービス品質の利点を得られず、常にトランザクション CJSA 下で実行されます。代わりに、アプリケーション項目を server.xml に追加して、アプリケーションを手動で定義することもできます。

**注:** CICS 自動構成で提供されるデフォルトを受け入れた場合、`dropins` ディレクトリーは自動的に作成されません。

### 手順

1. `dropins` ディレクトリーを作成して、そこにアプリケーションをデプロイするには、次のようにします。
   a. CICS Liberty 構成の `server.xml` に以下の行を追加します。

```
<applicationMonitor dropins="dropins" dropinsEnabled="true" pollingRate="5s"
    updateTrigger="disabled"/>
```

b. FTP を使用して、エクスポート済みファイルをバイナリー・モードで Liberty プロファイル・サーバーの dropins ディレクトリーに転送します。 ディレクトリー・パスは WLP_USER_DIR/servers/*server_name*/dropins です。 *server_name* は **com.ibm.cics.jvmserver.wlp.server.name** プロパティーの値です。

2. アプリケーションをサーバー構成ファイルに追加することでアプリケーションをデプロイするには、server.xml にアプリケーションの以下の属性を構成する必要があります。
   a. id: 一意でなければならず、サーバーによって内部的に使用されます。
   b. name: 一意でなければならず、アプリケーションに依存します。
   c. type: アプリケーションのタイプを指定します。サポートされるタイプは、WAR、EBA および EAR です。
   d. location: アプリケーションのロケーションを指定します。これは、絶対パスまたは URL で指定できます。

### タスクの結果

Liberty JVM サーバーは、デプロイされた WAR、EBA または EAR ファイルを自動的に検出してインストールします。

### 次のタスク

Web ブラウザーから Web アプリケーションにアクセスして、Web アプリケーションが正常に実行されていることを確認します。アプリケーション・ファイルを削除するには、dropins ディレクトリーから WAR または EBA ファイルを削除します。

アプリケーションのパッケージ化とデプロイには、CICS Explorer SDK を使用できます。詳しくは、 181 ページの『第 4 章 JVM サーバーへのアプリケーションのデプロイ』を参照してください。

## Liberty JVM サーバーへのミドルウェアのデプロイ

ミドルウェアは、DLL ファイル、JAR ファイル、OSGi バンドルのどの形式で提供されるかに応じてデプロイします。

### 手順

- DLL ファイルとして提供されるミドルウェアの場合は、JVM プロファイルの LIBPATH_SUFFIX オプションで参照されているディレクトリーにファイルをコピーします。 詳しくは、LIBPATH_PREFIX, LIBPATH_SUFFIXを参照してください。
- OSGi バンドルの JAR ファイルとして提供されるミドルウェアの場合は、server.xml ファイルの bundleRepository 定義で参照されているディレクトリーに JAR ファイルをコピーします。 詳しくは、バンドル・リポジトリーを参照してください。
- OSGi バンドルではない JAR ファイルとして提供されるミドルウェアの場合は、server.xml ファイルのグローバル・ライブラリー定義で参照されているディレクトリーに JAR ファイルをコピーします。 詳しくは、グローバル・ライブラリーを参照してください。

# JVM サーバー内の Java アプリケーションの呼び出し

JVM サーバーで稼働している Java アプリケーションを呼び出す方法は多数あります。使用される方式は JVM サーバーの特性によって異なります。

## このタスクについて

特定の URL を指定した HTTP 要求を使用することにより、Liberty JVM サーバーで実行される Web アプリケーションを呼び出すことができます。 EXEC CICS LINK または EXEC CICS START から直接 Web アプリケーションを起動することはできません。 OSGi JVM サーバーで稼働している Java アプリケーションを呼び出す場合は、プログラム定義されている Java に EXEC CICS LINK を行うか、Java としてターゲット・プログラムが定義されているトランザクションに EXEC CICS START を行うことができます。プログラム定義では、JVMSERVER と、呼び出す CICS 生成 OSGi サービスの名前を指定します。このような「リンク可能な」OSGi サービスは、マニフェストに CICS-MainClass ヘッダーが含まれている OSGi バンドルのインストール時に CICS によって作成されます。 CICS-MainClass ヘッダーには、アプリケーションへのエントリー・ポイントとして機能させる、OSGi バンドル内の Java クラスの main メソッドが指定されています。

OSGi サービスは、OSGi フレームワークに登録されている明確に定義されたインターフェースです。 OSGi バンドルやリモート・アプリケーションは OSGi サービスを使用して、OSGi バンドルにパッケージされているアプリケーション・コードを呼び出します。 OSGi バンドルは複数の OSGi サービスをエクスポートできます。詳しくは、OSGi バンドルの更新を参照してください。

**注:**

クラスパス・ベースの JVM サーバーでの Java 関数の呼び出しは、通常、バッチ、Axis2、および SAML などの、JVM サーバー固有の機能の一部として実行されます。これらの機能用に、DFHSJJI ベンダー・インターフェースが提供されています。

## 手順

- Web アーカイブ (WAR) ファイル、エンタープライズ・アーカイブ (EAR) ファイル、または、Web アプリケーション・バンドル (WAB) ファイルを含み、Liberty JVM サーバーで実行されるエンタープライズ・バンドル・アーカイブ (EBA) ファイルとして開発された Web アプリケーションの場合は、URL を使用してクライアントのブラウザーからアプリケーションを呼び出します。
- OSGi JVM サーバーにデプロイされている OSGi バンドルの場合は、以下のステップに従ってください。
  1. OSGi フレームワークで使用したい、アクティブな OSGi サービスのシンボル名を判別します。 CICS Explorer で「**Operations**」 > 「**Java**」 > 「**OSGi Services**」をクリックして、アクティブな OSGi サービスをリストします。
  2. 他の CICS アプリケーションに対して OSGi サービスを表す PROGRAM リソースを作成します。
     - JVM 属性で、YES を指定して、プログラムが Java プログラムであることを示します。

- JVMCLASS 属性で、OSGi サービスのシンボル名を指定します。この値は大/小文字の区別があります。
- JVMSERVER 属性で、OSGi サービスが実行される JVMSERVER リソースの名前を指定します。

3. 以下の 2 つの方法で Java アプリケーションを呼び出すことができます。
   - トランザクション ID を指定する 3270 または **EXEC CICS START** 要求を使用します。OSGi サービスの PROGRAM リソースを定義する TRANSACTION リソースを作成します。
   - **EXEC CICS LINK** 要求、ECI 呼び出し、または EXCI 呼び出しを使用します。要求をコーディングする際に、OSGi サービスの PROGRAM リソースの名前を指定します。

- Axis、バッチ、または SAML 機能については、 153 ページの『Axis2 用の JVM サーバーの構成』、 Modern Batch の概要、およびSAML 用の CICS の構成を参照してください。

## タスクの結果

他のコンポーネントが Java アプリケーションを使用できるようにするための定義を作成しました。 CICS は、ターゲット JVM サーバーで要求を受け取ると、指定された Java クラスまたは Web アプリケーションを新しい CICS Java スレッドで呼び出します。関連付けられた OSGi サービスまたは Web アプリケーションが登録されていないか、非アクティブである場合、呼び出し側プログラムにエラーが戻されます。

# 第 5 章 Java アプリケーションの管理

Java アプリケーションを使用可能にした後、CICS 領域をモニターして、アプリケーションの動作を理解することができます。アプリケーションのパフォーマンスを最適化するために環境を調整することができます。

## このタスクについて

統計とモニターを使用して、CICS 領域における Java アプリケーションの動作に関する情報を収集することができます。特に、JVM の動作を確認することができます。情報を収集した後、JVM または Language Environment エンクレーブに変更を加えて、パフォーマンスを改善することができます。

192 ページの『JVM サーバーにおける OSGi バンドルの更新』
OSGi フレームワーク内の OSGi バンドルを更新するプロセスは、バンドルのタイプおよびその依存関係によって異なります。JVM サーバーを再始動することなく、アプリケーションの OSGi バンドルを更新できます。ただし、ミドルウェア・バンドルを更新するには、JVM サーバーの再始動が必要です。

200 ページの『JVM サーバーからの OSGi バンドルの除去』
JVM サーバーから OSGi バンドルを除去したい場合は、CICS Explorer を使用して BUNDLE リソースを使用不可にして破棄します。

201 ページの『JVM サーバーへのアプリケーションの移動』
プールされた JVM で Java アプリケーションを実行している場合、JVM サーバーで実行するようにそれらのアプリケーションを移動することができます。JVM サーバーは、同じ JVM 内で Java アプリケーションに対する複数の要求を処理できるので、同じワークロードの実行に必要な JVM 数を減らすことができます。

202 ページの『JVM サーバーのスレッド限度の管理』
JVM サーバーでは、Java アプリケーションの実行に使用できるスレッド数が制限されています。また、各スレッドが 1 つの T8 TCB を使用するため、CICS 領域にもスレッド数の制限があります。 CICS 統計を使用してスレッド限度を調整すると、領域内の JVM サーバー数と、各 JVM サーバーで実行されるアプリケーションのパフォーマンスとのバランスを取ることができます。

203 ページの『CICS リスタート後の OSGi バンドル・リカバリー』
OSGi バンドルを含む CICS 領域をリスタートすると、CICS は BUNDLE リソースをリカバリーし、OSGi バンドルを JVM サーバーのフレームワークにインストールします。

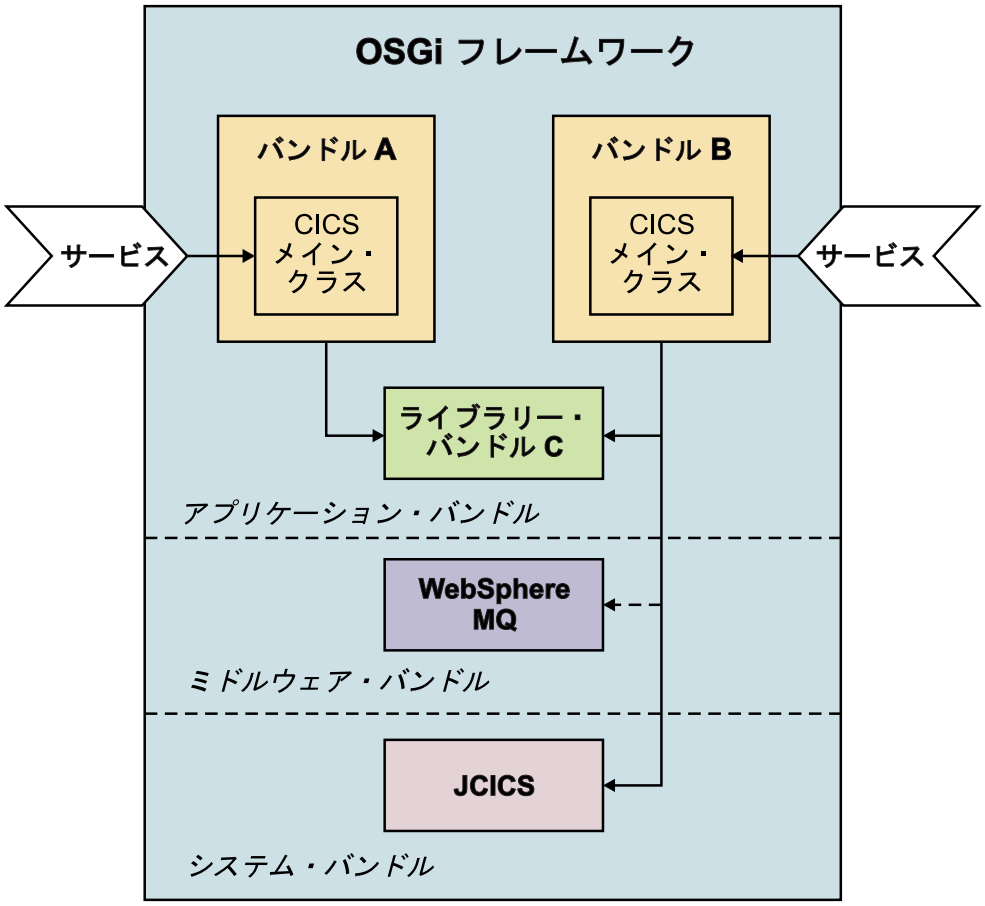
204 ページの『JVM stdout および stderr 出力をリダイレクトするための Java クラスの作成』
JVM プロファイルの `USEROUTPUTCLASS` オプションを使用して、JVM からの `stdout` ストリームと `stderr` ストリームを代行受信する Java クラスを指定します。このクラスを更新すると、適当なタイム・スタンプとレコード・ヘッダーを指定し、出力をリダイレクトすることができます。

# JVM サーバーにおける OSGi バンドルの更新

OSGi フレームワーク内の OSGi バンドルを更新するプロセスは、バンドルのタイプおよびその依存関係によって異なります。JVM サーバーを再始動することなく、アプリケーションの OSGi バンドルを更新できます。ただし、ミドルウェア・バンドルを更新するには、JVM サーバーの再始動が必要です。

## このタスクについて

標準的な JVM サーバーでは、次の図に示されているように、OSGi フレームワークに OSGi バンドルの混合が入っています。




バンドル A とバンドル B は別々の Java アプリケーションであり、別々の CICS バンドル内の OSGi バンドルとしてパッケージされています。両方のアプリケーションは、バンドル C でパッケージされている共通ライブラリーに依存しています。バンドル C は別個に管理され、更新されます。さらに、バンドル B は、WebSphere MQ ミドルウェア・バンドルと JCICS システム・バンドルに依存しています。

バンドル A と B の両方は、フレームワーク内の他のバンドルに影響を与えることなく、独立して更新できます。ただし、バンドル C を更新すると、それに依存する両方のバンドルに影響を与える可能性があります。バンドル C のどのエクスポート

されたパッケージも、OSGI フレームワーク内のメモリーにとどまります。このため、バンドル C での変更を有効にするには、バンドル A と B もこのフレームワークで更新する必要があります。

ミドルウェア・バンドルは、フレームワーク・サービスを含み、JVM サーバーのライフサイクルで管理されます。例えば、ネイティブ・コードをフレームワークに 1 回ロードさせたり、WebSphere MQ などの別の製品にアクセスするためにドライバーを追加したりすることができます。

CICS には、OSGi フレームワークとの対話を管理するためにシステム・バンドルが用意されています。これらのバンドルは、製品の一部として IBM から提供されます。システム・バンドルの例として、CICS サービスにアクセスするための JCICS API のほとんどを提供する `com.ibm.cics.server.jar` ファイルがあります。

『OSGi JVM サーバーの OSGi バンドルの更新』
Java 開発者から、更新されたバージョンの OSGi バンドルが提供された場合、このバンドルを参照している CICS バンドルを完全に置き換えるか、または新しいバージョンの OSGi バンドルを段階的に導入 (フェーズイン) することができます。

197 ページの『共通ライブラリーを含むバンドルの更新』
他の OSGi バンドルで使用するための共通ライブラリーを含む OSGi バンドルは、特定の順序で更新される必要があります。

199 ページの『OSGi ミドルウェア・バンドルの更新』
OSGi フレームワークで実行中のミドルウェア・バンドルを更新するには、JVM サーバーを停止してから、再始動する必要があります。

## OSGi JVM サーバーの OSGi バンドルの更新

Java 開発者から、更新されたバージョンの OSGi バンドルが提供された場合、このバンドルを参照している CICS バンドルを完全に置き換えるか、または新しいバージョンの OSGi バンドルを段階的に導入 (フェーズイン) することができます。

### このタスクについて

使用する更新方式は、以下の要因によって決まります。

- 更新時のサービス停止が許容されるかどうか。
- 更新時の CICS リソース変更が許容されるかどうか。

194 ページの『CICS リソースの変更を伴わない OSGi バンドルの段階的な導入 (フェーズイン)』
更新時のサービスの停止および CICS リソースの変更が許容されない場合は、この更新方法を使用して新しいバージョンの OSGi バンドルを段階的に導入します。

195 ページの『CICS リソースの変更を伴う OSGi バンドルの段階的な導入 (フェーズイン)』
更新処理時のサービスの停止が許容されない場合には、この更新方法を使用して新しいバージョンの OSGi バンドルを段階的に導入します。更新中に新しい CICS リソースが作成されます。

196 ページの『OSGi JVM サーバーの OSGi バンドルの置き換え』
更新処理時のサービスの停止が許容される場合には、この更新方法を使用しま

す。新しい CICS リソースは作成されませんが、既存の BUNDLE リソース定義を更新することが必要になる場合があります。

## CICS リソースの変更を伴わない OSGi バンドルの段階的な導入 (フェーズイン)

更新時のサービスの停止および CICS リソースの変更が許容されない場合は、この更新方法を使用して新しいバージョンの OSGi バンドルを段階的に導入します。

### 始める前に

新しいバージョンの OSGi バンドルの JAR ファイルは、古いバージョンの OSGi バンドルと同じ zFS ディレクトリー、つまり関連付けられている `osgibundle bundlepart` ファイルと同じディレクトリーに置く必要があります。デフォルトでは、このディレクトリーは BUNDLE リソース定義で指定されているディレクトリーです。この新しい OSGi バンドルのバージョンは、OSGi フレームワークに現在インストールされているものより上位のバージョンでなければならず、かつ OSGi バンドル参照が CICS バンドル・プロジェクトに追加されたときに定義されたバージョン範囲内になければなりません。

### 手順

OSGi JVM サーバーに新しいバージョンの OSGi バンドルを段階的に導入するには、以下の手順を使用します。

1. CICS Explorer の「バンドル」ビューで、OSGi バンドルを含む CICS バンドルを右クリックして「**フェーズイン**」をクリックし、次に「**OK**」をクリックします。新しいバージョンの OSGi バンドルが段階的に導入 (フェーズイン) され、新しいバージョンの OSGi バンドルによって実装されているすべてのサービスの新しいバージョンが OSGi フレームワークにインストールされ、古いバージョンのサービスが OSGi フレームワークから削除されます。
2. CICS Explorer の「OSGi サービス」ビューで、新しいバージョンの OSGi バンドルのすべての OSGi サービスがアクティブ状態であることを確認します。
3. CICS Explorer の「OSGi バンドル」ビューで、新しいバージョンの OSGi バンドルがリストされていてアクティブ状態であることを確認します。

### タスクの結果

すべての新しいサービス要求に、新しいバージョンの OSGi バンドルが使用されます。既存の要求には引き続き古いバージョンが使用されます。 OSGi バンドルのシンボル・バージョンが増え、Java コードが更新されたことを示します。

### 次のタスク

新しいバージョンの OSGi サービスが正常に動作していることが確認されたら、作業は完了です。必要に応じて、古いバージョンの OSGi JAR ファイルを zFS から削除できますが、これは必須ではありません。新しいバージョンで問題が起きた場合に古いバージョンに戻せるように、古いバージョンの OSGi JAR ファイルを残しておくのも良いでしょう。

新しいバージョンの OSGi サービスに満足できず、古いバージョンに戻したい場合は、以下の手順を使用します。

1. 新しいバージョンの OSGi バンドル JAR を zFS から削除します。
2. CICS Explorer の「バンドル」ビューで、OSGi バンドルを含む CICS バンドルを右クリックして「**フェーズイン**」をクリックし、次に「**OK**」をクリックします。古いバージョンの OSGi バンドルが現時点で zFS にある最上位のバージョンなので、そのバージョンが OSGi フレームワークに再インストールされて、問題のある新しいバージョンは削除されます。
3. CICS Explorer の「OSGi サービス」ビューで、古いバージョンの OSGi バンドルの OSGi サービスだけがリストされていて、それらがみなアクティブ状態であることを確認します。
4. CICS Explorer の「OSGi バンドル」ビューで、古いバージョンの OSGi バンドルだけがリストされていてアクティブ状態であることを確認します。

CICS のコールド・リスタート、ウォーム・リスタート、または緊急リスタートがある場合には、新しいバージョンの OSGi バンドルが自動的に復元されます。この動作が行われるようにするために、CICS リソース定義を変更する必要はありません。

## CICS リソースの変更を伴う OSGi バンドルの段階的な導入 (フェーズイン)

更新処理時のサービスの停止が許容されない場合には、この更新方法を使用して新しいバージョンの OSGi バンドルを段階的に導入します。更新中に新しい CICS リソースが作成されます。

### 始める前に

新しいバージョンの OSGi バンドルを含む CICS バンドルは既に定義済みで、zFS にエクスポートされています。この新しい OSGi バンドルのバージョンは、OSGi フレームワークに現在インストールされているバージョンより上位のバージョンとして OSGi バンドル・マニフェストに指定されている必要があります。同時に両方の OSGi バンドルをフレームワーク内で実行することが可能です。

### 手順

OSGi JVM サーバーに新しいバージョンの OSGi バンドルを段階的に導入するには、以下の手順を使用します。

1. CICS Explorer の「バンドル定義」ビューで、任意の場所を右クリックしてから「**新規**」をクリックし、 zFS 上に新しい CICS バンドルを取得するためのバンドル・プロジェクトを作成します。
2. CICS Explorer の「バンドル定義」ビューで、前の手順で作成した BUNDLE リソースを右クリックして「**インストール**」をクリックします。インストールのターゲットを選択してから「**OK**」をクリックして、 OSGi バンドルと CICS バンドルの OSGi サービスを OSGi フレームワークにインストールします。
3. CICS Explorer の「OSGi バンドル」ビューで、OSGi バンドルの状況を確認します。2 つのバージョンの OSGi バンドルがアクティブな状態でリストされます。
4. CICS Explorer の「OSGi サービス」ビューで、両方のバージョンの OSGi バンドルによって実装されている OSGi サービスがみなアクティブ状態であることを確認します。新しいサービス呼び出しには、最上位のセマンティック・バージョンを持つ OSGi バンドルを参照する OSGi サービスが使用されます。

### タスクの結果

OSGi フレームワークでは、更新された OSGi バンドルは、古いバージョンの OSGi バンドルと共に使用可能です。

すべての新しいサービス要求に、新しいバージョンの OSGi バンドルが使用されます。既存の要求には引き続き古いバージョンが使用されます。

### 次のタスク

新しいバージョンの OSGi サービスが正常に動作していることが確認されたら、以下の手順を使用して、古いバージョンを OSGi フレームワークから削除します。

1. 旧バージョンの OSGi バンドルを指し示す BUNDLE リソースを使用不可にします。CICS Explorer の「バンドル」ビューで、古い BUNDLE を右クリックして「**使用不可**」をクリックし、次に「**OK**」をクリックします。その OSGi バンドルによって実装されているすべてのサービスの古いバージョンが OSGi フレームワークから削除され、新しいバージョンのサービスのみが「OSGi サービス」ビューにリストされるようになります。「OSGi バンドル」ビューで、古い OSGi バンドルの状態がすべて解決済みになります。
2. 古いバージョンの OSGi バンドルを指し示す BUNDLE リソースを破棄します。CICS Explorer の「バンドル」ビューで、古い BUNDLE を右クリックして「**破棄**」をクリックし、次に「**OK**」をクリックします。「OSGi バンドル」ビューでは、古い OSGi バンドルが OSGi フレームワークから削除され、新しいバージョンの OSGi バンドルのみがリストされます。

新しいバージョンの OSGi バンドルがインストールされたことを確認するため、CICS 領域のコールド・スタートがある場合は、必ず、古いバージョンの BUNDLE 定義を含む CSD グループを参照している CICS グループ・リスト (GRPLIST システム初期化パラメーター) を、手順 1 で作成した新しい BUNDLE 定義を含む CSD グループを参照するように更新してください。

古いバージョンの OSGi バンドルを復元する場合は、上記の 2 つの手順を使用して、新しいバージョンの OSGi バンドルを指し示す BUNDLE リソースを使用不可にして破棄します。古いバージョンのサービスが CICS Explorer の「OSGi サービス」ビューにリストされ、このサービスが新しいサービス呼び出しに使用されます。

## OSGi JVM サーバーの OSGi バンドルの置き換え

更新処理時のサービスの停止が許容される場合には、この更新方法を使用します。新しい CICS リソースは作成されませんが、既存の BUNDLE リソース定義を更新することが必要になる場合があります。

### 始める前に

CICS バンドルを完全に置き換えるには、新しいバージョンの OSGi バンドルを含む、更新された CICS バンドルが zFS に存在しなければなりません。

### 手順

OSGi JVM サーバーにある既存の OSGi バンドルを新しいバージョンの OSGi バンドルに置き換えるには、以下の手順を使用します。

1. CICS Explorer の「バンドル」ビューで、更新する CICS バンドルの BUNDLE リソースを使用不可にして破棄します。その CICS バンドルの一部である OSGi サービスは、OSGi フレームワークから除去され、CICS Explorer の「OSGi サービス」ビューにリストされなくなります。

   **注:** OSGi バンドルによって実装されているサービスはこの時点から手順 3 が完了するまで OSGi フレームワークで使用できなくなるため、このサービスのすべてのユーザーがサービス停止の影響を被ります。
2. オプション: 更新された CICS バンドルを zFS 内の別のディレクトリーにデプロイした場合は、BUNDLE リソース定義を編集します。
3. CICS Explorer の「バンドル」ビューで、この BUNDLE リソース定義をインストールして、変更された OSGi バンドルを取得します。 CICS バンドル内の OSGi バンドルとサービスが、OSGi フレームワークにインストールされます。
4. CICS Explorer の「OSGi バンドル」ビューで、OSGi バンドルの状況を確認します。新しいバージョンの OSGi バンドルがアクティブな状態でリストされます。
5. CICS Explorer の「OSGi サービス」ビューで、新しいバージョンの OSGi バンドルによって実装されているすべての新しいバージョンの OSGi サービスがみなアクティブ状態であることを確認します。

### タスクの結果

すべての新しいサービス要求に、新しいバージョンの OSGi バンドルが使用されます。既存の要求には引き続き古いバージョンが使用されます。 OSGi バンドルのシンボル・バージョンが増え、Java コードが更新されたことを示します。

## 共通ライブラリーを含むバンドルの更新

他の OSGi バンドルで使用するための共通ライブラリーを含む OSGi バンドルは、特定の順序で更新される必要があります。

### 始める前に

新しいバージョンの OSGi バンドルを含む、更新された CICS バンドルが、zFS に存在しなければなりません。共通のライブラリーを別の CICS バンドルで管理する場合は、それらのライブラリーのライフサイクルを、それらに依存しているアプリケーションとは別個に管理することができます。

### このタスクについて

通常、OSGi バンドルには、別の OSGi バンドルへの依存関係についてサポートされるバージョンの範囲を指定します。範囲を使用することで、より柔軟にフレームワーク内で共存可能な変更を行うことができます。共通ライブラリーを含むバンドルを更新する場合、OSGi バンドルのバージョン番号が増えます。ただし、実行中のアプリケーションは、依存関係に対応するバージョンのバンドルを既に使用しています。最新バージョンのライブラリーを取得するには、アプリケーションの OSGi バンドルをリフレッシュする必要があります。そのため、特定のアプリケーション

は別のバージョンのライブラリーを使用するように更新して、その他のアプリケーションは古いバージョンを実行する状態で残すことが可能です。

共通ライブラリーを含む OSGi バンドルを更新する場合、その CICS の BUNDLE リソースを完全に置き換えることができます。しかし、クラスがライブラリーにロードされていないと、従属するバンドルがエラーを受け取る可能性があります。そのため、代わりの方法として、新しいバージョンのライブラリーをインストールして、元のバージョンのライブラリーと一緒にフレームワーク内で実行することができます。OSGi バンドルに異なるバージョン番号がある場合、OSGi フレームワークは両方のバンドルを並行して実行できます。

## 手順

OSGi JVM サーバーで既存の OSGi バンドルを置き換えるには、以下のようにします。

1. 新しいバージョンの CICS バンドル (共通ライブラリーを定義する OSGi バンドルを含む) を指し示す CICS BUNDLE リソースを定義してインストールします。CICS は、新しいバージョンの OSGi バンドルを OSGi フレームワークに定義します。既存の OSGi バンドルは、引き続き前のバージョンのライブラリーを使用します。
2. CICS Explorer の「OSGi バンドル」ビューで (**操作 > Java > OSGi バンドル**)、OSGi バンドルの状況を確認します。リストには、フレームワーク内で実行中の、シンボル名が同じでバージョンが異なる 2 つの OSGi バンドルの項目が表示されます。
3. 従属 Java アプリケーションの新規バージョンのライブラリーを取得するには、以下のいずれかの手順を使用します。
   - Java アプリケーションの CICS バンドルを置き換えます。
     a. Java アプリケーションの CICS BUNDLE リソースを使用不可にして破棄します。
     b. Java アプリケーションの CICS BUNDLE リソースを再インストールします。
   - 新しいバージョンの Java アプリケーションを段階的に導入 (フェーズイン) します。
     a. Java 開発者に、OSGi バンドルのバージョン情報を更新するよう依頼します。新しいバージョンの OSGi バンドルは、OSGi バンドル・マニフェストで指定された上位バージョンでなければならず、OSGi バンドル・プロジェクトが CICS バンドルに追加されたときに指定されたバージョン範囲内である必要もあります。必要に応じて、新しいバージョンの OSGi バンドルの依存関係を、共通ライブラリーが定義されている新しいバージョンの OSGi バンドルを必要とするように変更することもできます。
     b. 新しいバージョンの OSGi バンドルの JAR を CICS BUNDLE リソースのルート・ディレクトリーにコピーします。
     c. CICS Explorer の「**操作**」 > 「**バンドル**」ビューで、OSGi バンドルを含む CICS バンドルを右クリックして「**フェーズイン**」をクリックし、次に「**OK**」をクリックして、新しいバージョンの OSGi バンドルを段階的に導入します。

OSGi バンドルがフレームワークにロードされると、バンドルは最新バージョンの共通ライブラリーを取得します。

4. CICS Explorer の「バンドル」ビューで (「**操作**」 > 「**バンドル**」)、CICS BUNDLE リソースの状況を確認します。

### タスクの結果

共通ライブラリーを含む OSGi バンドルを更新し、最新バージョンのライブラリーを使用するように Java アプリケーションを更新しました。

## OSGi ミドルウェア・バンドルの更新

OSGi フレームワークで実行中のミドルウェア・バンドルを更新するには、JVM サーバーを停止してから、再始動する必要があります。

### このタスクについて

OSGi ミドルウェア・バンドルは、JVM サーバーの初期化中に OSGi フレームワークにインストールされます。例えば、パッチを適用したり、新しいバージョンを使用したりするために、ミドルウェア・バンドルを更新したい場合、変更されたバンドルを取得するために、JVM サーバーを停止してから、再始動する必要があります。

CICS Explorer を使用することによって、JVM サーバーのライフサイクルを管理したり、JVM プロファイルを編集したりすることができます。

### 手順

1. 新しいバージョンのミドルウェア・バンドルが、CICS が読み取りおよび実行アクセス権限を持つ、zFS 上のディレクトリーにあることを確認します。CICS には、ファイルへの読み取りアクセス権限も必要です。
2. zFS ディレクトリー名またはファイル名が JVM プロファイルに指定された値と異なる場合、JVM サーバーの JVM プロファイルの `OSGI_BUNDLES` オプションを編集します。
   a. CICS Explorer の「JVM サーバー」ビューを開き、zFS 内の JVM プロファイルの名前と場所を見つけます。 JVMSERVER リソースを参照するには、選択した領域または CICSplex に接続されている必要があります。
   b. 「z/OS UNIX ファイル」ビューを開き、JVM プロファイルがあるディレクトリーを参照します。
   c. JVM プロファイルを編集して、`OSGI_BUNDLES` オプションを更新します。
3. JVMSERVER リソースを使用不可にして、JVM サーバーをシャットダウンします。 JVMSERVER を使用不可にすると、その JVM サーバーにインストールされている OSGi バンドルを含むすべての BUNDLE リソースも使用不可になります。
4. JVMSERVER リソースを使用可能にして、更新された JVM プロファイルを使用して JVM サーバーを始動します。 JVM サーバーが始動し、新しいバージョンのミドルウェア・バンドルを OSGi フレームワークにインストールします。また、CICS は、使用不可になった BUNDLE リソースを使用可能にし、更新されたフレームワークに OSGi バンドルとサービスをインストールします。

### タスクの結果

OSGi フレームワークには、更新されたミドルウェア・バンドル、および JVM サーバーをシャットダウンする前にインストールされていた、Java アプリケーション用の OSGi バンドルとサービスが入っています。

## JVM サーバーからの OSGi バンドルの除去

JVM サーバーから OSGi バンドルを除去したい場合は、CICS Explorer を使用して BUNDLE リソースを使用不可にして破棄します。

### このタスクについて

BUNDLE リソースは、CICS バンドルで定義される OSGi バンドルと OSGi サービスの集合に対するライフサイクル管理を行います。OSGi フレームワークから OSGi バンドルを除去しても、インストールされている他の OSGi バンドルやサービスの状態に自動的に影響を与えることはありません。別のバンドルの前提条件であるバンドルを除去しても、そのバンドルを明示的にリフレッシュするまで従属バンドルの状態は一般的には変わりません。シングルトン・バンドルを使用する場合は例外です。他のバンドルが依存しているシングルトン・バンドルをアンインストールした場合、従属バンドルは、アンインストールされたバンドルのサービスを使用できません。報告される CICS BUNDLE リソースの状況は、OSGi バンドルの状況を正確には反映していない場合があります。

### 手順

1. 「**Operations**」 > 「**Java**」 > 「**OSGi Bundles**」をクリックして、OSGi バンドルが入っている BUNDLE リソースを検出します。
2. 「**Operations**」 > 「**Bundles**」をクリックして、BUNDLE リソースを使用不可にします。 CICS は、CICS バンドルで定義されている各リソースを使用不可にします。 OSGi バンドルとサービスについては、CICS は JVM サーバーの OSGi フレームワークに要求を送信して OSGi サービスの登録を解除し、OSGi バンドルを解決済み状態にします。すべての未完了トランザクションが完了しますが、CICS アプリケーションから OSGi サービスへの新しいリンクはすべて、エラーで戻ります。
3. BUNDLE リソースを破棄します。 CICS は OSGi フレームワークに要求を送信して、JVM サーバーから OSGi バンドルを除去します。

### タスクの結果

OSGi バンドルとサービスを OSGi フレームワークから除去しました。

### 次のタスク

OSGi フレームワークに存在しなくなった OSGi サービスを指す PROGRAM リソースがある場合、それらの PROGRAM リソースを使用不可にし、破棄する必要がある可能性があります。

# JVM サーバーへのアプリケーションの移動

プールされた JVM で Java アプリケーションを実行している場合、JVM サーバーで実行するようにそれらのアプリケーションを移動することができます。JVM サーバーは、同じ JVM 内で Java アプリケーションに対する複数の要求を処理できるので、同じワークロードの実行に必要な JVM 数を減らすことができます。

## 始める前に

アプリケーションがスレッド・セーフであり、1 つ以上の OSGi バンドルとしてパッケージされていることを確認してください。これらの OSGi バンドルは、1 つの CICS バンドルで zFS にデプロイされ、正しいターゲット JVMSERVER リソースを指定する必要があります。

Java 開発者は、CICS Explorer SDK を使用すると、OSGi を使用して Java アプリケーションを再パッケージすることができます。

## このタスクについて

アプリケーション用に既存の JVM サーバーを使用するか、アプリケーション用に JVM サーバーを作成することができます。スレッド限度と使用量が既に高い JVM サーバーにアプリケーションを移動しないでください。その JVM サーバーでロッキングの競合が生じる可能性があるからです。

## 手順

1. JVM サーバーを作成または更新します。
   - JVM サーバーを作成する場合は、135 ページの『JVM サーバーのセットアップ』を参照してください。プールされた JVM の JVM プロファイルの設定の多くは、JVM サーバーに適用されません。プールされた JVM プロファイルから DFHOSGI プロファイルにコピーできるオプションは、`LIBPATH_SUFFIX` オプションのみです。
   - 既存の JVM サーバーを使用する場合は、JVMSERVER リソースの THREADLIMIT 属性を増やして追加アプリケーションを処理するか、JVM サーバー・プロファイルのオプションを更新する必要がある可能性があります。JVM プロファイルを変更する場合、変更を有効にするために JVM サーバーを再始動します。
2. zFS でデプロイされたバンドルを指す BUNDLE リソースを作成します。BUNDLE リソースをインストールすると、CICS は、JVM サーバーの OSGi フレームワークに OSGi バンドルをロードします。OSGi フレームワークは、OSGi バンドルを解決し、OSGi サービスを登録します。 CICS Explorer を使用して、BUNDLE リソースが使用可能であることを確認してください。また、「OSGi Bundles」ビューおよび「OSGi Services」ビューを使用して、OSGi バンドルとサービスの状態を確認することもできます。
3. アプリケーションの PROGRAM リソースを更新します。
   a. EXECKEY 属性が CICS に設定されていることを確認します。 すべての JVM サーバーの作業は CICS キーで実行されます。
   b. JVM プロファイル名を除去し、JVMSERVER リソースの名前を入力します。

c. JVMCLASS 属性が、Java アプリケーションの OSGi サービスと一致することを確認します。

d. アプリケーションの PROGRAM リソースを再インストールします。

PROGRAM リソースは、OSGi サービスを使用して、OSGi バンドルを JVM サーバー外部の他の CICS アプリケーションから使用可能にします。

### タスクの結果

Java アプリケーションが呼び出されると、JVM サーバーで実行されます。

### 次のタスク

CICS Explorer の JVM サーバー・ビュー、および CICS 統計を使用して、JVM サーバーをモニターすることができます。パフォーマンスが最適でない場合は、スレッド限度を調整してください。

## JVM サーバーのスレッド限度の管理

JVM サーバーでは、Java アプリケーションの実行に使用できるスレッド数が制限されています。また、各スレッドが 1 つの T8 TCB を使用するため、CICS 領域にもスレッド数の制限があります。 CICS 統計を使用してスレッド限度を調整すると、領域内の JVM サーバー数と、各 JVM サーバーで実行されるアプリケーションのパフォーマンスとのバランスを取ることができます。

### このタスクについて

各 JVM サーバーには、Java アプリケーションを実行するために最大 256 個のスレッドがあります。 CICS 領域には最大 2000 個のスレッドを備えることができます。 CICS 領域で多数の JVM サーバー (例えば、8 つ以上) を実行している場合、どの JVM サーバーにも最大値を設定できるわけではありません。各 JVM サーバーのスレッド限度を調整して、CICS 領域内の JVM サーバー数と、Java アプリケーションのパフォーマンスとのバランスを取ることができます。

スレッド限度は JVMSERVER リソースで設定されるので、初期値を設定し、CICS 統計を使用して、Java ワークロードのテスト時にスレッド数を調整してください。

### 手順

1. JVMSERVER リソースを使用可能にし、Java アプリケーションのワークロードを実行します。
2. 適切な統計間隔を使用して JVMSERVER リソース統計を収集します。 CICS Explorer で「**操作**」 > 「**Java**」 > 「**JVM サーバー**」ビューを使用するか、DFH0STAT 統計プログラムを使用することができます。
3. タスクでスレッドを待機した回数と時間を確認します。 「JVMSERVER thread limit waits」フィールドと「JVMSERVER thread limit wait time」フィールドに、この情報が含まれています。
   - これらのフィールドの値が高く、多数のタスクが JVMTHRD 待機で中断する場合、JVM サーバーには使用可能な十分なスレッドがありません。スレッド数を増やすと、プロセッサーの使用量が増える可能性があるので、十分な MVS リソースが使用可能であることを確認してください。

- これらのフィールドの値が低く、ピーク時のタスク数が使用可能な最大スレッド数より少ない場合、スレッド限度を減らして、他の JVM サーバー用にスレッドを解放することができます。

4. MVS リソースが使用可能かどうかを確認するために、ディスパッチャー TCB プール統計と TCB モード統計を使用して、CICS 領域全体での T8 TCB の使用量を判断します。 JVM サーバーの各スレッドは 1 つの T8 TCB を使用し、1 つの領域内で 2000 個に制限されます。 T8 TCB は複数の JVM サーバー間で共用できませんが、すべての TCB は 1 つの THRD TCB プール内にあります。待機している TCB 数とプロセッサー使用量が少ない場合、十分な MVS リソースが使用可能であることを示します。
5. JVM サーバーで実行できるスレッド数を調整するには、JVMSERVER リソースの THREADLIMIT 値を変更します。
6. Java アプリケーションのワークロードを再度実行し、統計を使用して、待機しているタスクの数が減ったことを確認します。

### 次のタスク

JVM サーバーのパフォーマンスを調整するには、215 ページの『JVM サーバーのパフォーマンスの改善』を参照してください。

## CICS リスタート後の OSGi バンドル・リカバリー

OSGi バンドルを含む CICS 領域をリスタートすると、CICS は BUNDLE リソースをリカバリーし、OSGi バンドルを JVM サーバーのフレームワークにインストールします。

CICS バンドルにパッケージされる OSGi バンドルは、CSD に保管されません。BUNDLE リソース自体はカタログに保管されるため、CICS 領域のリスタート後、BUNDLE リソースが復元されると OSGi バンドルは動的に再作成されます。

CICS のコールド・リスタート、ウォーム・リスタート、または緊急リスタート後、JVM サーバーは、BUNDLE リソースのリカバリーとは非同期に始動されます。CICS リスタートで OSGi バンドルを正常に復元するには、JVM サーバーが完全に使用可能でなければなりません。したがって、BUNDLE リソースは CICS 開始の最後の段階時にリカバリーされますが、OSGi バンドルがインストールされるのは、JVM サーバーがその開始を完了したときのみです。

BUNDLE リソースとそれに含まれている OSGi バンドルは正しい順にインストールされ、CICS バンドルと OSGi バンドルの両方の間の依存関係がフレームワークで確実に解決されるようになります。CICS が OSGi バンドルをインストールできない場合、BUNDLE リソースは Disabled 状態でインストールされます。IBM CICS Explorer を使用すると、BUNDLE リソース、OSGi バンドル、および OSGi サービスの状態を表示できます。

# JVM stdout および stderr 出力をリダイレクトするための Java クラスの作成

JVM プロファイルの USEROUTPUTCLASS オプションを使用して、JVM からの stdout ストリームと stderr ストリームを代行受信する Java クラスを指定します。このクラスを更新すると、適当なタイム・スタンプとレコード・ヘッダーを指定し、出力をリダイレクトすることができます。

CICS は、この目的に使用できるサンプル Java クラス com.ibm.cics.samples.SJMergedStream および com.ibm.cics.samples.SJTaskStream を用意しています。/usr/lpp/cicsts/cicsts53/samples/com.ibm.cics.samples ディレクトリーに、これらの両方のクラスのサンプル・ソースが提供されています。/usr/lpp/cicsts/cicsts53 ディレクトリーは、z/OS UNIX で CICS ファイル用のインストール・ディレクトリーです。このディレクトリーは、DFHISTAR インストール・ジョブの **USSDIR** パラメーターで指定されます。また、これらのサンプル・クラスは、クラス・ファイル com.ibm.cics.samples.jar として出荷時に付属しています。このクラス・ファイルは、/usr/lpp/cicsts/cicsts53/lib ディレクトリーにあります。これらのクラスを変更するか、サンプルに基づいて独自のクラスを作成することができます。

257 ページの『JVM stdout、stderr、JVMTRACE、およびダンプ出力の場所の制御』には、以下に関する情報が記載されています。

- JVM からの出力のタイプで、USEROUTPUTCLASS オプションによって指定されるクラスによって代行受信されるものと、されないもの。使用するクラスは、代行受信する可能性があるすべてのタイプの出力を処理できなければなりません。
- 提供されたサンプル・クラスの動作。com.ibm.cics.samples.SJMergedStream クラスは、JVM 出力用とエラー・メッセージ用にマージされた 2 つのログ・ファイルを作成します。各レコードには、アプリケーション ID、日付、時刻、トランザクション ID、タスク番号、およびプログラム名を含むヘッダーが付けられます。これらのログ・ファイルは、一時データ・キューが使用可能な場合は、その一時データ・キューを使用して作成されます。一時データ・キューが使用不可であるか、Java アプリケーションで使用できない場合は、z/OS UNIX ファイルを使用して作成されます。com.ibm.cics.samples.SJTaskStream クラスは、単一のタスクからの出力を z/OS UNIX ファイルに送信し、タイム・スタンプとヘッダーを追加して、単一のタスクに固有の出力ストリームを提供します。

JVM サーバーが出力リダイレクト・クラスを使用するには、出力リダイレクト・クラスを含む OSGi バンドルを作成する必要があります。バンドル・アクティベーターがクラスのインスタンスをフレームワーク内のサービスとして登録し、プロパティー com.ibm.cics.server.outputredirectionplugin.name=*class_name* を確実に設定しておく必要があります。定数 com.ibm.cics.server.Constants.CICS_USER_OUTPUT_CLASSNAME_PROPERTY を使用して、プロパティー名を取得できます。次のコードの抜粋は、バンドル・アクティベーターでサービスを登録する方法を示しています。

```
Properties serviceProperties = new Properties();
    serviceProperties.put(Constants.CICS_USER_OUTPUT_CLASSNAME_PROPERTY, MyOwnStreamPlugin.class.getName());
    context.registerService(OutputRedirectionPlugin.class.getName(), new MyOwnStreamPlugin(), serviceProperties);
```

OSGi バンドルを JVM プロファイルの OSGI_BUNDLES オプションに追加するか、または最初のタスクの実行時にバンドルがフレームワークに確実にインストールされるようにすることができます。どちらの方法を使用しても、USEROUTPUTCLASS オプションでクラスを指定する必要があります。

独自のクラスを作成する場合は、以下を理解している必要があります。

- OutputRedirectionPlugin インターフェース
- 出力の予想される宛先
- 出力リダイレクト・エラーと内部エラーの処理

『出力リダイレクト・インターフェース』
CICS は、com.ibm.cics.server.jar に com.ibm.cics.server.OutputRedirectionPlugin と呼ばれるインターフェースを備えています。このインターフェースは、JVM からの stdout および stderr 出力を代行受信するクラスによって実装できます。提供のサンプルはこのインターフェースを実装します。

206 ページの『出力の予想される宛先』
CICS 提供のサンプル・クラスは、JVM からの出力を、CICS 領域に固有のディレクトリーに送信します。このディレクトリー名は、その CICS 領域に関連したアプリケーション ID を使用して作成されます。独自のクラスを作成する場合、必要に応じて、複数の CICS 領域から、同じ z/OS UNIX ディレクトリーまたはファイルに出力を送信することができます。

207 ページの『出力リダイレクト・エラーと内部エラーの処理』
ご使用のクラスで、CICS 機能を使用して出力をリダイレクトする場合、それらのクラスには、これらの機能を使用する際のエラーを処理するために適切な例外処理が含まれていなければなりません。

## 出力リダイレクト・インターフェース

CICS は、com.ibm.cics.server.jar に com.ibm.cics.server.OutputRedirectionPlugin と呼ばれるインターフェースを備えています。このインターフェースは、JVM からの stdout および stderr 出力を代行受信するクラスによって実装できます。提供のサンプルはこのインターフェースを実装します。

以下のサンプル・クラスが用意されています。

- このインターフェースを実装するスーパークラス com.ibm.cics.samples.SJStream。
- JVM プロファイルで指定されるクラスである、サブクラス com.ibm.cics.samples.SJMergedStream および com.ibm.cics.samples.SJTaskStream。

サンプル・クラスのように、ご使用のクラスがインターフェース OutputRedirectionPlugin を直接実装するか、このインターフェースを実装するクラスを拡張することを確認してください。スーパークラス com.ibm.cics.samples.SJStream から継承するか、同じインターフェースを持つクラス構造を実装することができます。どちらのメソッドを使用する場合でも、クラスは java.io.OutputStream を拡張する必要があります。

initRedirect() メソッドは、1 つ以上の出力リダイレクト・クラスで使用される 1 組のパラメーターを受け取ります。次のコードはインターフェースを示しています。

```
package com.ibm.cics.server;

import java.io.*;

public interface OutputRedirectionPlugin {

  public boolean initRedirect( String inDest,
                               PrintStream inPS,
                               String inApplid,
                               String inProgramName,
                               Integer inTaskNumber,
                               String inTransid
                              );
  }
```

スーパークラス com.ibm.cics.samples.SJStream には、com.ibm.cics.samples.SJMergedStream および com.ibm.cics.samples.SJTaskStream の共通コンポーネントが含まれています。これに含まれている initRedirect() メソッドは「false」を戻します。これは、このメソッドがサブクラス内の別のメソッドによってオーバーライドされる場合を除いて、出力のリダイレクトを事実上使用不可にします。これは、writeRecord() メソッドを実装しません。このようなメソッドは、出力リダイレクト・プロセスを制御するために任意のサブクラスによって提供されなければなりません。独自のクラス構造でこのメソッドを使用することができます。出力リダイレクトの初期化も、initRedirect() メソッドではなく、コンストラクターを使用して実行できます。

**inPS** パラメーターには、JVM のオリジナルの System.out 印刷ストリームまたはオリジナルの System.err 印刷ストリームのどちらかが入っています。基礎となるこれらのロギング宛先のどちらにでもロギングを書き込むことができます。これらの印刷ストリームのどちらかで close() メソッドを呼び出してはなりません。印刷ストリームは完全にクローズされたままになり、将来使用できないからです。

## 出力の予想される宛先

CICS 提供のサンプル・クラスは、JVM からの出力を、CICS 領域に固有のディレクトリーに送信します。このディレクトリー名は、その CICS 領域に関連したアプリケーション ID を使用して作成されます。独自のクラスを作成する場合、必要に応じて、複数の CICS 領域から、同じ z/OS UNIX ディレクトリーまたはファイルに出力を送信することができます。

例えば、単一のファイルを作成して、複数の異なる CICS 領域で実行される特定アプリケーションに関連した出力をそのファイルに含めることができます。

Thread.start() を使用してプログラマチックに開始されるスレッドでは、CICS 要求を行うことはできません。これらのアプリケーションの場合、JVM からの出力は、USEROUTPUTCLASS に指定されたクラスによって代行受信されますが、CICS 機能 (一時データ・キューなど) を使用してリダイレクトすることはできません。提供されたサンプル・クラスの場合と同様に、これらのアプリケーションからの出力を z/OS UNIX ファイルに送信することができます。

## 出力リダイレクト・エラーと内部エラーの処理

ご使用のクラスで、CICS 機能を使用して出力をリダイレクトする場合、それらのクラスには、これらの機能を使用する際のエラーを処理するために適切な例外処理が含まれていなければなりません。

例えば、一時データ・キュー CSJO および CSJE に書き込もうとするときに、これらのキューに CICS 提供の定義を使用する場合、次の例外が TDQ.writeData によってスローされます。

- IOErrorException
- LengthErrorException
- NoSpaceException
- NotOpenException

ご使用のクラスが出力を z/OS UNIX ファイルに送信する場合、それらのクラスには、z/OS UNIX への書き込み時に発生するエラーを処理する適切な例外処理が含まれていなければなりません。これらのエラーで最も一般的な原因は、セキュリティー例外です。

クラスを USEROUTPUTCLASS オプションで指定する JVM で実行される Java プログラムには、クラスでスローされる可能性がある例外を処理する適切な例外処理が含まれていなければなりません。CICS 提供のサンプル・クラスは、例外を内部で処理します。そのために、Try/Catch ブロックを使用してすべてのスロー可能な例外をキャッチしてから、問題を報告する 1 つ以上のメッセージを書き込みます。出力メッセージのリダイレクト中にエラーが検出されると、これらのエラー・メッセージは `System.err` に書き込まれ、リダイレクトに使用可能にします。しかし、エラー・メッセージのリダイレクト中にエラーが検出される場合、この問題を報告するメッセージは、要求を処理する JVM で使用される JVM プロファイルの STDERR オプションによって指定されるファイルに書き込まれます。このようにサンプル・クラスはすべてのエラーをトラップするため、これは、呼び出し側プログラムが出力リダイレクト・クラスによってスローされる例外を処理する必要がないことを意味します。このメソッドを使用すると、呼び出し側プログラムへの変更を避けることができます。クラスで発行されるエラー・メッセージを、失敗した宛先にリダイレクトしようとして、出力リダイレクト・クラスをループに入れないように注意してください。

# 第 6 章 Java パフォーマンスの改善

Java アプリケーションとそれらのアプリケーションが実行される JVM のパフォーマンスを改善するために、さまざまなアクションを実行することができます。

## このタスクについて

CICS 自体の微調整に加えて、以下の方法で Java アプリケーションのパフォーマンスをさらに高めることができます。

- Java アプリケーションが適切に作成されていることを確認する
- Java ランタイム環境 (JVM) を調整する
- JVM を実行する言語を調整する

## 手順

1. Java ワークロードのパフォーマンス目標を決定します。 最も一般的な目標には、プロセッサー使用率やアプリケーション応答時間の最小化があります。目標を決定したら、Java 環境を調整することができます。
2. Java アプリケーションを分析して、その Java アプリケーションが効率よく動作し、過剰なガーベッジを生成していないことを確認します。 IBM が提供するツールは、Java アプリケーションを分析して、特定の方法およびアプリケーション全体の効率とパフォーマンスを改善するのに役立ちます。
3. JVM サーバーを調整します。 統計と IBM ツールを使用すれば、ストレージの設定、ガーベッジ・コレクション、タスクの待機などの情報を分析して、JVM のパフォーマンスを調整することができます。
4. JVM が実行される Language Environment エンクレーブを調整します。 JVM が使用する MVS ストレージは、MVS Language Environment サービスの呼び出しによって取得されます。Language Environment のランタイム・オプションを変更して、MVS によって割り振られるストレージを調整することができます。
5. オプション: z/OS 共用ライブラリー領域を使用して、異なる CICS 領域内の JVM 間で DLL を共用する場合、ストレージの設定を調整できます。

   210 ページの『Java ワークロードにおけるパフォーマンス・ゴールの判別』
   所定のアプリケーション・ワークロードの全体的なパフォーマンスが最良のものになるように CICS JVM を調整するには、いくつかの異なる要因が関係しています。Java ワークロードに必要なパフォーマンス特性を決定する必要があります。これらの特性を設定すると、変更が必要なパラメーターとその変更方法を判別することができます。

   211 ページの『IBM Health Center を使用した Java アプリケーションの分析』
   Java アプリケーションのパフォーマンスを改善するには、IBM Health Center を使用してそのアプリケーションを分析することができます。このツールは、アプリケーションのパフォーマンスと効率の改善に役立つ推奨事項を提供します。

   212 ページの『ガーベッジ・コレクションおよびヒープ拡張』
   ガーベッジ・コレクションおよびヒープ拡張は、JVM の操作において重要な部

分を占めています。JVM におけるガーベッジ・コレクションの頻度は、JVM で実行するアプリケーションによって作成される不要情報、つまりオブジェクトの量に影響されます。

215 ページの『JVM サーバーのパフォーマンスの改善』
JVM サーバーで実行されているアプリケーションのパフォーマンスを改善するには、ガーベッジ・コレクションやヒープ・サイズを始めとする、環境のさまざまな部分を調整することができます。

220 ページの『JVM の Language Environment エンクレーブ・ストレージ』
JVM サーバーでは、(主に 64 ビット・ストレージで) 静的ストレージ要件と動的ストレージ要件の両方が適用されます。場合によっては、相当な量の 31 ビット・ストレージが使用されます。

228 ページの『z/OS 共用ライブラリー領域の調整』
共用ライブラリー領域は、 z/OS の機能であり、アドレス・スペース間のダイナミック・リンク・ライブラリー (DLL) ファイルの共用を可能にします。この機能により、 CICS 領域は JVM に必要な DLL を共用できるようになり、各領域が DLL を個別にロードする必要はなくなります。これにより、MVS が使用する実際のストレージの量、および領域へのファイルのロード所要時間を大幅に削減できます。

## Java ワークロードにおけるパフォーマンス・ゴールの判別

所定のアプリケーション・ワークロードの全体的なパフォーマンスが最良のものになるように CICS JVM を調整するには、いくつかの異なる要因が関係しています。Java ワークロードに必要なパフォーマンス特性を決定する必要があります。これらの特性を設定すると、変更が必要なパラメーターとその変更方法を判別することができます。

Java ワークロードの最も一般的なパフォーマンス目標は、以下のとおりです。

**最小限の総プロセッサー使用率**

この目標は、使用可能なプロセッサー・リソースを最も効率的に使用することに重点を置いています。この目標を達成するようにワークロードを調整すると、ワークロード全体のプロセッサーの合計使用率は最小化されますが、個々のタスクはプロセッサーを著しく消費する可能性があります。総プロセッサー使用率が最小限になるよう調整するには、JVM に大きいストレージ・ヒープ・サイズを指定して、ガーベッジ・コレクション数を最小化することが必要です。

**アプリケーションの最小応答時間**

この目標は、アプリケーション・タスクをできるだけ素早く呼び出し元に戻すことに重点を置いています。達成しなければならないサービス・レベル・アグリーメントが存在する場合、この目標は特に重要である可能性があります。この目標を達成するようにワークロードを調整すると、アプリケーションは一貫して素早く応答します。ただし、ガーベッジ・コレクションのためにプロセッサー使用率が高くなることがあります。アプリケーションの最小応答時間のための調整には、ヒープ・サイズを小さくしておくことと、おそらく gencon ガーベッジ・コレクション・ポリシーを使用することが必要です。

**最小限の JVM ストレージ・ヒープ・サイズ**

この目標は、JVM によって使用されるストレージの量を削減することに重点を置いています。 JVM で使用されるストレージの量を削減するには、JVM ヒープ・サイズを小さくします。

**注:** JVM ヒープ・サイズを小さくすると、ガーベッジ・コレクション・イベントの頻度が高まる可能性があります。

その他の要因が、アプリケーションの応答時間に影響を与える場合があります。その中でも最も重要なのが、Just In Time (JIT) コンパイラーです。 JIT コンパイラーは、実行時にアプリケーション・コードを動的に最適化し、多くの利点を提供しますが、これを行うには一定量のプロセッサー・リソースが必要になります。

## IBM Health Center を使用した Java アプリケーションの分析

Java アプリケーションのパフォーマンスを改善するには、IBM Health Center を使用してそのアプリケーションを分析することができます。このツールは、アプリケーションのパフォーマンスと効率の改善に役立つ推奨事項を提供します。

### このタスクについて

IBM Health Center は、IBM Support Assistant Workbench で使用できます。これらの無料のツールは、Getting Started guide に記載されているとおりに IBM からダウンロードできます。JVM でアプリケーションを単独で実行してみてください。JVM サーバーで混合ワークロードを実行している場合は、特定アプリケーションの分析がさらに難しくなることがあります。

### 手順

1. 必要な接続オプションを JVM サーバーの JVM プロファイルに追加します。IBM Health Center の資料に、ツールから JVM に接続するために追加する必要があるオプションが記述されています。
2. IBM Health Center を開始し、実行中の JVM に接続します。 IBM Health Center は JVM のアクティビティーをリアルタイムで報告するので、IBM Health Center が JVM をモニターするまでしばらく待ってください。
3. 「**Profiling**」リンクを選択して、アプリケーションのプロファイルを作成します。 さまざまなメソッドで費やす時間を確認できます。使用率が最大のメソッドを確認して、潜在的な問題を探してください。

   **ヒント:** 「**Analysis and Recommendations**」タブでは、最適化の候補になりうる特定のメソッドを識別できます。
4. 「**Locking**」リンクを選択して、アプリケーションにロッキングの競合がないか確認します。 Java ワークロードが、使用可能なすべてのプロセッサーを使用できるとは限らない場合、ロッキングが原因である可能性があります。アプリケーションでのロッキングにより、実行できる並列スレッドの量が減る可能性があります。
5. 「**Garbage Collection**」リンクを選択して、ヒープ使用量とガーベッジ・コレクションを確認します。 「**Garbage Collection**」タブでは、ヒープの使用量、およびガーベッジ・コレクションを実行するために JVM が一時停止する頻度が表示されます。

a. ガーベッジ・コレクションで費やされた時間の割合を確認します。 この情報は「Summary」セクションに表示されます。ガーベッジ・コレクションで費やされた時間が 2% を超える場合、ガーベッジ・コレクションの調整が必要な可能性があります。
b. ガーベッジ・コレクションの一時停止時間を確認します。 一時停止時間が 10 ミリ秒を超える場合、ガーベッジ・コレクションがアプリケーションの応答時間に影響を与えている可能性があります。
c. ガーベッジ・コレクションの比率をトランザクション数で除算して、各トランザクションが生成するおおよそのガーベッジ量を確認します。 ガーベッジの量がアプリケーションにとって多いと思われる場合は、さらにアプリケーションを調査する必要がある可能性があります。

### 次のタスク

アプリケーションを分析した後、Java ワークロードに合わせて Java 環境を調整することができます。

## ガーベッジ・コレクションおよびヒープ拡張

ガーベッジ・コレクションおよびヒープ拡張は、JVM の操作において重要な部分を占めています。JVM におけるガーベッジ・コレクションの頻度は、JVM で実行するアプリケーションによって作成される不要情報、つまりオブジェクトの量に影響されます。

### 割り振り失敗

JVM がストレージ・ヒープでスペース不足になり、それ以上のオブジェクトを割り振ることができない (割り振り失敗) 場合には、ガーベッジ・コレクションがトリガーされます。ガーベッジ・コレクターは、アプリケーションによって参照されなくなったストレージ・ヒープのオブジェクトをクリーンアップし、スペースの一部を解放します。ガーベッジ・コレクションは、ガーベッジ・コレクション・サイクルの期間に JVM で実行中の他のすべての処理を停止するため、ガーベッジ・コレクションに費やされる時間は、アプリケーションの実行には使用されていない時間ということになります。 JVM のガーベッジ・コレクション処理の詳しい説明については、IBM SDK, Java Technology Edition バージョン 7 の z/OS ユーザーズ・ガイドを参照してください。

割り振り失敗によってガーベッジ・コレクションが起動しても、十分なスペースが解放されなかった場合、ガーベッジ・コレクターはストレージ・ヒープを拡張します。ガーベッジ・コレクターは、ヒープ拡張時に、ヒープ用に予約されるストレージの最大量 (`-Xmx` オプションで指定された量) からストレージを取り、それをヒープのアクティブな部分 (`-Xms` オプションで指定されたサイズで始まる) に追加します。開始時に、`-Xmx` オプションで指定されたストレージの最大量が既に JVM に割り振られているため、ヒープ拡張によって、JVM に必要なストレージの量が増えることはありません。 `-Xms` オプションの値が、アプリケーションのヒープのアクティブな部分に十分なストレージを提供する場合、ガーベッジ・コレクターがヒープ拡張を実行する必要はありません。

JVM の存続時間中のある時点で、ガーベッジ・コレクターはストレージ・ヒープの拡張を停止します。これは、ヒープが、ガーベッジ・コレクションの頻度および処理によって解放されるスペースの量に関して、ガーベッジ・コレクターが適切であると判断する状態に達したためです。ガーベッジ・コレクターは割り振り失敗を除去するわけではありません。このため、ガーベッジ・コレクターがストレージ・ヒープの拡張を停止した後も、割り振り失敗により一部のガーベッジ・コレクションを起動することができます。パフォーマンス目標に応じて、ガーベッジ・コレクションの頻度が高すぎないかどうか考慮することができます。

## ガーベッジ・コレクションのオプション

ガーベッジ・コレクションにはさまざまなポリシーを使用でき、これらのポリシーは、アプリケーションとシステム全体のスループットと、ガーベッジ・コレクションが原因の一時停止時間とのトレードオフを行います。ガーベッジ・コレクションは `-Xgcpolicy` オプションによって制御されます。

**`-Xgcpolicy:optthruput`**
このポリシーは、高いスループットをアプリケーションで実現しますが、その代わりに、ガーベッジ・コレクションの処理時に一時停止が生じることがあります。

**`-Xgcpolicy:gencon`**
このポリシーは、ガーベッジ・コレクションの一時停止で費やされる時間を最小限に抑えるのに役立ちます。このガーベッジ・コレクション・ポリシーは JVM サーバーで使用してください。 JVMSERVER リソースを調べると、JVM サーバーで使用されているポリシーを確認することができます。JVM サーバー統計にあるフィールドは、ガーベッジ・コレクションのメジャー・イベントとマイナー・イベントの数、およびガーベッジ・コレクションで費やされたプロセッサー時間を示します。

JVM プロファイルを更新することによって、ガーベッジ・コレクション・ポリシーを変更できます。すべてのガーベッジ・コレクション・オプションについて詳しくは、IBM SDK, Java Technology Edition バージョン 7 の『ガーベッジ・コレクション・ポリシーの指定』を参照してください。

## 例: ガーベッジの生成量が多いマルチスレッド・アプリケーション

214 ページの図 2 は、`gencon` ガーベッジ・コレクション・ポリシーを使用する場合の、さまざまな段階での JVM サーバー内のストレージ・ヒープを示しています。JVM サーバーは同じアプリケーションに対する多数の並行要求を実行できます。したがって、アプリケーションはより大量のガーベッジを生成する可能性があります。 JVM サーバーのガーベッジ・コレクションは、JVM によって自動的に処理されます。

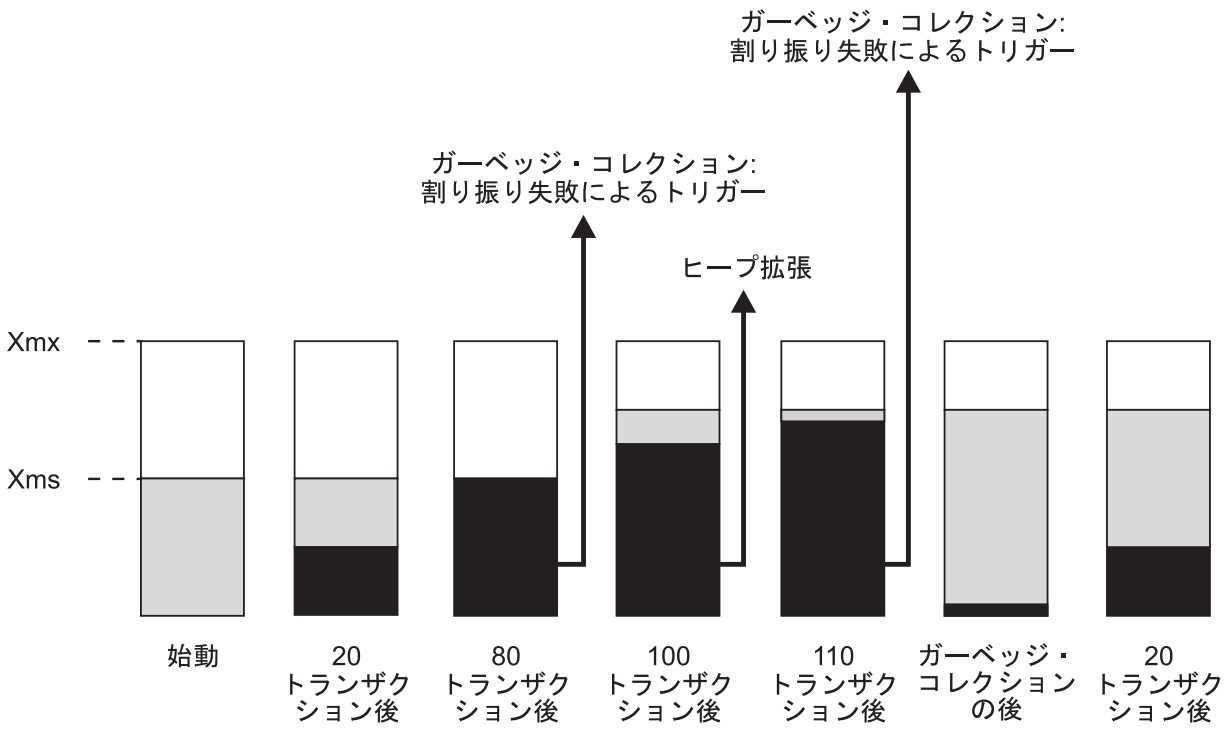



*図 2. ガーベッジの量が多い場合の JVM サーバーのストレージ・ヒープ*

最初の 20 件のトランザクションが行われる過程で、ストレージ・ヒープのアクティブな部分が満杯になり始めます。トランザクション数が 80 に達した後、ヒープは満杯になり、割り振り失敗が発生します。これにより、JVM 内で小規模なガーベッジ・コレクションが起動されます。ガーベッジ・コレクションにより、存続時間の短いオブジェクトはクリーンアップされます。ただし、アプリケーション要求がまだ実行中であるため、オブジェクトの一部は引き続き参照されており、ガーベッジ・コレクションに適格ではありません。

トランザクション数が 100 に達した後、ガーベッジ・コレクターは、現在必要なすべてのオブジェクトのための十分なスペースを見つけられないため、ストレージ・ヒープを拡張します。ストレージ・ヒープ用に予約されるストレージの最大量 (`-Xmx` オプションで指定された量) から、一部のストレージがヒープのアクティブな部分に追加されます。アプリケーションはオブジェクトの生成を継続しますが、ヒープ拡張によって十分なスペースが作成されたため、現行トランザクションを完了することができます。

トランザクション数が 110 に達した後、ストレージ・ヒープはほとんど占有された状態になります。再び割り振り失敗が発生したため、大規模なガーベッジ・コレクションが起動されます。前のトランザクションによって使用されていた、存続時間の長いオブジェクトの多くが JVM によってクリーンアップされます。さらに 20 件のトランザクションが行われた後、ヒープは再び満杯になり始めます。

`gencon` ポリシーを使用すると、大規模なガーベッジ・コレクションが実行される前に、ヒープ・サイズを管理するために多数の小規模なガーベッジ・コレクションが行われることがあります。JVM サーバー統計を使用して、ガーベッジ・コレクショ

ンが行われた回数、ヒープの占有率などの情報を調べることができます。

# JVM サーバーのパフォーマンスの改善

JVM サーバーで実行されているアプリケーションのパフォーマンスを改善するには、ガーベッジ・コレクションやヒープ・サイズを始めとする、環境のさまざまな部分を調整することができます。

## このタスクについて

CICS が提供する JVM サーバーに関する統計レポートには、タスクでスレッドを待機する時間、ヒープ・サイズ、ガーベッジ・コレクションの頻度、およびプロセッサー使用率の詳細が記載されています。また、JVM のモニターと分析を直接行って JVM サーバーを調整し、問題診断に役立つ、追加の IBM ツールを使用することもできます。統計を使用すると、JVM が効率よく動作していること、特にヒープ・サイズが適切で、ガーベッジ・コレクションが最適化されていることを確認することができます。

## 手順

1. JVM サーバーで使用されているプロセッサー時間の量を確認します。 ディスパッチャー統計は、T8 TCB が使用しているプロセッサー時間の量を示すことができます。JVM サーバー統計は、JVM がガーベッジ・コレクションに費やす時間、およびガーベッジ・コレクションの回数を示します。JVM ガーベッジ・コレクションは、アプリケーションの応答時間とプロセッサー使用率に悪影響を与える可能性があります。
2. CICS アドレス・スペースに使用可能なストレージ容量が十分にあることを確認します。CICS アドレス・スペースには、JVM サーバーに必要な Language Environment ヒープ・サイズが含まれます。
3. JVM におけるガーベッジ・コレクションとヒープを調整します。 ヒープが小さいと、ガーベッジ・コレクションの頻度が非常に高くなる可能性があります。一方、ヒープが大きすぎると、MVS ストレージの使用効率が悪くなるおそれがあります。 IBM Health Center を使用すると、ガーベッジ・コレクションの視覚化と調整を行い、それに応じてヒープを調整することができます。

## 次のタスク

メモリー使用量とヒープ・サイズをより詳しく分析するには、IBM Support Assistant の Memory Analyzer ツールを使用すると、Java プロセスのシステム・ダンプまたはヒープ・ダンプのスナップショットを使用して Java ヒープ・メモリーを分析することができます。

CICS 領域で 1 つ以上の JVM サーバーを始動するには、CICS に割り振られているストレージ容量の他に、JVM で使用できるストレージ容量が十分にあることを確認する必要があります。

216 ページの『JVM サーバーによるプロセッサー使用量の確認』
CICS モニター機能を使用すると、JVM サーバーで実行中のトランザクションで使用されるプロセッサー時間をモニターすることができます。 JVM サーバー内のすべてのスレッドは T8 TCB で実行されます。

217 ページの『JVM サーバーのストレージ所要量の計算』
CICS 領域で 1 つ以上の JVM サーバーを始動するには、CICS に割り振られているストレージ容量の他に、JVM で使用できるストレージ容量が十分にあることを確認する必要があります。

219 ページの『JVM サーバー・ヒープとガーベッジ・コレクションの調整』
JVM サーバーのガーベッジ・コレクションは、JVM によって自動的に処理されます。ガーベッジ・コレクション処理とヒープ・サイズを調整して、アプリケーションの応答時間とプロセッサー使用率が最適であることを確認できます。

220 ページの『JVM サーバー始動環境の調整』
複数の JVM サーバーを実行している場合、JVM 始動環境を調整することによりパフォーマンスが改善されます。

## JVM サーバーによるプロセッサー使用量の確認

CICS モニター機能を使用すると、JVM サーバーで実行中のトランザクションで使用されるプロセッサー時間をモニターすることができます。 JVM サーバー内のすべてのスレッドは T8 TCB で実行されます。

### このタスクについて

DFH$MOLS ユーティリティーを使用すると、SMF レコードを印刷したり、CICS Performance Analyzer などのツールを使用して SMF レコードを分析したりすることができます。

### 手順

1. CICS 領域でモニターをオンにして、パフォーマンス・クラスのモニター・データを収集します。
2. パフォーマンス・データ・グループ DFHTASK を確認します。 特に、次のフィールドを調べることができます。

| フィールド ID | フィールド名 | 説明 |
|---|---|---|
| 283 | MAXTTDLY | CICS 領域が使用可能なスレッドの限界に達したため、ユーザー・タスクが T8 TCB を取得するために待っている間に経過した時間。スレッドの限度は CICS 領域ごとに 2000 で、各 JVM サーバーは最大で 256 のスレッドを持つことができます。 |
| 400 | T8CPUT | ユーザー・タスクが、CICS T8 モードの TCB 上の CICS ディスパッチャー・ドメインによってディスパッチされている間のプロセッサー時間。T8 TCB に 1 つのスレッドが割り振られると、処理が完了するまでそのスレッドに対して同じ TCB が関連付けられた状態が続きます。 |
| 401 | JVMTHDWT | CICS システムが CICS 領域内の JVM サーバーに関するスレッドの限界に達したため、ユーザー・タスクが JVM サーバー・スレッドを取得するために待っている間に経過した時間。 |

3. プロセッサー使用量を改善するために、可能ならば、トレースの使用を削減または除去します。

a. 実稼働環境では、CICS マスター・システムのトレース・フラグをオフに設定して、CICS 領域を稼働させることを検討してください。 このフラグをオンに設定すると、Java プログラムの実行によってプロセッサーの消費が著しく増加します。 SYSTR=OFF で CICS を初期化するか、CETR トランザクションを使用することによって、フラグをオフに設定することができます。

b. 特殊なトランザクションのみの JVM トレースをアクティブにしていることを確認します。 JVM トレースは、短時間で大量の出力を生成することがあり、プロセッサー・コストを増加させます。JVM トレースの制御について詳しくは、255 ページの『Java の診断』を参照してください。

4. 実稼働環境で JVM プロファイルでの USEROUTPUTCLASS オプションを使用しないでください。 このオプションを指定すると、JVM のパフォーマンスに悪影響が出ます。 USEROUTPUTCLASS オプションによって、同一の CICS 領域を使用する開発者は、JVM 出力を分離し、適切な宛先に送信することができます。ただし、このためには、追加クラス・インスタンスの作成および呼び出しが必要です。

## JVM サーバーのストレージ所要量の計算

CICS 領域で 1 つ以上の JVM サーバーを始動するには、CICS に割り振られているストレージ容量の他に、JVM で使用できるストレージ容量が十分にあることを確認する必要があります。

### このタスクについて

初めて JVM サーバーを使用する際には、1 回限りのストレージ・コストが発生します。稼働中は、すべての JVM サーバーが 24 ビット、31 ビット、および 64 ビットのストレージを使用します。

JVM サーバーに必要なストレージは、CICS DSA ストレージからではなく、z/OS ストレージから取得されます。このストレージは、Java の観点では「ネイティブ・メモリー」と呼ばれています。このストレージの多くは Language Environment によって管理され、長期のストレージ割り振りと短期のストレージ割り振りの両方が含まれます。

このストレージは動的であり、割り振られたストレージを最も効果的に使用するためには効率的な LE ランタイム・オプションを使用する必要もあるため、正確な見積もりを行うことは極めて困難です。以下に略述する手順は、初期割り振りに必要な基本的で最小の見積もりを提供するためのものであり、31 ビットと 64 ビットの両方のストレージを拡張する余地を見越しておく必要があります。JVM サーバーがアクティブなときの実際のストレージ使用量は、DFH0STAT レポートに示されます。

最終的に安定化する前に、Language Environment z/OS ストレージのサブプールの使用量がワークロードに合わせて時間と共に増加していく場合もあります。

**注:** JVM サーバーのスレッドごとに、z/OS TCB と LE で使用するための少量の 24 ビット・ストレージが必要です。

## 手順

1. JVM サーバーを始動する前に、サンプル統計プログラム DFH0STAT を実行します。
   a. D O MVS,L コマンドの出力から、USS の **SHRLIBRGNSIZE**パラメーターに必要な 31 ビット・ストレージの量を得ます。
   b. **DFHAXRO HEAP64** パラメーターに指定されている、**HEAP64** の 31 ビットの初期割り振り量を加算します。例えば、HEAP64(100M,4M,KEEP,4M, 512K,KEEP,1K,1K,KEEP) の場合は、4 MB の 31 ビット・ストレージになります。 合計を DFH0STAT の「Private Area storage available above 16Mb」と比較して、合計が割り振られたスペースに収まるか確認します。
   •

   割り振られるストレージが実際の使用量により近くなるように 31 ビット・ストレージの設定を最適化したい場合は、**SHRLIBRGNSIZE** パラメーターと Language Environment オプションを調整します。 228 ページの『z/OS 共用ライブラリー領域の調整』および 226 ページの『DFHAXRO による JVM サーバーのエンクレーブの変更』を参照してください。
2. 以下のストレージ要件を合計して、各追加 JVM サーバーに必要な 64 ビット・ストレージの量を計算します。
   - **-Xmx** 値。このパラメーターのデフォルト値は、JVM によって設定されています。詳しくは、IBM Knowledge Center の Java 資料を参照してください。
   - 共用クラスを使用する場合は、**-Xcmsx** の値。
   - **HEAP64** パラメーターに指定されている 64 ビット・ストレージの初期値。上記の例では 100 MB です。
   - DFHAXRO の **LIBHEAP64** パラメーターの最初の値。デフォルト値は 3 MB です。
   - DFHAXRO の **STACK64** パラメーターの最初の値 (デフォルト値は 1 MB) を、JVM サーバーで使用する予定のスレッド数に **-Xgcthreads** または Java デフォルト値を足した数で乗算します。ただし、THREADSTACK64 が ON である場合、2 番目以降のスレッドはこのオプションの最初の値を使用します (通常のデフォルト設定値は 1 MB)。
3. **MEMLIMIT** 値を調べて、追加の JVM サーバーの実行に使用できる十分な 64 ビット・ストレージがあるかどうかを判別します。64 ビット・ストレージを使用する他の CICS 機能と、**HEAP64** 初期値に加えて JVM サーバーが大量のストレージを使用する可能性とを計算に入れる必要があります。DFHAXRO の Language Environment 64 ビット・ランタイム・オプションが最適化されていない場合、特にその必要があります。

   z/OS **MEMLIMIT** パラメーターは、CICS 領域の 64 ビット (2 GB 境界より上) ストレージの量を制限します。64 ビット・ストレージを使用する CICS 機能、およびこのパラメーターの確認と調整の方法については、「パフォーマンスの改善」の『MEMLIMIT の見積もり、確認、および設定』 を参照してください。

# JVM サーバー・ヒープとガーベッジ・コレクションの調整

JVM サーバーのガーベッジ・コレクションは、JVM によって自動的に処理されます。ガーベッジ・コレクション処理とヒープ・サイズを調整して、アプリケーションの応答時間とプロセッサー使用率が最適であることを確認できます。

## このタスクについて

ガーベッジ・コレクション処理は、アプリケーションの応答時間とプロセッサー使用率に影響を与えます。ガーベッジ・コレクションは、JVM におけるすべての作業を一時的に停止するので、アプリケーションの応答時間に影響を与える場合があります。小さいヒープ・サイズを設定する場合、メモリーを節約できますが、ガーベッジ・コレクションの頻度が高くなり、ガーベッジ・コレクションで費やされるプロセッサー時間が増える可能性があります。設定するヒープ・サイズが大きすぎると、JVM は、MVS ストレージの使用効率が悪くなるので、データ・キャッシュ・ミスやページングまでもが生じる可能性があります。CICS が提供する統計を使用すると、JVM サーバーを分析することができます。また、有利なデータ分析と調整オプションの推奨を行う IBM Health Center を使用することもできます。

## 手順

1. 適切な間隔で JVM サーバー統計とディスパッチャー統計を収集します。JVM サーバー統計により、メジャー・ガーベッジ・コレクションとマイナー・ガーベッジ・コレクションの数、およびガーベッジ・コレクション実行のために費やされた時間を知ることができます。ディスパッチャー統計は、CICS 領域全体で T8 TCB のプロセッサー使用率に関する情報を示すことができます。
2. T8 TCB のディスパッチャー TCB モード統計を使用して、JVM サーバー・スレッドで費やされたプロセッサー時間を確認します。 「Accum CPU Time / TCB」フィールドは、この TCB モードで接続中であるか、または接続されていたすべての TCB に要したプロセッサー時間の累積を示します。「TCB attaches」フィールドは、統計間隔で使用された T8 TCB の数を示します。これらの数値を使用して、各 T8 TCB が使用したおおよそのプロセッサー時間を算出してください。
3. JVM サーバー統計を使用して、ガーベッジ・コレクションで費やされた時間のパーセンテージを検出します。 統計間隔の時間を、ガーベッジ・コレクションで費やされた経過時間で除算してください。ガーベッジ・コレクションにおけるプロセッサー使用率を 2% 未満にすることを目標にします。パーセンテージがこれより高い場合、ガーベッジ・コレクションの頻度が低くなるようにヒープのサイズを大きくすることができます。
4. 解放されたヒープの値を、その間隔内で実行されたトランザクション数で除算して、トランザクションごとに収集されている不要情報の量を確認します。 T8 TCB のディスパッチャー統計を調べると、実行されたトランザクション数を確認できます。JVM サーバー内の各スレッドは 1 つの T8 TCB を使用します。
5. オプション: verbosegc ログ・データをファイルに書き込みます。それには、パラメーター **-Xverbosegclog:***path_to_file* を指定します。このデータは、Garbage Collection and Memory Visualizer という別の ISA ツールによって分析することが可能です。 JVM は、XML のガーベッジ・コレクション・メッセージを、JVM プロファイルの STDERR オプションで指定されるファイルに書き込みま

す。メッセージの例と説明は、IBM SDK for z/OS, Java Technology Edition バージョン 7 (『Troubleshooting and support』セクション) に記載されています。

**ヒント:** Memory Analyzer ツールのファイルを使用すると、より詳細な分析を実行することができます。

### タスクの結果

調整結果は、ユーザーの Java ワークロード、CICS および IBM SDK for z/OS の保守レベル、およびその他の要因によって異なります。ストレージやガーベッジ・コレクションの設定と JVM の調整の可能性について詳しくは、IBM SDK for z/OS, Java Technology Edition バージョン 7 (『Troubleshooting and support』セクション) に記載されています。

IBM Health Center および Memory Analyzer は、Java でのモニターと診断のための 2 つの IBM ツールです。それらは、IBM サポート・アシスタント・ワークベンチによって提供されます。これらのツールは、のサイトから無償でダウンロードできます。

## JVM サーバー始動環境の調整

複数の JVM サーバーを実行している場合、JVM 始動環境を調整することによりパフォーマンスが改善されます。

### このタスクについて

JVM サーバーの始動時に、サーバーでは `/usr/lpp/cicsts/cicsts53/lib` ディレクトリーに 1 セットのライブラリーがロードされる必要があります。多数の JVM サーバーを同時に始動すると、必要なライブラリーをロードするために時間がかかり、一部の JVM サーバーが、タイムアウトになったり、始動までに長時間かかったりする場合があります。 JVM サーバーの起動時間を短縮するために、JVM 開始環境を調整する必要があります。

### 手順

1. JVM サーバー用の共用クラス・キャッシュを作成して、ライブラリーを 1 回ロードします。 共用クラス・キャッシュを使用するには、**-Xshareclasses** オプションを各 JVM サーバーの JVM プロファイルに追加します。詳しくは、IBM SDK, Java Technology Edition 7.0.0 の『JVM 間でのクラス・データの共用』を参照してください。
2. OSGi フレームワークのタイムアウト値を増やします。 DFHOSGI.jvmprofile に含まれている `OSGI_FRAMEWORK_TIMEOUT` オプションは、CICS が JVM サーバーの始動およびシャットダウンを待機する時間の長さを指定します。値を超過した場合、JVM サーバーは正しく初期化またはシャットダウンされません。デフォルト値は 60 秒であるため、ご使用の環境に応じてこの値を増やしてください。

## JVM の Language Environment エンクレーブ・ストレージ

JVM サーバーでは、(主に 64 ビット・ストレージで) 静的ストレージ要件と動的ストレージ要件の両方が適用されます。場合によっては、相当な量の 31 ビット・ストレージが使用されます。

**注:** 使用される 31 ビット・ストレージの量は、以下に示すいくつかの要因で決まります。

- 構成パラメーター
- 他の製品の設計および使用
- JVM の設計
- Java ワークロード

例えば、**-Xcompressedrefs** を使用するとパフォーマンスが向上する可能性があります。ただし、31 ビット・ストレージが必要になるため、必ず、**-XXnosuballoc32bitmem** と共に使用して、JVM が必要に応じて圧縮参照のために 31 ビット・ストレージを動的に割り振れるようにする必要があります。これらのオプションについて詳しくは、IBM Knowledge Center の Java の資料を参照してください。ジャストインタイム・コンパイル (JIT) でも、コンパイルされるクラス・コードのために 31 ビット・ストレージが必要になります。

JVM は、Language Environment 事前初期設定モジュール **CELQPIPI** を使用して作成される Language Environment エンクレーブで、z/OS UNIX システム・サービスのプロセスとして実行されます。

JVM のストレージ要求が Language Environment によって処理されると、定義済みのランタイム・オプションに基づいて z/OS ストレージが割り振られます。

Language Environment ランタイム・オプションは、DFHAXRO で設定されます。これらのプログラムによって JVM エンクレーブに提供されるデフォルト値を表 18 に示しています。

*表 18. CICS で JVM エンクレーブに使用される Language Environment のランタイム・オプション*

| **Language Environment のランタイム・オプション** | **JVM サーバーのサンプル値** |
|---|---|
| ヒープ・ストレージ | `HEAP64(256M,4M,KEEP,4M,512K, KEEP,1K,1K,KEEP)` |
| ライブラリー・ヒープ・ストレージ | `LIBHEAP64(5M,3M)` |
| ストレージ内のどこにでも常駐可能なライブラリー・ルーチン・スタック・フレーム | `STACK64(1M,1M,32M)` |
| マルチスレッド・アプリケーションに対するオプションのヒープ・ストレージ管理 (64 ビット) | `HEAPPOOLS64(ALIGN)` |
| マルチスレッド・アプリケーションに対するオプションのヒープ・ストレージ管理 (31 ビット) | `HEAPPOOLS(ALIGN)` |
| ストレージ不足の状態のために予約されたストレージ量、および割り振り時と解放時のストレージの初期内容 | `STORAGE(NONE,NONE,NONE)` |

**注:** 現行の JVM サーバーの値については、ライブラリー SDFHSAMP 内の DFHAXRO メンバーを参照してください。

HEAP64 などの Language Environment ランタイム・オプションは、そのタイプのストレージの初期値の原則に基づいて機能します (100 MB 64 ビットなど)。HEAP64 で新規要求が収まらない場合は、指定されたサイズ (4 MB 以上) か、または、要求のサイズに制御情報を加えたサイズのいずれか大きい方だけ増分して割り振られます。要求を満たすために、適宜、追加で増分して割り振られます。増分が空になると、Language Environment は、ランタイム値に基づいて z/OS ストレージを保持 (KEEP) または解放 (FREE) します。

Language Environment ランタイム・オプションについて詳しくは、「*z/OS Language Environment カスタマイズ*」を参照してください。

Language Environment ランタイム・オプションは、サンプル・プログラム DFHAXRO を変更して再コンパイルすることでオーバーライドできます (226 ページの『DFHAXRO による JVM サーバーのエンクレーブの変更』を参照)。このプログラムは JVMSERVER リソースに対して設定するため、さまざまな名前を使用できます。これが、必要に応じて、JVM サーバーごとに別々のオプションが存在する理由です。

Language Environment エンクレーブ内で 1 つの JVM が必要とするストレージの量によっては、**REGION** および **MEMLIMIT** サイズの制限に使用するインストール・システム出口 IEALIMIT または IEFUSI の変更が必要になることがあります。すべての Java プログラム要求のルート先として、Java 所有領域 (JOR) を使用する方法が考えられます。このような領域で Java ワークロードのみを実行することにより、必要な CICS DSA ストレージの量を最小化し、最大量の MVS ストレージを JVM に割り振り可能にします。

『JVM サーバーの Language Environment ストレージ必要量の特定』
実際のストレージ必要量が分かれば、提供されている **DFHAXRO** オプションを変更する必要があるかどうかを判断できます。それによって、ストレージを増分割り振りする必要のない、あるいは、増分割り振りの回数を許容レベルに抑える値を選択できます。

226 ページの『DFHAXRO による JVM サーバーのエンクレーブの変更』
DFHAXRO は、JVM サーバーが実行される Language Environment エンクレーブにデフォルトの実行時オプション・セットを提供するサンプル・プログラムです。例えば、JVM ヒープとスタックにストレージ割り振りパラメーターを定義します。すべてのワークロード用に最適化されたデフォルトのランタイム・オプションを提供することは不可能です。実際のストレージ使用量を調べ、必要に応じてデフォルト値をオーバーライドして、使用されるストレージと割り振られるストレージの比率を最適化することを検討してください。

## JVM サーバーの Language Environment ストレージ必要量の特定

実際のストレージ必要量が分かれば、提供されている **DFHAXRO** オプションを変更する必要があるかどうかを判断できます。それによって、ストレージを増分割り振りする必要のない、あるいは、増分割り振りの回数を許容レベルに抑える値を選択できます。

## このタスクについて

DFHAXRO の HEAP64 ランタイム・オプションは、JVM サーバーの Language Environment エンクレーブのヒープ・サイズを制御します。このオプションには、64 ビット・ストレージ、31 ビット・ストレージ、および 24 ビット・ストレージの設定が含まれています。必要に応じて、DFHAXRO の代わりに独自のプログラムを使用できます。このプログラムは JVMSERVER リソースで指定する必要があります。

## 手順

1. DFHAXRO に RPTO(ON) および RPTS(ON) オプションを設定します。 これらのオプションは、DFHAXRO の指定されたソースでコメント化されています。これらのオプションを指定すると、Language Environment は、ストレージ・オプションについて報告し、実際に使用されているストレージを示すストレージ・レポートを作成します。
2. JVMSERVER リソースを使用不可にします。 JVM サーバーがシャットダウンし、Language Environment エンクレーブが除去されます。
3. JVMSERVER リソースを使用可能にします。 CICS は、DFHAXRO の Language Environment ランタイム・オプションを使用して、JVM サーバーのエンクレーブを作成します。JVM も開始します。
4. JVM サーバーで Java ワークロードを実行して、Language Environment エンクレーブで使用されるストレージに関するデータを収集します。
5. DFHAXRO から RPTO(ON) オプションと RPTS(ON) オプションを除去します。
6. JVMSERVER リソースを使用不可にして、ストレージ・レポートを生成します。 このストレージ・レポートには、初期の Language Environment エンクレーブ・ヒープ・ストレージの提案が含まれています。64 ビット・ユーザー・ヒープ統計の「Suggested initial size」項目に推奨値が含まれ、JVM サーバーで使用された Language Environment エンクレーブ・ヒープ・ストレージの合計量と等しくなります。

## タスクの結果

ストレージ・レポートは、z/OS UNIX の stderr ファイルに保存されます。また、**JOBLOG** または **DD://** 転送構文を使用して、CICS JES 出力に送ることもできます。ディレクトリーは、JVM プロファイルで JVM の出力をリダイレクトしたかどうかによって異なります。リダイレクトが存在しない場合、このファイルは、JVM の作業ディレクトリーに保管されます。プロファイルで WORK_DIR に値が設定されていない場合、このファイルは /tmp ディレクトリーに保管されます。

ストレージ・レポートの情報を使用して、**DFHAXRO HEAP64** オプションの Language Environment エンクレーブ・ヒープ・ストレージに適切な値を選択してください。目的は、初期割り振り量をレポートに提案されていたサイズまたはそれよりやや少ないサイズに設定して、増分割り振りが何度も行われないようにすることです。増分割り振りが行われる回数が多くなるほど、割り振られたストレージに対する使用量の比率が下がり、結果的に z/OS の観点から見ると不必要な使用が発生します。増分割り振りが多いと、ストレージを管理するために Language Environment が使用する CPU 時間の量も増加することになります。**LIBHEAP64** も使用できますが、通常は変更する必要はありません。

**注: HEAP64** の 31 ビットの初期サイズを増やす場合は、**HEAPPOOLS** の 31 ビット・ストレージの割り振りが過剰にならないように、**HEAPP** も変更する必要があります。**HEAPPOOLS** (および 64 ビットの **HEAPPOOLS64**) は、デフォルトの DFHAXRO ではアクティブであり、適切に構成した場合には非常に有効です。ストレージの最大使用量が、定義されている限度 (通常は 32 MB) に迫っていないか、**STACK64** を確認してください。この限度を超えると、ランタイム・エラーになります。

## 例

以下の例は、標準的な DFHAXRO オプションで 5 GB の JVM ヒープを設定した場合の **RPTOPTS** 出力です。

```
HEAPPOOLS(ALIGN,8,10,32,10,128,10,256,10,1024,10,2048,10,0,10,0,10,0,10,0,10,0,10,0,10)
HEAPPOOLS64(ALIGN,8,4000,32,2000,128,700,256,350,1024,100,2048,50,3072,50,4096,50,8192,
            25,16384,10,32768,5,65536,5)
HEAP64(100M,4M,KEEP,4194304,524288,KEEP,1024,1024,KEEP)
LIBHEAP64(3M,3M,FREE,16384,8192,FREE,8192,4096,FREE)
STACK64(1M,1M,32M)
THREADSTACK64(OFF,1M,1M,128M)
```

以下の例は、**RPTSTG** 出力の一部です。

```
STACK64 statistics:
Initial size:                                  1M
Increment size:                                1M
Maximum used by all concurrent threads:        1M
Largest used by any thread:                    1M - no change required
Number of increments allocated:                0
THREADSTACK64 statistics:
Initial size:                                  1M
Increment size:                                1M
Maximum used by all concurrent threads:        0M
Largest used by any thread:                    0M - not used
Number of increments allocated:                0
64bit User HEAP statistics:
Initial size:                                  100M
Increment size:                                4M
Total heap storage used:                       6099566592
Suggested initial size:                        5817M - use this
Successful Get Heap requests:                  783546
Successful Free Heap requests:                 780785
Number of segments allocated:                  135 - too many increments
Number of segments freed:                      0
31bit User HEAP statistics:
Initial size:                                  4194304
Increment size:                                524288
Totalheap storage used (suggested initial size):  137165672 - use this or a smaller value
Successful Get Heap requests:                  1345332
Successful Free Heap requests:                 1345260
Number of segments allocated:                  125 - too many increments
Number of segments freed:                      0
64bit Library HEAP statistics:
Initial size:                                  3M
Increment size:                                3M
Total heap storage used:                       4640032
Suggested initial size:                        5M
Successful Get Heap requests:                  113381
Successful Free Heap requests:                 112860
Number of segments allocated:                  1 - low, so no change required
Number of segments freed:                      0
31bit Library HEAP statistics:
Initial size:                                  16384
Increment size:                                8192
Total heap storage used (suggested initial size): 520
```

```
Successful Get Heap requests:                    33725
Successful Free Heap requests:                   33725
Number of segments allocated:                    1 - low, so no change required
Number of segments freed:                        0

Suggested Percentages for current CellSizes:
HEAPP(ALIGN,8,1,32,1,128,1,256,1,1024,1,2048,1,0)
```

**RPTSTG** 出力を確認する際には、**HEAP64** の増分サイズは Language Environment が割り振る最小のストレージ量であって、実際の増分はこの値よりもかなり大きい可能性があることに注意してください。したがって、1 回以上の増分割り振りが行われた場合に使用された z/OS ストレージの量を正確に算定することは不可能です。64 ビット HEAP には実際の増分割り振りの回数 (つまり 135) が報告されていますが、31 ビット **HEAP** については、レポートに示されている値より 1 つ少ない値 (つまり 125 ではなく 124) が実際の増分の回数です。

JVM サーバーの「suggested initial size (推奨初期サイズ)」は 5817 M です。この推奨初期サイズについて検討する際に、次の例を考慮してください。100MB + (135 x 4MB) は非常に少ないです。JVM サーバーには -Xmx5G が指定されており、これは 100M の初期割り振り量では収まらないので、Language Environment は 135 回のうちの 1 回の増分で 5 GB を少し超えるサイズを割り振ることになります。残った初期割り振りの 100 MB と、最小サイズが 4 MB の増分 134 回分が、残りの 697 MB に相当することになります。

増分が行われるときの Language Environment のストレージ管理の仕組みが原因で、割り振られる 31 ビットと 64 ビットの z/OS ストレージの量は、**RPTSTG** の「maximum used (最大使用量)」に示されている値よりもかなり大きくなる可能性があります。

推奨されている **DFHAXRO** の変更は、以下のとおりです。

```
* Heap storage
DC     C'HEAP64(5817M,'          Initial 64bit heap - change (Note 1)
DC     C'4M,'                    64bit Heap increment
DC     C'KEEP,'                  64bit Increments kept
DC     C'128M,'                  Initial 31bit heap - change (Note 2)
DC     C'2M,'                    31bit Heap increment - change (Note 3)
DC     C'FREE,'                  31bit Increments freed - change (Note 4)
DC     C'1K,'                    Initial 24bit heap
DC     C'1K,'                    24bit Heap increment
DC     C'KEEP) '                 24bit Increments kept

* Heap  pools
DC     C'HP64(ALIGN) '
DC     C'HEAPP(ALIGN,8,1,32,1,128,1,256,1,1024,1,2048,1,0) ' - change (Note 5)

* Library Heap storage
DC     C'LIBHEAP64(3M,3M) '      Initial 64bit heap - do not change (Note 6)

* 64bit stack storage
DC     C'STACK64(1M,1M,32M) ' - do not change (Note 7)
```

**注:**

1. **RPTSTG** 出力の 64 ビット「Suggested initial size (推奨初期サイズ)」に示されているとおり。
2. **RPTSTG** 出力の 31 ビット「Suggested initial size (推奨初期サイズ)」に示されているとおり。ただし、FREE を使用しているので多少減少しています。

3. 31 ビット **HEAP** 増分は、512 K ではなく 1 M または 2 M の値にするのが適切です。
4. 必要に応じて、31 ビット に**HEAP FREE** を使用すると、KEEP を指定した場合に比べて、「Total heap storage used (使用済み合計ヒープ・ストレージ)」に対応する z/OS ストレージの割り振り量が減少する可能性があります。
5. **RPTSTG** 出力で **HEAP POOLS** 統計の後に推奨内容が示されていますが、さらに最適化すると効果がある可能性があります。31 ビット・ヒープの初期サイズ 128 MB の 10% (デフォルト値) だと、割り振られるストレージの量が過剰になる可能性があります。 最低 6 つのプールがそれぞれ 128 MB の初期ヒープ・サイズの 10% を取ると、77 MB が割り振られることになります。プールの何パーセントが生産的に使用されているかに関わりなく、この割り振り分は「Total heap storage used (使用済み合計ヒープ・ストレージ)」値に含まれます (なぜなら、**HEAP POOLS** ストレージ・エクステントがそこに割り振られるからです)。
6. 増分は 1 回しか必要ありませんでした。これは問題ではありません。
7. 使用された最大サイズは 1 MB で、使用可能であった最大サイズは 32 MB だったので、変更は不要です。

以下に、もう一度同じ JVM サーバーで 31 ビット **HEAP FREE** を使用して実行した例を示します。「Number of segments (セグメント数)」は実行された **GETMAIN** と **FREEMAIN** の数です。JVM サーバーがアクティブであったとき、これは低い値を示していました。差が 2 であるということは、エンクレーブが初期割り振りと増分 1 回のみで終了したことを示しています。この量は「Total heap storage (合計ヒープ・ストレージ)」よりも少ない可能性があり、FREE が有効であったことを示しています。「Total heap storage used (使用済み合計ヒープ・ストレージ)」はより大きな値でした。しかし、JVM サーバーを実行するごとに合計は変化するので、1 つの **RPTSTG** だけに基づいて変更しても、ベストな設定が得られるとは限りません。

```
31bit User HEAP statistics:
Initial size: 134217728
Increment size: 2097152
Total heap storage used (suggested initial size): 154056664
Successful Get Heap requests: 3253239
Successful Free Heap requests: 3253176
Number of segments allocated: 149
Number of segments freed: 147
```

**RPTSTG** 出力を正しく解釈するには、「Language Environment デバッグ・ガイド」を読むことが重要です。

## DFHAXRO による JVM サーバーのエンクレーブの変更

DFHAXRO は、JVM サーバーが実行される Language Environment エンクレーブにデフォルトの実行時オプション・セットを提供するサンプル・プログラムです。例えば、JVM ヒープとスタックにストレージ割り振りパラメーターを定義します。すべてのワークロード用に最適化されたデフォルトのランタイム・オプションを提供することは不可能です。実際のストレージ使用量を調べ、必要に応じてデフォルト値をオーバーライドして、使用されるストレージと割り振られるストレージの比率を最適化することを検討してください。

## このタスクについて

このサンプル・プログラムを更新して Language Environment エンクレーブを調整するか、サンプルに基づいて独自のプログラムを作成することができます。このプログラムは JVMSERVER リソースで定義され、JVM サーバー用に作成される Language Environment エンクレーブの CELQPIPI 事前初期設定段階時に呼び出されます。

アセンブリー言語でこのプログラムを作成する必要があり、CICS 変換プログラムでこれを変換してはなりません。オプションは文字ストリングとして指定され、2 バイトのストリングの長さと、それに続く実行時オプションで構成されます。すべての Language Environment 実行時オプションの最大長は 255 バイトです。したがって、各オプションの省略バージョンを使用し、変更内容を全体で 200 バイト未満に制限してください。

## 手順

1. DFHAXRO プログラムを新規ロケーションにコピーして、実行時オプションを編集します。 CICS 領域に保守が適用される場合、プログラムの変更を反映したい場合があります。DFHAXRO のソースは、CICSTS53.CICS.SDFHSAMP ライブラリーにあります。
2. 各オプションの省略形を使用して、実行時オプションを編集します。 「*z/OS Language Environment プログラミング・ガイド*」に、Language Environment 実行時オプションに関する詳細情報が記載されています。
   - CICS はいくつかのオプションをこのリストに追加するため、高速処理のためにはオプションのリストのサイズを最小限に保つ必要があります。
   - HEAP64 オプションを使用して、初期のヒープ割り振りを指定します。
   - ALL31 オプション、POSIX オプション、および XPLINK オプションは、CICS によって強制的にオンにされます。 ABTERMENC オプションは、CICS によって (ABEND) に設定され、TRAP オプションは (ON,NOSPIE) に設定されます。
   - JVM サーバーによって作成されるファイルに対する権限を制御するには、以下の行を追加します。

     ```
     DC  C'ENVAR("_EDC_UMASK_DFLT=nnn")'
     ```

     ここで *nnn* は必要なマスク値です。
   - RPTO および RPTS オプションによって生成される出力は、CESE 一時データ・キューに書き込まれます。
   - 出力を生成するオプションがそれを行うのは、各 JVM の終了時です。生成後に CESE に送信される出力のボリュームを考慮に入れてください。
3. DFHASMVS プロシージャーを使用して、プログラムをコンパイルします。

## タスクの結果

JVMSERVER リソースを使用可能にすると、CICS は DFHAXRO プログラムで指定された実行時オプションを使って Language Environment エンクレーブを作成します。 CICS は実行時オプションを Language Environment に渡す前に、それらの長さを検査します。この長さが 255 バイトより長い場合、CICS は JVM サーバーの始動を試みず、CSMT にエラー・メッセージを書き込みます。指定した値は、

Language Environment に渡される前に CICS によって検査されません。

## z/OS 共用ライブラリー領域の調整

共用ライブラリー領域は、 z/OS の機能であり、アドレス・スペース間のダイナミック・リンク・ライブラリー (DLL) ファイルの共用を可能にします。この機能により、 CICS 領域は JVM に必要な DLL を共用できるようになり、各領域が DLL を個別にロードする必要はなくなります。これにより、MVS が使用する実際のストレージの量、および領域へのファイルのロード所要時間を大幅に削減できます。

共用ライブラリー領域用に予約されているストレージは、最初の JVM が領域で開始されるときにそれぞれの CICS 領域に割り振られます。割り振られるストレージの量は、z/OS の **SHRLIBRGNSIZE** パラメーターによって制御されます。このパラメーターは、SYS1.PARMLIB の BPXPRMxx メンバー内にあります。最小値は 16 MB で、z/OS のデフォルトは 64 MB です。必要なスペースの量を調査することによって、共用ライブラリー領域用に割り振られるストレージの量を調整できます。CICS 以外の他のアプリケーションが、その共用ライブラリー領域を使用しており、それに応じて **SHRLIBRGNSIZE** パラメーターを調整している可能性があることに注意してください。

共用ライブラリー領域用に割り振られるストレージの量を削減する場合は、最初に共用ライブラリー領域に無駄なスペースがないことを確認します。z/OS システムで通常の作業負荷をかけ、コマンド **D OMVS,L** を発行してライブラリー統計を表示します。共用ライブラリー領域に未使用のスペースがある場合は、**SHRLIBRGNSIZE** の設定を減らしてこのスペースを除去できます。CICS が、共用ライブラリー領域の唯一のユーザーの場合は、**SHRLIBRGNSIZE** を最小の 16 MB まで削減できます。これは、JVM が必要とする DLL が、約 10 MB の領域しか使用しないためです。

共用ライブラリー領域内のすべてのスペースが使用されている状態でも、CICS 領域におけるこのストレージ割り振りを削減したい場合は、以下の 3 つの手順が検討できます。

1. 共用ライブラリー領域サイズをファイルで必要なストレージの量より小さく設定することは可能です。共用ライブラリー領域がフルの場合には、代わりにファイルは専用ストレージにロードされ、共用機能の利点は得られません。この方法を選択した場合は、より重要なアプリケーションを先に再始動して、共用ライブラリー領域を確実に利用できるようにする必要があります。共用ライブラリー領域内のスペースのほとんどが、重要ではないアプリケーションによって使用されている場合は、この方法が最も適切です。
2. 共用ライブラリー領域内に配置される DLL は、拡張属性 +l によって識別されます。一部のファイルからこの属性を除去し、これらのファイルが共用ライブラリー領域に配置されることを防止することで、共用ライブラリー領域に必要なストレージ量を削減できます。この方法を選択する場合は、共用される頻度が少ないファイルを選択してください。また、拡張子 .so 付きのファイルは選択しないようにしてください。拡張子 .so が付いたファイルは、共用ライブラリー領域内に配置されない場合、ユーザー共用ライブラリーを通じて共用されますが、共用ライブラリー領域を使用した場合と比較して、この共用機能は非効率です。共用ライブラリー領域内のスペースのほとんどが、拡張子 .so が付いていない大きなファイルによって使用されている場合は、この方法が最適です。

3. CICS JVM に関連するすべてのファイルから拡張属性 +l を除去すると、 CICS 領域では共用ライブラリー領域が完全に使用されなくなり、 CICS 領域内で共用ライブラリー領域用のストレージが割り振られることもありません。この方法を選択した場合は、共用ライブラリー領域の共用機能を利用することはできません。 z/OS システム上の他のアプリケーションが大きな共用ライブラリー領域を必要としているが、 CICS 領域内でこのストレージ量を割り振りたくない場合は、この方法が最適です。

いずれかのファイルから拡張属性 +l を除去した場合は、これらのファイルを新規バージョンで置き換える際に (例えば、ソフトウェア・アップグレードの場合)、新バージョンのファイルにこの属性が設定されていないことを忘れずに確認してください。

z/OS UNIX の共用ライブラリーについて詳しくは、http://www.ibm.com/servers/eserver/zseries/zos/unix/perform/sharelib.html の z/OS UNIX System Services Web サイトを参照してください。

# 第 7 章 Java アプリケーションのセキュリティー

Java アプリケーションを保護して、許可されたユーザーだけがアプリケーションのデプロイやインストールを行ったり、これらのアプリケーションに Web からアクセスしたり CICS を経由してアクセスしたりできるようにすることが可能です。Java セキュリティー・マネージャーを使用して、Java アプリケーションが安全でない可能性のあるアクションを実行しないようにすることもできます。

Java アプリケーションのライフサイクル内のさまざまな時点で、以下のようなセキュリティーを追加できます。

- Java アプリケーション・リソースの定義やインストールに関するセキュリティー検査を実装します。Java アプリケーションは CICS バンドルにパッケージ化されているので、JVM サーバーにアプリケーションをインストールすることを許可されているユーザーが、このタイプのリソースをインストールできることを確認しなければなりません。
- 許可されたユーザーだけがアプリケーションにアクセスできるようにするための、アプリケーション・ユーザーに関するセキュリティー検査を実装します。
- CICSExecutorService を使って開始される CICS Java タスクに関するセキュリティー検査を実装します。このような CICS タスクはすべて、CJSA トランザクションおよびデフォルト・ユーザー ID の下で実行されます。
- Java セキュリティー・マネージャーを使用して、Java API に対するセキュリティー制限を実装します。

Java アプリケーションは、OSGi フレームワーク内か Liberty プロファイル・サーバー内で実行できます。Liberty プロファイルは、Web アプリケーションをホストするように設計されており、OSGi フレームワークが組み込まれています。 Liberty プロファイルには独自のセキュリティー・モデルがあるので、Liberty プロファイル・サーバーのセキュリティー構成は異なります。

232 ページの『OSGi アプリケーションに関するセキュリティーの構成』
CICS セキュリティーを使用して、JVM サーバーや Java アプリケーションのライフサイクルを管理したり、CICS を介してアプリケーションにアクセスしたりする権限を、適切なユーザーに与えます。

233 ページの『Liberty JVM サーバーに関するセキュリティーの構成』
CICS Liberty セキュリティー機能を使用すれば、Java Platform Enterprise Edition の役割によって、ユーザーを認証したり、Web アプリケーションへのアクセスを許可したりできます。そのようにして、CICS のトランザクション/リソース・セキュリティーとの統合も可能になります。また、CICS リソース・セキュリティーを使用して、CICS BUNDLE リソースにデプロイされる JVMSERVER リソースと Java Web アプリケーションの両方のライフサイクルを管理する権限を適切なユーザーに与えることができます。

242 ページの『Java 鍵ストアを使用した Liberty JVM サーバー用の SSL (TLS) の構成』
SSL を使用してデータを暗号化するように Liberty JVM サーバーを構成できま

す。オプションとして、クライアント証明書を使用してサーバーで認証するように構成することもできます。証明書は、Java 鍵ストアか、または RACF などの SAF 鍵リングに保管できます。

243 ページの『RACF を使用した Liberty JVM サーバー用の SSL (TLS) の構成』
SSL を使用してデータを暗号化するように Liberty JVM サーバーを構成できます。オプションとして、クライアント証明書を使用してサーバーで認証するように構成することもできます。証明書は、Java 鍵ストアか、または RACF などの SAF 鍵リングに保管できます。

245 ページの『Liberty JVM サーバーでの SSL (TLS) クライアント証明書認証のセットアップ』
SSL クライアント証明書認証を使用することで、クライアントおよびサーバーは証明書を相手に提供して相互に確認できます。この方法は、セキュリティーについての懸念のために追加レベルの認証が必要となる状況でよく使用されます。

247 ページの『z/OS Connect for CICS のセキュリティー』
z/OS Connect は、WebSphere Liberty プロファイル・アプリケーションであり、その構成と考慮事項は、他の WebSphere Liberty プロファイル・アプリケーションと同じです。z/OS Connect for CICS には、いくつかの追加要件があります。

248 ページの『Java セキュリティー・マネージャーの有効化』
デフォルトで、Java アプリケーションでは、Java API に要求されるアクティビティーにセキュリティーの制限がありません。Java セキュリティーを使用して、安全でない可能性があるアクションを Java アプリケーションが実行しないようにするために、そのアプリケーションが実行される JVM に対してセキュリティー・マネージャーを有効にすることができます。

## OSGi アプリケーションに関するセキュリティーの構成

CICS セキュリティーを使用して、JVM サーバーや Java アプリケーションのライフサイクルを管理したり、CICS を介してアプリケーションにアクセスしたりする権限を、適切なユーザーに与えます。

### 手順

必要に応じて、JVMSERVER リソースや BUNDLE リソースの作成、表示、更新、削除を行うための権限を、アプリケーション開発者やシステム管理者に与えます。JVMSERVER リソースは JVM サーバーの可用性を制御します。 BUNDLE リソースは、Java アプリケーションのデプロイメントの単位で、このアプリケーションの可用性を制御します。

### タスクの結果

OSGi フレームワーク内で実行される Java アプリケーションに関するセキュリティーを正常に構成しました。

# Liberty JVM サーバーに関するセキュリティーの構成

CICS Liberty セキュリティー機能を使用すれば、Java Platform Enterprise Edition の役割によって、ユーザーを認証したり、Web アプリケーションへのアクセスを許可したりできます。そのようにして、CICS のトランザクション/リソース・セキュリティーとの統合も可能になります。また、CICS リソース・セキュリティーを使用して、CICS BUNDLE リソースにデプロイされる JVMSERVER リソースと Java Web アプリケーションの両方のライフサイクルを管理する権限を適切なユーザーに与えることができます。

## 始める前に

CICS 領域で、SAF セキュリティーを使用するための構成が完了していることと、システム初期設定パラメーターとして SEC=YES が定義されていることを確認します。その後、Liberty JVM サーバーに Web アプリケーションをデプロイするために、JVMSERVER リソースや BUNDLE リソースの作成、表示、更新、削除を行うための権限を、アプリケーション開発者やシステム管理者に与えます。
JVMSERVER リソースは JVM サーバーの使用可否を制御するものです。BUNDLE リソースは Java アプリケーションのデプロイメント単位であり、そのアプリケーションの使用可否を制御するものです。CICS TS のセキュリティー機能 `cicsts:security-1.0` は SAF レジストリーを使用するため、この機能を追加する場合は、他のレジストリー・タイプが使用できなくなることに注意してください。他のレジストリー・タイプを同じ Liberty サーバー・インスタンスに追加しようとすると、障害が発生します。

## このタスクについて

このタスクでは、Liberty JVM サーバー用セキュリティーの構成方法、および Liberty セキュリティーと CICS セキュリティーの統合方法について説明します。

Web 要求を実行するためのデフォルトのトランザクション ID は CJSA です。ただし、JVMSERVER タイプの URIMAP を使用して、別のトランザクション ID で Web 要求を実行するように CICS を構成することもできます。通常は、Web アプリケーションの汎用的なコンテキスト・ルート (URI) に対応した URIMAP を指定し、そのアプリケーションを構成するサーブレット一式がトランザクション ID のスコープになるように設定できます。あるいは、具体性の高い URI を使用して、個々のサーブレットを別々のトランザクションで実行することも可能です。

## 手順

1. WebSphere Liberty プロファイルのエンジェル・プロセスを、認証および許可サービスを Liberty JVM サーバーに提供するように構成します。 Liberty サーバーのエンジェル・プロセスを参照してください。
2. `cicsts:security-1.0` 機能を、`server.xml` の featuremanager リストに追加します。

```
<featureManager>
   ...
   <feature>cicsts:security-1.0</feature>
</featureManager>
...
```

3. 次の例を使用して、System Authorization Facility (SAF) レジストリーを server.xml に追加します。

   ```
   <safRegistry id="saf"/>
   ```

4. server.xml に対する変更を保管します。

   **注:** あるいは、Liberty JVM サーバーを自動構成する場合に CICS 領域で **SEC** システム初期設定パラメーターが YES に設定されていると、Liberty JVM サーバーの再始動時に、Liberty JVM セキュリティーをサポートするように JVM サーバーが動的に構成されます。詳しくは、 139 ページの『Liberty JVM サーバーの構成』を参照してください。

   **SEC** システム初期設定パラメーターが NO に設定されていれば、認証または SSL サポートに依然として Liberty セキュリティーを使用できます。CICS セキュリティーがオフで、Liberty セキュリティーを使用する場合は、server.xml ファイルを手動で構成する必要があります。

   a. appSecurity-2.0 機能を featuremanager リストに追加します。
   b. ユーザーを認証するユーザー・レジストリーを追加します。 Liberty セキュリティーでは、SAF および基本ユーザー・レジストリーがサポートされます。詳しくは、WebSphere Application Server for z/OS 8.5.5 の『Liberty プロファイルのユーザー・レジストリーの構成』を参照してください。
   c. アプリケーション・リソースへのアクセスを許可するために security-role 定義を追加します。 239 ページの『ユーザーに Liberty JVM サーバー内のアプリケーションを実行する権限を与える』を参照してください。

## タスクの結果

cicsts:security-1.0 機能を使用すると、Liberty の z/OS セキュリティー機能を使用するように Web コンテナーが自動的に構成されます。さらに、SAF レジストリーが認証のために使用され、<application-bnd> エレメントで指定されている Java Platform Enterprise Edition の役割が許可のために使用されます。

## 次のタスク

- Liberty アプリケーション・セキュリティー認証規則を構成します。 237 ページの『Liberty JVM サーバーでのユーザー認証』を参照してください。
- Web アプリケーションの許可規則を定義します。 239 ページの『ユーザーに Liberty JVM サーバー内のアプリケーションを実行する権限を与える』および 240 ページの『JEE アプリケーション・ロール・セキュリティー』を参照してください。
- Liberty 認証キャッシュを変更します。

Secure Sockets Layer (SSL) を使用するための詳細については、242 ページの『Java 鍵ストアを使用した Liberty JVM サーバー用の SSL (TLS) の構成』を参照してください。

235 ページの『Liberty サーバーのエンジェル・プロセス』
エンジェル・プロセスでは、WebSphere Application Server Liberty プロファイル・サーバーに対して許可サービスが提供されます。

237 ページの『Liberty JVM サーバーでのユーザー認証』
Liberty JVM サーバーで実行されるすべての Web アプリケーションに対して

CICS セキュリティーを構成できますが、Web アプリケーションは、セキュリティー制約が Web アプリケーションに含まれている場合にのみユーザーを認証します。セキュリティー制約は、動的 Web プロジェクトまたは OSGi アプリケーション・プロジェクトのデプロイメント記述子 (web.xml) にアプリケーション開発者が定義します。セキュリティー制約とは、保護対象 (URL) および保護に使用するロールを定義するものです。

239 ページの『ユーザーに Liberty JVM サーバー内のアプリケーションを実行する権限を与える』
ユーザー ID に Liberty JVM サーバーでトランザクションを実行する権限を与えるには、CICS トランザクションおよびリソース・セキュリティーを使用するか、JEE アプリケーション・セキュリティー・ロールを使用して JEE アプリケーションへのアクセス権限を与えます。

240 ページの『JEE アプリケーション・ロール・セキュリティー』
JEE アプリケーション・ロール・セキュリティーは、使用する許可タイプに応じてさまざまな方法で構成できます。分散システムでは、基本レジストリーまたは LDAP レジストリーを、通常はアプリケーション固有の `<application-bnd>` エレメントと組み合わせて使用して、これらのレジストリーに基づいてユーザーを *ロール* にマップします。アプリケーションのデプロイメント記述子によって、アプリケーションのどの部分にどのロールがアクセスできるかが決まります。

## Liberty サーバーのエンジェル・プロセス

エンジェル・プロセスでは、WebSphere Application Server Liberty プロファイル・サーバーに対して許可サービスが提供されます。

エンジェル・プロセスは MVS コンソールから開始されます。 z/OS イメージ上で実行されるすべての WebSphere Application Server Liberty プロファイル・サーバーで、1 つのエンジェルを共用できます。各サーバーが実行しているコードのレベルや、各サーバーが CICS JVM サーバーで実行されるかどうかは関係ありません。

CICS Liberty セキュリティー機能を使用しない場合は、エンジェル・プロセスを実行したり、関連するセキュリティー規則を実装する必要はありません。

### エンジェル・プロセスの開始タスク

エンジェル・プロセスの開始タスク JCL プロシージャーは、CICS の USSHOME ディレクトリーにあります。例えば、次のパスです。

```
/usr/lpp/cicsts53/wlp/templates/zos/procs/bbgzangl.jcl
```

JCL は JES プロシージャー・ライブラリーにコピーして変更する必要があります。ROOT には、値 `USSHOME/wlp` を設定できます。次に例を示します。

```
/usr/lpp/cicsts53/wlp
```

エンジェル・プロセスは、Liberty JVM サーバーが始動する前に実行する必要があります。エンジェル・プロセスを開始または停止するには、次のオペレーター・コマンドを発行します。

```
START BBGZANGL
STOP BBGZANGL
```

エンジェル・プロセスに接続されている Liberty JVM サーバーを表示するには、次のオペレーター・コマンドを発行します。

```
MODIFY BBGZANGL,DISPLAY,SERVERS
```

返されるメッセージには、エンジェル・プロセスに接続されているアクティブな Liberty JVM サーバーとその他の非 CICS Liberty JVM サーバーを含む CICS 領域がリストされます。

## エンジェル・プロセス開始タスクの SAF 規則

エンジェル・プロセスを実行する際に使用するユーザー ID には SAF STARTED プロファイルが必要です。例えば、次のとおりです。

```
RDEFINE STARTED BBGZANGL.* UACC(NONE) STDATA(USER(WLPUSER))
SETROPTS RACLIST(STARTED) REFRESH
```

CICS Liberty JVM サーバーは、CICS 領域ユーザー ID の権限の下で実行されます。許可サービスを使用するには、このユーザー ID でエンジェル・プロセスに接続できる必要があります。 CICS Liberty JVM サーバーでサポートされる許可サービスは、CICS Liberty セキュリティー機能によって実装される z/OS ユーザー・レジストリー・サービスと SAF 許可サービス (SAFCRED) のみです。この機能を使用しない場合は、エンジェル・プロセスを実行する必要はありません。

CICS Liberty JVM サーバーがエンジェル・プロセスに接続できるようにするには、SERVER クラスにエンジェル・プロセス (BBG.ANGEL) 用のプロセスを作成します。例えば RACF で、それに対するアクセス権限を CICS 領域ユーザー ID (cics_region_user) に付与します。

```
RDEFINE SERVER BBG.ANGEL UACC(NONE)
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(cics_region_user)
```

CICS Liberty セキュリティー機能に必要なサービスに CICS Liberty JVM サーバーがアクセスできるようにするには、SERVER クラスに、SAF 許可ユーザー・レジストリー・サービスおよび SAF 許可サービス (SAFCRED) 用のプロファイルを作成します。例えば RACF で、それに対するアクセス権限を CICS 領域ユーザー ID (cics_region_user) に付与します。

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER) ACCESS(READ) ID(cics_region_user)
```

Liberty サーバーが z/OS 許可サービスを使用できるようにするには、許可モジュール BBGZSAFM 用の SERVER プロファイルを作成し、プロファイルに対して CICS 領域ユーザー ID (REGION1) を許可します。このアクションにより、Liberty サーバーが z/OS 許可サービスを使用できるようになります。領域ユーザー ID が REGION1 である CICS 領域が、許可されたモジュールにアクセスできるようにするには、次のようにします。

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(cics_region_user)
```

IFAUSAGE サービス (PRODMGR) 用の SERVER プロファイルを作成し、CICS 領域ユーザー ID を許可します。これにより、Liberty JVM サーバーは、CICS JVM サーバーが使用可能になるときには IFAUSAGE に登録し、使用不可になるときには登録解除できます。領域ユーザー ID が REGION1 である CICS 領域が、IFAUSAGE への登録および登録解除を行えるようにするには、次のようにします。

```
RDEFINE SERVER BBG.AUTHMOD.BBGZSAFM.PRODMGR UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.PRODMGR CLASS(SERVER) ACCESS(READ) ID(cics_region_user)
```

SERVER リソースをリフレッシュします。

```
SETROPTS RACLIST(SERVER) REFRESH
```

詳しくは、WebSphere Application Server for z/OS 8.5.5 製品資料内の『Liberty プロファイル: z/OS のプロセス・タイプ』を参照してください。

## Liberty JVM サーバーでのユーザー認証

Liberty JVM サーバーで実行されるすべての Web アプリケーションに対して CICS セキュリティーを構成できますが、Web アプリケーションは、セキュリティー制約が Web アプリケーションに含まれている場合にのみユーザーを認証します。セキュリティー制約は、動的 Web プロジェクトまたは OSGi アプリケーション・プロジェクトのデプロイメント記述子 (web.xml) にアプリケーション開発者が定義します。セキュリティー制約とは、保護対象 (URL) および保護に使用するロールを定義するものです。

<login-config> エレメントは、ユーザーが Web コンテナーへのアクセスを獲得する方法、および認証に使用する方式を定義します。サポートされるメソッドは、HTTP 基本認証、フォーム・ベース認証、または SSL クライアント認証のいずれかです。 CICS 用のアプリケーション・セキュリティーを定義する方法について詳しくは、CICS Explorer 製品資料内の『SSL security for Explorer connections』を参照してください。web.xml 内のエレメントの例を以下に示します。

```
<!-- Secure the application -->
<security-constraint>
 <display-name>com.ibm.cics.server.examples.wlp.tsq.web_SecurityConstraint</display-name>
  <web-resource-name>com.ibm.cics.server.examples.wlp.tsq.web</web-resource-name>
  <description>Protection area for com.ibm.cics.server.examples.wlp.tsq.web</description>
  <url-pattern>/*</url-pattern>
 </web-resource-collection>
 <auth-constraint>
  <description>Only SuperUser can access this application</description>
  <role-name>SuperUser</role-name>
 </auth-constraint>
 <user-data-constraint>
  <!-- Force the use of SSL -->
  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
 </user-data-constraint>
</security-constraint>

<!-- Declare the roles referenced in this deployment descriptor -->
<security-role>
 <description>The SuperUser role</description>
 <role-name>SuperUser</role-name>
</security-role>

<!--Determine the authentication method -->
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

**注:** あるサーブレットから別のサーブレットに要求を転送するために RequestDispatcher.forward() メソッドを使用すると、セキュリティー検査は、クライアントから要求された最初のサーブレットだけに行われます。

Liberty セキュリティーを使用して CICS で認証されたタスクでは、CICS でのトランザクションおよびリソース・セキュリティー検査を許可するために Liberty アプリケーション・セキュリティー・メカニズムのいずれかから派生したユーザー ID を使用できます。 CICS ユーザー ID は、以下の基準に従って決定されます。

1. Liberty アプリケーション・セキュリティー認証。

   SAF ユーザー・レジストリーとの統合は、CICS Liberty セキュリティー機能の一部であり、WebSphere Application Server Liberty プロファイルでサポートされるアプリケーション・セキュリティー・メカニズムはすべて CICS でもサポートされます。これには、カスタム・ログイン・モジュールまたは Trust Association Interceptor (TAI) を使用した HTTP 基本認証、フォーム・ログイン、SSL クライアント証明書認証、ID アサーションなどがあります。 Liberty で認証するすべての SAF ユーザー ID には、APPL クラスの Liberty JVM サーバーの APPLID への読み取りアクセスが付与されていなければなりません。この名前は、Liberty サーバー構成ファイル server.xml の safCredentials エレメントの profilePrefix 設定によって決定されます。

   ```
   <safCredentials profilePrefix="BBGZDFLT"/>
   ```

   APPL クラスは、特定の CICS 領域へのアクセスを制御するために CICS 端末ユーザーによっても使用され、Liberty JVM サーバーは、セキュリティー要件に応じて、CICS APPLID と同じプロファイルを使用できます。このエレメントを指定しない場合、デフォルトの profilePrefix である BBGZDFLT が使用されます。

   APPLID は定義する必要があります。また、ユーザーはその APPLID へのアクセス権限を持っている必要があります。 APPLID を BBGZDFLT として構成してアクティブにするには、以下のようにします。

   ```
   RDEFINE APPL BBGZDFLT UACC(NONE)
   SETROPTS CLASSACT(APPL)
   ```

   ユーザーを認証するためには、ユーザーにこの APPLID への読み取りアクセスが付与されていなければなりません。ユーザー AUSER を BBGZDFLT APPLID に照らして認証できるようにするには、以下のようにします。

   ```
   PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(AUSER)
   ```

   Liberty SAF で認証しないユーザー ID にも、APPLID への読み取りアクセスが付与されていなければなりません。 SAF で認証しないユーザー ID を Liberty サーバー構成ファイル server.xml の safCredentials エレメントで指定することができます。

   ```
   <safCredentials unauthenticatedUser="WSGUEST"/>
   ```

   このエレメントを指定しない場合、デフォルトの unauthenticatedUser は WSGUEST です。 SAF で認証しないユーザー ID である WSGUEST に BBGZDFLT APPLID への読み取りアクセスを与えるには、以下のようにします。

   ```
   PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(WSGUEST)
   ```

   WLP z/OS システム・セキュリティー・アクセス・ドメイン (WZSSAD) は、Liberty プロファイル・サーバーに付与された権限を参照します。これらの権限

によって、サーバーがユーザーの認証と許可を行う際に、どの System Authorization Facility (SAF) アプリケーション・ドメインおよびリソース・プロファイルの照会を許されるかを制御します。 CICS 領域のユーザー ID には、特定の APPLID ドメインで認証呼び出しを行うために WZSSAD 内で権限を付与する必要があります。特定の APPLID ドメインで認証する権限をサーバーに付与するには、CICS 領域の ID に、SERVER クラス内の BBG.SECPFX.<APPLID> プロファイルに対する読み取り権限を付与する必要があります。

```
RDEFINE SERVER BBG.SECPFX.BBGZDFLT UACC(NONE)
PERMIT BBG.SECPFX.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(cics_region_user)
```

詳しくは、WebSphere Application Server for z/OS 8.5.5 製品資料内の『Liberty プロファイル: WZSSAD を使用した z/OS セキュリティー・リソースへのアクセス』を参照してください。

2. 認証されないサブジェクトが Liberty から提供された場合は、URIMAP に定義されている USERID が使用されます。
3. USERID が URIMAP に定義されていない場合は、CICS のデフォルトのユーザー ID で要求が実行されます。

**注:** Liberty トランザクションのセキュリティー処理が CICS トランザクションの接続処理中に据え置かれるという方法のために、CICS Monitoring Facility (CMF) レコード、z/OS Workload Manager (WLM) 種別、タスク関連データ、および XAPADMGR 出口の UEPUSID グローバル・ユーザー出口フィールドで使用されるユーザー ID は、常に、領域のユーザー ID になります。

Liberty では、認証済みユーザー ID がキャッシュに入れられ、CICS とは異なり、キャッシュ期間中にユーザー ID の有効期限切れ検査がないことに注意してください。標準 Liberty 構成処理を使用して、キャッシュ・タイムアウトを構成できます。 WebSphere Application Server for z/OS 8.5.5 製品資料内の『Liberty プロファイル上の認証キャッシュの構成』を参照してください。

**関連タスク**:

245 ページの『Liberty JVM サーバーでの SSL (TLS) クライアント証明書認証のセットアップ』

SSL クライアント証明書認証を使用することで、クライアントおよびサーバーは証明書を相手に提供して相互に確認できます。この方法は、セキュリティーについての懸念のために追加レベルの認証が必要となる状況でよく使用されます。

## ユーザーに Liberty JVM サーバー内のアプリケーションを実行する権限を与える

ユーザー ID に Liberty JVM サーバーでトランザクションを実行する権限を与えるには、CICS トランザクションおよびリソース・セキュリティーを使用するか、JEE アプリケーション・セキュリティー・ロールを使用して JEE アプリケーションへのアクセス権限を与えます。

### このタスクについて

CICS セキュリティーを使用すると、既存のセキュリティー・プロシージャーを使用できますが、個別の Web アプリケーションがそれぞれの URIMAP からアクセスさ

れるようにする必要があります。一方、役割ベースのセキュリティーを使用すると、別の JEE アプリケーション・サーバーの既存の標準 JEE セキュリティー定義を再利用できます。 237 ページの『Liberty JVM サーバーでのユーザー認証』を参照してください。CICS 権限を排他的に使用する場合は、server.xml 内のアプリケーションの定義に含まれる特別なサブジェクト ALL_AUTHENTICATED_USERS を使用して、Liberty のロール・ベースの検査をこれ以上行わないようにすることを選択できます。Liberty アプリケーションを CICS バンドルでデプロイする場合、これは CICS によって自動的に構成されます。

```
<application id="com.ibm.cics.server.examples.wlp.tsq.app"
    name="com.ibm.cics.server.examples.wlp.tsq.app" type="eba"
    location="${server.output.dir}/installedApps/com.ibm.cics.server.examples.wlp.tsq.app.eba">
 <application-bnd>
  <security-role name="cicsAllAuthenticated">
   <special-subject type="ALL_AUTHENTICATED_USERS"/>
  </security-role>
 </application-bnd>
</application>
```

この特別なサブジェクトを使用して Web アプリケーションのデプロイメント記述子 (web.xml) 内のすべての URL に cicsAllAuthenticated ロール・アクセスを与えると、認証された任意のユーザー ID を使用して Web アプリケーションにアクセスできるようになるため、トランザクションに対する許可は、CICS トランザクション・セキュリティーを使用して制御する必要があります。アプリケーションを dropins ディレクトリーに直接デプロイする場合、dropins はセキュリティーをサポートしないため、アプリケーションは CICS セキュリティーを使用するようには構成されません。

CICS トランザクションまたはリソースのセキュリティーを使用するには、以下の手順に従う必要があります。

### 手順

1. 各 Web アプリケーションにタイプ JVMSERVER の URIMAP を定義します。通常、URIMAP を Web アプリケーションの一般コンテキスト・ルート (URI) に一致するように指定し、トランザクション ID の有効範囲が、そのアプリケーションを構成するサーブレットのセットになるようにします。あるいは、より正確な URI を使用して、個別のトランザクションでそれぞれのサーブレットが実行されるようにします。
2. CICS トランザクションまたはリソースのセキュリティー・プロファイルを使用して、URIMAP に指定したトランザクションを使用することを Web アプリケーションのすべてのユーザーに許可します。

## JEE アプリケーション・ロール・セキュリティー

JEE アプリケーション・ロール・セキュリティーは、使用する許可タイプに応じてさまざまな方法で構成できます。分散システムでは、基本レジストリーまたは LDAP レジストリーを、通常はアプリケーション固有の `<application-bnd>` エレメントと組み合わせて使用して、これらのレジストリーに基づいてユーザーを*ロール*にマップします。アプリケーションのデプロイメント記述子によって、アプリケーションのどの部分にどのロールがアクセスできるかが決まります。

## このタスクについて

z/OS には、追加のレジストリー・タイプである SAF レジストリーがあります。Liberty JVM サーバーは、cicsts:security-1.0 機能がインストールされると、このタイプを認証に暗黙的に使用します。許可のためにそれを使用することも選択できます。 SAF レジストリー・タイプでは、ユーザーとロール間のマッピング (EJB ロール) がサポートされています。そのため、SAF レジストリー自体から直接、マッピング情報を取得できます。

Liberty JVM サーバーで SAF 許可なしで JEE ロールを使用する場合、CICS バンドルを使用してアプリケーションをインストールすることはできません。その理由は、CICS バンドルがインストールされているアプリケーションの場合、<application-bnd> エレメントが自動的に作成され、特別な対象 ALL_AUTHENTICATED_USERS が使用されるため、ユーザー自身でこのエレメントを定義することがないからです。代わりに、server.xml に <application> エレメントを直接作成し、必要なロールとユーザーを含めて <application-bnd> を構成する必要があります。

ただし、JEE ロールと SAF 許可を使用することを選択した場合は、引き続き CICS バンドルを Web アプリケーションのライフサイクルに対して使用できます。 SAF レジストリーで判別されるロール・マッピングを使用することを優先して、Liberty では <application-bnd> は無視されます。ロール・マッピングは、EJB ロールに属するユーザーによって決まります。

## 手順

1. <safAuthorization id="saf"/> エレメントを server.xml に追加します。
2. 記述されている接頭部スキームへの参照を使用して、必要な EJB ロールを作成します。
3. ユーザーをこれらの EJB ロールに追加します。

   デフォルトでは、SAF 許可を使用すると、ユーザーが特定のロールに属しているかどうかを判別するために <profile_prefix>.<resource>.<role> というパターンがアプリケーションにより使用されます。 profile_prefix 部分はデフォルトでは BBGZDFLT ですが、<safCredential> エレメントを使用して変更できます。詳しくは、WebSphere Application Server for z/OS 8.5.5 製品資料内の『Liberty プロファイル: WZSSAD を使用した z/OS セキュリティー・リソースへのアクセス』を参照してください。

   ロール・マッピングの設定は、server.xml の <safRoleMapper> エレメントを使用して変更できます。このエレメントのデフォルトは、

   ```
   <safRoleMapper profilePattern="myprofile.%resource%.%role%" toUpperCase="true"/>
   ```

   です。

   以下の RACF コマンドを使用して、特定の EJB ロールに対する権限をユーザーに許可できます。ここで、WEBUSER は認証済みユーザー ID です。

   ```
   RDEFINE EJBROLE BBGZDFLT.MYAPP.ROLE UACC(NONE)
   PERMIT BBGZDFLT.MYAPP.ROLE CLASS(EJBROLE) ACCESS(READ) ID(WEBUSER)
   ```

4. オプション: CICS サーブレット・サンプルをデプロイし、JEE ロール・セキュリティーを SAF 許可とともに使用する場合は、デプロイしたサーブレットごとに SAF EJBROLE を作成します。例えば、BBGZDFLT のデフォルト APPL クラスを使用する場合は、以下の EJBROLE セキュリティー定義を定義します。

```
RDEFINE EJBROLE BBGZDFLT.com.ibm.cics.server.examples.wlp.hello.war.cicsAllAuthenticated UACC(NONE)
RDEFINE EJBROLE BBGZDFLT.com.ibm.cics.server.examples.wlp.tsq.app.cicsAllAuthenticated UACC(NONE)
RDEFINE EJBROLE BBGZDFLT.com.ibm.cics.server.examples.wlp.jdbc.app.cicsAllAuthenticated UACC(NONE)
SETROPTS RACLIST(EJBROLE) REFRESH
```

許可を必要とする Web ユーザー ID ごとに、定義したロールに読み取り権限を付与します。

```
PERMIT BBGZDFLT.com.ibm.cics.server.examples.wlp.hello.war.cicsAllAuthenticated
       CLASS(EJBROLE) ID(user) ACCESS(READ)
PERMIT BBGZDFLT.com.ibm.cics.server.examples.wlp.tsq.app.cicsAllAuthenticated
       CLASS(EJBROLE) ID(user) ACCESS(READ)
PERMIT BBGZDFLT.com.ibm.cics.server.examples.wlp.jdbc.app.cicsAllAuthenticated
       CLASS(EJBROLE) ID(user) ACCESS(READ)
SETROPTS RACLIST(EJBROLE) REFRESH
```

### タスクの結果

ロールと、ロールに含めるユーザーを定義することにより、CICS セキュリティー、JEE ロール・セキュリティー、あるいはその両方を使用して、Web アプリケーションに対するアクセス権限を付与することができます。

## Java 鍵ストアを使用した Liberty JVM サーバー用の SSL (TLS) の構成

SSL を使用してデータを暗号化するように Liberty JVM サーバーを構成できます。オプションとして、クライアント証明書を使用してサーバーで認証するように構成することもできます。証明書は、Java 鍵ストアか、または RACF などの SAF 鍵リングに保管できます。

### このタスクについて

Liberty JVM サーバーで SSL を使用可能にするには、**ssl-1.0** Liberty フィーチャー、鍵ストア、および HTTPS ポートを追加する必要があります。Liberty プロファイルの構成ファイル、`server.xml` を自動構成します。CICS では自動的に `server.xml` ファイルが作成され、更新されます。自動構成では、必ず Java 鍵ストアが作成されます。

Liberty JVM サーバーへの Web 要求では、CICS ソケット・ドメインではなく、TCP/IP ソケットおよび SSL 処理の JVM サポートが使用されることを理解することが重要です。

### 手順

SSL を構成するために自動構成を使用するには、以下のステップを実行します。

1. JVM プロファイルで JVM システム・プロパティー **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true** を使用して、自動構成を有効にします。

2. JVM プロファイルで JVM システム・プロパティー **-Dcom.ibm.cics.jvmserver.wlp.server.https.port** を設定して、SSL ポートを設定します。
3. JVM サーバーを再始動して、必要な構成エレメントを server.xml に追加します。

### タスクの結果

Liberty JVM サーバーのための SSL が正常に構成されます。

**関連タスク**:

245 ページの『Liberty JVM サーバーでの SSL (TLS) クライアント証明書認証のセットアップ』
SSL クライアント証明書認証を使用することで、クライアントおよびサーバーは証明書を相手に提供して相互に確認できます。この方法は、セキュリティーについての懸念のために追加レベルの認証が必要となる状況でよく使用されます。

『RACF を使用した Liberty JVM サーバー用の SSL (TLS) の構成』
SSL を使用してデータを暗号化するように Liberty JVM サーバーを構成できます。オプションとして、クライアント証明書を使用してサーバーで認証するように構成することもできます。 証明書は、Java 鍵ストアか、または RACF などの SAF 鍵リングに保管できます。

## RACF を使用した Liberty JVM サーバー用の SSL (TLS) の構成

SSL を使用してデータを暗号化するように Liberty JVM サーバーを構成できます。オプションとして、クライアント証明書を使用してサーバーで認証するように構成することもできます。証明書は、Java 鍵ストアか、または RACF などの SAF 鍵リングに保管できます。

### このタスクについて

Liberty JVM サーバーで SSL を使用可能にするには、**ssl-1.0** Liberty フィーチャー、鍵ストア、および HTTPS ポートを追加する必要があります。server.xml を手動で構成します。server.xml ファイルを編集して、必要なエレメントと値を追加します。RACF 鍵リングを使用する場合は、手動の手順に従う必要があります。

Liberty JVM サーバーへの Web 要求では、CICS ソケット・ドメインではなく、TCP/IP ソケットおよび SSL 処理の JVM サポートが使用されることを理解することが重要です。

### 手順

SSL を手動で構成するには、署名証明書を作成する必要があります。この署名証明書を使用して、サーバー証明書を作成します。そして、その署名証明書を、署名証明書を使用してサーバー証明書の認証を行うクライアントの Webブラウザーにエクスポートします。

1. 認証局 (CA) 証明書 (署名証明書) を作成します。RACF コマンドを使用する例を以下に示します。

```
RACDCERT GENCERT
  CERTAUTH
  SUBJECTSDN(CN('CICS Sample Certification Authority')
   O('IBM')
   OU('CICS'))
   SIZE(1024)
  WITHLABEL('CICS-Sample-Certification')
```

2. ステップ 2 の署名証明書を使用したサーバー証明書を作成します。ここで、<userid> は CICS 領域ユーザー ID です。hostname は、Liberty サーバー HTTPS ポートで使用するように構成されたサーバーのホスト名です。

```
RACDCERT ID(<userid>) GENCERT
  SUBJECTSDN(CN('<hostname>')
   O('IBM')
   OU('CICS'))
   SIZE(1024)
   SIGNWITH (CERTAUTH LABEL('CICS-Sample-Certification'))
  WITHLABEL('<userid>-Liberty-Server')
```

3. 署名証明書およびサーバー証明書を RACF 鍵リングに接続します。 以下のコマンドで RACF を使用できます。<keyring> の値は、使用する鍵リングの名前に置き換えてください。<userid> の値は CICS 領域ユーザー ID に置き換えてください。

```
RACDCERT ID(<userid>) CONNECT(RING(<keyring>)
       LABEL('CICS-Sample-Certification')
       CERTAUTH)

RACDCERT ID(<userid>) CONNECT(RING(<keyring>)
       LABEL('<userid>-Liberty-Server'))
```

署名証明書を CER ファイルにエクスポートします。

```
RACDCERT CERTAUTH EXPORT(LABEL('CICS-Sample-Certification'))
       DSN('<userid>.CERT.LIBCERT')
       FORMAT(CERTDER)
       PASSWORD('password')
```

エクスポートされた証明書を、FTP を使用してバイナリー形式でワークステーションに転送し、認証局証明書としてブラウザーにインポートします。

4. server.xml ファイルを編集し、SSL 機能と鍵ストアを追加します。HTTPS ポート (次の例の値は 9443) を設定し、CICS 領域を再始動します。SAF 鍵リングを URL 形式 safkeyring://<userid>/<keyring> で指定する必要があります。<userid> 値には CICS 領域ユーザー ID を設定し、<keyring> 値には鍵リングの名前を設定する必要があります。パスワード・フィールドは、SAF 鍵リングへのアクセスには使用されないため、password に設定する必要があります。

```
<featureManager>
  ...
       <feature>ssl-1.0</feature>
</featureManager>
...
<httpEndpoint host="*" httpPort="9080" httpsPort="9443"
    id="defaultHttpEndpoint"/>
...
.
<keyStore filebased="false" id="racfKeyStore"
        location="safkeyring://<userid>/<keyring>"
        password="password"
        readOnly="true"
        type="JCERACFKS"/>
```

```
<ssl id="defaultSSLConfig" keyStoreRef="racfKeyStore"
      sslProtocol="SSL_TLS"
      serverKeyAlias="<userid>-Liberty-Server" />
```

### タスクの結果

Liberty JVM サーバーのための SSL が正常に構成されます。

**関連タスク**:

『Liberty JVM サーバーでの SSL (TLS) クライアント証明書認証のセットアップ』
SSL クライアント証明書認証を使用することで、クライアントおよびサーバーは証明書を相手に提供して相互に確認できます。この方法は、セキュリティーについての懸念のために追加レベルの認証が必要となる状況でよく使用されます。

242 ページの『Java 鍵ストアを使用した Liberty JVM サーバー用の SSL (TLS) の構成』
SSL を使用してデータを暗号化するように Liberty JVM サーバーを構成できます。オプションとして、クライアント証明書を使用してサーバーで認証するように構成することもできます。 証明書は、Java 鍵ストアか、または RACF などの SAF 鍵リングに保管できます。

## Liberty JVM サーバーでの SSL (TLS) クライアント証明書認証のセットアップ

SSL クライアント証明書認証を使用することで、クライアントおよびサーバーは証明書を相手に提供して相互に確認できます。この方法は、セキュリティーについての懸念のために追加レベルの認証が必要となる状況でよく使用されます。

### 始める前に

242 ページの『Java 鍵ストアを使用した Liberty JVM サーバー用の SSL (TLS) の構成』のタスクを完了する必要があります。まだ CICS Liberty のセキュリティーをセットアップ済みでない場合は、先に進む前に、『Securing』の『Liberty JVM サーバーのセキュリティーの構成 (Configuring security for a Liberty JVM server)』を完了させてください。

### このタスクについて

以下のセットアップ情報では、RACF 鍵ストアを使用して、SSLクライアント証明書認証の証明書を保管していることを想定しています。

### 手順

1. 署名証明書を使用して個人証明書を作成し、この個人証明書を RACF ユーザー ID に関連付けます。 次に、個人証明書を CER 形式でデータ・セットにエクスポートしてから、ワークステーションにバイナリーで FTP 転送します。その個人証明書を Web ブラウザーに個人証明書としてインポートします。証明書が Web ブラウザーにインポートされると、SSL クライアント証明書を提供して Liberty サーバーの HTTPS ポートに接続できるようになります。次の RACF コマンドを実行します。ここで `<clientuserid>` は、RACF ユーザー ID を `<hostname>` はクライアント・コンピューターのホスト名を表します。

```
RACDCERT ID(<clientuserid>) GENCERT
  SUBJECTSDN(CN('<hostname>')
   O('IBM')
   OU('CICS'))
   SIZE(1024)
   SIGNWITH (CERTAUTH LABEL('CICS-Sample-Certification'))
  WITHLABEL('<clientuserid>-certificate')
```

この手順で既に行ったように、個人証明書をエクスポートします。

```
RACDCERT ID(<clientuserid>)
  EXPORT(LABEL('<clientuserid>-certificate'))
   DSN('USERID.CERT.CLICERT')
   FORMAT(PKCS12DER)
   PASSWORD('password')
```

SSL クライアント証明書認証をサポートするように、server.xml の SSL 要素を更新します。

```
<ssl id="defaultSSLConfig" keyStoreRef="racfKeyStore"
   sslProtocol="SSL_TLS"
   serverKeyAlias="<userid>-Liberty-Server"
   clientAuthenticationSupported="true"/>
```

さらに、すべてのクライアントが有効な SSL クライアント証明書を必ず提供するようにしたい場合は、次のようにして clientAuthentication 属性を SSL 要素に追加します。

```
<ssl id="defaultSSLConfig" keyStoreRef="racfKeyStore"
   sslProtocol="SSL_TLS"
   serverKeyAlias="<userid>-Liberty-Server"
   clientAuthenticationSupported="true"
   clientAuthentication="true"/>
```

2. 手順 2 のクライアント・ユーザー ID の ID を使用して CICS で Web 要求を認証できます。その後、web.xml に CLIENT-CERT の login-config 要素を指定して、Web アプリケーションをデプロイします。web.xml ファイルは、デプロイする Web アプリケーションのソース・ファイルにあります。

   ```
   <login-config>
     <auth-method>CLIENT-CERT</auth-method>
   </login-config
   ```

   代わりに、SSL クライアント証明書認証が構成されていない場合に HTTP 基本認証へのフェイルオーバーを許可するには、server.xml に webAppSecurity 要素を追加します。

   ```
   <webAppSecurity allowFailOverToBasicAuth="true" />
   ```

3. 最後に、CICS トランザクション・セキュリティーをセットアップして、CICS トランザクションへのアクセスを認証済みクライアント・ユーザー ID に基づいて許可します。 詳しくは、 239 ページの『ユーザーに Liberty JVM サーバー内のアプリケーションを実行する権限を与える』 のトピックを参照してください。

**関連タスク**:

『Securing』の『SSL 認証』

**関連情報**:

# z/OS Connect for CICS のセキュリティー

z/OS Connect は、WebSphere Liberty プロファイル・アプリケーションであり、その構成と考慮事項は、他の WebSphere Liberty プロファイル・アプリケーションと同じです。z/OS Connect for CICS には、いくつかの追加要件があります。

z/OS Connect for CICS は、動的サービス・ディスカバリーを可能にする RESTful 管理インターフェースを備えています。このインターフェースは、個々の JSON サービスと同じホスト名およびポート番号でホストされます。z/OS Connect for CICS は、このインターフェースと個々の JSON サービスを保護するために Transport Layer Security (TLS) を使用する必要があります。

z/OS Connect for CICS へのクライアント接続はすべて、HTTPS プロトコルを使用する必要があります。z/OS Connect for CICS は、デフォルトでは、クライアント認証による CICS への TLS 接続を必要とするように構成されます。このデフォルトをそのまま使用する場合は、クライアント証明書を SAF ユーザー ID に関連付ける必要があります。アプリケーションは、この証明書から得られた ID を使用して実行されます。

z/OS Connect for CICS は、HTTP 基本認証プロトコルをサポートするように構成できます。このプロトコルを使用すると、クライアントは TLS と SAF ユーザー ID/パスワードとを組み合わせて使用して接続できるようになります。HTTP 基本認証をサポートできるようにするには、Liberty サーバー構成ファイル (`server.xml`) に次の行を追加します。`<webAppSecurity allowFailOverToBasicAuth="true" />`

詳しくは、WebSphere Application Server for z/OS 8.5.5 製品資料内の『z/OS Connect のセキュリティーの構成』を参照してください。

z/OS Connect のセキュリティーを適切に構成するには、`zos.connect.access.roles.zosConnectAccess` ロールのメンバーでなければなりません。このロールは RACF で EJBROLE リソース・プロファイルにマップされており、ユーザーとグループが含まれています。EJBROLES を使用するには SAF を有効にする必要があります。`<safAuthorization id="saf"/>` エレメントを `server.xml` 構成ファイルに追加してください。EJBROLES について詳しくは、240 ページの『JEE アプリケーション・ロール・セキュリティー』のトピックを参照してください。

WebSphere Liberty プロファイルの IBM Knowledge Center には、WebSphere Application Server for z/OS 8.5.5 製品資料内の『Liberty プロファイル上のアプリケーションの許可の構成』に関する情報が記載されています。 CICS のトランザクション・セキュリティーを有効にする場合、詳細については、『Securing』の『トランザクション・セキュリティー』を参照してください。

**関連情報**:

# Java セキュリティー・マネージャーの有効化

デフォルトで、Java アプリケーションでは、Java API に要求されるアクティビティーにセキュリティーの制限がありません。Java セキュリティーを使用して、安全でない可能性があるアクションを Java アプリケーションが実行しないようにするために、そのアプリケーションが実行される JVM に対してセキュリティー・マネージャーを有効にすることができます。

## このタスクについて

セキュリティー・マネージャーは、コード・ソースに割り当てられる 1 組の権限 (システム・アクセス権) であるセキュリティー・ポリシーを実施します。 Java プラットフォームにはデフォルトのポリシー・ファイルが用意されています。ただし、Java セキュリティーがアクティブであるときに、Java アプリケーションを CICS で正常に実行できるようにするには、アプリケーションの実行に必要な権限を CICS に付与する追加のポリシー・ファイルを指定する必要があります。

この追加のポリシー・ファイルは、セキュリティー・マネージャーが有効になっている JVM の種類ごとに指定する必要があります。CICS には、独自のポリシーを作成するために使用できるサンプルが用意されています。

**注:** Liberty JVM サーバーでは、Java セキュリティー・マネージャーの有効化はサポートされていません。

- OSGi セキュリティー・エージェントのサンプルは、プロジェクトに `com.ibm.cics.server.examples.security` という名前の OSGi ミドルウェア・バンドルを作成します。その中には、セキュリティー・プロファイルが含まれています。このプロファイルは、インストール先のフレームワーク内にあるすべての OSGi バンドルに適用されます。
- `example.permissions` ファイルには、JVM サーバーで実行中のアプリケーションに固有の許可が含まれています。それには、アプリケーションが System.exit() メソッドを使用していないことを確認するチェックが含まれます。
- CICS に、zFS の中で OSGi バンドルの配置先となるディレクトリーに対する読み取りアクセス権限と実行アクセス権限が必要です。

JVM サーバーの OSGi フレームワークで実行されるアプリケーションの場合:

## 手順

1. CICS Explorer SDK でプラグイン・プロジェクトを作成して、提供される OSGi セキュリティー・エージェントのサンプルを選択します。
2. プロジェクトで、`example.permissions` ファイルを選択して、セキュリティー・ポリシーの許可を編集します。
   a. CICS zFS および DB2 のインストール・ディレクトリーが正しく指定されていることを確認します。
   b. 必要に応じて他の権限を追加します。
3. zFS 内の適切なディレクトリー (例えば `/u/bundles`) に OSGi バンドルをデプロイします。
4. JVM サーバーの JVM プロファイルを編集して、他のすべてのバンドルより前になるように、`OSGI_BUNDLES` オプションに OSGi バンドルを追加します。

```
OSGI_BUNDLES=/u/bundles/
com.ibm.cics.server.examples.security_1.0.0.jar
```

5. JVM プロファイルに次の Java プロパティーを追加して、セキュリティーを有効にします。

   ```
   -Djava.security.policy=all.policy
   ```

6. JVM プロファイルに次の Java 環境変数を追加して、OSGi フレームワーク内でセキュリティーを有効にします。

   ```
   org.osgi.framework.security=osgi
   ```

7. OSGi フレームワークを Java 2 セキュリティーで開始できるようにするには、以下のポリシーを追加します。

   ```
   grant { permission java.security.AllPermission; };
   ```

8. 変更を保管し、JVMSERVER リソースを使用可能にして、ミドルウェア・バンドルを JVM サーバーにインストールします。
9. オプション: Java 2 セキュリティーをアクティブにします。
   a. Java 2 セキュリティー・ポリシー・メカニズムをアクティブにするには、それを適切な JVM プロファイルに追加します。また、Java 2 セキュリティー・ポリシーを編集して、適切な権限を付与する必要があります。
   b. Java 2 セキュリティー・ポリシー・メカニズムがアクティブな状態で Java アプリケーションから JDBC または SQLJ を使用するには、IBM Data Server Driver for JDBC and SQLJ を使用します。
   c. Java 2 セキュリティー・ポリシー・メカニズムをアクティブにするには、JVM プロファイルを編集します。 『デプロイ』の『Java セキュリティー・マネージャーの有効化』に記載されている Java 2 セキュリティー・ポリシーのセットアップ方法を参照してください。
   d. JDBC ドライバーに対する権限を付与するには、例 1 に示されている行を追加して、Java 2 セキュリティー・ポリシーを編集します。db2*xxx* の代わりに、すべての DB2 ライブラリーが配置されているディレクトリーを指定してください。権限は、このレベルより下のすべてのディレクトリーとファイルに適用されます。これにより、JDBC および SQLJ を使用できるようになります。
   e. 例 2 に示されている行を追加して、Java 2 セキュリティー・ポリシーを編集して読み取り許可を付与します。読み取り許可を追加しないで Java プログラムを実行すると、AccessControlExceptions と予測不能な結果が生成されます。Java 2 セキュリティー・ポリシーで JDBC および SQLJ を使用できます。

**例 1:**

```
grant codeBase "file:/usr/lpp/db2xxx/-" {
  permission java.security.AllPermission;
};
```

**例 2:**

```
grant {

// allows anyone to read properties
permission java.util.PropertyPermission "*", "read";

};
```

## タスクの結果

Java アプリケーションが呼び出されると、JVM は、クラスのコード・ソースを判別し、セキュリティー・ポリシーを調べてから、適切な権限をクラスに付与します。

# 第 8 章 Java アプリケーションのトラブルシューティング

Java アプリケーションに問題がある場合は、CICS と JVM で提供される診断機能を使用して、問題の原因を調べることができます。

## このタスクについて

CICS は、Java に関連した問題の診断に役立ついくつかの統計、メッセージ、およびトレースを提供します。Java に付属の診断ツールとインターフェースは、JVM で何が発生しているかについて CICS よりも詳しい情報を提供できます。これは、CICS が JVM 内のアクティビティーの多くを認識しないからです。

JVM の分析をリアルタイムとオフラインで実行する無料のツール (例えば、JConsole や IBM Health Center) を使用できます。詳細については、IBM SDK, Java Technology Edition バージョン 7 の『診断ツールの使用』を参照してください。

Liberty JVM サーバーで実行している Web アプリケーションのトラブルシューティングについては、269 ページの『第 9 章 Liberty JVM サーバーおよび Java Web アプリケーションのトラブルシューティング』を参照してください。ログ・ファイルが存在する場所については、257 ページの『JVM stdout、stderr、JVMTRACE、およびダンプ出力の場所の制御』を参照してください。

## 手順

1. JVM サーバーを始動できない場合は、Java インストールのセットアップが正しいことを確認してください。 CICS メッセージおよび JVM の `stderr` ファイルにあるエラーを使用して、問題の原因を判別してください。
   a. 正しいバージョンの Java SDK がインストールされていることと、CICS が z/OS UNIX 内の Java SDK にアクセスできることを確認してください。 サポートされている SDK のリストについては、『高水準言語サポート』を参照してください。
   b. **`USSHOME`** システム初期設定パラメーターが CICS 領域で設定されていることを確認します。 このパラメーターは、z/OS UNIX 上のファイルのホームを指定します。
   c. **`JVMPROFILEDIR`** システム初期設定パラメーターが CICS 領域で正しく設定されていることを確認します。 このパラメーターは、z/OS UNIX 上の JVM プロファイルの場所を指定します。
   d. JVM プロファイルが入っている z/OS UNIX ディレクトリーへの読み取りアクセス権限と実行アクセス権限が CICS 領域にあることを確認します。
   e. JVM の作業ディレクトリーへの書き込みアクセス権限が CICS 領域にあることを確認します。 このディレクトリーは、JVM プロファイルの `WORK_DIR` オプションで指定されます。
   f. JVM プロファイルの `JAVA_HOME` オプションが、Java SDK が入っているディレクトリーを指し示していることを確認します。
   g. SDFJAUTH が CICS 開始 JCL の STEPLIB 連結にあることを確認します。

h. WebSphere MQ または DB2 の DLL ファイルを使用している場合は、これらのファイルの 64 ビット・バージョンが CICS から使用可能であることを確認してください。

i. Language Environment エンクレーブを構成するために DFHAXRO を変更する場合は、実行時オプションが 200 バイトを超えないことと、それらのオプションが有効であることを確認してください。 CICS は、指定されたオプションを Language Environment に渡す前に、それらを検証しません。 Language Environment からのエラー・メッセージがないかどうか、SYSOUT を確認してください。

2. セットアップが正しい場合は、診断情報を収集して、アプリケーションと JVM に何が起きているかを調べます。

a. JVM プロファイルに PRINT_JVM_OPTIONS=YES を追加します。 このオプションを指定すると、クラスパスの内容を始めとして、開始時に JVM に渡されるすべてのオプションが SYSPRINT に出力されます。JVM がプロファイル内でこのオプションを指定して開始されるたびに、情報が生成されます。

b. JVM からの情報およびエラー・メッセージがないか、dfhjvmout ファイルと dfhjvmerr ファイルを調べます。 これらのファイルは、JVM プロファイルの WORK_DIR オプションで指定されるディレクトリーにあります。JVM プロファイルで STDOUT オプションと STDERR オプションが変更された場合、これらのファイルの名前が異なる可能性があります。

3. アプリケーションが障害を起こしたか、アプリケーションのパフォーマンスが低下する場合は、アプリケーションをデバッグします。

- java.lang.ClassNotFoundException エラーを受け取り、トランザクションが AJ05 コードで異常終了する場合は、アプリケーションが OSGi フレームワークの IBM またはベンダーのクラスにアクセスできない可能性があります。この問題の修正方法について詳しくは、『アップグレード』の『JVM サーバーにおける Java アプリケーションのアップグレード』を参照してください。
- CEDX トランザクションを使用して、アプリケーション・トランザクションをデバッグします。Liberty JVM サーバーの場合、URI マップを使用してインバウンド・アプリケーション要求をアプリケーション・トランザクションに対応させているとしたら、そのトランザクションをデバッグします。デフォルトのトランザクション CJSA を使用する場合、DFHEDFTC トランザクション・クラスで MAXACTIVE 属性を 1 に設定する必要があります。この設定が必要となる理由は、多数の CJSA タスクが実行されている場合があり、誤ったトランザクションをデバッグする可能性があるためです。実稼働環境では、CJSA トランザクションで CEDX を使用しないでください。
- JVM サーバーでデバッガーを使用するには、JVM プロファイルでいくつかのオプションを設定する必要があります。詳しくは、 265 ページの『Java アプリケーションのデバッグ』を参照してください。
- OSGi バンドルおよびサービスの状態を確認する場合は、OSGi コンソールを使用してください。 JVM プロファイルに次のプロパティーを設定します。-Dosgi.console=host:port (ここで、host は Liberty サーバーが稼働しているシステムのホスト名、port は同じシステム上の空きポート) および -Dosgi.console.encoding=ISO8859-1。 Liberty サーバーを使用している場合は、osgiConsole-1.0 フィーチャーを server.xml に追加します。 JVM プロ

ファイルに指定したホストおよびポートのプロパティーを指定して Telnet セッションを使用することで、OSGi コンソールに接続します。

**注:** OSGi コンソールで `exit` コマンドを入力すると、JVMSERVER が動作する環境に system.exit(0) 呼び出しを実行します。 system.exit(0) は、すべてのスレッドとワークロードを即時に停止し、そのまま処理を継続する場合は、JVM と CICS が確定できない状態になります。このような場合、それ以降の複雑さを回避するために即時シャットダウンを実行するように、CICS は設計されています。このため、JVM プロファイルと `server.xml` の両方の書き込み権限を制御することが重要です。Liberty JVM サーバーは、さらに保護を提供するために、OSGi コンソールが実行可能になる前に、osgiconsole-1.0 機能を含めることを要求します。OSGi コンソールは、主に開発とデバッグの補助機能であり、実稼働環境での実行は想定されていません。

4. メモリー不足エラーが表示される場合、64 ビット・ストレージが不十分であるか、アプリケーションにメモリー・リークがあるか、またはヒープ・サイズが不十分である可能性があります。
   a. CICS 統計またはツール (IBM Health Center など) を使用して、JVM をモニターします。アプリケーションにメモリー・リークがある場合、ガーベッジ・コレクション後に残っているライブ・データの量が、ヒープが使い果たされるまで時間と共に徐々に増えます。 JVM サーバー統計は、最後のガーベッジ・コレクション後のヒープのサイズ、およびヒープの最大サイズとピーク・サイズを報告します。詳しくは、211 ページの『IBM Health Center を使用した Java アプリケーションの分析』を参照してください。
   b. Language Environment のストレージ・レポートを実行して、ストレージの量が十分かどうかを確認してください。 詳しくは、220 ページの『JVM の Language Environment エンクレーブ・ストレージ』を参照してください。
5. Java アプリケーションのインストール時や実行時にエンコード・エラーが表示される場合、セットアップしたコード・ページの組み合わせが競合しているか、サポートされていない可能性があります。 z/OS 上の JVM は通常、ファイルのエンコードに EBCDIC コード・ページを使用します。非 Liberty JVM サーバーのデフォルトは IBM1047 (または cp1047) ですが、JVM では必要に応じて他のコード・ページをファイルのエンコードに使用できます。 CICS では文字データの処理に EBCDIC コード・ページが必要で、すべての JCICS 呼び出しも EBCDIC コード・ページを使用しなければなりません。コード・ページは、CICS 領域に関する **LOCALCCSID** システム初期設定パラメーターで設定されます。
   a. JVM サーバー・ログを調べて、**LOCALCCSID** の値に関連した警告メッセージが発行されたかどうかを確認します。 このパラメーターが EBCDIC でないコード・ページ、JVM でサポートされていないコード・ページ、またはサポートされていない EBCDIC コード・ページ (930 など) に設定されている場合は、JVM サーバーは cp1047 を使用します。
   b. JCICS 呼び出しは、**LOCALCCSID** システム初期設定パラメーターで指定されているコード・ページを使用します。 アプリケーションが別のコード・ページを想定している場合は、エンコード・エラーが表示されます。 JCICS に別のコード・ページを使用するには、JVM プロファイル内で **-Dcom.cics.jvmserver.override.ccsid=** パラメーターを設定します。

c. JVM プロファイル内で **-Dcom.cics.jvmserver.override.ccsid=** パラメーターを使用している場合は、CCSID が EBCDIC コード・ページであることを確認してください。 JCICS 呼び出しの使用時には、アプリケーションは EBCDIC を使用しなければなりません。

d. Axis2 JVM サーバー内で SOAP 処理を実行している場合は、**-Dfile.encoding** JVM プロパティーで EBCDIC プロパティーが指定されていることを確認してください。 UTF-8 などの EBCDIC でないコード・ページを指定すると、Web サービス要求は失敗し、応答には壊れたデータが含まれます。

6. メッセージ DFHSJ0904 が発行され、次の例のような問題が報告される場合、JVM サーバーのスレッド限界に達している可能性があります。

```
Exception 'java.lang.Exception: CICSThreadExecutor: no work found for Task 45921.
The work this Task was started to perform has already timed-out.'
```

次のように問題解決を試みてください。

- HTTP 要求で JVM サーバー・スレッドを取得するために待機する時間を制御するために、**-Dcom.ibm.cics.jvmserver.threadjoin.timeout** 設定を変更します。
- JVMSERVER リソースの **THREADLIMIT** 値を増やしてください。
- **THREADLIMIT** が既に最大許可値に設定されている場合、単一の JVM サーバーが処理可能な作業量を超えて実行を試みている可能性があります。複数の JVM サーバーまたは複数の領域間のワークロード・バランシングを検討してください。

  または、他の制約が原因で CICS システムが応答しなくなっている可能性もあります。パフォーマンス上の問題を診断するための標準的な手順に従ってください。 「パフォーマンスの改善」の『CICS システムのパフォーマンスの向上』を参照してください。

## 次のタスク

問題の原因を確定できない場合は、IBM サポートにお問い合わせください。 Java の問題を報告するための MustGather にリストされているとおりに、必要な情報を確実に提供してください。

255 ページの『Java の診断』
通常の CICS 診断情報源の多くには、Java アプリケーションに適用される情報が含まれています。CICS 提供の情報に加えて、問題判別に使用できる、JVM 固有の複数のインターフェースがあります。

257 ページの『JVM stdout、stderr、JVMTRACE、およびダンプ出力の場所の制御』
JVM サーバーで実行される Java アプリケーションからの出力は、z/OS UNIX ファイルに書き込むことができます。 JVM サーバーの STDOUT、STDERR、および JVMTRACE オプションで z/OS UNIX ファイルを指定します。指定しなければ、JES ログに転送されます。

263 ページの『JVM サーバーの OSGi ログ・ファイルの管理』
OSGi フレームワークは、JVM サーバーの作業ディレクトリーにある 1 組のロ

グ・ファイルにエラーを書き込みます。ご使用の環境にデフォルトが適切でない場合は、JVM サーバーごとにログ・ファイルの数とサイズを管理できます。

263 ページの『JVM サーバーの CICS コンポーネント・トレース』
Java によって作成されるログに加えて、CICS は、SJ (JVM) および AP ドメインで、トレース・レベル 0、1、および 2 において、いくつかの標準トレース・ポイントを提供します。これらのトレース・ポイントは、CICS が JVM サーバーのセットアップと管理を行う際に取るアクションをトレースします。

264 ページの『JVM サーバーのトレースの活動化と管理』
SJ および AP コンポーネントのトレースをオンにすると、JVM サーバーのトレースを活動化できます。少量のトレースが内部トレース・テーブルに書き込まれますが、Java はまた、ロギング情報を JVM サーバーごとに zFS の固有のファイルに書き込みます。このファイルはラップしないので、そのサイズを zFS で管理する必要があります。

265 ページの『Java アプリケーションのデバッグ』
CICS における JVM は、Java 2 Platform で提供される標準デバッグ・メカニズムである Java Platform Debugger Architecture (JPDA) をサポートします。このアーキテクチャーは、リモート・デバッガーと JVM との接続を可能にする 1 組の API を提供します。

266 ページの『CICS JVM プラグイン・メカニズム』
JVM の標準 JPDA デバッグ・インターフェースに加えて、CICS Java ミドルウェアには CICS 提供の 1 組の代行受信ポイント (プラグイン) があります。これは、アプリケーションのデバッグに役立つ場合があります。これらのプラグインを使用して、アプリケーション Java コードの実行直前と直後に追加の Java プログラムを挿入できます。

## Java の診断

通常の CICS 診断情報源の多くには、Java アプリケーションに適用される情報が含まれています。CICS 提供の情報に加えて、問題判別に使用できる、JVM 固有の複数のインターフェースがあります。

### Java の CICS 診断ツール

CICS には、実行中の Java アプリケーションに関して収集できる統計とモニター・データがあります。エラーが発生すると、トランザクションは異常終了し、メッセージが該当するログに書き込まれます。JVM (SJ) ドメインに適用される異常終了とメッセージのリストについては、『Reference』->『Diagnostics』の『CICS メッセージ』 を参照してください。 Java に関連したメッセージの形式は DFHSJxxxx です。

また、トレースをオンにして、追加の診断情報を生成することもできます。 JVM ドメインのトレース・ポイントは、『Reference』->『Diagnostics』の『JVM ドメインのトレース・ポイント』 にリストされています。

初期化後に最初の JVM が CICS 領域で開始すると、CICS は、メッセージ DFHSJ0207 を発行して、使用されている Java のバージョンを表示します。

Java SDK が提供する診断ツールとインターフェースは、JVM で何が発生しているかについてより詳細な情報を提供します。 JVM からのメッセージと診断情報は、

JVM の stderr ログ・ファイルに書き込まれます。Java の問題を検出した場合は、必ずこのファイルを調べてください。例えば、CICS がメッセージを発行して、JVM が異常終了したことを示す場合、stderr ログ・ファイルが第一の診断情報源です。257 ページの『JVM stdout、stderr、JVMTRACE、およびダンプ出力の場所の制御』では、JVM からの出力の場所を制御する方法、および JVM 内部からのメッセージならびに JVM で実行中の Java アプリケーションからの出力のリダイレクト方法を示しています。

CICS 用の Java アプリケーションを開発する際には、CICS におけるスレッド・セーフティーとトランザクション分離の要件を考慮することが重要です。 Java アプリケーションが、最初に使用されるときに正しく機能するものの、それ以降の使用時に正しく動作しない場合、この問題はおそらく、分離の問題が原因です。

## OSGi 診断ファイル

OSGi フレームワークは、JVM サーバーにおける OSGi バンドルおよびサービスの問題のトラブルシューティングに使用できる診断ファイルを zFS で作成します。

**OSGi キャッシュ**

OSGi キャッシュは、JVM サーバーの *$WORK_DIR*/*applid*/*jvmserver*/configuration/org.eclipse.osgi ディレクトリーにあります。*$WORK_DIR* は JVM サーバーの作業ディレクトリー、*applid* は CICS APPLID、*jvmserver* は JVMSERVER リソースの名前です。OSGi キャッシュには、フレームワークのメタデータ、およびフレームワークの実行に必要なその他の情報が入っています。JVM サーバーが始動するときに、キャッシュが置き換えられます。

**OSGi ログ**

OSGi フレームワークでエラーが発生すると、OSGi ログが、JVM サーバーの *$WORK_DIR*/*applid*/*jvmserver*/configuration/ ディレクトリーに作成されます。ファイル拡張子は .log です。

## JVM 診断ツール

CICS 資料には、一部の Java 診断ツールとインターフェースに関する情報が記載されています。

- 264 ページの『JVM サーバーのトレースの活動化と管理』では、CETR トランザクションによって提供されるコンポーネント・トレースを使用して、JVM サーバーと JVM サーバー内で実行されるタスクのライフサイクルをトレースする方法を説明しています。JVM サーバーは補助トレースも GTF トレースも使用しません。代わりに、JVM サーバーごとに固有に名前が指定される zFS 上のファイルにトレースが書き込まれます。
- 265 ページの『Java アプリケーションのデバッグ』 では、リモート・デバッガーを使用して、JVM で実行されている Java アプリケーションのアプリケーション・コードをステップスルーする方法について説明しています。また、CICS は CICS Java ミドルウェアに一連の代行受信ポイント (すなわちプラグイン) を用意しているため、デバッグ、ロギング、またはその他の目的のための追加の Javaプログラムを、アプリケーション Java コードの実行直前および直後に挿入できます。詳しくは、266 ページの『CICS JVM プラグイン・メカニズム』を参照してください。

JVM にはより多くの診断ツールとインターフェースが使用できます。JVM の問題判別に使用できる追加の機能については、IBM SDK for z/OS, Java Technology Edition バージョン 7 (『Troubleshooting and support』セクション) を参照してください。次の機能が、役に立つ診断情報を提供します。

- CICS が提供するインターフェースを使用することなく、JVM の内部トレース機能を直接使用できます。内部トレース機能の制御と、各種宛先への JVM トレース情報の出力に使用できるシステム・プロパティーに関する情報は、「*Diagnostics Guide*」に記載されています。これらのシステム・プロパティーを使用すると、JVM 内の任意のメソッドまたはクラスからトレースを出力し、メソッド呼び出しで任意のパラメーターと戻りの型の値を検出することができます。
- JVM にメモリー・リークが検出される場合、JVM にヒープ・ダンプを要求できます。ヒープ・ダンプは、JVM のヒープ内にあるすべてのライブ・オブジェクト (引き続き使用中のオブジェクト) のダンプを生成します。また、IBM Health Center や Memory Analyzer ツールを使用して、メモリー・リークを分析することもできます。これらのツールはどちらも、IBM Support Assistant で入手可能です。Java ツールについて詳しくは、IBM Monitoring and Diagnostic Tools for Java - Health Centerを参照してください。
- IBM 64-bit SDK for z/OS, Java テクノロジー・エディション に付属の HPROF プロファイラーは、JVM で実行されるアプリケーションのパフォーマンス情報を提供します。したがって、プログラムのどの部分が最大のメモリーまたはプロセッサー時間を使用しているかが分かります。
- JVM は、モニター、プロファイル作成、および RAS (信頼性・可用性・保守性) 用のインターフェースを提供します。

CICS 環境に固有ではなく、IBM JVM に使用可能なすべてのインターフェース、オプション、またはシステム・プロパティーでは、IBM JVM 資料を第一の情報源として使用してください。

## JVM stdout、stderr、JVMTRACE、およびダンプ出力の場所の制御

JVM サーバーで実行される Java アプリケーションからの出力は、z/OS UNIX ファイルに書き込むことができます。 JVM サーバーの STDOUT、STDERR、および JVMTRACE オプションで z/OS UNIX ファイルを指定します。指定しなければ、JES ログに転送されます。

デフォルトでは、JVM サーバーで実行される Java アプリケーションからの出力は、z/OS UNIX ファイル・システムに書き込まれます。 z/OS UNIX ファイル・システムは、`$WORK_DIR/APPLID/JVMSERVER` のディレクトリー構造内でファイル名規則 `DATE.TIME.<dfhjvmxxx>` に従います。これらのオーバーライドを使用して、出力を JES ログに転送することもできます。詳しくは、DD ステートメントを使用した JVM サーバー出力の JES への転送 を参照してください。

デフォルトをオーバーライドする場合は、`STDOUT`、`STDERR`、および `JVMTRACE` オプションに zFS ファイル名を指定します。ただし、固定のファイル名を使用した場合、その JVM プロファイルを使用して作成されたすべての JVM の出力が同じファイルに追加されます。さまざまな JVM からの出力が、レコード・ヘッダーなしで混ざり合うことになります。この状態では、問題判別の役に立ちません。

これらの値をカスタマイズする場合は、STDOUT、STDERR、および JVMTRACE オプションに可変ファイル名を指定することをお勧めします。CICS 領域の存続期間中、個々の JVM ごとに固有のファイルにすることができます。さらに、追加の識別情報を含めることもできます。

- *APPLID* シンボルを使用して、ファイル名に CICS 領域の APPLID を含めることができます。

ファイル名に含められるその他の識別情報としては、*DATE* シンボルと *TIME* シンボルがあります。

*DATE* は、JVM サーバー始動時にプロファイルで解析された日付 (形式は Dyymmdd) に置換されます。

*TIME* は、JVM サーバー始動時にプロファイルで解析された時刻 (形式は Thhmmss) に置換されます。

*JVMSERVER* は、JVMSERVER リソースの名前に置換されます。

さらなるカスタマイズを、USEROUTPUTCLASS オプションを使用してプログラムのレベルで実施できますが、これは Liberty では機能しません。JVM プロファイルに指定する USEROUTPUTCLASS オプションには、Java クラスを指定します。Java クラスは、JVM からの出力を代行受信して、カスタム・ロケーション (CICS 一時データ・キューなど) にリダイレクトします。出力レコードにタイム・スタンプとヘッダーを追加し、JVM で実行中の個々のトランザクションからの出力を識別することができます。CICS には、これらのタスクを実行するサンプル・クラスが用意されています。

JVM から出力される JAVADUMP ファイルの場所は、JVM プロファイルの WORK_DIR オプションで指定した、z/OS UNIX 上の作業ディレクトリーです。JAVADUMP ファイルは、それらの名前にあるタイム・スタンプで一意的に識別されます。これらのファイルの名前をカスタマイズすることはできません。より詳細な Java TDUMP は、JAVA_DUMP_TDUMP_PATTERN オプションで指定したファイルに書き込まれます。CICS に付属のサンプル JVM プロファイルで示されているように、この値で *APPLID*、*DATE*、*TIME*、および *JVMSERVER* シンボルを使用すると、JVM ごとに固有の名前にすることができます。

JVM は、JAVADUMP または TDUMP を生成すると、情報をその stderr ストリームに書き込みます。JAVADUMP ファイルと TDUMP ファイルの内容の詳細については、IBM SDK for z/OS, Java Technology Edition バージョン 7 (『Troubleshooting and support』セクション)を参照してください。

259 ページの『DD ステートメントを使用した JVM サーバー出力の JES への転送』
出力を特定の場所にリダイレクトするように JVM サーバーを更新できます。

259 ページの『JVM stdout および stderr ストリームのリダイレクト』
アプリケーション開発時に、開発者は USEROUTPUTCLASS オプションを使用して、CICS 領域で開発者固有の stdout 項目および stderr 項目を分離し、開発者が選択した識別可能な宛先に送信できます。Java クラスを使用して出力をリダイレクトし、出力レコードにタイム・スタンプとヘッダーを追加できます。ただし、この方法ではダンプ出力を代行受信することはできません。

262 ページの『Java ダンプ・オプションの制御』
JVM にダンプ・オプションを指定するには、JVM プロファイルで -Xdump オプションを使用できます。

## DD ステートメントを使用した JVM サーバー出力の JES への転送

出力を特定の場所にリダイレクトするように JVM サーバーを更新できます。

JVM サーバーの STDOUT、STDERR、および JVMTRACE 出力を JES ログに転送できます。これによって、JVM サーバーのログ・ファイル出力を MSGUSR といった他の CICS ログと共に管理できるようになります。

JOBLOG パラメーターを使用する場合、STDOUT と JVMTRACE は、SYSPRINT が定義されていれば SYSPRINT に、定義されていなければ動的な SYSnnn に転送されます。STDERR は、SYSOUT が定義されていれば SYSOUT に、定義されていなければ動的な SYSnnn に転送されます。以下に例を示します。

```
STDOUT=JOBLOG
STDERR=JOBLOG
JVMTRACE=JOBLOG
```

JES に定義した任意の MVS データ定義 (DD) に出力を転送することもできます。例えば、以下のように CICS 領域 JCL に DD ステートメント JVMOUT と JVMERR を指定した場合

```
//JVMOUT   DD SYSOUT=*
//JVMERR   DD SYSOUT=*
```

JVM プロファイルで以下の JVM プロファイル・オプションを使用して、STDOUT ストリームと STDERR ストリームを JVMOUT と JVMERR の宛先に転送できます。

```
STDOUT=//DD:JVMOUT
STDERR=//DD:JVMERR
```

JVM サーバー出力の発信元を明示するために、JES に転送されるすべての STDOUT および STDERR 項目は、接頭辞文字列として JVM サーバー名を付けて書き込まれます。複数の JVM サーバーが同じ宛先を共用している場合に、これは便利です。この動作を無効にするには、JVM プロファイル・オプション IDENTITY_PREFIX を使用します。このオプションを FALSE に設定すると、接頭辞文字列は使用されなくなります。

宛先を指定しない場合、出力は zFS のデフォルト・ファイルにリダイレクトされますが、特定の zFS ファイルに送信されるように設定することも可能です。257 ページの『JVM stdout、stderr、JVMTRACE、およびダンプ出力の場所の制御』を参照してください。

## JVM stdout および stderr ストリームのリダイレクト

アプリケーション開発時に、開発者は USEROUTPUTCLASS オプションを使用して、CICS 領域で開発者固有の stdout 項目および stderr 項目を分離し、開発者が選択した識別可能な宛先に送信できます。Java クラスを使用して出力をリダイレクト

し、出力レコードにタイム・スタンプとヘッダーを追加できます。ただし、この方法ではダンプ出力を代行受信することはできません。

USEROUTPUTCLASS オプションを指定すると、JVM のパフォーマンスに悪影響が出ます。実稼働環境で最高のパフォーマンスを発揮するには、このオプションは使用しないでください。

アプリケーションまたはシステム・コードのどちらかによって System.out() または System.err() に書き込まれた出力は、出力リダイレクト・クラスによってリダイレクトできます。ただし、JVM から発行されるいくつかのメッセージには、JVM プロファイルの STDOUT オプションと STDERR オプションで指定した z/OS UNIX ファイルが使用されます。また、これらのファイルは、USEROUTPUTCLASS オプションで指定されたクラスが目的の宛先にデータを書き込めない場合にも使用されます。したがって、これらのファイルに適切なファイル名を指定する必要があります。

USEROUTPUTCLASS オプションを使用するには、適当な Java クラスの名前を指定して、JVM プロファイルで USEROUTPUTCLASS=[java class] を指定してください。このクラスは java.io.OutputStream を拡張します。提供されたサンプル JVM プロファイルには、コメント化されたオプション USEROUTPUTCLASS=com.ibm.cics.samples.SJMergedStream が含まれています。このオプションは、提供されたサンプル・クラスを指定します。com.ibm.cics.samples.SJMergedStream クラスを使用して JVM からの出力をそのプロファイルで処理するには、このオプションをアンコメントしてください。また、CICS は代わりのサンプル Java クラス com.ibm.cics.samples.SJTaskStream も提供します。

JVM サーバーの場合、OSGi フレームワークで出力リダイレクト・クラスを実行するには、そのクラスを OSGi バンドルとしてパッケージ化します。詳しくは、204 ページの『JVM stdout および stderr 出力をリダイレクトするための Java クラスの作成』を参照してください。

**注:** 出力リダイレクト・サンプルは、OSGi およびクラスパス JVM サーバーで機能するものであり、Liberty JVM サーバーでは機能しません。

『サンプル・クラス com.ibm.cics.samples.SJMergedStream および com.ibm.cics.samples.SJTaskStream』
CICS 要求を発行できる Java アプリケーション・スレッドの場合、JVM からの出力を代行受信し、その出力を一時データ・キューに書き込むことができます。JVM のアクティビティーと CICS のアクティビティーを相互に関連付けるログが生成されます。

## サンプル・クラス com.ibm.cics.samples.SJMergedStream および com.ibm.cics.samples.SJTaskStream

CICS 要求を発行できる Java アプリケーション・スレッドの場合、JVM からの出力を代行受信し、その出力を一時データ・キューに書き込むことができます。JVM のアクティビティーと CICS のアクティビティーを相互に関連付けるログが生成されます。

出力を代行受信するときに、タイム・スタンプ、タスク ID とトランザクション ID、およびプログラム名を追加できます。したがって、複数の JVM からの出力が

含まれる、マージされたログ・ファイルを作成することが可能です。このログ・ファイルを使用すると、JVM のアクティビティーを CICS のアクティビティーと相互に関連付けることができます。サンプル・クラス com.ibm.cics.samples.SJMergedStream は、マージされたログ・ファイルを作成するようにセットアップされています。

com.ibm.cics.samples.SJMergedStream クラスは、JVM からの出力を一時データ・キュー CSJO (stdout ストリームの場合) および CSJE (stderr ストリームおよび内部メッセージの場合) に送信します。これらの一時データ・キューは、グループ DFHDCTG で提供され、CSSL にリダイレクトされますが、必要に応じて再定義することができます。

このクラスは、出力のリダイレクトによって、日付、時刻、APPLID、TRANSID、タスク番号、およびプログラム名を含むヘッダーを各レコードに追加します。その結果、JVM 出力用とエラー・メッセージ用に 2 つのマージされたログ・ファイルが作成されます。これらのログ・ファイルでは、出力とメッセージの送信元を容易に特定できます。

これらのクラスは、/usr/lpp/cicsts/cicsts53/lib ディレクトリーにあるファイル com.ibm.cics.samples.jar に出荷時に入っています。ここで、/usr/lpp/cicsts/cicsts53 は、z/OS UNIX 上の CICS ファイルのインストール・ディレクトリーです。これらのクラスのソースもサンプルとして提供されているので、必要に応じてそれらのクラスを変更するか、サンプルに基づいて独自のクラスを作成することができます。これらのクラスは OSGi バンドル JAR としてパッケージされます。これらのクラスは、CLASSPATH JVM サーバーにデプロイするか、または OSGi JVM サーバーに OSGI_BUNDLES JVM サーバー・オプションを使用したミドルウェア・バンドルとしてデプロイすることができます。詳しくは、204 ページの『JVM stdout および stderr 出力をリダイレクトするための Java クラスの作成』を参照してください。

CICS によって接続されたスレッド以外のスレッドで実行される Java アプリケーションは、CICS 要求を発行できません。JVM からの出力は、CICS 機能を使用してリダイレクトできません。com.ibm.cics.samples.SJMergedStream クラスは、引き続き出力を代行受信し、各レコードにヘッダーを追加します。この出力は、前述のとおり、z/OS UNIX ファイル */work_dir/applid*/stdout/CSJO および */work_dir/applid*/stderr/CSJE に書き込まれます。これらのファイルが使用不可である場合、出力は、JVM プロファイルの STDOUT および STDERR オプションで指定された z/OS UNIX ファイルに書き込まれます。

JVM 出力用にマージされたログ・ファイルを作成する代わりに、単一のタスクからの出力を z/OS UNIX ファイルに送信することもできます。また、タイム・スタンプとヘッダーを追加して、単一のタスクに固有の出力ストリームを提供できます。CICS に付属のサンプル・クラス com.ibm.cics.samples.SJTaskStream は、この目的のためにセットアップされています。このクラスは、各タスクの出力を 2 つの z/OS UNIX ファイルに送信します。一方は、stdout ストリーム用、もう一方は stderr ストリーム用です。これらのストリーム内の各出力項目には、タスク番号を使用して固有の名前が付けられます (YYYYMMDD.task.*tasknumber* という形式)。これらの z/OS UNIX ファイルは、STDOUT および STDERR というディレクトリーにそれぞれ

保管されます。このプロセスは、CICS によって接続されたスレッドで実行される Java アプリケーションでも、その他のスレッドで実行される Java アプリケーションでも同じです。

### エラー処理

JVM から出されるメッセージの長さは可変です。CSSL キューの最大レコード長 (133 バイト) では、受信したメッセージによっては格納できない可能性があります。キューの最大レコード長を超えるメッセージを受信した場合、このサンプルの出力リダイレクト・クラスはエラー・メッセージを発行します。メッセージのテキストが影響を受ける可能性があります。

133 バイトより長いメッセージを JVM から受信することが分かった場合には、CSJO と CSJE を別々の一時データ・キューとして再定義します。それらのキューを区画外宛先にして、キューのレコード長を増やしてください。そのキューを物理データ・セットまたはシステム出力データ・セットに割り振ることができます。この場合、システム出力データ・セットの方が便利な場合があります。出力を表示するためにキューをクローズする必要がないためです。一時データ・キューの定義方法については、『Reference』の『TDQUEUE リソース』-> System definitionを参照してください。CSJO と CSJE を再定義する場合は、グループ DFHDCTG で定義される一時データ・キューと同じように、コールド・スタート時にできるだけ早くそれらのキューがインストールされるようにしてください。

一時データ・キュー CSJO および CSJE にアクセスできない場合、出力は、z/OS UNIX ファイル */work_dir/applid*/stdout/CSJO および */work_dir/applid*/stderr/CSJE に書き込まれます。ここで、*work_dir* は JVM プロファイルの WORK_DIR オプションで指定されたディレクトリー、*applid* は CICS 領域に関連付けられたアプリケーション ID です。これらのファイルが使用不可である場合、出力は、JVM プロファイルの STDOUT および STDERR オプションで指定された z/OS UNIX ファイルに書き込まれます。

サンプルの出力リダイレクト・クラスでエラーが検出された場合、1 つ以上のエラー・メッセージが発行されます。出力メッセージの処理中にエラーが発生した場合、エラー・メッセージは System.err に送信され、リダイレクトの対象になります。ただし、エラー・メッセージの処理中にエラーが発生した場合は、JVM プロファイルの STDERR オプションで指定されたファイルに、新しいエラー・メッセージが送信され、Java クラスでの再帰的ループが回避されます。これらのクラスは、呼び出し側の Java プログラムに例外を戻しません。

## Java ダンプ・オプションの制御

JVM にダンプ・オプションを指定するには、JVM プロファイルで -Xdump オプションを使用できます。

Java ダンプ・オプションに関する情報は、IBM SDK for z/OS, Java Technology Edition バージョン 7 (『Troubleshooting and support』セクション) に記載されています。

# JVM サーバーの OSGi ログ・ファイルの管理

OSGi フレームワークは、JVM サーバーの作業ディレクトリーにある 1 組のログ・ファイルにエラーを書き込みます。ご使用の環境にデフォルトが適切でない場合は、JVM サーバーごとにログ・ファイルの数とサイズを管理できます。

## このタスクについて

OSGi フレームワークは、zFS の *$WORK_DIR*/*applid*/*jvmserver*/configuration ディレクトリーにあるログ・ファイルにエラーを書き込みます。ここで、*$WORK_DIR* は JVM サーバーの作業ディレクトリー、*applid* は CICS APPLID、*jvmserver* は JVMSERVER リソースの名前です。JVM プロファイルに *$WORK_DIR* の値を設定しない場合、ログは CICS 領域の z/OS UNIX ホーム・ディレクトリーに出力されます。

OSGi フレームワークは、サイズが 1000 KB に達するまでログ・ファイルに書き込み続けます。その後、OSGi フレームワークは、さらにエラー・メッセージを書き出すために別のログ・ファイルを作成します。そのディレクトリーに最大 10 個のログ・ファイルを保持できます。10 個目のログ・ファイルが満杯になると、OSGi フレームワークは最も古いログ・ファイルに上書きします。したがって、各 JVM サーバーでは、zFS のログ・ファイルに最大 10,000 KB のストレージを割り振ることができます。

ファイル数とストレージ使用量を増減するために、OSGi フレームワークで使用されるログ・ファイルの数とサイズを変更するオプションを JVM プロファイルに追加できます。

## 手順

- ログ・ファイルの最大数を変更するには、**eclipse.log.backup.max** パラメーターを JVM プロファイルに追加します。
- 各ログ・ファイルの最大サイズを変更するには、**eclipse.log.size.max** パラメーターを JVM プロファイルに追加します。

## 例

次の例は、2 つのパラメーターが指定された JVM プロファイルを示しています。この例では、OSGi フレームワークは最大 5 つのログ・ファイルを使用でき、各ログ・ファイルの最大サイズは 500 KB です。

```
#Parameters to control the number and size of OSGi logs
#
-Declipse.log.backup.max=5
-Declipse.log.size.max=500
#
#
```

# JVM サーバーの CICS コンポーネント・トレース

Java によって作成されるログに加えて、CICS は、SJ (JVM) および AP ドメインで、トレース・レベル 0、1、および 2 において、いくつかの標準トレース・ポイントを提供します。これらのトレース・ポイントは、CICS が JVM サーバーのセットアップと管理を行う際に取るアクションをトレースします。

CETR トランザクションを使用して、SJ および AP ドメイン・トレース・ポイントをレベル 0、1 および 2 で活動化できます。SJ ドメインのすべての標準トレース・ポイントの詳細については、『Reference』->『Diagnostics』の『JVM ドメインのトレース・ポイント』を参照してください。

### SJ および AP コンポーネント・トレース

SJ コンポーネントは、SJ ドメインでの例外および処理をトレースし、内部トレース・テーブルに書き込みます。AP コンポーネントは OSGi フレームワークの OSGi バンドルのインストールをトレースします。SJ のレベル 3 とレベル 4 のトレースは、Java ログを作成し、zFS のトレース・ファイルに書き込みます。トレース・ファイルの名前と場所は、JVM プロファイル内の `JVMTRACE` オプションによって決まります。

SJ のレベル 4 トレースは、トレース・ファイルの中に詳細なロギング情報を作成します。このトレース・レベルを使用する場合、zFS 内にファイル用の十分なスペースがあることを確認する必要があります。トレースの起動および管理について詳しくは、『JVM サーバーのトレースの活動化と管理』を参照してください。

## JVM サーバーのトレースの活動化と管理

SJ および AP コンポーネントのトレースをオンにすると、JVM サーバーのトレースを活動化できます。少量のトレースが内部トレース・テーブルに書き込まれますが、Java はまた、ロギング情報を JVM サーバーごとに zFS の固有のファイルに書き込みます。このファイルはラップしないので、そのサイズを zFS で管理する必要があります。

### このタスクについて

JVM サーバーのトレースは、補助トレースも GTF トレースも使用しません。CICS は、内部トレース・テーブルに一部の情報を書き込みます。しかし、ほとんどの診断情報は Java によって記録され、zFS 内のファイルに書き込まれます。このファイルには、各 JVM サーバーに由来する一意的な名前が付けられます。デフォルトのファイル名の形式は *&DATE;.&TIME;*.dfhjvmtrc です。JVMSERVER リソースを使用可能にすると、このファイルが `$WORK_DIR/&APPLID;/&JVMSERVER;` ディレクトリーに CICS によって作成されます。JVM プロファイルでトレース・ファイルの名前と場所を変更できます。JVM サーバーの実行時にトレース・ファイルを削除または名前変更すると、CICS はそのファイルを再作成せず、ロギング情報が別のファイルに書き込まれることもありません。

### 手順

1. CETR トランザクションを使用して、JVM サーバーのトレースを活動化します。 JVM サーバーのトレースおよびロギングの情報を作成するには、2 つのコンポーネントを使用できます。
   - JVM サーバーを始動および停止するために CICS が行うアクションをトレースするには、SJ コンポーネントを選択します。JVM は、診断情報を zFS ファイルに記録します。

- OSGi バンドルのインストールをトレースするには、AP コンポーネントを選択します。

2. SJ および AP コンポーネントのトレース・レベルを設定します。
   - SJ のレベル 0 は、例外のみのトレースを作成します。例えば、JVM サーバーの初期化時のエラーや、OSGi フレームワーク内の問題です。SJ のレベル 1 とレベル 2 は、SJ ドメインからさらに CICS トレースを作成します。このトレースは、内部トレース・テーブルに書き込まれます。
   - SJ のレベル 3 は、JVM から追加のログを作成します。例えば、OSGi フレームワーク内の警告メッセージや情報メッセージです。この情報は、zFS のトレース・ファイルに書き込まれます。
   - SJ のレベル 4 および AP のレベル 2 は、CICS および JVM からデバッグ情報を作成します。これは、JVM サーバー処理に関するより詳細な情報を提供します。この情報は、zFS のトレース・ファイルに書き込まれます。
3. 各トレース項目には、日時のタイム・スタンプがあります。JVMTRACE プロファイル・オプションを使用して、このトレース・ファイルの名前と場所を変更できます。
4. デフォルトの JVMTRACE 設定を使用している場合、JVMSERVER リソースを有効にすると、CICS は JVM の存続期間にわたって新しい固有のトレース・ファイルを作成します。 JVMSERVER リソースを無効にすると、トレース・ファイルを削除することができます。情報を別個に保持する場合はファイルの名前を変更することができます。
5. ファイルの数を管理するには、LOG_FILES_MAX オプションを設定することで、JVM サーバー始動時に保持される古いトレース・ファイルの数を制御できます。

## Java アプリケーションのデバッグ

CICS における JVM は、Java 2 Platform で提供される標準デバッグ・メカニズムである Java Platform Debugger Architecture (JPDA) をサポートします。このアーキテクチャーは、リモート・デバッガーと JVM との接続を可能にする 1 組の API を提供します。

### このタスクについて

JDPA をサポートする任意のツールを使用して、CICS で実行される Java アプリケーションをデバッグすることができます。例えば、z/OS 上の Java SDK に付属の Java Debugger (JDB) を使用できます。JPDA リモート・デバッガーを接続するには、JVM プロファイルでいくつかのオプションを設定する必要があります。

IBM は、Health Center を含む、Java 用のモニターおよび診断ツールを提供しています。 IBM Health Center は、IBM Support Assistant Workbench で使用できます。これらの無料のツールは、Getting Started guide に記載されているとおりに IBM からダウンロードできます。

### 手順

1. 以下のように、デバッグ・オプションを JVM プロファイルに追加して、JVM をデバッグ・モードで開始します。

```
-agentlib:jdwp=transport=dt_socket,server=y,address=port,suspend=n
```

リモートでデバッガーに接続するための空きポートを選択します。 JVM プロファイルが複数の JVM サーバーで共用される場合、別の JVM プロファイルをデバッグに使用できます。

**注:** 中断のデフォルト値は `y` ですが、これにより、デバッグ・エージェントの接続試行時に JVM サーバーがロックされる可能性があります。 `n` の中断値を指定して、JVM サーバーがロックされないようにしてください。

2. Liberty JVM サーバーのデバッグを行うときに、以下のプロパティーを JVM プロファイルに追加します。

```
-Dwas.debug.mode=true
-Dcom.ibm.websphere.ras.inject.at.transform=true
```

3. デバッガーを JVM に接続します。 接続中にエラーが発生する場合 (ポート値が間違っている場合など)、JVM の標準出力ストリームと標準エラー・ストリームにメッセージが書き込まれます。
4. デバッガーを使用して、JVM の初期状態を確認します。例えば、開始されたスレッドの ID やロードされたシステム・クラスを確認します。 JVM は実行を中断します。Java アプリケーションは開始していません。
5. 完全 Java クラス名とソース・コード行番号を指定して、Java アプリケーションの適切なポイントでブレークポイントを設定します。アプリケーション・クラスは通常、まだロードされていないので、デバッガーは、クラスがロードされるまでこのブレークポイントの活動化が延期されることを示します。 JVM を CICS ミドルウェア・コードを使用してアプリケーションのブレークポイントまで実行させてください。このポイントで JVM は再度実行を中断します。
6. ロードされたクラスと変数を調べ、追加のブレークポイントを設定して、必要に応じてコードをステップスルーします。
7. デバッグ・セッションを終了します。アプリケーションを最後まで実行させることができます。その時点でデバッガーと CICS JVM 間の接続はクローズします。一部のデバッガーは、JVM の強制終了をサポートします。その結果、異常終了し、CICS システム・コンソールにエラー・メッセージが表示されます。

## CICS JVM プラグイン・メカニズム

JVM の標準 JPDA デバッグ・インターフェースに加えて、CICS Java ミドルウェアには CICS 提供の 1 組の代行受信ポイント (プラグイン) があります。これは、アプリケーションのデバッグに役立つ場合があります。これらのプラグインを使用して、アプリケーション Java コードの実行直前と直後に追加の Java プログラムを挿入できます。

アプリケーションに関する情報 (例えば、クラス名やメソッド名) はプラグインで使用できます。プラグインは JCICS API を使用して、アプリケーションに関する情報を取得することもでき、また、標準 JPDA インターフェースと併用して、CICS 専用の追加デバッグ機能を提供することもできます。プラグインは、CICS ユーザー出口と同様に、デバッグ以外の目的にも使用できます。

Java 出口は、Java プログラムが呼び出される直前と直後に呼び出されるメソッドを提供する CICS Java ラッパー・プラグインです。

プラグインをデプロイするには、そのプラグインを OSGi バンドルとしてパッケージ化します。詳しくは、182 ページの『JVM サーバーにおける OSGi バンドルのデプロイ』を参照してください。

2 つの Java プログラミング・インターフェースが提供されます。

両方のインターフェースが com.ibm.cics.server.jar で提供され、Javadoc で文書化されます。詳しくは、43 ページの『CICS 用 Java クラス・ライブラリー (JCICS)』を参照してください。

Java プログラミング・インターフェースは次のとおりです。

- DebugControl: com.ibm.cics.server.debug.DebugControl。このプログラミング・インターフェースは、ユーザーによって提供される実装に対して行うことができるメソッド呼び出しを定義します。
- Plugin: com.ibm.cics.server.debug.Plugin。これは、プラグイン実装の登録に使用する汎用プログラミング・インターフェースです。

以下は DebugControl インターフェースの例です。

```
public interface DebugControl
{
    // called before an application object method or program main is invoked
    public void startDebug(java.lang.String className,java.lang.String methodName);

    // called after an application object method or program main is invoked
    public void stopDebug(java.lang.String className,java.lang.String methodName);

    // called before an application object is deleted
    public void exitDebug();

}
public interface Plugin
{
    // initaliser, called when plug-in is registered
    public void init();
}
```

以下は DebugControl および Plugin インターフェースの実装例です。

```
import com.ibm.cics.server.debug.*;

public class SampleCICSDebugPlugin
    implements Plugin, DebugControl
{
    // Implementation of the plug-in initialiser
    public void init()
    {
        // This method is called when the CICS Java middleware loads and
        // registers the plug-in. It can be used to perform any initialization
        // required for the debug control implementation.
    }

    // Implementations of the debug control methods
    public void startDebug(java.lang.String className,java.lang.String methodName)
    {
        // This method is called immediately before the application method is
        // invoked. It can be used to start operation of a debugging tool. JCICS
        // calls such as Task.getTask can be used here to obtain further
        // information about the application.
    }

    public void stopDebug(java.lang.String className,java.lang.String methodName)
```

```
    {
        // This method is called immediately after the application method is
        // invoked. It can be used to suspend operation of a debugging tool.
    }

    public void exitDebug()
    {
        // This method is called immediately before an application object is
        // deleted. It can be used to terminate operation of a debugging tool.
    }

    public static void main(com.ibm.cics.server.CommAreaHolder ca)
    {
    }
}
```

# 第 9 章 Liberty JVM サーバーおよび Java Web アプリケーションのトラブルシューティング

Java Web アプリケーションで問題が生じた場合、CICS および Liberty プロファイルで提供される診断を使用して、問題の原因を判別することができます。

CICS は、Liberty JVM サーバーで実行中の Java Web アプリケーションに関連した問題の診断に役立つ統計、メッセージ、およびトレースを提供します。Web アプリケーションを実行するために使用される Liberty プロファイル・テクノロジーも、zFS 内に使用可能な診断を生成します。一般的なセットアップ・エラーおよびアプリケーションの問題については、251 ページの『第 8 章 Java アプリケーションのトラブルシューティング』を参照してください。

## 問題の回避

CICS は、領域 APPLID と JVMSERVER リソース名の値を使用して、zFS ファイル・システム上に固有のファイル名とディレクトリー名を作成します。許容文字の一部は、UNIX システム・サービス・シェルで特殊な意味を持ちます。例えば、ドル記号 ($) は環境変数名の先頭を意味します。領域 APPLID と JVM サーバー名では、非英数字は使用しないでください。 UNIX システム・サービス・シェルでこれらの文字を使用する場合、エスケープ文字として円記号 (¥) を使用しなければならない場合があります。例えば、JVM サーバー MY$JVMS を呼び出す場合には、次のようにします。

```
cat CICSPRD.MY\$JVMS.D20140319.T124122.dfhjvmout
```

## Liberty JVM サーバーを始動できない

1. Liberty JVM サーバーを始動できない場合は、セットアップが正しいことを確認します。問題の原因として考えられる要因を判別するには、`WLP_OUTPUT_DIR` の下にある Liberty `messages.log` ファイルの CICS メッセージとエラーを使用します。
2. JVM プロファイルの中の **`-Dfile.encoding`** JVM プロパティーが、ISO-8859-1 または UTF-8 のどちらかを指定していることを確認します。これらの 2 つのコード・ページは、Liberty プロファイル・サーバーでサポートされています。それ以外の値に設定すると、JVM サーバーは始動できません。

## dropins ディレクトリーにデプロイした後、Web アプリケーションが利用不可になる

1. `dfhjvmerr` で CWWK0221E エラー・メッセージを受け取っている場合、JVM プロファイルと `server.xml` でホスト名とポート番号に正しい値を設定していることを確認します。このエラー・メッセージは、正しいポート番号またはホスト名が設定されていないことを意味します。ホスト名が無効であるか、ポート番号が使用中である可能性があります。

## Liberty JVM サーバーを使用可能にした後、CICS CPU が増大する

1. Liberty JVM サーバーが dropins ディレクトリーをあまりにも頻繁にスキャンすると、過剰な入出力および CPU 使用量が発生します。Liberty JVM サーバーがアプリケーションの dropins ディレクトリーをスキャンする頻度は、構成可能です。構成テンプレートに指定されるデフォルトの間隔は 5 秒ですが、この値を増やすことも、あるいはアプリケーションのスキャンを無効にすることもできます。

   この問題を修正するには、server.xml ファイルの中の次の XML を編集します。

```
<config monitorInterval="5s" updateTrigger="polled"/>
<applicationMonitor updateTrigger="disabled" pollingRate="5s" dropins="dropins" dropinsEnabled="false"/>
```

   CICS バンドルの中に Web アプリケーションをインストールした場合、<config> エレメントの中のポーリングは無効にせず、アプリケーションのスキャンを無効にしてください。この設定の編集方法について詳しくは、WebSphere Application Server for z/OS 8.5.5 の『動的更新の制御』を参照してください。

## アプリケーションが使用可能にならない

1. WAR ファイルを dropins ディレクトリーにコピーしても、アプリケーションが使用可能にならないという問題が発生することがあります。Liberty messages.log ファイルの中でエラー・メッセージを確認してください。CWWKZ0013E エラー・メッセージを受け取っている場合、既に同じ名前の Web アプリケーションが Liberty JVM サーバーで実行されています。この問題を修正するには、Web アプリケーションの名前を変更してから、dropins ディレクトリーにデプロイします。

## Web アプリケーションが認証を要求しない

セキュリティーを構成しましたが、Web アプリケーションが認証を要求していません。

1. Web アプリケーションに対して CICS セキュリティーを構成できますが、Web アプリケーションは、WAR ファイルの中にセキュリティーの制限が含まれる場合に限り、セキュリティーを使用します。アプリケーション開発者によって、セキュリティーの抑制が動的 Web プロジェクトの web.xml ファイル内に定義されたことを確認します。
2. server.xml ファイルに、正しいセキュリティー構成情報が含まれていることを確認します。すべての構成エラーは dfhjvmerr に報告され、有用な情報がいくつか提供されていることがあります。CICS セキュリティーを使用している場合、CICS セキュリティー機能が指定されていることを確認してください。CICS セキュリティーのスイッチがオフの場合、アプリケーション・ユーザーを認証するための基本ユーザー・レジストリーを指定したことを確認します。
3. server.xml が、EJBRoles を利用するように <safAuthorization> 用に構成されているか、または <application-bnd> エレメントでローカル・ロール・マッピング用に構成されているかを確認します。 <application-bnd> エレメントは、server.xml または installedApps.xml の <application> エレメントにあります。ローカル・ロール・マッピング用に CICS によって追加されたデフォルト・セキュリティー・ロールは「cicsAllAuthenticated」です。

## Web アプリケーションが HTTP 403 エラー・コードを返す

Web アプリケーションは、Web ブラウザーに HTTP 403 エラー・コードを返します。HTTP 403 許可障害を受け取る場合は、使用しているユーザー ID が取り消されているか、アプリケーション・トランザクションの実行を許可されていないかのどちらかです。

1. CICS メッセージ・ログのエラー・メッセージ ICH408I を調べて、どの種類の許可障害が発生したかを確認します。問題を解決するには、ユーザー ID のパスワードが正しいことと、トランザクションの実行が許可されていることを確認します。
2. アプリケーションがクラス com.ibm.ws.webcontainer.util.Base64Decode の例外を返す場合は、`dfhjvmerr` のエラー・メッセージを確認します。構成エラー・メッセージ (例えば、CWWKS4106E または CWWKS4000E) がある場合、サーバーは、異なるエンコード方式で作成された構成ファイルにアクセスしようとしています。このタイプの構成エラーは、**`file.encoding`** 値を変更して JVM サーバーを再始動するときに発生することがあります。この問題を解決するには、以前のエンコード方式に戻して JVM サーバーを再始動するか、構成ファイルを削除することができます。JVM サーバーは、始動するときに、正しいファイル・エンコード方式でファイルを再作成します。

## Web アプリケーションが HTTP 500 エラー・コードを返す

Web アプリケーションは、Web ブラウザーに HTTP 500 エラーを返します。HTTP 500 エラーを受け取る場合、構成エラーが発生しました。

1. CICS メッセージ・ログの DFHSJ メッセージを調べてください。そのメッセージに、このエラーの特定の原因に関する詳細情報が記されていることがあります。
2. URIMAP を使用して特定のトランザクションのアプリケーション要求を実行している場合、URIMAP が正しいトランザクションを使用していることを確認します。
3. URIMAP が正しいトランザクションを使用している場合、SCHEME 属性と USAGE 属性が正しく設定されていることを確認します。SCHEME は、アプリケーション要求 (HTTP または HTTPS のどちらか) と一致している必要があります。USAGE 属性は、JVMSERVER に設定されている必要があります。

## Web アプリケーションが HTTP 503 エラー・コードを返す

Web アプリケーションは、Web ブラウザーに HTTP 503 エラーを返します。HTTP 503 エラーを受け取る場合、アプリケーションは使用不可です。

1. CICS メッセージ・ログの DFHSJ メッセージを調べて、追加情報を確認します。
2. アプリケーションに対して TRANSACTION リソースと URIMAP リソースが使用可能であることを確認します。これらのリソースが CICS バンドルの中にアプリケーションの一部としてパッケージされている場合は、BUNDLE リソースの状況を確認できます。
3. 要求は、完了前に消去された可能性があります。ログの中のエラー・メッセージは、要求が消去された理由を説明しています。

## Web アプリケーションが例外を返す

Web アプリケーションは、Web ブラウザーに例外を返しました。例えば、アプリケーションは、クラス com.ibm.ws.webcontainer.util.Base64Decode の例外を戻しています。

1. `dfhjvmerr` でエラー・メッセージを確認します。
2. 構成エラー・メッセージ (例えば、CWWKS4106E または CWWKS4000E) がある場合、サーバーは、異なるエンコード方式で作成された構成ファイルにアクセスしようとしています。このタイプの構成エラーは、**`file.encoding`** 値を変更して JVM サーバーを再始動するときに発生することがあります。この問題を解決するには、以前のエンコード方式に戻して JVM サーバーを再始動するか、構成ファイルを削除することができます。JVM サーバーは、始動するときに、正しいファイル・エンコード方式でファイルを再作成します。

## エラー・メッセージ CWWKB0109E

Liberty サーバーが正常なシャットダウンに失敗した場合、WLP_ZOS_PLATFORM=TRUE に設定されている次の Liberty JVM サーバーで、エラー・メッセージ CWWKB0109E が `messages.log` ファイルに書き込まれます。このエラーは修正する必要はなく、無視してかまいません。

## productInfo スクリプトを使用して Liberty プロファイルの整合性を検証

CICS のインストール後、またはサービスの適用後に、productInfo スクリプトを使用して Liberty プロファイル・インストールの整合性を検証できます。

1. ディレクトリーを `CICS USSHOME` ディレクトリーに移動します。
2. productInfo では Java を使用するので、PATH に Java が組み込まれていることを確認する必要があります。または、JVM プロファイルで **`JAVA_HOME`** 環境変数を **`JAVA_HOME`** の値に設定します。例えば、次のようにします。

   ```
   export JAVA_HOME=/usr/lpp/java/J7.0_64
   ```

3. 検証オプション `wlp/bin/productInfo validate` を指定して productInfo スクリプトを実行します。エラーが出ないようにしてください。 Liberty プロファイル productInfo スクリプトについて詳しくは、Liberty プロファイルのインストールの整合性を検証を参照してください。

## Liberty コマンドを実行するために wlpenv スクリプトを使用する

IBM サービスから、**`productInfo`** や **`server dump`** などの 1 つ以上の Liberty プロファイル提供コマンドの実行を依頼されることがあります。こうしたコマンドを実行するには、必要な環境を設定するためのラッパーとして wlpenv スクリプトを使用できます。このスクリプトは、JVM プロファイルが正常に構文解析された後に、Liberty JVM サーバーを使用可能にするたびに作成および更新されます。このスクリプトは各 CICS 領域にある JVM サーバーごとに固有であるため、JVM プロファイルとして指定された *`WORK_DIR`* に作成され、*`APPLID.JVMSERVER`*`.wlpenv` という名前になります。ここで、*APPLID* は CICS 領域 APPLID の値で、*JVMSERVER* は JVMSERVER リソースの名前です。

UNIX システム・サービス・シェルで wlpenv スクリプトを実行するには、JVM プロファイルで指定されている *WORK_DIR* にディレクトリーを変更し、引数として Liberty プロファイル・コマンドを指定してスクリプトを実行します。例えば、次のようにします。

```
./wlpenv productInfo version
```

```
./wlpenv server dump --archive=package_file_name.dump.pax --include=heap
```

なお、**server dump** コマンドには、サーバー名を指定しないでください。サーバー名は、JVM サーバーが最後に使用可能になったときに設定されていた値に wlpenv スクリプトによって設定されるからです。

Liberty コマンドについて詳しくは、WebSphere Application Server for z/OS 8.5.5 の『Liberty プロファイル: productInfo コマンド』およびWebSphere Application Server for z/OS 8.5.5 の『コマンド・プロンプトからの Liberty プロファイル・サーバー・ダンプの生成』を参照してください。

# 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。 本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。 これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。 ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。 本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。 実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510
東京都中央区日本橋箱崎町19番21号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

**以下の保証は、国または地域の法律に沿わない場合は、適用されません。**

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書には、技術的に正確でない記述や誤植がある場合があります。 本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。 IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN 本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

**プライバシー・ポリシーに関する考慮事項**

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品 (「ソフトウェア・オファリング」) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項を確認ください。

CICSPlex SM Web ユーザー・インターフェース:

WUI メイン・インターフェース: このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、認証、お客様の利便性の向上、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名、およびその他の個人を識別可能な情報を、セッションごとの Cookie および持続的な Cookie を使用して収集する場合があります。これらの Cookie を無効にすることはできません。

WUI データ・インターフェース: このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、認証、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名、およびその他の個人を識別可能な情報を、セッションごとの Cookie を使用して収集する場合があります。これらの Cookie を無効にすることはできません。

WUI Hello World ページ: このソフトウェア・オファリングは、展開される構成に応じて、個人を識別可能な情報を収集することのないセッションごとの Cookie を使用する場合があります。これらの Cookie を無効にすることはできません。

CICS Explorer: このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、認証、およびシングル・サインオン構成の目的のために、それぞれのお客様のユーザー名とパスワードを、セッションごとおよび持続的な設定を使用して収集する場合があります。ユーザーのパスワードを暗号化した形式でディスク上に保管することは、サインオンの際にチェック・ボックスにチェック・マークを付けるというユーザーによる明示的な操作によってのみ有効にできますが、上述のそれらの設定を無効にすることはできません。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、 お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、『IBM のプライバシー・ポリシー』(http://www.ibm.com/privacy) および 『IBM オンラインでのプライバシー・ステートメント』(http://www.ibm.com/privacy/details) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』および「IBM Software Products and Software-as-a-Service Privacy Statement」(http://www-01.ibm.com/software/info/product-privacy/) を参照してください。

『商標』

## 商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corporation の商標または登録商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、 http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標登録です。

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

# 参考文献

## CICS Transaction Server for z/OS の CICS ブック

### 一般

*CICS Transaction Server for z/OS Program Directory - base*、GI13-3375
*CICS Transaction Server for z/OS Program Directory activation module - base*、GI13-3376
*CICS Transaction Server for z/OS Program Directory activation module - Developer Trial*、GI13-3377
*CICS Transaction Server for z/OS Program Directory activation module - Value Unit Edition*、GI13-3378
*CICS Transaction Server for z/OS 新機能*、GC43-2879
*CICS Transaction Server for z/OS CICS TS バージョン 5.3 へのアップグレード*、GC43-2974
*CICS Transaction Server for z/OS インストール・ガイド*、GC43-2877

### CICS へのアクセス

*CICS インターネット・ガイド*、SC43-2878
*CICS Web サービス・ガイド*、SC43-2876

### 管理

*CICS System Definition Guide*、SC34-7428
*CICS Customization Guide*、SC34-7404
*CICS Resource Definition Guide*、SC34-7425
*CICS Operations and Utilities Guide*、SC34-7420
*CICS RACF Security Guide*、SC34-7423
*CICS Supplied Transactions*、SC34-7427

### プログラミング

*CICS アプリケーション・プログラミング・ガイド*、SC43-2882
*CICS アプリケーション・プログラミング・リファレンス*、SC43-2875
*CICS System Programming Reference*、SC34-7429
*CICS Front End Programming Interface User's Guide*、SC34-7412
*CICS C++ OO Class Libraries*、SC34-7405
*CICS Distributed Transaction Programming Guide*、SC34-7410
*CICS Business Transaction Services*、SC34-7403
*CICS での Java アプリケーション*、SC43-2885

### 診断

*CICS Problem Determination Guide*、GC34-7422
*CICS パフォーマンス・ガイド*、SC43-2881
*CICS Messages and Codes Vol 1*、GC34-7418
*CICS Messages and Codes Vol 2*、GC34-7419
*CICS Diagnosis Reference*、GC34-7409
*CICS Recovery and Restart Guide*、SC34-7424

*CICS Data Areas*、GC34-7406
*CICS Trace Entries*、SC34-7430
*CICS Debugging Tools Interfaces Reference*、GC34-7408

### 通信

*CICS 相互通信ガイド*、SC43-2880
*CICS External Interfaces Guide*、SC34-7411

### データベース

*CICS DB2 Guide*、SC34-7407
*CICS IMS Database Control Guide*、SC34-7413
*CICS Shared Data Tables Guide*、SC34-7426

## CICS Transaction Server for z/OS の CICSPlex SM ブック

### 一般

*CICSPlex SM 概念および計画*、SC43-2883
*CICSPlex SM Web ユーザー・インターフェース・ガイド*、SC43-2884

### 管理

*CICSPlex SM Administration*、SC34-7438
*CICSPlex SM Operations Views Reference*、SC34-7447
*CICSPlex SM Monitor Views Reference*、SC34-7446
*CICSPlex SM Managing Workloads*、SC34-7444
*CICSPlex SM Managing Resource Usage*、SC34-7443
*CICSPlex SM Managing Business Applications*、SC34-7442

### プログラミング

*CICSPlex SM Application Programming Guide*、SC34-7439
*CICSPlex SM Application Programming Reference*、SC34-7440

### 診断

*CICSPlex SM Resource Tables Reference Vol 1*、SC34-7449
*CICSPlex SM Resource Tables Reference Vol 2*、SC34-7450
*CICSPlex SM Messages and Codes*、GC34-7445
*CICSPlex SM Problem Determination*、GC34-7448

## 他の CICS 資料

以下の資料には CICS に関する詳しい情報が含まれますが、これらの資料は CICS Transaction Server for z/OS、バージョン 5 リリース 3 の一部としては提供されません。

*Designing and Programming CICS Applications*、SR23-9692
*CICS Application Migration Aid Guide*、SC33-0768
*CICS ファミリー: API の構成*、SC88-7261
*CICS ファミリー クライアント・サーバー プログラミングの手引き*、SC88-7429
*CICS Family: Interproduct Communication*、SC34-6853
*CICS Family: Communicating from CICS on System/390*、 SC34-6854

*CICS Transaction Gateway (OS/390 版) 管理の手引き*、SD88-7246
*CICS Family: General Information*、GC33-0155
*CICS 4.1 Sample Applications Guide*、SC33-1173
*CICS/ESA 3.3 XRF Guide*、SC33-0661

## 他の IBM 資料

以下の資料には、関連する IBM 製品に関する情報が記載されています。
*IBM Developer Kit and Runtime Environment, Java 2 Technology Edition Diagnostics Guide*、SC34-6358
*Persistent Reusable Java Virtual Machine User's Guide*、SC34-6201

# アクセシビリティー

アクセシビリティー機能は、運動障害または視覚障害など身体に障害を持つユーザーがソフトウェア・プロダクトを快適に使用できるようにサポートします。

CICS システムのセットアップ、実行、および保守に必要なほとんどの作業は、以下のいずれかの方法で行うことができます。

- CICS にログオンした 3270 エミュレーターを使用する
- TSO にログオンした 3270 エミュレーターを使用する
- 3270 エミュレーターを MVS システム・コンソールとして使用する

IBM パーソナル・コミュニケーションズは、身体障害のある方々のためのアクセシビリティー機能を持つ 3270 エミュレーションを提供します。 CICS システムで必要なアクセシビリティー機能を提供するためにこの製品を使用することができます。

# 索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

## [ア行]



## [カ行]



## [サ行]



## [タ行]

## K



## L



## M



## O



## P



## S



## T



## U



## W

# Z



# ［特殊文字］



SC43-2885-00

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21