z/OS용 CICS Transaction Server
버전 5 릴리스 2

CICS의 Java 애플리케이션

SC43-0655-00

z/OS용 CICS Transaction Server
버전 5 릴리스 2

CICS의 Java 애플리케이션

SC43-0655-00

참고

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 먼저 219 페이지의 『주의사항』의 정보를 읽으십시오.

이 개정판은 새 개정판에 별도로 명시하지 않는 한, z/OS용 IBM CICS Transaction Server 버전 5 릴리스 2(제품 번호 5655-Y04) 및 모든 후속 릴리스와 수정에 적용됩니다.

목차

머리글

이 매뉴얼은 고객이 IBM® CICS® Transaction Server 버전 5 릴리스 2의 서비스를 확보하기 위해 프로그램을 쓸 수 있는 의도한 프로그래밍 인터페이스를 문서화합니다.

『이 책의 내용』
이 안내서는 CICS에서 Java™ 애플리케이션과 엔터프라이즈 Bean을 개발 및 사용하는 방법을 알려줍니다.

『대상 독자』

『Information Center의 주제 위치』
이 서적의 주제는 CICS Information Center에서도 찾을 수 있습니다. Information Center는 정보가 표시되는 방식을 구조화시키기 위해 컨텐츠 유형을 사용합니다.

이 책의 내용

이 안내서는 CICS에서 Java 애플리케이션과 엔터프라이즈 Bean을 개발 및 사용하는 방법을 알려줍니다.

대상 독자

이 안내서는 다음 사용자를 대상으로 합니다.

- CICS에 대한 경험이 거의 없고 Java 프로그램을 개발하고 실행하는 데 필요한 것보다 CICS에 대해 많은 정보가 필요하지 않은 전문 Java 애플리케이션 프로그래머
- Java 지원을 위한 CICS 요구사항을 이해해야 하는 전문 CICS 사용자와 시스템 프로그래머

Information Center의 주제 위치

이 서적의 주제는 CICS Information Center에서도 찾을 수 있습니다. Information Center는 정보가 표시되는 방식을 구조화시키기 위해 컨텐츠 유형을 사용합니다.

Information Center 컨텐츠 유형은 일반적으로 업그레이드, 구성, 설치와 같은 태스크 중심적입니다. 다른 컨텐츠 유형으로는 참조, 개요, 시나리오 또는 학습서 기반 정보가 있습니다. 다음 맵핑은 이 서적의 주제와 외부 Information Center에 대한 링크가 있는 Information Center 컨텐츠 유형 간의 관계를 보여줍니다.

표 1. PDF 주제 대 Information Center 컨텐츠 유형 맵핑. 이 표는 Information Center의 컨텐츠 유형 주제와 PDF의 주제 간 관계를 나열합니다.

이 서적의 주제 세트	Information Center의 위치
• 1 페이지의 제 1 장『Java 시작하기』 • 27 페이지의 제 2 장『CICS용 Java 애플리케이션 개발』 • 95 페이지의 제 3 장『Java 지원 설정』 • 139 페이지의 제 4 장『JVM 서버에 애플리케이션 전개』 • 149 페이지의 제 5 장『Java 애플리케이션 관리』 • 163 페이지의 제 6 장『Java 성능 개선』 • 195 페이지의 제 8 장『Java 애플리케이션 문제점 해결』	• 시작하기 • 구성 • 관리 • 애플리케이션 개발 • 전개 • 성능 개선 • 문제점 해결 및 지원

z/OS용 CICS Transaction Server, 버전 5 릴리스 2의 변경사항

이번 릴리스의 변경사항에 대한 정보는 Information Center의 *새로운 기능*, 또는 다음 서적을 참조하십시오.

- *z/OS용 CICS Transaction Server 새로운 기능*
- *CICS TS 버전 5.1에서 z/OS용 CICS Transaction Server 업그레이드*
- *CICS TS 버전 4.2에서 z/OS용 CICS Transaction Server 업그레이드*
- *CICS TS 버전 4.1에서 z/OS용 CICS Transaction Server 업그레이드*
- *CICS TS 버전 3.2에서 z/OS용 CICS Transaction Server 업그레이드*
- *CICS TS 버전 3.1에서 z/OS용 CICS Transaction Server 업그레이드*

릴리스 후 텍스트의 기술 변경사항은 각각의 변경된 또는 새 정보 행의 왼쪽에 새로 막대(|)로 표시됩니다.

제 1 장 Java 시작하기

엔터프라이즈에서 Java를 사용하려는 경우, CICS는 CICS 영역이 제어하는 JVM(Java Virtual Machine)에서 Java 애플리케이션을 개발하고 실행하기 위한 런타임 환경과 도구를 제공합니다. JVM 서버에서 실행되는 Java 워크로드는 zAAP(IBM System z® Application Assist Processor)에서 실행될 수 있습니다.

Java 애플리케이션 개발

OSGi 서비스 플랫폼을 준수하는 재사용 가능한 모듈형 Java 애플리케이션을 작성할 수 있습니다. 이러한 애플리케이션은 CICS와 다른 플랫폼 간에 쉽게 이식할 수 있으며 OSGi는 종속 항목 및 버전 관리에 대한 세분성(granularity)을 제공합니다.

JCICS(Java CICS) API를 사용하여 CICS 서비스(예를 들어, 파일 또는 임시 저장영역 큐에서 읽기)에 액세스하는 애플리케이션을 작성할 수 있습니다. Java 애플리케이션은 다른 CICS 애플리케이션에 링크될 수 있으므로 DB2® 및 IMS™의 데이터에 액세스할 수 있습니다. Java 애플리케이션은 JVM 서버에서 실행됩니다.

Liberty 프로파일과 함께 제공되는 웹 도구를 사용하여 애플리케이션의 웹 프리젠테이션 계층을 작성할 수 있습니다. CICS는 애플리케이션과 동일한 JVM 서버에서 JSP 페이지와 웹 servlet을 실행할 수 있습니다.

Axis2 JVM 서버의 웹 서비스

서비스 제공자와 서비스 요청자가 이기종 환경에서 작업을 수행하도록 Java 웹 서비스를 작성할 수 있습니다. Java 웹 서비스는 JVM 서버에서 실행되며 SOAP 처리는 Liberty JVM 서버의 Apache CXF 또는 JVM 서버의 Apache Axis2 웹 서비스 엔진에서 수행됩니다. 또한 표준 Java API 및 어노테이션을 사용하여 Java 웹 서비스를 작성하고 데이터 변환 핸들 XML을 수행하거나 구조화 데이터 작업을 수행할 수 있습니다.

CICS에 대한 Java 연결

JCA(Java Connector Architecture)를 사용하여 CICS Transaction Gateway를 통해 외부 Java 애플리케이션을 CICS에 연결할 수 있습니다. JCA는 외부 Java 환경에서 CICS 애플리케이션을 호출하기 위한 관련 기술입니다. 이러한 방식으로 호출되는 CICS 애플리케이션은 Java 또는 지원되는 다른 언어로 구현될 수 있습니다.

2 페이지의『CICS의 Java 지원』

CICS는 CICS 영역이 제어하는 JVM(Java Virtual Machine)에서 Java 엔터프라

이즈 애플리케이션을 개발하고 실행하기 위한 런타임 환경과 도구를 제공합니다. Java 애플리케이션은 다른 언어로 작성된 CICS 서비스 및 애플리케이션과 상호작용할 수 있습니다.

16 페이지의 『OSGi를 준수하는 Java 애플리케이션』
CICS에는 JVM 서버에서 OSGi 사양을 준수하는 Java 애플리케이션을 실행하기 위한 OSGi 프레임워크의 Equinox 구현이 포함됩니다.

20 페이지의 『Liberty JVM 서버의 웹 애플리케이션 및 웹 서비스』
CICS는 경량 Java servlet과 JSP(JavaServer Pages)를 실행할 수 있는 웹 컨테이너를 제공합니다. 개발자는 Java servlet 및 JSP 스펙의 다양한 특성을 사용하여 CICS용 최신 웹 애플리케이션을 작성할 수 있습니다. 웹 컨테이너는 JVM 서버에서 실행되며 WebSphere® Application Server Liberty 프로파일 기술을 기반으로 빌드됩니다.

22 페이지의 『Java 웹 서비스』
CICS에는 Java 웹 서비스를 실행하기 위한 Axis2 기술이 포함됩니다. Axis2는 Apache 기반의 개방형 소스 웹 서비스 엔진이며 Java 환경에서 SOAP 메시지를 처리하기 위해 CICS와 함께 제공됩니다.

관련 정보:

JVM 서버 설정
Java 애플리케이션, 웹 애플리케이션, Axis2 또는 CICS 보안 토큰 서비스를 JVM 서버에서 실행하려면 CICS 자원을 설정하고 JVM으로 선택항목을 전달하는 JVM 프로파일을 작성해야 합니다.

CICS의 Java 지원

CICS는 CICS 영역이 제어하는 JVM(Java Virtual Machine)에서 Java 엔터프라이즈 애플리케이션을 개발하고 실행하기 위한 런타임 환경과 도구를 제공합니다. Java 애플리케이션은 다른 언어로 작성된 CICS 서비스 및 애플리케이션과 상호작용할 수 있습니다.

z/OS®의 Java는 Java 애플리케이션 실행을 위한 포괄적인 지원을 제공합니다. CICS는 IBM 64-bit SDK for z/OS, Java Technology Edition, 버전 7 또는 버전 7 릴리스 1을 사용합니다. 두 버전 중 하나를 사용해야 하지만 CICS 영역에서 실행되는 모든 JVM 서버는 동일한 버전을 사용해야 합니다. SDK에는 Java API 세트 전체와 개발 도구 세트를 지원하는 Java Runtime Environment가 포함됩니다. z/OS에서 Java 채택을 장려하기 위해 특정 System z 하드웨어에서 특수 프로세서를 사용할 수 있습니다. 이 프로세서를 IBM System z Application Assist Processor(zAAP)라고 하며 CICS의 Java 워크로드를 포함하여 적절한 Java 워크로드를 실행할 수 있도록 추가 프

로세서 용량을 제공할 수 있습니다. z/OS 플랫폼에서 Java에 대한 자세한 정보를 찾을 수 있으며 http://www.ibm.com/servers/eserver/zseries/software/java/에서 SDK의 64 비트 버전을 다운로드할 수 있습니다.

CICS는 Eclipse 기반 도구와 Java 애플리케이션을 위한 JVM 서버 런타임 환경을 제공합니다.

CICS Explorer® SDK

CICS Explorer SDK는 Eclipse 기반 통합 개발 환경(IDE)을 위한 무료 다운로드입니다. SDK는 OSGi 서비스 플랫폼 스펙을 준수하는 애플리케이션 개발 및 전개를 지원합니다. OSGi 서비스 플랫폼은 구성요소 모델을 사용하여 애플리케이션을 개발하고 해당 애플리케이션을 프레임워크에 OSGi 번들로 전개하기 위한 메커니즘을 제공합니다. *OSGi 번들*은 애플리케이션 구성요소의 전개 단위이며 버전 제어 정보, 종속 항목, 애플리케이션 코드가 포함됩니다. OSGi의 주된 이점은 *OSGi 서비스*라는 올바르게 정의된 인터페이스를 통해서만 액세스하는 재사용 가능 구성요소에서 애플리케이션을 작성할 수 있다는 것입니다. 또한 Java 애플리케이션의 라이프 사이클과 종속 항목을 세부적으로 관리할 수 있습니다.

CICS Explorer SDK는 CICS의 지원되는 릴리스에 대한 Java 애플리케이션 개발을 지원합니다. SDK에는 CICS용 애플리케이션 개발을 시작하는 CICS 서비스 및 예제에 액세스하기 위한 클래스의 JCICS(Java CICS) 라이브러리가 포함됩니다. 도구를 사용하여 기존 Java 애플리케이션을 OSGi로 변환할 수도 있습니다.

CICS Explorer SDK에서는 CICS에 전개할 수 있는 CICS 번들에 Liberty 애플리케이션을 패키징할 수 있도록 지원합니다.

JVM 서버

JVM 서버는 CICS의 Java 애플리케이션을 위한 전략적 런타임 환경입니다. JVM 서버는 다른 Java 애플리케이션으로부터의 여러 동시 요청을 단일 JVM에서 처리할 수 있습니다. JVM 서버를 사용하면 CICS 영역에서 Java 애플리케이션을 실행하는 데 필요한 JVM 수가 감소합니다. JVM 서버를 사용하려면 Java 애플리케이션이 스레드세이프(threadsafe) 상태여야 하며 OSGi 스펙을 준수해야 합니다. JVM 서버는 다음과 같은 이점을 제공합니다.

- zAAP 프로세서에서 적절한 Java 워크로드가 실행될 수 있어 트랜잭션 비용이 감소합니다.
- 스레드세이프(threadsafe) 상태의 Java 프로그램 및 웹 서비스와 같은 다양한 유형의 작업을 JVM 서버에서 실행할 수 있습니다.
- JVM 서버를 다시 시작하지 않고 OSGi 프레임워크에서 애플리케이션 라이프 사이클을 관리할 수 있습니다.

- OSGi를 사용하여 패키지화되는 Java 애플리케이션을 CICS와 다른 플랫폼 간에 보다 쉽게 이식할 수 있습니다.
- 웹 애플리케이션이 Liberty JVM 서버에 전개될 수 있습니다.

5 페이지의『OSGi 서비스 플랫폼』
OSGi 서비스 플랫폼은 구성 요소 모델을 사용하여 애플리케이션을 개발하고 OSGi 프레임워크에 해당 애플리케이션을 전개하기 위한 메커니즘을 제공합니다. OSGi 구조는 Java 애플리케이션을 작성 및 관리하는 데 도움을 주는 여러 계층으로 분리됩니다.

6 페이지의『JVM 서버 런타임 환경』
*JVM 서버*는 단일 JVM에서 여러 Java 애플리케이션에 대한 많은 동시 요청을 처리할 수 있는 런타임 환경입니다. JVM 서버를 사용하여 스레드세이프(threadsafe) 상태의 Java 애플리케이션을 OSGi 프레임워크에서 실행하고 Liberty 프로파일에서 웹 애플리케이션을 실행하며 Axis2 웹 서비스 엔진에서 웹 서비스 요청을 처리할 수 있습니다.

9 페이지의『JVM 프로파일』
JVM 프로파일은 JVM의 특성을 결정하는 Java 실행 프로그램 선택항목과 시스템 등록 정보를 포함하는 텍스트 파일입니다. JVM 프로파일은 표준 텍스트 편집기를 사용하여 편집할 수 있습니다.

10 페이지의『JVM 구조』
CICS에서 실행되는 JVM은 JVM 프로파일에 정의되고 64비트 저장영역을 사용하는 클래스 및 클래스 경로 세트를 사용합니다. 각 JVM은 MVS 저장영역을 가장 효율적으로 사용하도록 조정할 수 있는 Language Environment 인클레이브에서 실행됩니다.

13 페이지의『CICS 태스크 및 스레드 관리』
CICS는 OTE(Open Transaction Environment)를 사용하여 JVM 서버 작업을 실행합니다. 각 태스크는 JVM 서버에서 스레드로 실행되며 T8 TCB를 사용하여 연결됩니다. OSGi 사용에 따른 주요 이점은 OSGi 프레임워크의 애플리케이션이 ExecutorService를 사용하여 CICS에서 비동기식으로 추가 태스크를 실행하는 스레드를 작성한다는 것입니다. CICS는 특수한 측정을 사용하여 이탈 태스크를 처리합니다.

16 페이지의『공유 클래스 캐시』
공유 클래스 캐시는 단일 캐시에 저장된 Java 애플리케이션 클래스를 공유하기 위해 여러 JVM에 대한 메커니즘을 제공합니다. IBM SDK for z/OS는 공유 클래스 캐시를 지원합니다. 클래스 캐시는 OSGi JVM 서버 및 Apache Axis2와 같은 비 OSGi JVM 서버에 사용할 수 있습니다.

OSGi 서비스 플랫폼

OSGi 서비스 플랫폼은 구성 요소 모델을 사용하여 애플리케이션을 개발하고 OSGi 프레임워크에 해당 애플리케이션을 전개하기 위한 메커니즘을 제공합니다. OSGi 구조는 Java 애플리케이션을 작성 및 관리하는 데 도움을 주는 여러 계층으로 분리됩니다.

OSGi 프레임워크는 OSGi 서비스 플랫폼 스펙의 핵심입니다. CICS는 OSGi 프레임워크의 Equinox 구현을 사용합니다. 자세한 정보는 의 내용을 참조하십시오.OSGi 프레임워크는 JVM 서버가 시작될 때 초기화됩니다. Java 애플리케이션에 OSGi를 사용하는 경우 다음과 같은 주요 이점을 제공합니다.

- JVM을 다시 시작하지 않고 또한 해당 JVM에 전개되는 다른 Java 애플리케이션에 영향을 주지 않고 새 Java 애플리케이션과 새 버전 Java 애플리케이션을 활성 프로덕션 시스템에 전개할 수 있습니다.
- Java 애플리케이션은 이식성이 우수하고 리엔지니어링이 용이하며 변경 요구사항에 보다 잘 순응합니다.
- 동적 라이프 사이클을 포함하는 OSGi 번들 세트로 애플리케이션을 전개하는 선택항목을 제공하여 POJO(Plain Old Java Object) 프로그래밍 모델을 따를 수 있습니다.
- 애플리케이션 번들 종속 항목과 버전을 보다 쉽게 관리할 수 있습니다.

OSGi 구조는 다음과 같은 계층으로 구성됩니다.

- 모듈 계층
- 라이프 사이클 계층
- 서비스 계층

모듈 계층

전개 단위는 OSGi 번들입니다. 모듈 계층에서는 OSGi 프레임워크가 번들의 모듈 측면을 처리합니다. OSGi 프레임워크로 이 처리를 수행할 수 있는 메타데이터가 번들 Manifest 파일에 제공됩니다.

OSGi의 한 가지 중요한 이점은 Manifest 파일의 메타데이터를 사용하는 클래스 로더 모델입니다. OSGi에는 글로벌 클래스 경로가 없습니다. 번들이 OSGi 프레임워크에 설치되면 모듈 계층이 해당 메타데이터를 처리하며 설치된 다른 모듈이 선언한 내보내기 및 버전 정보에 대해 선언된 외부 종속 항목을 조정합니다. OSGi 프레임워크는 모든 종속 항목을 분석하며 각 번들에 필요한 독립 클래스 경로를 계산합니다. 이 접근 방식을 사용하면 다음과 같은 요구사항이 충족되므로 일반 Java 클래스 로딩의 단점이 해결됩니다.

- 각 번들은 자신이 명시적으로 내보낸 Java 패키지에 대한 가시성만 제공합니다.
- 각 번들은 패키지 종속 항목을 명시적으로 선언합니다.

- 패키지는 특정 버전에서 내보낼 수 있으며, 특정 버전 또는 특정 버전 범위에서 가져올 수 있습니다.
- 여러 개의 패키지 버전을 여러 클라이언트에서 동시에 사용할 수 있습니다.

라이프 사이클 계층

OSGi의 번들 라이프 사이클 관리 계층을 통해 JVM의 라이프 사이클에 관계 없이 번들을 동적으로 설치, 시작, 중지, 설치 제거할 수 있습니다. 라이프 사이클 계층을 통해, 모든 종속 항목이 해결된 경우에만 번들이 시작될 수 있으므로 런타임에서 ClassNotFoundException 예외 발생을 줄일 수 있습니다. 해결되지 않은 종속 항목이 있는 경우에는 OSGi 프레임워크가 문제점을 보고하고 번들을 시작하지 않습니다.

각 번들은 번들 Manifest에서 식별되고 프레임워크가 이벤트를 시작 및 중지하기 위해 호출하는 번들 활성기 클래스를 제공할 수 있습니다.

서비스 계층

OSGi의 서비스 계층은 본질적으로 내구성이 적은 서비스 레지스트리 구성요소를 통해 서비스 지향 구조를 지원합니다. 번들은 서비스를 서비스 레지스트리에 공개하므로, 다른 번들은 서비스 레지스트리에서 이러한 서비스를 검색할 수 있습니다. 이 서비스는 번들 간 협업을 위한 주요 수단입니다. OSGi 서비스는 POJO(Plain Old Java Object)이며 사용자 정의 특성(이름/값 쌍)으로 저장된 선택적 메타데이터와 함께 하나 이상의 Java 인터페이스 이름으로 서비스 레지스트리에 공개됩니다. 검색 번들은 인터페이스 이름으로 서비스 레지스트리에서 서비스를 조회할 수 있으며, 사용자 정의 특성을 기준으로 조회되는 서비스를 잠재적으로 필터링할 수 있습니다.

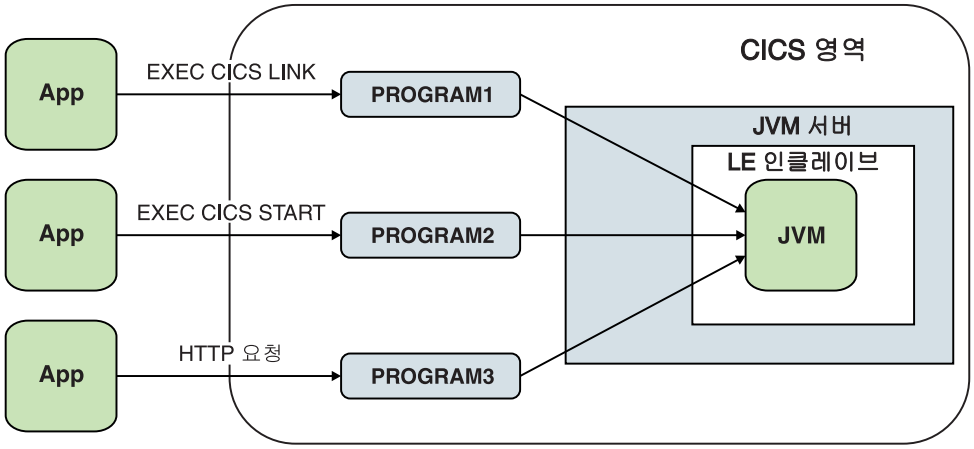
서비스는 완전히 동적이며 일반적으로 서비스를 제공하는 번들과 동일한 라이프 사이클을 갖습니다.

JVM 서버 런타임 환경

*JVM 서버*는 단일 JVM에서 여러 Java 애플리케이션에 대한 많은 동시 요청을 처리할 수 있는 런타임 환경입니다. JVM 서버를 사용하여 스레드세이프(threadsafe) 상태의 Java 애플리케이션을 OSGi 프레임워크에서 실행하고 Liberty 프로파일에서 웹 애플리케이션을 실행하며 Axis2 웹 서비스 엔진에서 웹 서비스 요청을 처리할 수 있습니다.

JVM 서버는 JVMSERVER 자원으로 표시됩니다. JVMSERVER 자원을 사용할 수 있으면 CICS가 MVS™에서 저장영역을 요청하고 Language Environment® 인클레이브를 설정하며 인클레이브에서 64비트 JVM을 실행합니다. CICS는 JVMSERVER 자원에 지정된 JVM 프로파일을 사용하여 올바른 선택항목으로 JVM을 작성합니다. 이 프로파일에서 JVM 선택항목과 시스템 등록 정보를 지정하고 고유 라이브러리를 추가할 수 있습니다. 예를 들어, 고유 라이브러리를 추가하여 Java 애플리케이션에서 DB2 또는 WebSphere MQ에 액세스할 수 있습니다.

JVM 서버 사용의 이점 중 하나는 동일한 JVM에서 여러 애플리케이션에 대한 많은 요청을 실행할 수 있다는 것입니다. 다음 다이어그램에서는 세 개 애플리케이션이 다른 액세스 방법을 사용하여 CICS 영역에서 세 개 Java 프로그램을 동시에 호출합니다. 각 Java 프로그램이 동일한 JVM 서버에서 실행됩니다.



Java 애플리케이션

Java 애플리케이션을 JVM 서버에서 실행하려면 스레드세이프(threadsafe) 상태여야 하며 CICS 번들에서 하나 이상의 OSGi 번들로 패키지화되어야 합니다. JVM 서버는 OSGi 번들과 서비스를 실행할 수 있는 OSGi 프레임워크를 구현합니다. OSGi 프레임워크는 서비스를 등록하고 번들 간의 종속 항목과 버전을 관리합니다. OSGi는 프레임워크에서 모든 클래스 경로 관리를 처리하므로 JVM 서버를 중지한 후 다시 시작하지 않고 Java 애플리케이션을 추가, 갱신, 제거할 수 있습니다.

OSGi를 사용하여 패키지화되는 Java 애플리케이션의 전개 단위는 CICS 번들입니다. BUNDLE 자원은 CICS에 대한 애플리케이션을 나타내므로 애플리케이션의 라이프 사이클을 관리하는 데 사용할 수 있습니다. CICS Explorer SDK는 CICS 번들 프로젝트에 OSGi 번들을 전개하기 위한 지원을 zFS에 제공합니다.

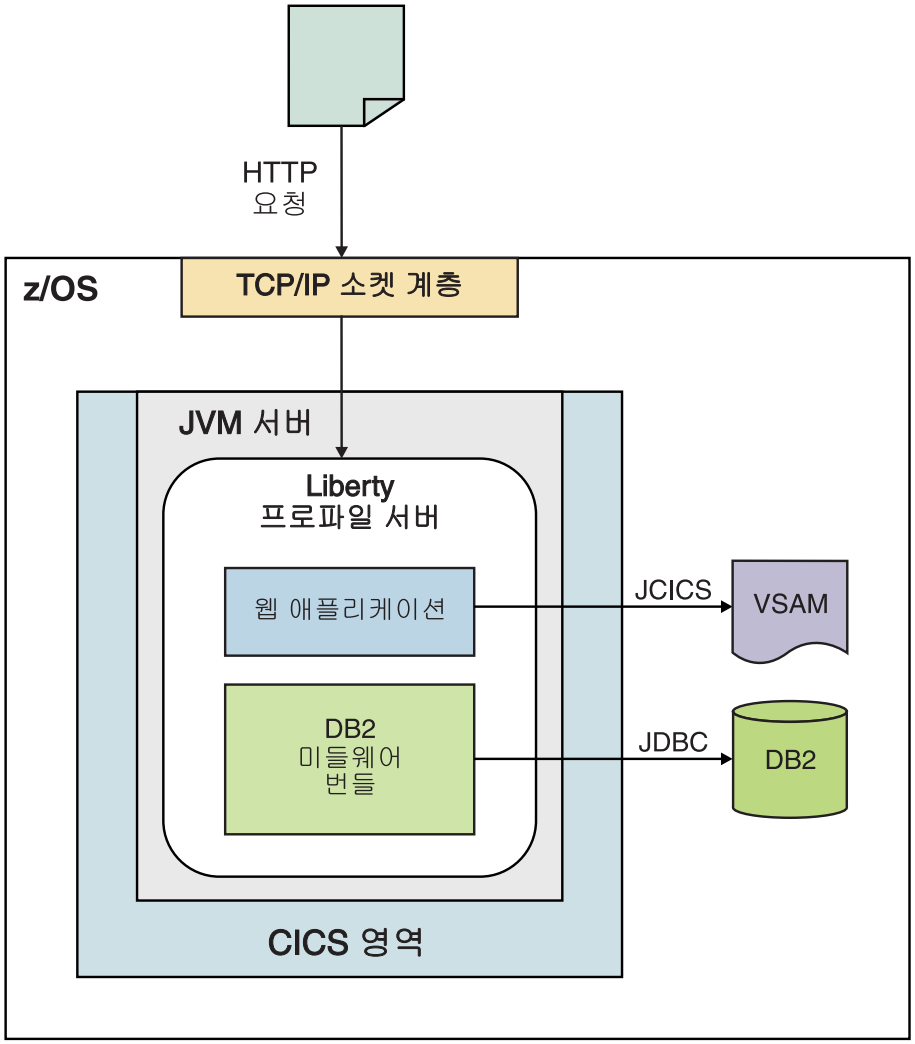
OSGi 프레임워크 외부에서 Java 애플리케이션에 액세스하려면 PROGRAM 자원을 사용하여 애플리케이션이 실행 중인 JVM 서버와 OSGi 서비스 이름을 식별하십시오. OSGi 서비스는 CICS 기본 클래스를 가리킵니다.

JVM 서버에서 OSGi 프레임워크 사용에 대한 자세한 정보는 16 페이지의 『OSGi를 준수하는 Java 애플리케이션』의 내용을 참조하십시오.

Java 웹 애플리케이션

JVM 서버는 Java 애플리케이션을 OSGi 프레임워크에서 실행하는 것뿐 아니라 WebSphere Application Server Liberty 프로파일 실행 또한 지원합니다. Liberty 프로파일은 웹 애플리케이션 실행을 위한 경량 애플리케이션 서버입니다. Liberty 프로파

일은 OSGi 프레임워크에서도 실행되므로 Java 웹 구성요소와 OSGi 번들을 동일한 JVM 서버에서 실행할 수 있습니다. 웹 애플리케이션은 JCICS를 사용하여 CICS의 자원과 서비스에 액세스하고 DB2의 데이터에 액세스할 수 있습니다. Liberty 프로파일 서버에서 실행되는 애플리케이션은 CICS의 웹 지원을 통해서가 아닌 z/OS의 TCP/IP 소켓 계층을 통해 액세스합니다.



Java 웹 애플리케이션은 전개할 Liberty 모델을 따를 수 있습니다. 이 모델에서는 개발자가 WAR 파일을 Liberty 서버의 drop-in 디렉토리에 직접 전개하거나 CICS 번들을 작성하는 CICS 애플리케이션 모델을 사용할 수 있습니다. CICS 번들은 라이프 사이클 관리를 제공하며 OSGi 번들, WAR 파일과 같은 많은 구성요소를 포함하는 애플리케이션을 함께 패키지화할 수 있습니다.

웹 애플리케이션에서 OSGi 번들에 액세스하려면 애플리케이션을 EBA(Enterprise Bundle Archive) 파일로 전개해야 합니다. EBA를 개발하려면 Rational® Application

Developer를 사용하거나 Eclipse IDE, CICS Explorer SDK, IBM WebSphere Application Server Developer Tools for Eclipse 버전 8.5.5의 조합을 사용할 수 있습니다. 이 조합의 도구는 무료로 사용할 수 있지만 CICS Explorer SDK를 제외하고는 IBM 지원 센터를 이용할 수 없습니다.

Liberty 프로파일 서버 사용에 대한 자세한 정보는 20 페이지의 『Liberty JVM 서버의 웹 애플리케이션 및 웹 서비스』의 내용을 참조하십시오.

웹 서비스

JVM 서버를 사용하여 웹 서비스 요청자 및 제공자 애플리케이션을 위한 SOAP 처리를 실행할 수 있습니다. 파이프라인이 Axis2(Java 기반의 SOAP 엔진)를 사용하는 경우에는 SOAP 처리가 JVM 서버에서 발생합니다. 웹 서비스에 JVM 서버를 사용하는 데 따른 이점은 zAAP 프로세서에 작업을 할당할 수 있다는 것입니다.

웹 서비스에 JVM 서버 사용에 대한 자세한 정보는 22 페이지의 『Java 웹 서비스』의 내용을 참조하십시오.

JVM 프로파일

JVM 프로파일은 JVM의 특성을 결정하는 Java 실행 프로그램 선택항목과 시스템 등록 정보를 포함하는 텍스트 파일입니다. JVM 프로파일은 표준 텍스트 편집기를 사용하여 편집할 수 있습니다.

CICS가 Java 프로그램을 실행하는 요청을 수신하면 JVM 프로파일의 이름이 Java 실행 프로그램으로 전달됩니다. Java 프로그램은 JVM 프로파일에서 선택항목을 사용하여 작성된 JVM에서 실행됩니다.

CICS는 JVMPROFILEDIR 시스템 초기화 매개변수가 지정하는 z/OS UNIX 시스템 서비스 디렉토리에 있는 JVM 프로파일을 사용합니다. 이 디렉토리에는 CICS가 JVM 프로파일을 읽기 위한 올바른 권한이 필요합니다.

샘플 JVM 프로파일

CICS에는 Java 환경 구성에 유용한 여러 가지 샘플 JVM 프로파일이 포함됩니다. 이러한 프로파일은 CICS 설치 프로세스에서 사용자 정의됩니다. 이러한 파일은 CICS가 기본값으로 또는 시스템 프로그램용으로 사용합니다.

JVM 프로파일은 CICS 실행 프로그램이 Java용으로 사용하는 선택항목을 나열합니다. 일부 선택항목은 CICS에만 해당되고 다른 선택항목은 JVM 런타임 환경의 표준입니다. 예를 들어, JVM 프로파일은 저장영역 힙의 *초기 크기*와 가능한 확장 *크기*를 제어합니다. 이 프로파일은 또한 JVM이 생성하는 메시지 및 덤프 출력의 대상을 정의할 수 있습니다. JVM 프로파일은 JVMSERVER 자원 정의의 JVMPROFILE 속성에서 이름이 지정됩니다. JVMSERVER 속성을 참조하십시오.

샘플을 복사하고 해당 애플리케이션에 맞게 사용자 정의할 수 있습니다. CICS와 함께 제공되는 샘플 JVM 프로파일은 /usr/lpp/cicsts/cicsts52/JVMProfiles 디렉토리에 있습니다(z/OS UNIX). 설치 디렉토리의 샘플을 **JVMPROFILEDIR** 시스템 초기화 매개변수에 지정한 디렉토리에 복사하십시오. 이러한 파일에 대한 변경사항을 포함하는 APAR을 적용하는 경우 설치 위치의 샘플 JVM 프로파일을 겹쳐씁니다. 수정사항을 유지하려면 고유 애플리케이션 클래스를 추가하거나 선택항목을 변경하기 전에 샘플을 항상 다른 위치에 복사하십시오.

다음 표에는 각 샘플 JVM 프로파일의 주요 특성이 요약되어 있습니다.

표 2. CICS와 함께 제공되는 샘플 JVM 프로파일

JVM 프로파일	**특성**
DFHJVMAX.jvmprofile	Axis2 JVM 서버에 제공되는 샘플 프로파일. JVM 프로파일은 JVMSERVER 자원에서 지정됩니다. CICS는 DFHJVMAX.jvmprofile을 사용하여 JVM 서버를 초기화합니다.
DFHJVMST.jvmprofile	보안 토큰 서비스를 위해 JVM 서버에 제공된 샘플 프로파일. JVM 프로파일은 JVMSERVER 자원에서 지정됩니다. CICS는 DFHJVMST.jvmprofile을 사용하여 JVM 서버를 초기화합니다.
DFHOSGI.jvmprofile	OSGi JVM 서버에 제공되는 샘플 프로파일. JVM 프로파일은 JVMSERVER 자원에서 지정됩니다. CICS는 DFHJVMAX.jvmprofile을 사용하여 JVM 서버를 초기화합니다.
DFHWLP.jvmprofile	Liberty JVM 서버에 제공되는 샘플 프로파일. JVM 프로파일은 JVMSERVER 자원에서 지정됩니다. CICS는 DFHWLP.jvmprofile을 사용하여 Liberty JVM 서버를 초기화합니다.

JVM 구조

CICS에서 실행되는 JVM은 JVM 프로파일에 정의되고 64비트 저장영역을 사용하는 클래스 및 클래스 경로 세트를 사용합니다. 각 JVM은 MVS 저장영역을 가장 효율적으로 사용하도록 조정할 수 있는 Language Environment 인클레이브에서 실행됩니다.

IBM 64-bit SDK for z/OS, Java Technology Edition의 버전 7에 대한 자세한 정보는 IBM SDK for z/OS, Java Technology Edition, 버전 7 사용자 안내서의 내용을 참조하십시오.

11 페이지의 『JVM의 클래스 및 클래스 경로』
CICS에서 실행되는 JVM은 최초 클래스(시스템 및 표준 확장 클래스), 고유 C DLL 라이브러리 파일, 애플리케이션 클래스 등 다른 유형의 클래스 또는 라이브러리 파일을 사용할 수 있습니다.

12 페이지의 『JVM의 저장영역 힙』
JVMs for IBM 64-bit SDK for z/OS, Java Technology Edition 버전 7의 런타임 저장영역은 단일 64비트 저장영역 힙으로 관리됩니다.

12 페이지의 『JVM이 구성되는 경우』
JVM이 필요한 경우 JVM용 CICS 실행 프로그램이 MVS에서 저장영역을 요청하고 Language Environment 인클레이브를 설정하며 Language Environment 인클레이브에서 JVM을 실행합니다. 각 JVM은 병렬로 실행되는 JVM 간의 분리를 위해 고유한 Language Environment 인클레이브에서 구성됩니다.

13 페이지의『JVM 및 z/OS 공유 라이브러리 영역』
공유 라이브러리 영역은 주소 공간이 동적 링크 라이브러리(DLL) 파일을 공유할 수 있도록 해주는 z/OS 기능입니다.

JVM의 클래스 및 클래스 경로

CICS에서 실행되는 JVM은 최초 클래스(시스템 및 표준 확장 클래스), 고유 C DLL 라이브러리 파일, 애플리케이션 클래스 등 다른 유형의 클래스 또는 라이브러리 파일을 사용할 수 있습니다.

JVM은 이러한 구성요소 각각의 목적을 인식하고 구성요소 로드 방법과 구성요소 저장 위치를 판별합니다. JVM의 클래스 경로는 JVM 프로파일에서 선택항목으로 정의되며 선택적으로 JVM 등록 정보 파일에서 참조됩니다.

- *최초 클래스*는 JVM에서 기본 서비스를 제공하는 z/OS JVM 코드입니다. 최초 클래스는 시스템 클래스와 표준 확장 클래스로 분류될 수 있습니다.
- *고유 C 동적 링크 라이브러리(DLL) 파일*은 `.so` 확장자를 갖습니다(z/OS UNIX). JVM을 실행하는 데 필요한 라이브러리도 있고 애플리케이션 코드 또는 서비스로 로드할 수 있는 추가 고유 라이브러리도 있습니다. 예를 들어, 추가 고유 라이브러리에는 DB2 JDBC 드라이버를 사용하기 위한 DLL 파일이 포함됩니다.
- *애플리케이션 클래스*는 JVM에서 실행되는 애플리케이션용 클래스이며 사용자가 작성한 애플리케이션에 속하는 클래스가 포함됩니다. Java 애플리케이션 클래스에는 또한 JCICS 인터페이스 클래스인 JDBC, JNDI와 같이 CICS에 대한 표준 JVM 표준 설정에 포함되지 않은 자원에 액세스하는 서비스를 제공하기 위해 IBM 또는 다른 벤더가 제공하는 클래스가 포함됩니다. Java 애플리케이션 클래스가 클래스 캐시에 로드되면 클래스가 보존되어 동일한 JVM에서 실행되는 다른 애플리케이션이 재사용할 수 있습니다.

클래스 또는 고유 라이브러리가 지정될 수 있는 클래스 경로는 라이브러리 경로이자 표준 클래스 경로입니다.

- *라이브러리 경로*는 애플리케이션 코드 또는 서비스로 로드된 JVM 및 추가 고유 라이브러리를 실행하는 데 필요한 파일을 포함하여 JVM이 사용하는 고유 C 동적 링크 라이브러리(DLL) 파일을 지정합니다. 각 DLL 파일의 사본이 하나만 로드되고 모든 JVM이 공유하지만 각 JVM은 DLL에 대한 정적 데이터 영역 사본을 소유합니다.

  JVM의 기본 라이브러리 경로는 JVM 프로파일에서 **USSHOME** 시스템 초기화 매개변수 및 **JAVA_HOME** 선택항목으로 지정된 디렉토리를 사용하여 자동으로 빌드됩니다. 기본 라이브러리 경로는 JVM 프로파일에 표시되지 않습니다. 이 경로에는 CICS가 사용하는 고유 라이브러리와 JVM을 실행하는 데 필요한 모든 DLL 파일이 포함됩니다. **LIBPATH_SUFFIX** 선택항목 또는 **LIBPATH_PREFIX** 선택항목을 사용하여 라이브러리 경로를 확장할 수 있습니다. **LIBPATH_SUFFIX**는 라이브러리 경로 끝에서 IBM

제공 라이브러리 다음에 항목을 추가합니다. **LIBPATH_PREFIX**는 시작 부분에 항목을 추가합니다(이름이 같은 IBM 제공 라이브러리 대신 로드되는 경우). 문제점 판별 목적으로 이 작업을 수행해야 할 수도 있습니다.

라이브러리 경로에 포함하는 DLL 파일과 LP64 선택항목을 컴파일하고 함께 링크하십시오. 기본 라이브러리 경로에서 제공되는 DLL 파일과 DB2 JDBC 드라이버와 같은 서비스가 사용하는 DLL 파일은 LP64 선택항목으로 빌드됩니다.

- *표준 클래스 경로*는 OSGi 사용 JVM 서버에 사용해서는 안됩니다. OSGi 프레임워크가 애플리케이션을 포함하는 OSGi 번들의 정보에서 애플리케이션의 클래스 경로를 자동으로 판별하기 때문입니다. 표준 클래스 경로는 OSGi에 대해 구성되지 않는 JVM 서버가 사용하도록 보존됩니다(예를 들어, CICS의 Axis2 환경). Axis2와 같이 표준 클래스 경로를 사용하는 예외적인 시나리오의 경우에는 클래스 경로 항목에 총칭 문자 접미부를 사용하여 특정 디렉토리의 모든 JAR 파일을 지정할 수 있습니다.

  CICS는 또한 **USSHOME** 시스템 초기화 매개변수로 지정된 디렉토리의 `/lib` 서브디렉토리를 사용하여 JVM에 대한 기본 클래스 경로를 자동으로 빌드합니다. 이 클래스 경로에는 CICS와 JVM이 제공하는 JAR 파일이 포함됩니다. JVM 프로파일에는 이 경로가 표시되지 않습니다.

시스템 클래스와 표준 확장 클래스(최초 클래스)는 JVM에서 부트 클래스 경로에 이미 포함되어 있으므로 클래스 경로에 포함하지 않아도 됩니다.

JVM의 저장영역 힙

JVMs for IBM 64-bit SDK for z/OS, Java Technology Edition 버전 7의 런타임 저장영역은 단일 64비트 저장영역 힙으로 관리됩니다.

각 JVM의 힙은 JVM에 대한 Language Environment 인클레이브의 64비트 저장영역에서 할당됩니다. 각 힙의 *크기*는 JVM 프로파일에서 선택항목으로 판별합니다.

단일 저장영역 힙은 *힙*이라고도 하고 *사용공간 정리 힙*이라고도 합니다. *초기 저장영역 할당*은 JVM 프로파일의 **-Xms** 선택항목으로 설정되고 *최대 크기*는 **-Xmx** 선택항목으로 설정됩니다.

JVM에 최적화된 성능을 달성하기 위해 힙 *크기*를 조정할 수 있습니다. 172 페이지의 『JVM 서버 힙 및 사용공간 정리 조정』의 내용을 참조하십시오.

JVM이 구성되는 경우

JVM이 필요한 경우 JVM용 CICS 실행 프로그램이 MVS에서 저장영역을 요청하고 Language Environment 인클레이브를 설정하며 Language Environment 인클레이브에서 JVM을 실행합니다. 각 JVM은 병렬로 실행되는 JVM 간의 분리를 위해 고유한 Language Environment 인클레이브에서 구성됩니다.

Language Environment 인클레이브는 Language Environment 사전 초기화 모듈, CELQPIPI로 작성되고 JVM은 z/OS UNIX 프로세스로 실행됩니다. 따라서 JVM은 CICS Language Environment 서비스가 아닌 MVS Language Environment 서비스를 사용합니다. JVM에 사용되는 저장영역은 MVS 64비트 저장영역으로, MVS Language Environment 서비스를 호출하여 획득합니다. 이 저장영역은 CICS 주소 공간에 있지만 CICS 동적 저장영역(DSA)에는 포함되지 않습니다.

JVM의 Language Environment 인클레이브는 JVM의 저장영역 요구사항에 따라 확장될 수 있습니다. CICS에서 Language Environment 인클레이브에 사용하는 Language Environment 런타임 선택항목은 Language Environment 인클레이브 힙 저장영역의 초기 크기와 추가 증분을 제어합니다.

CICS가 인클레이브에 요청하는 저장영역 크기가 JVM 프로파일로 지정된 저장영역의 크기에 최대한 가깝도록 CICS가 Language Environment 인클레이브에 사용하는 런타임 선택항목을 조정할 수 있습니다. 따라서 MVS 저장영역을 가장 효율적으로 사용할 수 있습니다. 저장영역 조정에 대한 자세한 정보는 174 페이지의 『JVM용 Language Environment 인클레이브 저장영역』의 내용을 참조하십시오.

JVM 및 z/OS 공유 라이브러리 영역

공유 라이브러리 영역은 주소 공간이 동적 링크 라이브러리(DLL) 파일을 공유할 수 있도록 해주는 z/OS 기능입니다.

이 기능을 사용하면 각 CICS 영역이 JVM에 필요한 DLL을 개별적으로 로드하지 않고 공유할 수 있습니다. 따라서 MVS가 사용하는 실제 저장영역의 크기와 영역이 파일을 로드하는 데 소요되는 시간을 크게 줄일 수 있습니다.

공유 라이브러리 영역에 대해 예약되는 저장영역은 첫 번째 JVM이 영역에서 시작될 때 각 CICS 영역에 할당됩니다. 할당되는 저장영역 크기는 z/OS의 **SHRLIBRGNSIZE** 매개변수로 제어됩니다. 공유 라이브러리 영역에 대해 할당되는 저장영역 크기 조정에 대한 자세한 정보는 179 페이지의 『z/OS 공유 라이브러리 영역 조정』의 내용을 참조하십시오.

CICS 태스크 및 스레드 관리

CICS는 OTE(Open Transaction Environment)를 사용하여 JVM 서버 작업을 실행합니다. 각 태스크는 JVM 서버에서 스레드로 실행되며 T8 TCB를 사용하여 연결됩니다. OSGi 사용에 따른 주요 이점은 OSGi 프레임워크의 애플리케이션이 ExecutorService를 사용하여 CICS에서 비동기식으로 추가 태스크를 실행하는 스레드를 작성한다는 것입니다. CICS는 특수한 측정을 사용하여 이탈 태스크를 처리합니다.

CICS가 JVM 서버를 사용 가능으로 설정하면 JVM 서버는 Language Environment 프로세스 스레드에서 실행됩니다. 이 스레드는 TP TCB의 하위입니다. 모든 CICS 태

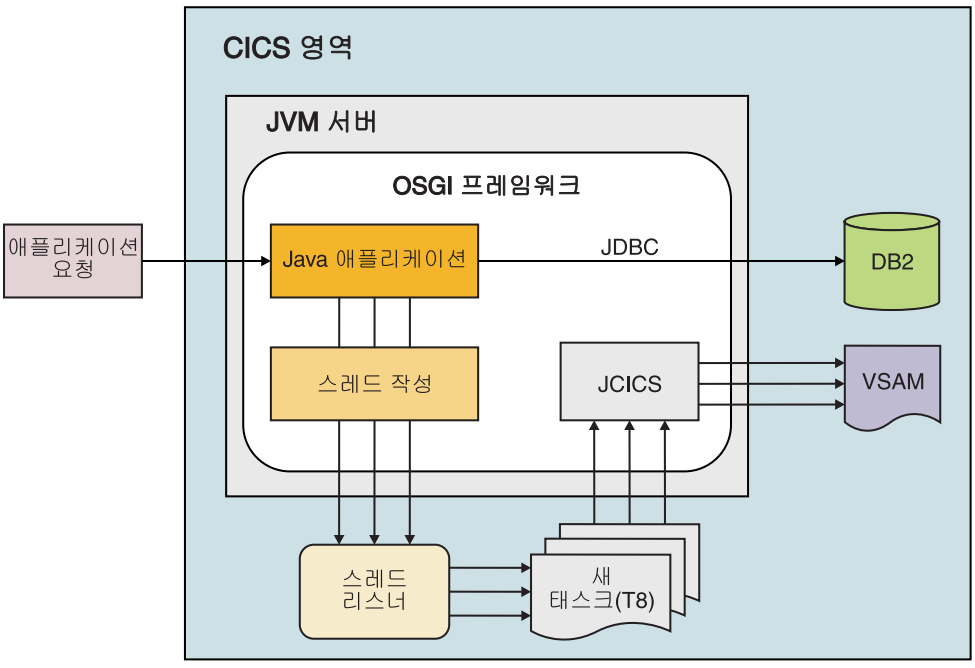
스크는 T8 TCB를 사용하여 JVM의 스레드에 첨부됩니다. JVMSERVER 자원에서 THREADLIMIT 속성을 설정하여 JVM 서버가 사용할 수 있는 T8 TCB 수를 제어할 수 있습니다.

JVM 서버용으로 작성되는 T8 TCB는 가상 풀에 있으며 동일한 CICS 영역에서 실행 중인 다른 JVM 서버가 이를 재사용할 수 없습니다. 모든 JVM 서버에서 CICS 영역에 존재할 수 있는 최대 T8 TCB 수는 2000이고 특정 JVM 서버에 대한 최대값은 256입니다.

멀티스레드된 애플리케이션

OSGi 프레임워크에서 실행되는 Java 애플리케이션은 또한 ExecutorService OSGi 서비스를 사용하여 비동기식으로 CICS 태스크를 시작할 수 있습니다. JVM 서버는 시동 시 ExecutorService를 OSGi 서비스로 등록합니다. ExecutorService는 JCICS API를 사용하여 CICS 서비스에 액세스할 수 있는 스레드를 작성하며 CICS에서 제공하는 구현을 자동으로 사용합니다. 이러한 접근 방법은 애플리케이션이 스레드를 작성하기 위해 특정 JCICS API 메소드를 사용하지 않아도 됨을 의미합니다. 그러나 애플리케이션이 CICSExecutorService를 사용하여 별도 CICS 가능 스레드에서 작업을 실행할 수도 있습니다.

JVM 서버를 사용하는 경우 JVM 서버 리스너라는 장기 실행 태스크를 작성하기 위한 CJSL 트랜잭션을 시작합니다. 이 리스너는 애플리케이션으로부터의 새 스레드 요청을 대기하며 CJSA 트랜잭션을 실행하여 T8 TCB에서 디스패치되는 CICS 태스크를 작성합니다. 이 프로세스가 다음 다이어그램에 나와 있습니다.



고급 시나리오에서는 애플리케이션이 OSGi 서비스를 사용하여 많은 스레드를 비동기식으로 실행할 수 있습니다. 이러한 스레드는 모두 JCICS를 통해 CICS 서비스에 액세스하며 T8 TCB에서 실행됩니다.

JVM 서버를 위한 실행 키

Java 프로그램은 올바른 실행 키에서 실행되는 JVM을 사용해야 합니다. JVM 서버는 CICS 키에서 실행됩니다. JVM 서버를 사용하려면 Java 프로그램을 위한 PROGRAM 자원이 EXECKEY 속성을 CICS로 설정해야 합니다. CICS는 T8 TCB를 사용하여 JVM을 실행하고 CICS 키의 MVS 저장영역을 획득합니다.

이탈 태스크

CICS JVM 서버 인프라는 태스크 이탈 발견 메커니즘 사용을 지원합니다. JVM 기반 태스크가 유효한 이탈 한계보다 오랫동안 프로세스의 제어를 보유하는 경우 JVM의 연관 스레드가 AICA 또는 비슷한 이상 종료로 종료됩니다.

스레드가 종료되면 JVM이 불일치 상태로 남아 있을 수 있습니다. Java 프로그래밍 언어 및 JVM은 이 방식의 스레드 중지를 허용하도록 설계되지 않았습니다. 예를 들어, 종료된 스레드로 인해 잠금이 유지될 수 있고, 파이널라이저가 실행되지 않았을 수 있고, 자원이 할당된 상태로 남아 있을 수 있고, 공유 저장영역이 불일치 상태로 남아 있을 수 있습니다. 애플리케이션 및 시스템 상태 둘 다 영향을 받을 수 있습니다.

이탈 처리로 인해 스레드가 종료된 경우 CICS가 자동으로 JVM 서버를 다시 시작하여 JVM을 다시 일치 상태로 복구합니다. 다시 시작하면 JVM 서버가 단계적으로 중지되고 이후 즉시 다시 설정됩니다. JVM 서버는 이 간격 동안 새 작업을 처리할 수 없습니다. CICS는 다시 시작이 발생했음을 보고하기 위해 메시지 DFHSJ1009를 발행합니다.

JVM 서버에서 실행 중인 태스크에 대한 이탈 조건은 동일한 JVM 서버의 모든 사용에 대한 임시 사용 가능성 문제점을 발생시킬 수 있습니다. 이와 같은 이유로 CICS는 구성된 이탈 제한시간 값을 10배로 곱하여 수정합니다(최대값은 45분). 이 새로운 값이 유효한 이탈 값입니다. 이와 같이 이탈 제한시간을 높게 설정하면 비효율적인(하지만 작동 중인) 애플리케이션에 대해 이탈 조건이 발견되는 가능성이 감소됩니다. 예를 들어, 트랜잭션 정의가 RUNAWAY=SYSTEM을 지정하고 ICVR 시스템 초기화 매개변수가 기본 한계 500밀리초를 나타내는 경우 해당 태스크가 JVM 서버에서 실행될 때 유효한 이탈 값은 50000밀리초입니다.

공유 클래스 캐시

공유 클래스 캐시는 단일 캐시에 저장된 Java 애플리케이션 클래스를 공유하기 위해 여러 JVM에 대한 메커니즘을 제공합니다. IBM SDK for z/OS는 공유 클래스 캐시를 지원합니다. 클래스 캐시는 OSGi JVM 서버 및 Apache Axis2와 같은 비OSGi JVM 서버에 사용할 수 있습니다.

CICS가 아닌 Java 7은 JVM 서버에서 클래스 캐시 기능 사용을 지원합니다. 따라서 CICS SPI 또는 CEMT 명령을 사용하여 JVM 서버 클래스 캐시를 사용 또는 사용 안함으로 설정할 수 없습니다. JVM 서버 클래스 캐시를 사용 가능 또는 사용 안함으로 설정하려면 JVM 명령행 매개변수를 사용합니다. 또한 JVM 명령행 매개변수를 사용하여 클래스 캐시 *크기*를 설정합니다.

클래스 캐시에는 다음 구성요소(파일, 오브젝트, 변수, 컴파일된 클래스)가 로드되지 *않습니다*.

- JVM 프로파일 라이브러리 경로에 지정된 고유 C DLL 파일. 파일에 액세스해야 하는 모든 JVM이 각 DLL 파일의 단일 사본을 사용하므로 이 파일은 로드되지 않습니다.
- 애플리케이션 오브젝트 및 변수. 작업 데이터가 JVM에 저장되므로 로드되지 않습니다.
- JIT(Just-In-Time) 컴파일 클래스. 워크로드마다 컴파일 프로세스가 다를 수 있으므로 로드되지는 않지만 JVM에 저장됩니다.

캐시 사용

클래스 캐시는 기본적으로 JVM에서 사용할 수 없습니다. 클래스 캐시를 사용 가능으로 설정하려면 JVM 명령행 매개변수를 사용합니다. 예를 들어, 다음과 같습니다.

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

캐시 크기 지정

클래스 캐시 *크기*를 지정하려면 JVM 명령행 매개변수를 사용합니다. 예를 들어, 다음 매개변수는 *크기*를 20M으로 설정합니다.

```
-Xscmx20M
```

Java 클래스 데이터 공유에 대한 자세한 정보는 JVM 간 클래스 데이터 공유를 참조하십시오.

OSGi를 준수하는 Java 애플리케이션

CICS에는 JVM 서버에서 OSGi 사양을 준수하는 Java 애플리케이션을 실행하기 위한 OSGi 프레임워크의 Equinox 구현이 포함됩니다.

OSGi 서비스 플랫폼 스펙(5 페이지의 『OSGi 서비스 플랫폼』의 설명 참조)은 모듈 및 동적 Java 애플리케이션을 실행하고 관리하기 위한 프레임워크를 제공합니다. JVM 서버의 기본 구성에는 OSGi 프레임워크의 Equinox 구현이 포함됩니다. JVM 서버의 OSGi 프레임워크에 전개되는 Java 애플리케이션은 OSGi 사용의 이점과 CICS에서 애플리케이션을 실행하는 데 따른 서비스 품질을 활용할 수 있습니다.

다음과 같은 이유로 Java 애플리케이션을 사용할 수 있습니다.

- 트랜잭션 비용을 줄이기 위해 zAAP에서 실행될 수 있는 Java 워크로드를 작성합니다.
- 다른 플랫폼에서 OSGi를 사용하는 Java 애플리케이션 작성 경험이 있고 CICS에서 Java 애플리케이션을 작성하려고 합니다.
- 애플리케이션의 사용 가능성과 애플리케이션이 실행되는 JVM에 영향을 주지 않고 독립적으로 재사용 및 갱신할 수 있는 모듈형 구성요소의 세트로 Java 애플리케이션을 제공할 수 있습니다.

OSGi를 준수하는 Java 애플리케이션을 효과적으로 개발, 전개, 관리하려면 CICS Explorer SDK 및 CICS Explorer를 사용해야 합니다.

- CICS Explorer SDK는 기존 Eclipse IDE(Integrated Development Environment)를 강화하여 Java 개발자가 CICS에서 Java 애플리케이션을 작성하고 전개하는 데 도움을 줄 수 있는 도구와 지원을 제공합니다. 이 도구를 사용하면 기존 Java 애플리케이션을 OSGi 번들로 변환할 수 있습니다.
- CICS Explorer는 OSGi 번들, OSGi 서비스, 실행되는 JVM 서버에 대한 보기를 시스템 관리자에게 제공하는 Eclipse 기반 시스템 관리 도구입니다. 이 도구를 사용하면 Java 애플리케이션을 사용 또는 사용 안함으로 설정하고 프레임워크에서 OSGi 번들 및 서비스의 상태를 확인하고 JVM 서버의 성능에 대한 예비적 통계를 가져올 수 있습니다.

OSGi 작업을 수행하려는 Java 개발자 또는 시스템 관리자에게는 이처럼 무료로 사용할 수 있는 도구에 대한 액세스 권한이 필요합니다.

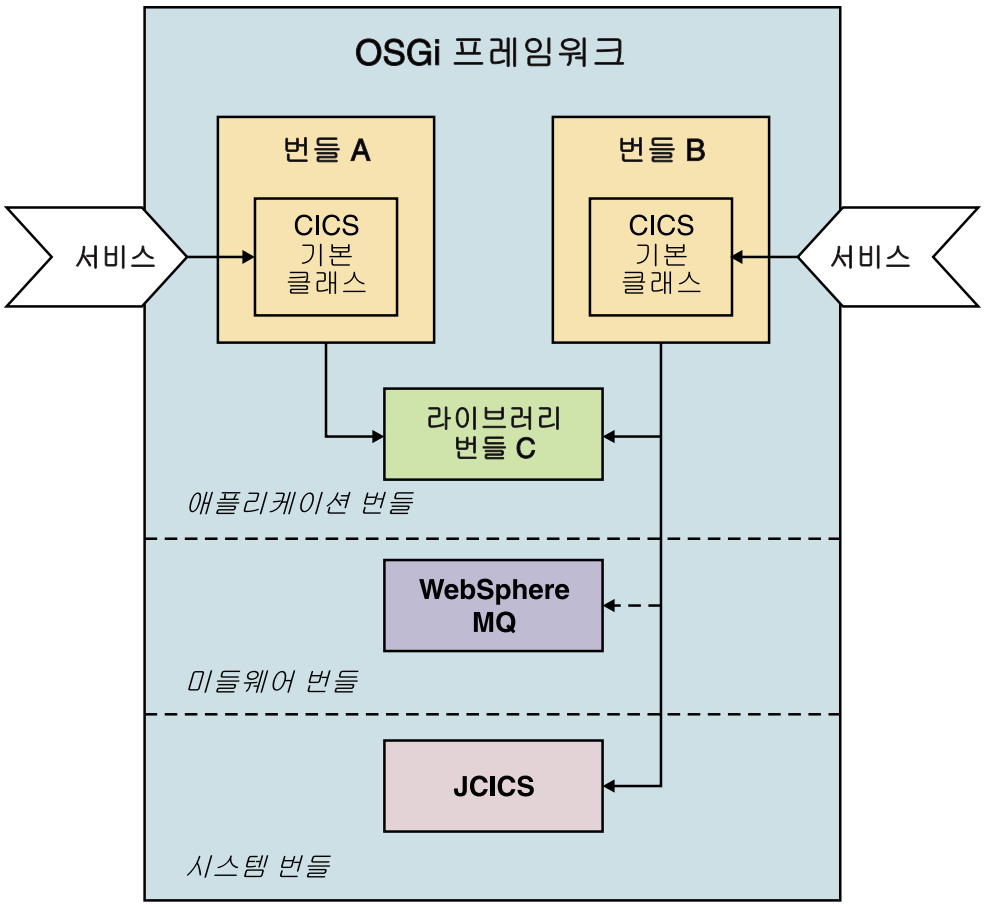
다음 예제는 CICS에서 OSGi를 사용하는 Java 애플리케이션을 실행할 수 있는 방법을 설명합니다.

동일한 JVM 서버에서 여러 Java 애플리케이션 실행

JVM 서버는 동일한 JVM에서 동시에 여러 요청을 처리할 수 있습니다. 따라서 동일한 애플리케이션을 동시에 여러 번 호출하거나 동일한 JVM 서버에서 여러 애플리케이션을 실행할 수 있습니다.

JVM 서버 간에 애플리케이션을 분할하는 방법을 결정한 경우 OSGi 모델을 사용하여 애플리케이션을 OSGi 번들 세트로 구성요소화하는 방법을 계획할 수 있습니다. 또한

애플리케이션에 서비스를 제공하기 위해 프레임워크에서 필요한 지원 OSGi 번들을 결정해야 합니다. OSGi 프레임워크에는 다른 유형의 OSGi 번들이 포함될 수 있습니다(다음 다이어그램 참조).



애플리케이션 번들

애플리케이션 번들은 애플리케이션 코드를 포함하는 OSGi 번들입니다. OSGi 번들은 자체적으로 포함되거나 프레임워크에서 다른 번들에 종속됩니다. 이러한 종속 항목은 프레임워크가 관리하므로 분석되지 않는 종속 항목이 있는 OSGi 번들은 프레임워크에서 실행할 수 없습니다. 애플리케이션을 CICS의 프레임워크 외부에서 액세스할 수 있게 하려면 OSGi 번들이 CICS 기본 클래스를 해당 OSGi 서비스로 선언해야 합니다. PROGRAM 자원이 CICS 기본 클래스를 가리키는 경우 OSGi 프레임워크 외부의 다른 애플리케이션이 Java 애플리케이션에 액세스할 수 있습니다. 하나 이상의 애플리케이션에 대한 공통 라이브러리를 포함하는 OSGi 번들이 있는 경우 Java 개발자가 CICS 기본 클래스를 선언하지 않을 수 있습니다. 이 OSGi 번들은 프레임워크의 다른 OSGi 번들만 사용할 수 있습니다.

Java 애플리케이션의 전개 단위는 CICS 번들입니다. CICS 번들은 임의 수의 OSGi 번들을 포함할 수 있으며 하나 이상의 JVM 서버에 전개될 수 있습니다. JVM 서버 관리와 별도로 애플리케이션 번들을 추가, 갱신, 제거할 수 있습니다.

미들웨어 번들

미들웨어 번들은 시스템 서비스(예를 들어, WebSphere MQ에 연결)를 구현하는 클래스를 포함하는 OSGi 번들입니다. 다른 예로는 고유 코드를 포함하여 OSGi 프레임워크에서 한 번만 로드되어야 하는 OSGi 번들을 들 수 있습니다. 미들웨어 번들은 해당 클래스를 사용하는 애플리케이션이 아닌 JVM 서버의 라이프 사이클로 관리됩니다. 미들웨어 번들은 JVM 서버의 JVM 프로파일에서 지정되므로 JVM 서버가 시작될 때 CICS로 로드됩니다.

시스템 번들

시스템 번들은 애플리케이션에 키 서비스를 제공하기 위해 CICS와 OSGI 프레임워크 간의 상호작용을 관리하는 OSGi 번들입니다. 대표적인 예로 CICS 서비스 및 자원에 대한 액세스를 제공하는 JCICS OSGi 번들을 들 수 있습니다.

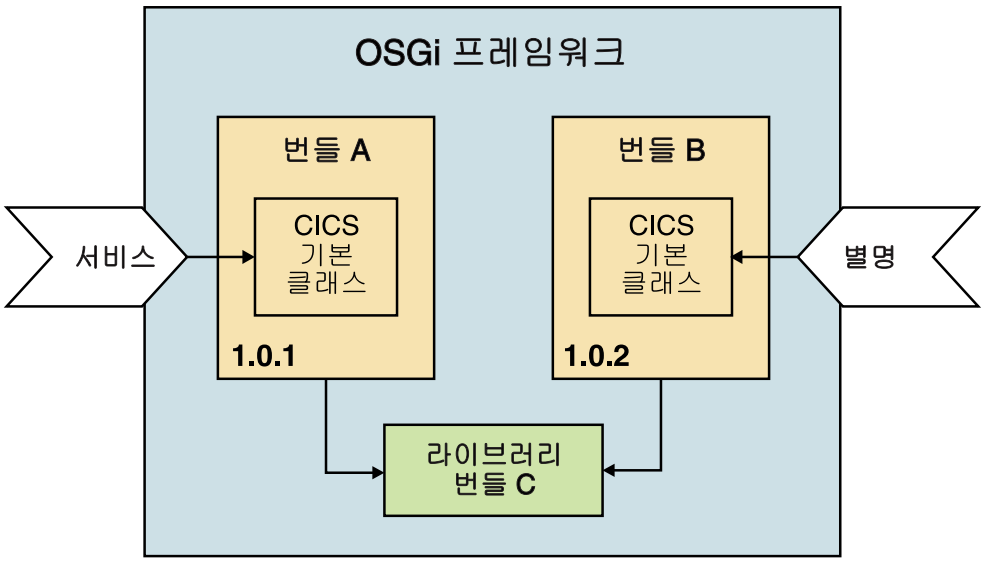
Java 애플리케이션 관리를 단순화하려면 다음 우수 사례를 따르십시오.

- 동일한 CICS 번들에서 애플리케이션을 비교하는 단단한 결합의 OSGi 번들을 전개하십시오. 단단한 결합 번들은 OSGi 서비스를 사용하지 않고 서로 직접 클래스를 내보냅니다. 이러한 OSGi 번들을 CICS 번들에서 함께 전개하면 번들을 함께 갱신하고 관리할 수 있습니다.
- 애플리케이션 간의 종속 항목 작성을 방지하십시오. 대신 별도 OSGi 번들에 공통 라이브러리를 작성하고 고유한 CICS 번들에서 관리하십시오. 라이브러리는 애플리케이션과 별도로 갱신할 수 있습니다.
- 번들 간에 종속 항목을 작성할 때 버전을 사용하여 OSGi 우수 사례를 따르십시오. 버전 범위를 사용한다는 것은 애플리케이션이 종속되는 번들에 대한 호환 갱신을 허용한다는 것입니다.
- JVM 서버에 대한 이름 지정 규칙을 설정하고 시스템 프로그래머와 Java 개발자 간의 규칙에 동의하십시오.
- 싱글톤 OSGi 번들은 사용하지 마십시오. 다른 번들이 종속되어 있는 싱글톤 번들을 버리면 종속 번들이 실패할 수 있습니다.

JVM 서버에서 동일한 Java 애플리케이션의 여러 버전 실행

OSGi 프레임워크는 프레임워크에서 OSGi 번들의 여러 버전 실행을 지원하므로 사용 가능성을 방해하지 않고 애플리케이션 갱신을 단계적으로 진행할 수 있습니다. 그러나 프레임워크에서 동일한 OSGi 서비스의 여러 버전을 가질 수 없습니다. OSGi 번들의 다른 버전이 동일한 CICS 기본 클래스를 포함하는 경우 별명을 사용하여 중복 서비스

를 덮어쓸 수 있습니다. 별명은 CICS 기본 클래스의 선언으로 지정되며 OSGi 프레임워크에서 번들의 갱신 버전에 대한 OSGi 서비스로 등록됩니다. 애플리케이션을 사용할 수 있도록 다른 PROGRAM 자원에 대한 별명을 지정하십시오.



관련 정보:

OSGi 애플리케이션에 대한 JVM 서버 구성
OSGi 번들에 패키지화되는 Java 애플리케이션을 전개하려는 경우 JVM 서버에서 OSGi 프레임워크를 실행하도록 구성합니다.

JCICS 예제 시작하기

Liberty JVM 서버의 웹 애플리케이션 및 웹 서비스

CICS는 경량 Java servlet과 JSP(JavaServer Pages)를 실행할 수 있는 웹 컨테이너를 제공합니다. 개발자는 Java servlet 및 JSP 스펙의 다양한 특성을 사용하여 CICS용 최신 웹 애플리케이션을 작성할 수 있습니다. 웹 컨테이너는 JVM 서버에서 실행되며 WebSphere Application Server Liberty 프로파일 기술을 기반으로 빌드됩니다.

Liberty 프로파일은 빠르게 시작되고 다른 플랫폼에서 실행될 수 있는 애플리케이션 개발을 위한 경량 웹 컨테이너입니다. Java 개발자가 애플리케이션을 빠르게 개발 및 테스트할 수 있도록 최적화되어 웹 서버를 구성 및 시작하는 데 최소한의 노력만 필요합니다. Java 개발자는 무료로 제공되는 Eclipse 도구를 사용하여 애플리케이션과 웹 서버를 간단하게 전개할 수 있도록 패키지화합니다.사용 가능한 웹 서비스 지원에는 JAX-RS(Java API for RESTful Web Services) 및 JAX-WS(Java API for XML Web Services)가 포함됩니다. Liberty 프로파일에 대한 자세한 정보는 Liberty 프로파일 개요를 참조하십시오.

Liberty 프로파일 기술은 JVM 서버에서 웹 컨테이너로 실행되도록 CICS와 함께 설치됩니다. Liberty JVM 서버는 Liberty 프로파일에서 사용 가능한 특성의 서브세트를 지원합니다. OSGi 애플리케이션 Java servlet, JSP 페이지를 실행할 수 있습니다. 지원되는 특성에 대한 자세한 정보는 Liberty 특성을 참조하십시오.

Liberty JVM 서버와 연관 도구는 다음과 같은 이유로 사용할 수 있습니다.

- 3270 화면을 웹 브라우저 및 RESTful 클라이언트로 바꿔 CICS 애플리케이션의 표시 인터페이스를 최신화하려고 합니다.
- Java 표준 기반 개발 도구를 사용하여 다른 기존 CICS 애플리케이션으로 웹 클라이언트를 패키지화, 공존, 관리하려고 합니다.
- WebSphere Application Server에서 Liberty 프로파일 애플리케이션을 이미 사용하고 있으며 CICS에서 실행하기 위해 이식하려고 합니다.
- CICS에서 이미 Jetty 또는 유사한 servlet 엔진을 사용하고 있으며 Liberty 프로파일을 기반으로 하는 웹 컨테이너로 이주하려고 합니다.
- DataSource 정의를 사용하여 Java에서 DB2 데이터베이스에 액세스하려고 합니다 (CICS DB2 연결 정의 참조).
- JTA(Java Transaction API)를 사용하여, 유형 4 JDBC 데이터베이스 드라이버를 통해 원격 자원 관리자를 갱신하여 CICS 복구 가능 자원에 대한 갱신을 조정하려고 합니다.
- JAX-RS를 사용하여 REST(Representational State Transfer) 원리를 따르는 서비스를 전개해야 합니다.
- JAX-WS를 사용하여 표준 어노테이션 기반 모델 지원을 통해 애플리케이션을 개발해야 합니다.

관련 개념:

구성에서 웹 애플리케이션에 대한 Liberty JVM 서버 구성

Java 웹 애플리케이션 개발

관련 정보:

Liberty 프로파일에 대한 JVM 서버 구성
Java servlet 및 JSP 페이지를 사용하는 Java 웹 애플리케이션을 전개하려는 경우 Liberty JVM 서버에서 웹 컨테이너를 실행하도록 구성합니다. 웹 컨테이너를 Liberty 프로파일 기술을 기반으로 합니다.

servlet 시작하기 예제

Java 웹 서비스

CICS에는 Java 웹 서비스를 실행하기 위한 Axis2 기술이 포함됩니다. Axis2는 Apache 기반의 개방형 소스 웹 서비스 엔진이며 Java 환경에서 SOAP 메시지를 처리하기 위해 CICS와 함께 제공됩니다.

Axis2는 여러 웹 서비스 스펙을 지원하는 웹 서비스 SOAP 엔진의 Java 구현입니다. 또한 Axis2에서 실행될 수 있는 Java 애플리케이션을 작성하는 방법을 설명하는 프로그래밍 모델을 제공합니다. Axis2는 Java 환경에서 웹 서비스를 처리하기 위해 CICS와 함께 제공되므로 적절한 Java 처리를 zAAP 프로세서로 전달하도록 지원합니다.

JVM 서버는 기존 웹 서비스를 변경하지 않고 Java SOAP 파이프라인에서 인바운드 및 아웃바운드 SOAP 메시지를 처리하기 위한 Axis2 실행을 지원합니다. 그러나 Java 애플리케이션에서 웹 서비스를 작성하고 동일한 JVM 서버에서 실행할 수도 있습니다. JVM 서버의 Axis2 저장소에 애플리케이션을 전개함으로써 Java 애플리케이션 및 SOAP 처리 모두 zAAP에서 실행할 수 있습니다.

Java 웹 서비스를 사용하는 이유는 다음과 같습니다.

- 다른 플랫폼의 Axis2 웹 서비스에 대한 경험이 있고 CICS에서 웹 서비스를 작성하려고 합니다.
- 표준 Java API를 사용하여 Axis2와 통합되는 Java 데이터 바인딩을 작성합니다.
- CICS 웹 서비스 보조자로 처리하기 어려운 WSDL 문서를 복잡하게 만들었습니다.

다음 예제는 웹 서비스에서 Java를 사용할 수 있는 방법에 대해 설명합니다.

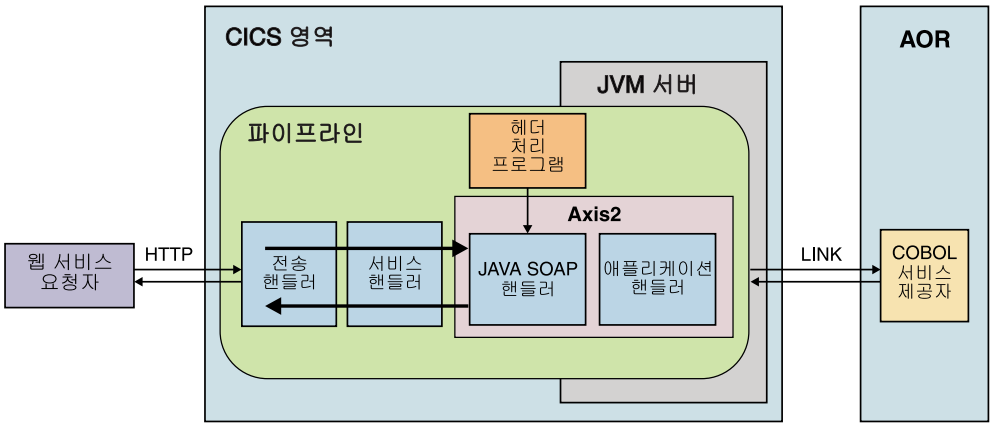
JVM 서버에서의 SOAP 메시지 처리

웹 서비스 파이프라인에서 발생하는 대부분의 SOAP 처리는 SOAP 핸들러와 애플리케이션 핸들러가 수행합니다. 선택적으로 JVM 서버에서 이 SOAP 처리를 실행하고 zAAP를 사용하여 작업을 실행할 수 있습니다. COBOL, C, C++ 또는 PL/I로 작성된 웹 서비스 애플리케이션도 계속 사용할 수 있습니다.

기존 웹 서비스가 있는 경우 JVM 서버를 사용하도록 파이프라인 구성을 갱신할 수 있습니다. 웹 서비스는 변경하지 않아도 됩니다. 파이프라인이 SOAP 헤더 처리 프로그램을 사용하는 경우에는 Axis2 프로그래밍 모델을 사용하여 Java로 프로그램을 재작성하는 것이 가장 좋습니다. 헤더 처리 프로그램은 추가 데이터 변환을 수행하지 않고 Axis2와 Java 오브젝트를 공유할 수 있습니다. 예를 들어, 헤더 처리 프로그램이 COBOL에 있는 경우 데이터를 Java에서 COBOL로 변환한 후 다시 반대 방향으로 변환해야 하므로 SOAP 처리 성능이 저하될 수 있습니다.

다음 다이어그램의 시나리오는 웹 서비스 제공자인 COBOL 애플리케이션 예제입니다. 요청은 Java를 지원하도록 구성된 파이프라인에서 처리됩니다. SOAP 핸들러와 애플리

케이션 핸들러는 Axis2가 처리하고 JVM 서버에서 실행되는 Java 프로그램입니다. 애플리케이션 핸들러는 데이터를 XML에서 COBOL로 변환하고 애플리케이션에 링크됩니다.



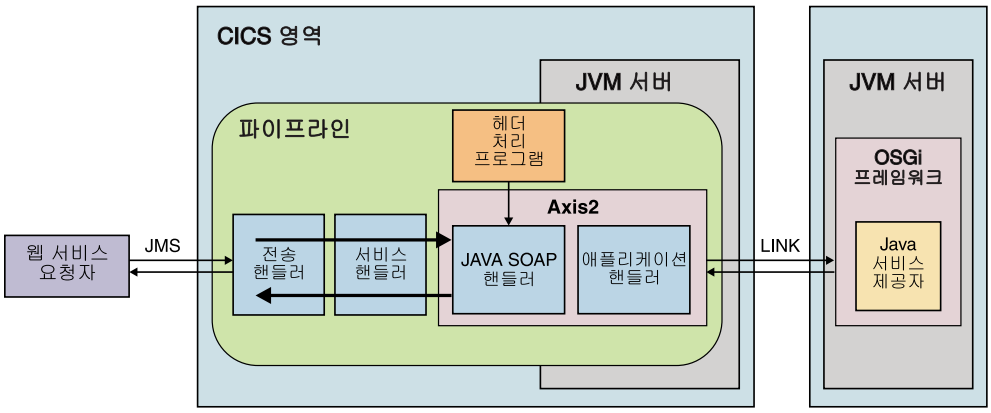
환경을 계획할 때 JVM 서버의 전용 영역 세트를 사용하는지 확인하십시오. 이 예제에서 COBOL 애플리케이션은 JVM 서버가 실행되는 CICS 영역과 다른 애플리케이션 소유 영역(AOR)에서 실행됩니다. 워크로드 관리를 사용하여 워크로드 균형을 맞출 수 있습니다(예를 들어, 애플리케이션 핸들러의 **EXEC CICS LINK** 또는 웹 서비스 요청자의 인바운드 요청에서).

CICS 웹 서비스 보조자로부터의 출력을 사용하는 Java 애플리케이션 작성

언어 구조를 해석하고 CICS 웹 서비스 보조자에서 생성된 데이터 바인딩을 사용하는 Java 애플리케이션을 작성할 수 있습니다. 웹 서비스 보조자는 WSDL에서 언어 구조를 또는 언어 구조에서 WSDL을 생성할 수 있습니다. 보조자는 또한 SOAP 처리 중에 XML과 대상 언어 사이에 데이터를 변환하는 방법을 설명하는 웹 서비스 바인딩을 생성합니다.

보조자를 사용하여 언어 구조를 생성하는 경우 JZOS 또는 J2C를 사용하여 언어 구조 작업을 수행함으로써 Java 클래스를 생성할 수 있습니다. 이러한 도구는 Java 개발자가 다른 CICS 애플리케이션과 상호작용할 수 있는 방법을 제공합니다. 이 예제에서는 이러한 도구를 사용하여 CICS가 XML에서 데이터를 변환한 후 인바운드 SOAP 메시지를 처리할 수 있는 Java 애플리케이션을 작성합니다. 자세한 정보는 38 페이지의 『Java 구조화 데이터와의 상호작용』의 내용을 참조하십시오.

다음 다이어그램의 시나리오는 웹 서비스 제공자인 Java 애플리케이션 예제입니다. SOAP 처리는 JVM 서버에서 Axis2가 수행합니다. 애플리케이션 핸들러는 하나 이상의 OSGi 번들로 패키지화되고 전개되는 Java 애플리케이션에 링크되며 JVM 서버에서 실행됩니다.



이 접근 방법의 이점은 웹 서비스 보조자가 데이터 바인딩을 생성했으므로 웹 서비스가 CICS에서 WEBSERVICE 자원으로 표시된다는 것입니다. 통계, 자원 관리, CICS의 기타 기능을 사용하여 웹 서비스를 관리할 수 있습니다. 단점은 Java 개발자가 친숙하지 않은 프로그래밍 언어의 언어 구조 작업을 수행해야 한다는 것입니다.

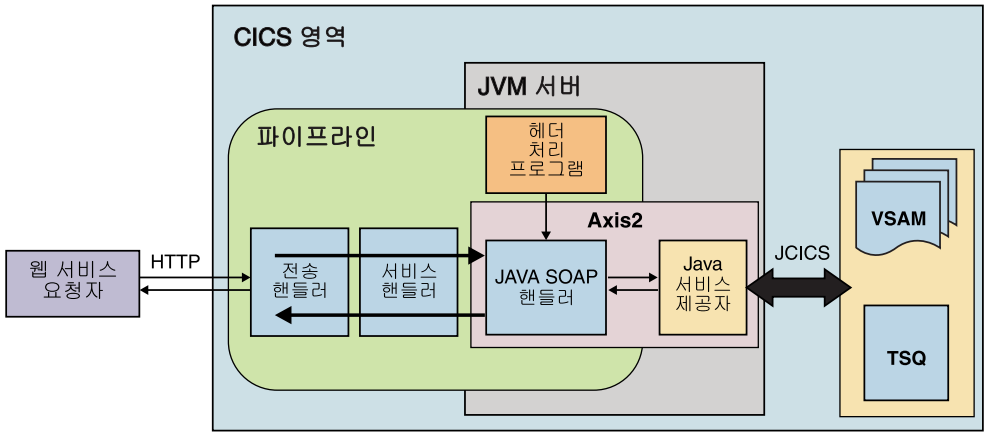
이러한 유형의 애플리케이션에 대해 사용자 환경을 계획하는 경우 별도 JVM 서버를 사용하여 애플리케이션을 실행하십시오.

- 다른 워크로드의 경우 JVM 서버를 보다 효과적으로 관리하고 조정할 수 있습니다.
- 인바운드 요청과 **EXEC CICS LINK**에서 워크로드 관리를 사용하여 워크로드 균형을 맞추고 환경 스케일을 조정할 수 있습니다.
- CICS의 OSGi 지원을 이용하여 Java 애플리케이션을 관리할 수 있습니다.

Java 데이터 바인딩을 사용하는 Java 애플리케이션 작성

SOAP용 XML 메시지를 생성하고 구문 분석하는 Java 애플리케이션을 작성할 수 있습니다. Java 7 API는 XML 작업을 위한 표준 Java 라이브러리를 제공합니다. 예를 들어, JAXB(Java Architecture for XML Binding)를 사용하여 Java 데이터 바인딩을 작성하고 JAX-WS(Java API for XML Web Services) 라이브러리를 사용하여 XML을 생성하고 구문 분석할 수 있습니다. 이러한 라이브러리를 사용하는 경우 애플리케이션은 SOAP 파이프라인 처리와 동일한 JVM 서버에서 Axis2로 실행될 수 있습니다.

다음 다이어그램의 시나리오는 웹 서비스 제공자이자 JVM 서버에서 Axis2 SOAP 엔진으로 처리하는 Java 애플리케이션의 예제입니다.



Java 애플리케이션은 Java 데이터 바인딩을 사용하고 Java SOAP 핸들러와 상호작용하므로 애플리케이션 핸들러가 없습니다. 이 예제에서는 웹 서비스 요청자가 HTTP를 사용하여 CICS 영역에 연결하지만 JMS를 사용할 수도 있습니다. Java 애플리케이션은 JCICS를 사용하여 CICS 서비스에 액세스합니다(이 예제의 경우 VSAM 파일과 임시 저장영역 큐).

이 접근 방식의 이점은 Java 개발자가 친숙한 기술을 사용하여 애플리케이션을 작성한다는 것입니다. 또한 Java 개발자는 웹 서비스 보조자가 처리할 수 없는 복잡한 WSDL 문서에 대한 작업으로 바인딩을 생성할 수 있습니다. 그러나 이 접근 방법에는 다음과 같은 몇 가지 제한사항이 있습니다.

- 이 유형의 애플리케이션에는 WS 보안을 사용할 수 없으므로 보안을 사용하려면 SSL을 사용하여 연결을 보호해야 합니다.
- 파이프라인 처리에서 사용자 ID에 대한 컨텍스트 전환이 발생하지 않습니다. 요청에서 사용자 ID를 변경하려면 URIMAP 자원을 사용하십시오.
- 웹 서비스 보조자의 웹 서비스 바인딩을 사용하지 않으므로 WEBSERVICE 자원은 없습니다.
- 애플리케이션이 웹 서비스 요청자인 경우에는 파이프라인 처리가 무시됩니다. 따라서 파이프라인에서 사용 가능한 서비스 품질(QoS)을 얻을 수 없습니다.

CICS 영역에서 워크로드 관리를 구현하는 경우 이 워크로드 유형을 라우팅하는 방법을 계획해야 합니다. Java 애플리케이션은 SOAP 처리와 동일한 JVM 서버에서 실행되므로 CICS는 라우팅 기회를 제공하지 않습니다. 그러나 라우팅이 필요한 경우 JAX-WS 애플리케이션의 DPL(Distributed Program Link)를 다른 프로그램에 구현할 수 있습니다.

제 2 장 CICS용 Java 애플리케이션 개발

CICS 서비스를 사용하고 CICS 제어하에 실행되는 Java 애플리케이션을 작성할 수 있습니다. CICS Explorer SDK를 사용하여 CICS 자원에 액세스하고 다른 언어로 작성된 프로그램과 상호 작용하기 위해 JCICS 클래스 라이브러리를 사용하는 애플리케이션을 개발할 수 있습니다. 웹 서비스 또는 CICS Transaction Gateway와 같은 다양한 프로토콜과 기술을 사용하여 Java 프로그램에 연결할 수도 있습니다.

CICS는 Java 애플리케이션을 지원하기 위한 런타임 환경과 도구를 제공합니다. CICS Explorer SDK는 Java 애플리케이션을 개발하여 CICS에 전개하기 위한 지원을 제공하는 Eclipse 기반 도구입니다. 이 도구에는 CICS 자원 및 서비스에 액세스하는 애플리케이션을 개발하기 위한 JCICS 클래스 라이브러리가 포함되어 있습니다. 예를 들어, VSAM 파일, 트랜지언트 데이터 큐, 임시 저장영역에 액세스할 수 있습니다. JCICS를 사용하여 COBOL, C와 같은 다른 언어로 작성된 CICS 애플리케이션에 링크할 수도 있습니다.

CICS Explorer SDK는 다음과 같은 기능을 제공합니다.

- CICS의 특정 릴리스에서 지원되는 클래스만 사용하도록 대상 환경을 구성할 수 있습니다. SDK는 애플리케이션 개발을 위한 올바른 라이브러리를 가져옵니다.
- OSGi 번들, OSGi 애플리케이션 프로젝트(EBA 파일), 웹 사용 OSGi 번들 프로젝트(WAB 파일), 동적 웹 프로젝트(WAR 파일)를 하나 이상의 CICS 번들에 패키지화하여 전개할 수 있습니다. 그러나 WAR 파일은 OSGi 번들에 액세스할 수 없으므로 WAR 파일과 OSGi 번들을 동일한 CICS 번들에 패키지화하는 데 따른 이점은 거의 없습니다. CICS는 servlet 또는 JSP 페이지와 같은 웹 구성요소를 포함하는 애플리케이션을 실행하기 위해 웹 컨테이너를 제공합니다. 웹 컨테이너는 WebSphere Application Server의 Liberty 기술을 기반으로 빌드됩니다.
- CICS 번들을 애플리케이션 프로젝트에 패키지화하여 플랫폼에 전개할 수 있습니다.

SDK에는 CICS용 Java 애플리케이션을 처음 개발할 때 시작하는 데 도움을 줄 수 있는 샘플 세트가 포함되어 있습니다.

29 페이지의『CICS에 대해 알아두어야 할 사항』

CICS는 다른 많은 사용자가 동일한 파일과 프로그램을 사용하여 동일한 애플리케이션을 실행하기 위한 요청을 제출하는 동시에 사용자가 요청으로 애플리케이션을 실행하기 위한 서비스를 제공하는 트랜잭션 처리 서브시스템입니다. CICS는 자원 공유, 데이터 무결성, 실행 우선순위를 관리하며 동시에 빠른 응답 시간을 유지합니다.

34 페이지의『CICS Explorer SDK를 사용하여 애플리케이션 개발』
CICS Explorer SDK(Software Development Kit)는 OSGi 및 웹 프로젝트에 대한 지원을 포함하여 Java 애플리케이션을 개발하여 CICS에 전개하기 위한 환경을 제공합니다.

36 페이지의『JVM의 상태 제어』
CICS에서 실행할 Java 애플리케이션을 설계 및 개발하는 경우 애플리케이션이 JVM을 잘못된 상태로 유지하지 않는지 또는 원하지 않는 방식으로 상태를 수정하지 않는지 확인하십시오. CICS 서비스를 사용하여 JVM 상태를 쉽게 제어할 수 있습니다.

38 페이지의『Java 구조화 데이터와의 상호작용』
CICS Java 프로그램은 원래 다른 프로그래밍 언어에 사용하도록 설계된 데이터와 자주 상호작용합니다. 예를 들어, Java 프로그램은 COBOL 카피북에 정의된 COMMAREA를 사용하여 COBOL 프로그램에 링크되거나 데이터가 C++ 헤더 파일을 사용하여 정의된 경우 VSAM 파일에서 레코드를 읽을 수 있습니다. 가져오기 프로그램을 사용하여 이러한 양식의 구조화된 데이터와 상호작용할 수 있습니다.

40 페이지의『JCICS를 사용한 Java 개발』
CICS 서비스에 액세스하기 위해 CICS Java 클래스 라이브러리(JCICS)를 사용하는 Java 애플리케이션을 작성할 수 있습니다. JCICS는 Java에서 다른 CICS 지원 언어
(예: COBOL)에 대해 제공되는 **EXEC CICS** API(Application Programming Interface)에 해당합니다.

77 페이지의『Java 애플리케이션에서 데이터 액세스』
DB2 및 VSAM의 데이터를 액세스하고 갱신할 수 있는 Java 애플리케이션을 작성할 수 있습니다. 또는 다른 언어의 프로그램에 링크하여 DB2, VSAM, IMS에 액세스할 수 있습니다.

78 페이지의『CICS에서 Java 애플리케이션으로부터의 연결성』
CICS 환경의 Java 프로그램은 TCP/IP 소켓을 열고 외부 프로세스와 통신할 수 있습니다. Java 프로그램을 게이트웨이로 사용하여 다른 언어의 CICS 프로그램이 사용할 수 없는 다른 엔터프라이즈 애플리케이션에 연결할 수 있습니다. 예를 들어, 원격 servlet 또는 데이터베이스와 통신할 Java 프로그램을 작성할 수 있습니다.

79 페이지의『JDBC 및 SQLJ를 사용하여 Java 프로그램에서 DB2 데이터 액세스』
CICS에서 실행되는 Java 프로그램은 여러 가지 방법을 사용하여 DB2 데이터베이스에 저장된 데이터에 액세스할 수 있습니다.

87 페이지의『Liberty JVM 서버에서 실행할 Java 웹 애플리케이션 개발』
Java servlet 및 JSP 페이지를 사용하는 Java™ 웹 애플리케이션을 전개하려는 경우 Liberty JVM 서버에서 웹 컨테이너를 실행하도록 구성합니다.

CICS에 대해 알아두어야 할 사항

CICS는 다른 많은 사용자가 동일한 파일과 프로그램을 사용하여 동일한 애플리케이션을 실행하기 위한 요청을 제출하는 동시에 사용자가 요청으로 애플리케이션을 실행하기 위한 서비스를 제공하는 트랜잭션 처리 서브시스템입니다. CICS는 자원 공유, 데이터 무결성, 실행 우선순위를 관리하며 동시에 빠른 응답 시간을 유지합니다.

CICS 애플리케이션은 제품 주문 처리 또는 회사 급여 준비와 같은 비즈니스 운영을 함께 수행하는 관련 프로그램의 콜렉션입니다. CICS 애플리케이션은 프로그램과 파일에 액세스하기 위한 인터페이스와 CICS 서비스를 사용하여 CICS 제어하에 실행됩니다.

transaction 요청을 제출하여 CICS 애플리케이션을 실행합니다. 트랜잭션이라는 용어는 CICS에서 특별한 의미를 갖습니다. CICS 사용 및 보다 일반적인 산업 용도의 차이에 대한 설명은 『CICS 트랜잭션』의 내용을 참조하십시오. 트랜잭션 실행은 필수 기능을 구현하는 하나 이상의 애플리케이션 실행으로 구성됩니다.

CICS용 Java 애플리케이션을 개발하려면 CICS 프로그램, 트랜잭션, 태스크 간의 관계를 이해해야 합니다. 이러한 용어는 CICS 문서 전반에 걸쳐 사용되며 많은 프로그래밍 명령에 나타납니다. 또한 CICS가 런타임 환경에서 Java 애플리케이션을 다루는 방식을 이해해야 합니다.

『CICS 트랜잭션』
트랜잭션은 단일 요청으로 시작되는 처리 단위입니다.

30 페이지의 『CICS 태스크』
태스크는 단일 트랜잭션 실행 인스턴스입니다.

30 페이지의 『CICS 애플리케이션』
Java 프로그램의 경우 JCICS(Java Class Library for CICS)를 사용하여 CICS 서비스와 다른 언어로 작성된 애플리케이션에 대한 링크에 액세스할 수 있습니다.

31 페이지의 『CICS 서비스』
Java 프로그램은 JCICS 프로그래밍 인터페이스를 통해 CICS 서비스(데이터 관리, 통신, 작업 단위, 프로그램, 진단 서비스)에 액세스할 수 있습니다.

33 페이지의 『CICS의 Java 런타임 환경』
CICS는 스레드세이프 상태의 Java 애플리케이션을 실행하기 위한 JVM 서버 환경을 제공합니다. 스레드세이프 상태가 아닌 애플리케이션은 JVM 서버를 사용할 수 없습니다.

CICS 트랜잭션

트랜잭션은 단일 요청으로 시작되는 처리 단위입니다.

요청은 일반적으로 단말기에서 사용자가 수행합니다. 그러나 웹 페이지, 원격 워크스테이션 프로그램, 또는 다른 CICS 영역의 애플리케이션에서 작성될 수도 있고 사전 정의

된 시간에 자동적으로 트리거될 수도 있습니다. 제품 개요의 CICS 웹 지원 개념 및 구조 및 제품 개요에서 CICS 외부 인터페이스의 개요에서는 다양한 CICS 트랜잭션 실행 방법을 설명합니다.

단일 트랜잭션은 실행 시 필요한 처리를 수행하는 하나 이상의 *애플리케이션*으로 구성됩니다.

그러나 *트랜잭션*이라는 용어는 CICS에서 단일 이벤트와 동일한 유형의 다른 모든 트랜잭션 모두의 평균을 구하는 데 사용됩니다. TRANSACTION 자원 정의로 CICS에 대한 각 트랜잭션 유형을 설명합니다. 이 정의는 트랜잭션 유형에 이름(트랜잭션 ID 또는 TRANSID)을 제공하며 먼저 호출할 프로그램, 트랜잭션 실행 중에 필요한 인증 유형 등 수행될 작업에 대한 여러 가지 정보를 CICS에 알려줍니다.

해당 TRANSID를 CICS에 제출하여 트랜잭션을 실행합니다. CICS는 TRANSACTION 정의에 기록된 정보를 사용하여 올바른 실행 환경을 설정하며 첫 번째 프로그램을 시작합니다.

*트랜잭션*이라는 용어는 이제 IT 업계 전반에서 *복구 단위*(CICS의 경우 *작업 단위*)를 설명하는 데 광범위하게 사용됩니다. 트랜잭션은 일반적으로 복구 가능한 완벽한 논리적 작업으로, 프로그래밍된 명령 또는 시스템 장애에 따라 모두 확약 또는 취소될 수 있습니다. 대부분의 경우 CICS 트랜잭션의 범위 또한 단일 작업 단위이지만 CICS 문서를 읽을 때 의미 차이를 고려해야 합니다.

CICS 태스크

태스크는 단일 트랜잭션 실행 인스턴스입니다.

*태스크*라는 단어는 CICS에서 특정 의미를 갖습니다. CICS가 트랜잭션 실행 요청을 수신하면 트랜잭션 유형 실행의 이 *단일 인스턴스*와 연관된 새 태스크를 시작합니다. 즉, CICS 태스크는 일반적으로 특정 사용자 대신 개인용 데이터 세트를 사용한 단일 트랜잭션 실행입니다. 태스크를 *스레드*로 간주할 수도 있습니다. 태스크는 우선순위와 준비 상태에 따라 CICS가 *디스패치*합니다. 트랜잭션이 완료되면 태스크가 종료됩니다.

CICS 애플리케이션

Java 프로그램의 경우 JCICS(Java Class Library for CICS)를 사용하여 CICS 서비스와 다른 언어로 작성된 애플리케이션에 대한 링크에 액세스할 수 있습니다.

CICS 애플리케이션은 COBOL, C, C++ , Java, PL/I 또는 어셈블러 언어로 작성될 수 있습니다. 대부분의 처리 논리는 표준 언어 명령문으로 표시되지만 CICS 서비스를 요청하기 위해 애플리케이션이 제공된 API(Application Programming Interface)를 사용합니다. COBOL, C, C++, PL/I, 또는 어셈블러 프로그램은 **EXEC CICS** API(Application Programming Interface) 또는 C++ 클래스 라이브러리를 사용할 수 있습니

다. Java 프로그램은 JCICS 클래스 라이브러리를 사용합니다. JCICS는 41 페이지의 『CICS용 Java 클래스 라이브러리(JCICS)』에서 설명합니다.

CICS 서비스

Java 프로그램은 JCICS 프로그래밍 인터페이스를 통해 CICS 서비스(데이터 관리, 통신, 작업 단위, 프로그램, 진단 서비스)에 액세스할 수 있습니다.

CICS 서비스 관리자는 일반적으로 제목에 제어 단어가 있습니다(예를 들어, 『단말기 제어』 및 『프로그램 제어』). 이러한 용어는 CICS 정보에서 광범위하게 사용됩니다.

데이터 관리 서비스

CICS는 다음 데이터 관리 서비스를 제공합니다.

- VSAM(Virtual Storage Access Method) 데이터 세트 액세스를 위한 레코드 레벨 공유와 무결성. CICS는 데이터 백아웃(트랜잭션 또는 시스템 장애) 및 정방향 복구(매체 장애)를 지원하기 위해 활동을 로그합니다. CICS 파일 제어는 VSAM 데이터를 관리합니다.

  CICS는 또한 두 가지 전용 파일 구조를 구현하며 구조를 조작하기 위한 명령을 제공합니다.

  임시 저장영역

  임시 저장영역(TS)은 여러 트랜잭션이 데이터를 사용할 수 있도록 하기 위한 수단입니다. 데이터는 프로그램에서 필요한 대로 작성되는 큐에 보존됩니다. 큐는 순차적으로 또는 항목 번호로 액세스할 수 있습니다.

  임시 저장영역 큐는 기본 메모리에 있거나 저장 장치에 기록될 수 있습니다.

  임시 저장영역 큐는 이름 지정된 스크래치패드로 간주될 수 있습니다.

  트랜지언트 데이터

  트랜지언트 데이터(TD)는 또한 여러 트랜잭션이 사용할 수 있으며 큐에 보존됩니다. 그러나 TS 큐와 달리 TD 큐는 사전 정의해야 하며 순차적으로만 읽을 수 있습니다. 각 항목은 읽을 때 큐에서 제거됩니다.

  트랜지언트 데이터 큐는 항상 데이터 세트에 기록됩니다. 특정 수의 항목을 기록하면 특정 트랜잭션이 시작되기 위해 트랜지언트 데이터 큐가 트리거로서 활동하도록 정의할 수 있습니다. 예를 들어, 트리거된 트랜잭션은 큐를 처리할 수 있습니다.

- 데이터베이스 제품과의 인터페이스를 통해 다른 데이터베이스(DB2 포함)의 데이터 액세스

통신 서비스

CICS는 SNA 및 TCP/IP 프로토콜을 사용하여 다양한 단말기(디스플레이, 프린터, 워크스테이션)에 대한 액세스 권한을 부여하는 명령을 제공합니다. CICS 단말기 제어는 SNA 및 TCP/IP 네트워크 관리를 제공합니다.

SNA 프로토콜을 사용하여 고급 프로그램간 통신(APPC) 명령으로 원격 시스템의 다른 프로그램을 시작하고 통신하는 프로그램을 작성할 수 있습니다. CICS APPC는 피어 투 피어 분산 애플리케이션 모델을 구현합니다.

다음 CICS 전용 통신 서비스가 제공됩니다.

Function Shipping
원격 CICS 영역에서 기존으로 정의되는 자원(파일, 큐, 프로그램) 액세스를 위한 프로그램 요청은 CICS에서 소유 영역으로 자동 라우트됩니다.

DPL(Distributed Program Link)
원격 CICS 영역에서 기존으로 정의된 프로그램에 대한 프로그램 링크 요청은 고유 영역으로 자동 라우트됩니다. CICS는 분산 애플리케이션의 무결성을 유지하기 위한 명령을 제공합니다.

비동기 처리
CICS는 프로그램이 동일한 또는 원격 CICS 영역에서 다른 트랜잭션을 시작하고 선택적으로 데이터를 전달할 수 있는 명령을 제공합니다. 새 트랜잭션은 독립적으로 새 태스크에 예약됩니다. 이 기능은 다른 소프트웨어 제품이 제공하는 포크 오퍼레이션과 유사합니다.

트랜잭션 라우팅
원격 CICS 영역에서 기존으로 정의되는 트랜잭션을 실행하는 요청은 소유 영역으로 자동 라우트됩니다. 사용자에 대한 응답의 경로는 요청을 수신한 영역으로 다시 지정됩니다.

작업 단위 서비스

CICS가 트랜잭션을 실행하기 위해 새 태스크를 작성하면 새 작업 단위(UOW)가 자동으로 시작됩니다. 즉, BEGIN 명령이 필요하지 않으므로 CICS가 제공하지 않습니다. CICS 트랜잭션은 항상 트랜잭션 내에서 실행됩니다.

CICS는 복구 가능 작업 완료를 확약하거나 롤백하기 위해 SYNCPOINT 명령을 제공합니다. 동기점이 완료되면 CICS가 자동으로 다른 작업 단위를 시작합니다. SYNCPOINT 명령을 실행하지 않고 프로그램을 종료하면 CICS가 암시적 동기점을 취하고 트랜잭션 확약을 시도합니다.

확약 범위에는 복구 가능으로 정의된 모든 CICS 자원과 CICS에서 제공되는 인터페이스를 통해 관심항목을 등록한 다른 자원 관리자가 포함됩니다.

프로그램 서비스

CICS는 프로그램이 다른 프로그램에 대한 제어를 링크하거나 전송하고 리턴할 수 있는 명령을 제공합니다.

진단 서비스

CICS는 프로그램을 추적하고 덤프를 생성하기 위해 사용할 수 있는 명령을 제공합니다.

CICS의 Java 런타임 환경

CICS는 스레드세이프 상태의 Java 애플리케이션을 실행하기 위한 JVM 서버 환경을 제공합니다. 스레드세이프 상태가 아닌 애플리케이션은 JVM 서버를 사용할 수 없습니다.

JVM 서버는 단일 JVM에서 태스크를 실행할 수 있는 런타임 환경입니다. 이 환경에서는 각 Java 태스크에 필요한 가상 저장영역의 크기가 감소하며 CICS에서 많은 태스크를 동시에 실행할 수 있습니다.

CICS 태스크는 동일한 JVM 서버 프로세스에서 스레드로 동시에 실행됩니다. JVM은 여러 애플리케이션을 동시에 실행할 수 있는 모든 CICS 태스크가 공유합니다. 모든 정적 데이터와 정적 클래스 또한 공유됩니다. 따라서 CICS에서 JVM 서버를 사용하려면 Java 애플리케이션이 스레드세이프(threadsafe) 상태여야 합니다. 각 스레드는 T8 TCB에서 실행되며 JCICS API를 사용하여 CICS 서비스에 액세스할 수 있습니다.

애플리케이션에서 System.exit() 메소드를 사용하지 마십시오. 이 메소드를 실행하면 JVM 서버와 CICS가 모두 종료되므로 애플리케이션의 상태와 사용 가능성에 영향을 줍니다.

멀티스레드된 애플리케이션

새 스레드를 시작하거나 스레드를 시작하는 라이브러리를 호출하는 애플리케이션 코드를 작성할 수 있습니다. 애플리케이션에서 스레드를 작성하려는 경우 기본적으로 OSGi 레지스트리에서 일반 ExecutorService를 사용합니다. 애플리케이션이 JVM 서버에서 실행되는 경우 ExecutorService는 자동으로 CICS ExecutorService를 사용하여 CICS 스레드를 작성합니다. 이 접근 방법은 애플리케이션이 다른 환경으로 쉽게 이식될 수 있으므로 특정 JCICS API 메소드를 사용하지 않아도 됨을 의미합니다.

그러나 CICS에 고유한 애플리케이션을 작성하는 경우 JCICS API에서 CICSExecutorService 클래스를 사용하도록 선택하여 새 스레드를 요청할 수 있습니다.

어떤 방법을 선택하더라도 새로 작성된 스레드는 CICS 태스크로 실행되며 CICS 서비스에 액세스할 수 있습니다. JVM 서버를 사용할 수 없는 경우 CICS는 JVM에서 실행되는 모든 CICS 태스크가 완료될 때까지 대기합니다. ExecutorService 또는

CICSExecutorService 클래스를 사용함으로써 CICS가 실행 중인 태스크를 인식할 수 있으며 JVM 서버가 종료되기 전에 애플리케이션 작업이 완료되는지 확인할 수 있습니다.

JCICS 오브젝트는 오브젝트를 작성한 태스크에서만 사용할 수 있습니다. 태스크 간에 오브젝트를 공유하려고 시도하면 예상치 못한 결과가 나타날 수 있습니다.

CICS ExecutorService 사용에 대한 세부사항은 45 페이지의 『스레드』의 내용을 참조하십시오.

JVM 서버 시동 및 종료

정적 데이터는 JVM 서버에서 실행되는 모든 스레드가 공유하므로 OSGi 번들 활성기 클래스를 작성하여 정적 데이터를 초기화하고 JVM이 종료될 때 올바른 상태를 유지하도록 할 수 있습니다. JVM 서버는 예를 들어, JVM 구성을 변경하거나 문제점을 수정하기 위해 관리자가 사용 불가능으로 설정할 때까지 실행됩니다. 번들 활성기 클래스를 제공함으로써 상태가 애플리케이션에 올바르게 설정되었는지 확인할 수 있습니다. CICS에는 JVM 서버를 계속 시작 또는 중지하기 전에 이러한 클래스가 완료될 때까지 대기하는 시간을 지정하는 제한시간이 있습니다. 시동 및 종료 클래스에서는 JCICS를 사용할 수 없습니다.

CICS Explorer SDK를 사용하여 애플리케이션 개발

CICS Explorer SDK(Software Development Kit)는 OSGi 및 웹 프로젝트에 대한 지원을 포함하여 Java 애플리케이션을 개발하여 CICS에 전개하기 위한 환경을 제공합니다.

이 태스크 정보

SDK를 사용하여 새 애플리케이션을 작성하거나 OSGi 스펙을 준수하기 위해 기존 Java 애플리케이션을 다시 패키지화할 수 있습니다. OSGi 서비스 플랫폼은 구성요소 모델을 사용하여 애플리케이션을 개발하고 해당 애플리케이션을 프레임워크에 OSGi 번들로 전개하기 위한 메커니즘을 제공합니다. *OSGi 번들*은 애플리케이션의 전개 단위이며 버전 제어 정보, 종속 항목, 애플리케이션 코드가 포함됩니다. OSGi의 주된 이점은 *Java 패키지*라는 올바르게 정의된 인터페이스를 통해서만 액세스하는 재사용 가능 구성요소에서 애플리케이션을 작성할 수 있다는 것입니다. 그런 다음 *OSGi 서비스*를 사용하여 Java 패키지에 액세스할 수 있습니다. 또한, Java 애플리케이션의 라이프 사이클과 종속 항목을 세부적으로 관리할 수 있습니다. OSGi로 애플리케이션 개발에 대한 정보는 OSGi Alliance 웹 사이트를 참조하십시오.

또한 SDK를 사용하여 Java servlet 및 JSP 페이지를 포함하는 OSGi 애플리케이션 프로젝트와 동적 웹 프로젝트 작업을 수행할 수 있습니다. CICS 서비스에 액세스하기 위해 JCICS를 사용하는 비즈니스 로직과 최신 웹 계층을 포함하는 애플리케이션을 작성

할 수 있습니다. 웹 애플리케이션이 다른 OSGi 번들에서 코드에 액세스해야 하는 경우 OSGi 애플리케이션 프로젝트(EBA 파일)로 전개되어야 합니다. 애플리케이션 Manifest에 나머지 OSGi 번들을 포함하거나 나머지 번들을 Liberty bundle_repository에 공통 라이브러리로 설치해야 합니다. EBA 파일에는 애플리케이션에 시작점을 제공하고 해당 시작점을 웹 브라우저에 URL로 표시하기 위한 웹 사용 OSGi 번들(WAB 파일)이 포함되어야 합니다.

SDK를 사용하여 CICS의 지원되는 릴리스에서 실행할 Java 애플리케이션을 개발할 수 있습니다. CICS 릴리스마다 다른 Java 버전을 지원하며 CICS의 특성을 더 지원하기 위해 향후 릴리스에서 JCICS API가 확장되었을 수도 있습니다. 잘못된 클래스 사용을 방지하기 위해 SDK는 대상 플랫폼을 설정하는 특성을 제공합니다. 개발하는 CICS 릴리스를 정의할 수 있으며 SDK는 사용할 수 없는 Java 클래스를 자동으로 숨깁니다.

SDK 도움말은 애플리케이션을 개발 및 전개하기 위해 다음 단계를 각각 수행할 수 있는 방법에 대한 전체 세부사항을 제공합니다.

프로시저

1. Java 개발을 위한 대상 플랫폼을 설정하십시오.

   대상 플랫폼에서는 애플리케이션 환경에서 CICS의 대상 릴리스에 적합한 Java 클래스만 사용해야 합니다.

2. Java 애플리케이션 개발을 위한 OSGi 번들 프로젝트 또는 플러그인 프로젝트를 작성하십시오.

   a. 프로젝트의 기본 버전은 `1.0.0.qualifier`입니다. **버전** 필드에서 버전 번호의 끝에서 ".qualifier"를 제거하십시오. qualifier는 현재 지원하지 않습니다.

3. 우수 사례를 사용하여 Java 애플리케이션을 개발하십시오. CICS용 Java 애플리케이션을 처음 개발하는 경우 CICS Explorer SDK와 함께 제공되는 예제를 사용하여 시작할 수 있습니다. Java 애플리케이션에서 JCICS를 사용하려면 com.ibm.cics.server 패키지를 가져와야 합니다.

4. 옵션: 동적 웹 애플리케이션(WAR) 또는 웹 사용 OSGi 번들 프로젝트(WAB)를 작성하여 나중에 애플리케이션 프리젠테이션을 개발하십시오. servlet과 JSP 페이지를 동적 웹 프로젝트에서 작성할 수 있습니다. WAR 파일의 경우 또한 Liberty API 번들에 대한 액세스를 제공하도록 대상 플랫폼을 수정해야 합니다. 세부사항은 대상 환경 설정을 참조하십시오.

5. 전개할 애플리케이션을 패키지화하십시오.

   a. 웹 사용 OSGi 번들 프로젝트(WAB)를 전개하는 경우 OSGi 애플리케이션 프로젝트(EBA)를 작성하십시오.

   b. EBA 또는 웹 애플리케이션(WAR 파일)을 참조하는 하나 이상의 CICS 번들 프로젝트를 작성하십시오. CICS 번들은 CICS의 애플리케이션에 대한 전개 단

위입니다. 함께 갱신 및 관리하려는 웹 애플리케이션을 CICS 번들 프로젝트에 포함하십시오. 애플리케이션을 전개하려는 JVMSERVER 자원의 이름을 알아야 합니다.

CICS 자원의 서브세트(예: PROGRAM, URIMAP, TRANSACTION 자원)를 CICS 번들 프로젝트에 추가할 수도 있습니다. 이러한 자원은 애플리케이션의 일부로서 동적으로 설치 및 관리됩니다.

c. 옵션: 애플리케이션을 CICS 플랫폼에 전개하려면 CICS 번들을 참조하는 애플리케이션 프로젝트를 작성하십시오. 애플리케이션은 CICS를 통해 CICSplex에서 애플리케이션을 전개 및 관리하기 위한 단일 관리 지점을 제공합니다. 자세한 정보는 애플리케이션 개발에서 전개를 위한 애플리케이션 패키징의 내용을 참조하십시오.

6. 애플리케이션 프로젝트 또는 CICS 번들 프로젝트를 내보내 Java 애플리케이션을 zFS에 전개하십시오. 또는 프로젝트를 소스 저장소에 저장하여 전개할 수 있습니다.

결과

CICS Explorer SDK를 사용하여 애플리케이션을 개발하고 내보냈습니다.

다음에 수행할 작업

애플리케이션을 JVM 서버에 설치하십시오. CICS에서 자원을 작성할 수 있는 권한이 없는 경우에는 시스템 프로그래머 또는 관리자가 대신 애플리케이션을 작성할 수 있습니다. 시스템 프로그래머 또는 관리자에게 내보낸 번들이 있는 위치와 대상 JVM 서버의 이름을 알려주어야 합니다. 세부사항은 140 페이지의 『JVM 서버에 OSGi 번들 전개』의 내용을 참조하십시오.

JVM의 상태 제어

CICS에서 실행할 Java 애플리케이션을 설계 및 개발하는 경우 애플리케이션이 JVM을 잘못된 상태로 유지하지 않는지 또는 원하지 않는 방식으로 상태를 수정하지 않는지 확인하십시오. CICS 서비스를 사용하여 JVM 상태를 쉽게 제어할 수 있습니다.

JVM 상태 보호

애플리케이션이 JVM 상태를 변경하는 경우 애플리케이션이 또한 원래 상태로 재설정되는지 확인하십시오. 예를 들어, 애플리케이션이 기본 시간대를 재설정하고 해당 시간대를 기반으로 계산을 수행할 수 있습니다. 동일한 JVM을 사용하는 다른 애플리케이션은 새 기본 시간대를 사용하며 이는 적절하지 않을 수 있습니다.

Java 애플리케이션은 JVM 서버 런타임 환경에서 실행되며 다른 스레드의 동일한 JVM에서 실행될 수도 있는 다른 애플리케이션과 분리되지 않습니다. 애플리케이션이 JVM을 변경하면 해당 JVM에서 실행되는 다른 모든 애플리케이션에 영향을 줍니다.

애플리케이션에서 System.exit() 메소드를 사용하지 마십시오. System.exit() 메소드를 사용하면 JVM 서버와 CICS가 모두 종료되어 애플리케이션 상태에 영향을 줄 수 있습니다.

JVM의 정적 상태 제어

JVM에서 원하지 않는 상태를 유지하지 마십시오. 상태는 JVM 서버 내 모든 실행 애플리케이션 간에 공유됩니다.

애플리케이션은 변경 가능한 클래스 필드의 상태에 종속되는 경우 고유한 정적 저장영역을 다시 초기화해야 합니다. JVM에는 모든 애플리케이션 및 시스템 클래스(애플리케이션에 영향을 줄 수 있지만 정적 초기화 프로그램에서 사용되는 애플리케이션과 값에서 명시적으로 사용되지 않는 클래스 포함)에 대한 정적 변수의 값이 존재합니다.

대부분의 경우 정적 변수는 저장영역 재초기화를 방지하기 위해 사용되므로 해당 변수를 지속시킴으로써 성능을 향상시킬 수 있습니다. 애플리케이션에서 해당 변수 값을 재설정해야 하는 경우 애플리케이션이 직접 값을 재설정해야 합니다. 애플리케이션 설계의 일부로 의도적으로 포함되지 않은 변경 가능한 클래스 필드와 정적 초기화 프로그램을 식별하여 제거하십시오.

가능한 경우 클래스 필드를 개인용 또는 최종으로 정의하십시오. 원시 메소드를 최종 클래스 필드에 쓸 수 있으며 비개인용 메소드가 클래스 필드가 참조하는 오브젝트를 획득하고 오브젝트 또는 배열의 상태를 변경할 수 있습니다.

특정 프로그램 호출에서 다음 호출까지 정보를 유지하려는 경우 Java 애플리케이션을 쉽게 설계할 수 있도록 상태를 전달하는 기능을 사용할 수 있습니다. 정적 상태와 정적 상태를 통해 참조되는 오브젝트 인스턴스는 JVM에서 지속되므로 애플리케이션이 동일한 JVM에서 동일한 애플리케이션을 더 실행하기 위해 사용할 수 있는 지속적 항목을 작성할 수 있습니다.

예를 들어, 한 조작이 복잡한 데이터 구조를 구성하기 위해 DB2 정보를 읽습니다. 이 조작은 많은 비용이 소요되므로 절대적으로 필요한 경우에만 반복합니다. 복잡한 데이터 구조는 애플리케이션 정적 저장영역에 저장될 수 있으며 나중에 동일한 JVM에서 애플리케이션을 실행할 때 액세스할 수 있으므로 불필요한 초기화를 방지할 수 있습니다. 오브젝트가 정적 저장영역, 즉 정적 클래스 필드에 고착되는 경우에는 사용공간 정리를 위한 후보가 되지 않습니다.

JVM 서버에서는 시스템 프로그래머가 JVM 서버를 사용 안함으로 설정할 때까지 모든 애플리케이션에 대해 정적 상태가 지속됩니다. JVM 서버를 다시 시작하는 동안 오브젝트 상태를 유지하기 위해 OSGi 번들 활성기 클래스를 제공할 수 있습니다. 이러한 클래스에는 JCICS 호출이 포함될 수 없습니다.

사용 후 DB2 연결, 소켓, 기타 태스크 수명 시스템 자원 닫기

Java 애플리케이션은 JVM 서버 런타임 환경에서 실행되므로 다른 애플리케이션에서 DB2에 대한 여러 연결을 가질 수 있습니다. 따라서 DB2에서 태스크가 완료된 경우에는 연결을 닫는 것이 가장 올바르지만 반드시 필요하지는 않습니다. 태스크가 완료되면 연결이 삭제되기 때문입니다.

java.net 패키지를 사용하여 소켓을 관리하기 위해 애플리케이션에서 스레드를 시작하는 경우에는 애플리케이션이 연결을 관리하고 닫아야 합니다. java.net 클래스를 사용하여 작성된 소켓은 CICS 소켓 도메인이 아닌 JVM의 고유 소켓 기능을 사용합니다. CICS는 이러한 소켓을 사용하여 수행되는 통신을 관리 또는 모니터할 수 없습니다.

애플리케이션이 사용하는 다른 태스크 수명 시스템 자원에도 동일한 사항이 적용되며 사용 후에는 해제되어야 합니다.

가능한 스레드세이프(threadsafe) 상태 이슈에 대한 테스트 애플리케이션

JVM 서버 런타임 환경은 스레드세이프(threadsafe) 상태가 아닌 Java 애플리케이션 사용을 지원하지 않으므로 항상 스레드세이프(threadsafe) 상태의 Java 애플리케이션을 작성해야 합니다. 여러 스레드가 동시에 사용하더라도 오브젝트가 항상 올바른 상태를 유지해야 합니다. Java 애플리케이션이 지정된 JVM에서의 최초 사용 시 올바르게 작동하지만 이후 사용 시 올바르게 작동하지 않는 경우 스레드세이프(threadsafe) 이슈에 따른 문제점일 수 있습니다.

Java 구조화 데이터와의 상호작용

CICS Java 프로그램은 원래 다른 프로그래밍 언어에 사용하도록 설계된 데이터와 자주 상호작용합니다. 예를 들어, Java 프로그램은 COBOL 카피북에 정의된 COMMAREA를 사용하여 COBOL 프로그램에 링크되거나 데이터가 C++ 헤더 파일을 사용하여 정의된 경우 VSAM 파일에서 레코드를 읽을 수 있습니다. 가져오기 프로그램을 사용하여 이러한 양식의 구조화된 데이터와 상호작용할 수 있습니다.

JZOS 및 J2C를 사용하여 Java로 애플리케이션 데이터 가져오기

CICS는 카피북 가져오기 프로그램을 지원하므로 Java에서 다른 프로그래밍 언어의 구조화 데이터를 사용할 수 있습니다. 지원되는 가져오기 프로그램은 JZOS 도구와 Rational이 제공합니다. Rational 도구는 J2C라고도 하는 JCA(Java EE Connector Architecture)를 사용합니다.

가져오기 프로그램은 소스 프로그램에 포함된 데이터 유형을 맵핑하므로 애플리케이션이 데이터 구조의 개별 필드에 액세스할 수 있습니다. JZOS 또는 Rational J2C 도구로 데이터와 상호작용하여 Java 클래스를 생성할 수 있으므로 Java와 CICS의 다른 프로그램 간에 데이터를 전달할 수 있습니다.

CICS는 다음 가져오기 프로그램의 Java 아티팩트를 지원합니다.

- RAD(Rational Application Developer) 및 Rational Developer for System z의 J2C 도구로부터의 데이터 바인딩 Bean
- IBM JZOS Batch Toolkit for z/OS SDK의 레코드

IBM Redbooks® 서적인 CICS용 Java 애플리케이션 개발은 기존 COBOL 애플리케이션을 조작하는 Heritage Trader 애플리케이션이라는 예제 애플리케이션을 사용합니다. 다음 주제에 대한 정보가 제공됩니다.

- JZOS와 J2C를 설치하기 위한 지시사항
- JCICS로 COBOL 애플리케이션 이주
- J2C용 Java 데이터 바인딩 클래스 작성
- JZOS로 랩퍼 클래스 생성
- JCICS API를 사용하여 웹, 파일, DB2 액세스를 위한 예제 구현

J2C 요구사항

엔터프라이즈 애플리케이션을 작성하기 위해 사용할 수 있는 Java EE Connector 아티팩트를 작성할 수 있습니다. RAD J2C 마법사를 사용하면 COBOL 및 다른 애플리케이션 데이터 구조에 맵핑하는 클래스 또는 클래스 세트를 쉽게 작성할 수 있습니다.

Rational J2C 가져오기 프로그램을 사용하려면 Windows 또는 Linux 워크스테이션에 RAD가 필요합니다.

JZOS 요구사항

IBM JZOS Batch Toolkit for z/OS SDK는 z/OS에서 Java 일괄처리 기능을 제공하는 도구 세트입니다. JZOS에는 Java 애플리케이션을 일괄처리 작업 또는 시작된 태스크로 직접 실행하기 위한 실행 프로그램과, Java 애플리케이션에서 직접 사용 가능한 기존 z/OS 데이터 및 키 시스템 서비스에 액세스할 수 있는 Java 메소드 세트가 포함됩니다.

JZOS는 COBOL 카피북 및 어셈블러 DSECT로부터의 자동 레코드 클래스 생성을 지원합니다.

JZOS 다운로드에는 PDF 형식의 *JZOS COBOL Record Generator 사용자 안내서* 및 *JZOS Assembler Record Generator 사용자 안내서*가 포함됩니다.

IBM Redbooks

CICS용 Java 애플리케이션 개발

Java EE 커넥터 구조를 나타내는 CICS용 Java 커넥터

z/OS 볼륨 2의 Java 독립형 애플리케이션

J2C 정보

RAD: EIS(Enterprise Information System) 연결

RAD: COBOL 가져오기 프로그램 개요

CICS Transaction Gateway 프로그래밍 안내서

JZOS 정보

JZOS Java 실행 프로그램 및 툴킷 개요

IBM SDK for z/OS, Java Technology Edition의 JZOS 일괄처리 실행 프로그램 및 툴킷 기능 설치 및 사용자 안내서

JCICS를 사용한 Java 개발

CICS 서비스에 액세스하기 위해 CICS Java 클래스 라이브러리(JCICS)를 사용하는 Java 애플리케이션을 작성할 수 있습니다. JCICS는 Java에서 다른 CICS 지원 언어 (예: COBOL)에 대해 제공되는 **EXEC CICS** API(Application Programming Interface)에 해당합니다.

JCICS를 사용하여 CICS 자원에 액세스하고 다른 언어로 작성된 프로그램과 상호 작용하는 Java 애플리케이션을 작성할 수 있습니다. **EXEC CICS** API의 기능은 대부분 지원됩니다. 라이브러리는 CICS를 포함하는 `com.ibm.cics.server.jar` 파일에 CICS Explorer SDK와 함께 제공됩니다.

41 페이지의 『CICS용 Java 클래스 라이브러리(JCICS)』
JCICS는 **EXEC CICS** API 명령의 기능을 대부분 지원합니다.

46 페이지의 『데이터 인코딩』
JVM은 문자 인코딩을 위해 CICS의 다른 코드 페이지를 사용할 수 있습니다. CICS는 항상 EBCDIC 코드 페이지를 사용해야 하지만 JVM은 ASCII와 같은 다른 인코딩을 사용할 수 있습니다. JCICS API를 사용하는 애플리케이션을 개발하는 경우 올바른 인코딩을 사용하는지 확인해야 합니다.

48 페이지의 『JCICS API 서비스 및 예제』
Java 이외 프로그램이 **EXEC CICS** API를 통해 사용할 수 있는 많은 선택항목과 서비스를 Java 프로그램이 JCICS를 통해 사용할 수 있습니다. 이 정보는 **EXEC CICS** API와 JCICS 사이에 맵핑을 제공하며 예제 Java 코드의 발췌본을 제공합니다.

75 페이지의『JCICS 사용』
JCICS 라이브러리에서 일반 Java 클래스와 같은 클래스를 사용합니다. 애플리케이션은 필수 유형의 참조를 선언하며 new 연산자를 사용하여 클래스의 새 인스턴스가 작성됩니다.

76 페이지의『Java 제한사항』
Java 애플리케이션을 개발할 때 다양한 제한사항이 적용됩니다. CICS에서 애플리케이션이 실행될 때는 다른 문제점이 발생할 수 있습니다.

CICS용 Java 클래스 라이브러리(JCICS)

JCICS는 **EXEC CICS** API 명령의 기능을 대부분 지원합니다.

JCICS 클래스에 대한 자세한 정보는 클래스 정의에서 생성된 Javadoc에 나와 있습니다. Javadoc은 JCICS 클래스 참조를 참고하십시오.

42 페이지의『JavaBeans』
JCICS의 일부 클래스는 JavaBeans로 사용할 수 있습니다. 이는 해당 클래스를 Eclipse와 같은 애플리케이션 개발 도구로 사용자 정의하고 직렬화할 수 있으며 JavaBeans API를 사용하여 조작할 수 있음을 의미합니다.

42 페이지의『라이브러리 구조』
각 JCICS 라이브러리 구성요소는 네 가지 카테고리(인터페이스, 클래스, 예외 또는 오류)로 분류됩니다.

43 페이지의『CICS 자원』
프로그램 또는 임시 저장영역 큐와 같은 CICS 자원은 자원 이름과 같은 다양한 등록 정보 값으로 식별되는 해당 Java 클래스의 인스턴스로 표시됩니다.

43 페이지의『데이터 전달 인수』
채널과 컨테이너를 사용하거나 통신 영역(COMMAREA)을 사용하여 프로그램 간에 데이터를 전달할 수 있습니다.

44 페이지의『순차 클래스』
JCICS 순차 클래스의 목록입니다.

45 페이지의『Task.out 및 Task.err』
각 Java 관련 CICS 태스크마다 CICS가 표준 출력 및 표준 오류 스트림으로 사용할 수 있는 두 개 Java PrintWriters 클래스를 자동으로 작성합니다. 표준 출력 및 표준 오류 스트림은 Task 클래스인 out 및 err의 공용 필드입니다.

45 페이지의『스레드』
JVM 서버 환경에서는 OSGi 프레임워크에서 실행되는 애플리케이션이 ExecutorService를 사용하여 CICS 태스크에서 실행되는 스레드를 비동기식으로 작성할 수 있습니다.

JavaBeans

JCICS의 일부 클래스는 JavaBeans로 사용할 수 있습니다. 이는 해당 클래스를 Eclipse와 같은 애플리케이션 개발 도구로 사용자 정의하고 직렬화할 수 있으며 JavaBeans API를 사용하여 조작할 수 있음을 의미합니다.

JCICS에서 사용할 수 있는 JavaBeans은 다음과 같습니다.

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

이러한 Bean은 이벤트를 정의하지 않으며 등록 정보와 메소드로 구성됩니다. 또한 다음 세 가지 방법 중 하나로 런타임에 인스턴스화될 수 있습니다.

- 클래스의 new 메소드를 직접 호출하여. 이 방법이 선호됩니다.
- 등록 정보 값을 수동으로 설정하고 클래스 이름에 대해 Beans.instantiate()를 호출하여
- 등록 정보 값을 설계 시 설정하고 `.ser` 파일의 Beans.instantiate()를 호출하여

처음 두 가지 선택항목 중 하나를 선택하는 경우에는 런타임에 적절한 set 메소드를 호출하여 CICS 자원 이름을 포함하는 등록 정보 값을 설정해야 합니다.

라이브러리 구조

각 JCICS 라이브러리 구성요소는 네 가지 카테고리(인터페이스, 클래스, 예외 또는 오류)로 분류됩니다.

인터페이스
: 몇몇 인터페이스는 상수 세트를 정의하기 위해 제공됩니다. 예를 들어, TerminalSendBits 인터페이스는 `java.util.BitSet`를 구성하는 데 사용되는 상수 세트를 제공합니다.

클래스
: 제공된 클래스는 JCICS 기능을 대부분 제공합니다. `API` 클래스는 ABEND와 예외를 제외한 CICS API의 일부에 해당하는 모든 클래스에 대한 공통 초기화를 제공하는 추상 클래스입니다. 예를 들어, `Task` 클래스는 CICS 태스크에 해당하는 메소드 및 변수 세트를 제공합니다.

오류 및 예외

Java 언어는 예외와 오류를 모두 Throwable 클래스의 서브클래스로 정의합니다. JCICS는 CicsError를 Error의 서브클래스로 정의합니다. CicsError는 심각한 오류에 사용되는 다른 모든 CICS 오류 클래스의 수퍼클래스입니다.

JCICS는 CicsException을 Exception의 서브클래스로 정의합니다. CicsException은 모든 CICS 예외 클래스(InvalidQueueIdException과 같이 CICS QIDERR 조건을 나타내는 CicsConditionException 클래스)의 수퍼클래스입니다.

자세한 정보는 52 페이지의 『오류 처리 및 비정상 종료』 의 내용을 참조하십시오.

CICS 자원

프로그램 또는 임시 저장영역 큐와 같은 CICS 자원은 자원 이름과 같은 다양한 등록 정보 값으로 식별되는 해당 Java 클래스의 인스턴스로 표시됩니다.

CICS Explorer, CEDA 트랜잭션 또는 CICSPlex® SM WUI를 사용하여 CICS 자원을 정의합니다. 암시적 원격 액세스를 사용하려면 원격 자원을 가리키는 자원을 로컬로 정의합니다.

CICS 자원 정의에 대한 자세한 정보는 *CICS 자원 정의 안내서CICSPlex 시스템 관리자 개념 및 계획* 매뉴얼을 참조하십시오.

데이터 전달 인수

채널과 컨테이너를 사용하거나 통신 영역(COMMAREA)을 사용하여 프로그램 간에 데이터를 전달할 수 있습니다.

COMMAREA를 사용하는 경우 한 번에 최대 32KB를 전달할 수 있습니다. 채널과 컨테이너를 사용하면 프로그램 간에 32KB보다 많이 전달할 수 있습니다. COMMAREA 또는 채널과 기타 모든 매개변수는 인수로서 해당 메소드에 전달됩니다.

메소드 중 다수는 과부하가 발생합니다. 즉, 유형과 그 수가 다른 하나 이상의 인수를 취하는 다른 버전을 가집니다. 인수가 없거나 최소 필수 인수를 갖는 하나의 메소드와 모든 인수를 갖는 다른 메소드가 존재할 수 있습니다. 예를 들어, Program 클래스에는 다음과 같은 다양한 link() 메소드가 포함됩니다.

link()

이 메소드는 데이터를 전달하는 COMMAREA 또는 다른 선택항목을 사용하지 않고 단순 LINK를 수행합니다.

link(com.ibm.cics.server.CommAreaHolder)

이 메소드는 데이터를 전달하기 위해 COMMAREA를 사용하되 다른 선택항목 없이 단순 LINK를 수행합니다.

link(com.ibm.cics.server.CommAreaHolder, int)

이 메소드는 데이터를 전달하기 위한 COMMAREA와 COMMAREA 내부 데이터의 길이를 지정하기 위한 DATALENGTH 값을 사용하여 분산 LINK를 수행합니다.

link(com.ibm.record.IByteBuffer)

이 메소드는 VisualAge for Java와 함께 제공되는 Java 레코드 프레임워크의 IByteBuffer 인터페이스를 구현하는 오브젝트를 사용하여 LINK를 수행합니다.

link(com.ibm.cics.server.Channel)

이 메소드는 하나 이상의 컨테이너에서 데이터를 전달하기 위해 채널을 사용하여 LINK를 수행합니다.

관련 정보:

54 페이지의『채널 및 컨테이너 예제』

*컨테이너*는 프로그램 간에 정보를 전달하기 위해 설계된 이름 지정된 데이터 블록입니다. 컨테이너는 *채널* 세트로 그룹화됩니다. 이 정보는 Java 애플리케이션에서 채널과 컨테이너를 사용하는 방법을 설명하고 일부 코드 예제를 제공합니다.

순차 클래스

JCICS 순차 클래스의 목록입니다.

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

- TSQType

관련 참조:

70 페이지의『직렬화 서비스』
JCICS는 자원 사용을 태스크로 예약할 수 있는 CICS 직렬화 서비스를 지원합니다.

Task.out 및 Task.err

각 Java 관련 CICS 태스크마다 CICS가 표준 출력 및 표준 오류 스트림으로 사용할 수 있는 두 개 Java `PrintWriters` 클래스를 자동으로 작성합니다. 표준 출력 및 표준 오류 스트림은 `Task` 클래스인 `out` 및 `err`의 공용 필드입니다.

CICS 태스크가 단말기(이 경우 단말기 이름은 *프린시펄 기능*)에서 구동되는 경우 CICS는 표준 출력 및 표준 오류 스트림을 태스크 단말기에 맵핑합니다.

태스크의 프린시펄 기능이 단말기가 아닌 경우에는 표준 출력 및 표준 오류 스트림이 System.out 및 System.err로 전송됩니다.

스레드

JVM 서버 환경에서는 OSGi 프레임워크에서 실행되는 애플리케이션이 ExecutorService를 사용하여 CICS 태스크에서 실행되는 스레드를 비동기식으로 작성할 수 있습니다.

CICS는 Java ExecutorService 인터페이스의 구현을 제공합니다. 이 구현은 JCICS API를 사용하여 CICS 서비스에 액세스할 수 있는 스레드를 작성합니다. JVM 서버는 시동 시 CICS ExecutorService를 OSGi 서비스로 등록합니다. JCICS를 사용할 수 있는 태스크를 작성하려면 Java Thread 클래스 대신 이 서비스를 사용하십시오.

CICS에서 제공하는 ExecutorService는 OSGi 프레임워크에서 높은 우선순위로 등록되므로 애플리케이션이 스레드를 작성하는 데 사용할 수 있습니다. 일반적으로 애플리케이션은 특정 구현을 사용하기 위해 서비스를 필터링하지 않는 경우 가장 높은 우선순위 ExecutorService를 사용합니다.

애플리케이션에서 스레드를 작성하려는 경우 기본적으로 OSGi 레지스트리에서 일반 ExecutorService를 사용합니다. 애플리케이션이 JVM 서버에서 실행되는 경우 OSGi 레지스트리는 자동으로 CICS ExecutorService를 사용하여 CICS 스레드를 작성합니다. 이 접근 방법은 애플리케이션이 구현에서 분리되므로 스레드를 작성하기 위해 JCICS API 메소드를 사용하지 않아도 됨을 의미합니다.

그러나 CICS에 고유한 애플리케이션을 작성하는 경우 JCICS API에서 CICSExecutorService 클래스를 사용하도록 선택하여 새 스레드를 요청할 수 있습니다.

CICSExecutorService

이 클래스는 java.util.concurrent.ExecutorService 인터페이스를 구현합니다. CICSExecutorService 클래스는 새 JCICS 사용 스레드에서 실행할 Runnable Java 오

브젝트를 제출하는 데 사용할 수 있는 정적 메소드인 runAsCICS()를 제공합니다. runAsCICS() 메소드는 애플리케이션의 CICSExecutorService 인스턴스를 얻기 위해 OSGi 레지스트리 찾아보기를 수행하는 유틸리티 메소드입니다.

이 클래스는 Java ExecutorService 인터페이스의 구현으로 등록되므로 ExecutorService를 요청하는 애플리케이션에는 JVM 서버에서 실행될 때만 CICSExecutorService가 제공됩니다.

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

제한사항

JCICS를 사용할 수 있는 스레드를 작성하려면 execute() 메소드를 사용해야 합니다. submit() 메소드를 사용하는 경우 애플리케이션은 JCICS를 실행할 수 없는 Java 스레드를 가져옵니다.

OSGi 프레임워크에서 실행되지 않는 애플리케이션의 경우(예를 들어, Axis2 Java 프로그램) ExecutorService를 사용할 수 없으므로 초기 애플리케이션 스레드에서만 JCICS에 액세스할 수 있습니다. 또한 다음 조치를 수행하기 전에 초기 스레드 이외 다른 모든 스레드가 완료되는지 확인해야 합니다.

- com.ibm.cics.server.Program 클래스의 link 메소드
- com.ibm.cics.server.TerminalPrincipalFacility 클래스의 setNextTransaction(String) 메소드
- com.ibm.cics.server.TerminalPrincipalFacility 클래스의 setNextCOMMAREA(byte[]) 메소드
- com.ibm.cics.server.Task 클래스의 commit() 메소드
- com.ibm.cics.server.Task 클래스의 rollback() 메소드
- com.ibm.cics.server 클래스에서 `AbendException` 예외 리턴

관련 참조:

70 페이지의『스레드 및 태스크 예제』
CICSExecutorService를 사용하여 CICS에서 태스크를 시작하는 스레드를 작성할 수 있습니다. 이 서비스를 사용하여 스레드를 작성하는 경우 JCICS API를 사용하여 CICS 서비스를 액세스할 수 있는 CICS 태스크가 작성됩니다.

데이터 인코딩

JVM은 문자 인코딩을 위해 CICS의 다른 코드 페이지를 사용할 수 있습니다. CICS는 항상 EBCDIC 코드 페이지를 사용해야 하지만 JVM은 ASCII와 같은 다른 인코딩을 사용할 수 있습니다. JCICS API를 사용하는 애플리케이션을 개발하는 경우 올바른 인코딩을 사용하는지 확인해야 합니다.

JCICS API는 기본 JVM이 아닌 CICS 영역에 지정된 코드 페이지를 사용합니다. 따라서 JVM이 다른 파일 인코딩을 사용하는 경우 애플리케이션이 다른 코드 페이지를 처리해야 합니다. CICS는 CICS가 사용하는 코드 페이지 판별을 용이하게 하기 위해 여러 가지 Java 등록 정보를 제공합니다.

- **com.ibm.cics.jvmserver.supplied.ccsid** 등록 정보는 CICS 영역에 지정된 코드 페이지를 리턴합니다. 기본적으로 JCICS API는 문자 인코딩을 위해 이 코드 페이지를 사용합니다. 그러나 이 값은 JVM 서버 구성에서 덮어쓸 수 있습니다.
- **com.ibm.cics.jvmserver.override.ccsid** 등록 정보는 JVM 프로파일에서 덮어쓰기 값을 리턴합니다. 이 값은 CICS 영역이 사용하는 코드 페이지가 아닌 JCICS API가 문자 인코딩에 사용하는 코드 페이지입니다.
- **com.ibm.cics.jvmserver.local.ccsid** 등록 정보는 JCICS API가 JVM 서버에서 문자 인코딩에 사용하는 코드 페이지를 리턴합니다.

JCICS에 대한 인코딩을 변경하기 위해 Java 애플리케이션에서 이러한 등록 정보를 설정할 수 없습니다. 코드 페이지를 변경하려면 시스템 관리자에게 JVM 프로파일을 갱신하여 JVM 시스템 등록 정보 **-Dcom.ibm.cics.jvmserver.override.ccsid**를 추가하도록 요청해야 합니다.

인코딩 예제

java.lang.String 매개변수를 입력으로 허용하는 모든 JCICS 메소드는 데이터가 CICS로 전달되기 전에 올바른 코드 페이지로 자동으로 인코딩됩니다. 마찬가지로 JCICS API에서 리턴되는 모든 java.lang.String 값은 올바른 코드 페이지에서 인코딩됩니다. JCICS API는 대부분의 클래스에 헬퍼 메소드를 제공합니다. 이러한 헬퍼 메소드는 문자열 및 데이터 작업을 수행하여 애플리케이션 대신 코드 페이지를 판별 및 설정합니다.

애플리케이션이 String.getBytes() 또는 new String(byte[] bytes) 메소드를 사용하는 경우 애플리케이션이 메소드가 올바른 인코딩을 사용하는지 확인해야 합니다. 애플리케이션에서 이러한 메소드를 사용하려는 경우 Java 등록 정보를 사용하여 데이터를 올바르게 인코딩할 수 있습니다.

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

다음 예제는 애플리케이션이 COMMAREA에서 필드를 읽을 때 JCICS 인코딩을 사용하는 방법을 보여줍니다.

```
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 서비스 및 예제

Java 이외 프로그램이 **EXEC CICS** API를 통해 사용할 수 있는 많은 선택항목과 서비스를 Java 프로그램이 JCICS를 통해 사용할 수 있습니다. 이 정보는 **EXEC CICS** API와 JCICS 사이에 맵핑을 제공하며 예제 Java 코드의 발췌본을 제공합니다.

50 페이지의 『Java 프로그램에서의 CICS 예외 처리』
CICS ABEND 및 예외가 Java 예외 처리 구조에 통합되어 CICS에서 발생하는 문제점을 처리합니다.

52 페이지의 『Java 웹 애플리케이션에서의 CICS 예외 처리』
CICS ABEND 및 예외가 Liberty 웹 컨테이너에 대한 Java 예외 처리 구조에 통합되어 CICS 애플리케이션에서 발생하는 문제점을 처리합니다. 웹 애플리케이션에서 처리되지 않는 Java 예외는 웹 컨테이너가 발견하고 servlet 예외 처리 프로세스를 구동합니다. 이 처리의 일부로 CICS에서 확약되지 않은 CICS 작업 단위를 롤백합니다.

52 페이지의 『오류 처리 및 비정상 종료』
Java 프로그램에서 ABEND를 시작하려면 Task.abend() 또는 Task.forceAbend() 메소드 중 하나를 호출해야 합니다.

53 페이지의 『APPC 맵핑 대화』
APPC 비맵핑 대화 지원은 JCICS API에서 사용할 수 없습니다.

54 페이지의 『BMS(Basic Mapping Support)』
BMS(Basic Mapping Support)는 CICS 프로그램과 단말기 장치 간의 API(Application Programming Interface)입니다. JCICS는 몇몇 BMS API(Application Programming Interface)를 지원합니다.

54 페이지의 『채널 및 컨테이너 예제』
*컨테이너*는 프로그램 간에 정보를 전달하기 위해 설계된 이름 지정된 데이터 블록입니다. 컨테이너는 *채널* 세트로 그룹화됩니다. 이 정보는 Java 애플리케이션에서 채널과 컨테이너를 사용하는 방법을 설명하고 일부 코드 예제를 제공합니다.

59 페이지의 『진단 서비스』
JCICS API(Application Programming Interface)는 이러한 CICS 추적 및 덤프 명령을 지원합니다.

59 페이지의 『문서 서비스』
이 섹션에서는 DOCUMENT API(Application Programming Interface)의 명령에 대한 JCICS 지원을 설명합니다.

59 페이지의 『환경 서비스』
CICS 환경 서비스는 애플리케이션과 관련된 CICS 데이터 영역, 매개변수, 자원 속성에 대한 액세스를 제공합니다.

63 페이지의 『파일 서비스』
JCICS는 CICS 파일 및 인덱스의 각 유형에 대한 **EXEC CICS** API 명령에 맵핑되는 클래스와 메소드를 제공합니다.

66 페이지의『HTTP 및 TCP/IP 서비스』

HttpHeader, NameValueData, FormField 클래스의 Getter는 해당 API 명령에 대한 HTTP 헤더, 이름 및 값 쌍, 양식 필드 값을 리턴합니다.

67 페이지의『JTA(Java Transaction API)』

JTA(Java Transaction API)를 사용하여 트랜잭션 갱신을 여러 자원 관리자로 조정할 수 있습니다.

68 페이지의『프로그램 서비스』

JCICS는 CICS 프로그램 제어 명령인 LINK, RETURN 및 INVOKE APPLICATION을 지원합니다.

69 페이지의『스케줄링 서비스』

JCICS는 태스크에 대해 저장된 데이터를 검색하고 간격 제어 요청을 취소하며 지정된 시간에 태스크를 시작할 수 있는 CICS 스케줄링 서비스를 지원합니다.

70 페이지의『직렬화 서비스』

JCICS는 자원 사용을 태스크로 예약할 수 있는 CICS 직렬화 서비스를 지원합니다.

70 페이지의『저장영역 서비스』

CICS 서비스(예: **EXEC CICS** GETMAIN)를 사용한 명시적인 저장영역 관리는 지원되지 않습니다. 표준 Java 저장영역 관리 기능이 태스크 전용 저장영역에 대한 요구를 충족시키는 데 충분한지 확인해야 합니다.

70 페이지의『스레드 및 태스크 예제』

CICSExecutorService를 사용하여 CICS에서 태스크를 시작하는 스레드를 작성할 수 있습니다. 이 서비스를 사용하여 스레드를 작성하는 경우 JCICS API를 사용하여 CICS 서비스를 액세스할 수 있는 CICS 태스크가 작성됩니다.

71 페이지의『임시 저장영역 큐 서비스』

JCICS는 CICS 임시 저장영역 명령으로 DELETEQ TS, READQ TS, WRITEQ TS를 지원합니다.

72 페이지의『단말기 서비스』

JCICS는 이러한 CICS 단말기 서비스 명령을 지원합니다.

73 페이지의『트랜지언트 데이터 큐 서비스』

JCICS는 CICS 트랜지언트 데이터 명령인 DELETEQ TD, READQ TD, WRITEQ TD를 지원합니다. INTO 선택항목을 제외한 모든 선택항목이 지원됩니다.

74 페이지의『작업 단위(UOW) 서비스』

JCICS는 CICS SYNCPOINT 서비스를 위한 지원을 제공합니다.

74 페이지의『웹 서비스 예제』

JCICS는 애플리케이션에서 웹 서비스 작업을 위해 사용할 수 있는 모든 API 명령을 지원합니다.

Java 프로그램에서의 CICS 예외 처리

CICS ABEND 및 예외가 Java 예외 처리 구조에 통합되어 CICS에서 발생하는 문제점을 처리합니다.

모든 일반 CICS ABEND는 단일 Java 예외, `AbendException`에 맵핑되는 반면 각 CICS 조건은 개별 Java 예외에 맵핑됩니다. 따라서 Java의 ABEND 처리 모델이 다른 프로그래밍 언어와 유사하게 됩니다. 단일 핸들러가 모든 ABEND를 제어할 수 있으므로 핸들러가 특정 ABEND를 조회한 후 다음 작업을 결정해야 합니다.

CICS 자체가 조건을 나타내는 예외를 발견하면 ABEND로 전환됩니다.

Java 예외 처리는 다른 언어의 조건 처리 및 ABEND와 완전하게 통합되므로 ABEND가 표준 언어에 독립적인 방식으로 Java 및 Java가 아닌 프로그램 간에 전파될 수 있습니다. 조건을 유발하거나 발견한 프로그램을 떠나기 전에 ABEND에 조건이 맵핑됩니다.

그러나 다른 프로그래밍 언어의 경우 이상 종료 처리 모델에는 Java 예외 처리 구조 및 JavaAPI의 기반 기술 구현의 본질에 따른 몇 가지 차이점이 있습니다.

- 다른 프로그래밍 언어로 처리할 수 없는 ABEND를 Java 프로그램이 발견할 수 있습니다. 이러한 ABEND는 일반적으로 동기점 처리 중에 발생합니다. Java 애플리케이션을 인터럽트하는 이러한 ABEND를 방지하기 위해 확인되지 않은 예외의 확장으로 맵핑됩니다. 따라서 선언하거나 처리하지 않아도 됩니다.
- 프로그램 종료와 같은 여러 내부 CICS 이벤트 또한 Java 예외로 맵핑되어 Java 애플리케이션이 처리할 수 있습니다. 또한 일반적인 사례 중단을 방지하기 위해 이러한 이벤트가 확인되지 않은 예외 확장으로 맵핑되므로 처리 또는 선언하지 않아도 됩니다.

다음 세 가지 클래스 예외 계층은 CICS와 관련이 있습니다.

1. CicsError: java.lang.Error를 확장하며 AbendError 및 UnknownCicsError의 기반이 됩니다.
2. CicsRuntimeException: java.lang.RuntimeException을 확장하며 차례로 다음 클래스로 확장됩니다.

   AbendCancelException
   CICS ABEND CANCEL을 나타냅니다.

   AbendException
   일반 CICS ABEND를 나타냅니다.

   EndOfProgramException
   링크된 프로그램이 정상적으로 종료되었음을 나타냅니다.
3. CicsException: java.lang.Exception을 확장하며 서브클래스를 포함합니다.

CicsConditionException.

모든 CICS 조건의 기본 클래스입니다.

『CICS 오류 처리 명령』

CICS 조건 처리는 위와 같이 Java 예외 구조에 통합됩니다. Java에서 해당 "**EXEC CICS**" 명령이 지원되는 방식이 아래에 설명되어 있습니다.

『CICS 조건』

Java의 조건 처리 모델은 다른 CICS 프로그래밍 언어와 다릅니다.

CICS 오류 처리 명령:

CICS 조건 처리는 위와 같이 Java 예외 구조에 통합됩니다. Java에서 해당 "**EXEC CICS**" 명령이 지원되는 방식이 아래에 설명되어 있습니다.

HANDLE ABEND

CICS 지원 언어의 프로그램으로 생성된 ABEND를 처리하려면 catch 절에 AbendException이 나타나는 Java try-catch 명령문을 사용하십시오.

HANDLE CONDITION

특정 조건(예: PGMIDERR)을 처리하려면 해당 예외(예: InvalidProgramException)의 이름을 지정하는 catch 절을 사용합니다 또는 모든 CICS 조건을 발견하려는 경우 CicsConditionException 이름을 지정하는 catch 절을 사용하십시오.

IGNORE CONDITION

이 명령은 Java 애플리케이션과 관련이 있습니다.

POP HANDLE 및 PUSH HANDLE

이 명령은 Java 애플리케이션에서는 관련이 없습니다. CICS ABEND 및 조건을 나타내는 데 사용되는 Java 예외는 범위 내 catch 블록으로 발견됩니다.

CICS 조건:

Java의 조건 처리 모델은 다른 CICS 프로그래밍 언어와 다릅니다.

COBOL에서는 각 조건마다 예외 처리 레이블을 정의합니다. CICS 명령 처리 중에 해당 조건이 발생하는 경우 레이블로 제어가 전송됩니다.

C 및 C++에서는 조건에 대한 예외 처리 레이블을 정의할 수 없습니다. 조건을 발견하려면 각 CICS 명령 다음에 EIB의 RESP 필드를 확인해야 합니다.

Java의 경우 CICS 명령으로 리턴된 조건은 Java 예외에 맵핑됩니다. 모든 CICS 명령을 try-catch 블록에 포함하고 각 조건에 특정 처리를 수행하거나 특정 예외와 관련이 없는 경우 단일 널(null) 예외 처리(catch) 절을 포함할 수 있습니다. 또는 보다 넓은 범위에서 예외 처리(catch) 절이 처리할 수 있도록 조건을 전파할 수 있습니다.

Java 웹 애플리케이션에서의 CICS 예외 처리

CICS ABEND 및 예외가 Liberty 웹 컨테이너에 대한 Java 예외 처리 구조에 통합되어 CICS 애플리케이션에서 발생하는 문제점을 처리합니다. 웹 애플리케이션에서 처리되지 않는 Java 예외는 웹 컨테이너가 발견하고 servlet 예외 처리 프로세스를 구동합니다. 이 처리의 일부로 CICS에서 확약되지 않은 CICS 작업 단위를 롤백합니다.

HttpServlet 인터페이스를 확장하는 모든 Java 웹 애플리케이션이 HttpServlet 인터페이스에 정의된 대로 `IOException` 또는 `ServletException`을 제외하고 확인된 모든 예외를 처리해야 합니다. 확인된 예외에는 처리되지 않은 CICS 조건을 나타내는 `com.ibm.cics.server.CicsConditionException`의 모든 서브클래스가 포함됩니다. 따라서 CICS 조건을 발견하는 모든 예외 처리 코드는 작업 단위가 롤백되어야 하는 모든 오류 조건을 식별하고 예제에 설명된 대로 태스크 오브젝트에서 rollback() 메소드를 사용하여 명시적으로 동기점 롤백을 호출하거나 `AbendException`을 전달해야 합니다.

```
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.RuntimeException의 서브클래스인 확인되지 않은 Java 예외는 웹 애플리케이션을 포함한 Java 애플리케이션에서 전달될 수 있고 `com.ibm.cics.server.AbendException` 및 `com.ibm.cics.server.AbendCancelException`을 포함합니다. 따라서 `AbendException`을 전달하거나 트랜잭션 이상 종료를 처리하지 않는 웹 애플리케이션이 servlet 예외 처리 프로세스 및 연관된 작업 단위 롤백 처리를 구동합니다.

JTA(Java Transaction API)를 사용하여 작업 단위를 확약하는 웹 애플리케이션은 Liberty 트랜잭션 관리자의 제어에 따라 확약됩니다. 세부사항은 67 페이지의 『JTA(Java Transaction API)』의 내용을 참조하십시오.

오류 처리 및 비정상 종료

Java 프로그램에서 ABEND를 시작하려면 Task.abend() 또는 Task.forceAbend() 메소드 중 하나를 호출해야 합니다.

메소드	JCICS 클래스	EXEC CICS 명령
abend(), forceAbend()	Task	ABEND

ABEND

Java 프로그램에서 ABEND를 시작하려면 Task.abend() 메소드 중 하나를 호출하십시오. 이로 인해 CICS에서 이상 종료 조건이 설정되며 AbendException이 전달됩니다. AbendException이 애플리케이션 오브젝트의 상위 레벨에서 발견되지 않거나 호출 프로그램(있는 경우)에 등록된 ABEND 핸들러로 처리되는 경우 CICS가 종료되고 트랜잭션을 롤백합니다.

다른 abend() 메소드는 다음과 같습니다.

- abend(String *abcode*): ABEND 코드 *abcode*로 ABEND를 야기합니다.
- abend(String *abcode*, boolean *dump*): ABEND 코드 *abcode*로 ABEND를 야기합니다. **dump** 매개변수가 false이면 덤프를 가져오지 않습니다.
- abend(): ABEND 코드 및 덤프 없이 ABEND를 야기합니다.

ABEND CANCEL

처리할 수 없는 ABEND를 시작하려면 Task.forceAbend() 메소드 중 하나를 호출하십시오. 상술한 바와 같이 Java 프로그램에서 발견될 수 있는 AbendCancelException이 전달됩니다. 이러한 경우 예외를 다시 전달하여 **ABEND_CANCEL** 처리를 완료함으로써 CICS로 제어가 리턴되면 CICS가 트랜잭션을 종료하고 롤백하도록 해야 합니다. AbendCancelException은 알림 목적으로만 발견하고 다시 전달하십시오.

다른 forceAbend() 메소드는 다음과 같습니다.

- forceAbend(String *abcode*): ABEND 코드 *abcode*로 **ABEND CANCEL**을 야기합니다.
- forceAbend(String *abcode*, boolean *dump*): ABEND 코드 *abcode*로 **ABEND CANCEL**을 야기합니다. **dump** 매개변수가 false이면 덤프를 가져오지 않습니다.
- forceAbend(): ABEND 코드 및 덤프 없이 **ABEND CANCEL**을 야기합니다.

APPC 맵핑 대화

APPC 비맵핑 대화 지원은 JCICS API에서 사용할 수 없습니다.

APPC 맵핑 대화:

메소드	JCICS 클래스	EXEC CICS 명령
initiate()	AttachInitiator	ALLOCATE, CONNECT PROCESS
converse()	Conversation	CONVERSE
get*() 메소드	Conversation	EXTRACT ATTRIBUTES
get*() 메소드	Conversation	EXTRACT PROCESS
free()	Conversation	FREE

메소드	JCICS 클래스	EXEC CICS 명령
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

BMS(Basic Mapping Support)

BMS(Basic Mapping Support)는 CICS 프로그램과 단말기 장치 간의 API(Application Programming Interface)입니다. JCICS는 몇몇 BMS API(Application Programming Interface)를 지원합니다.

메소드	JCICS 클래스	EXEC CICS 명령
sendControl()	TerminalPrincipalFacility	SEND CONTROL
sendText()	TerminalPrincipalFacility	SEND TEXT
	지원되지 않음	SEND MAP, RECEIVE MAP

채널 및 컨테이너 예제

*컨테이너*는 프로그램 간에 정보를 전달하기 위해 설계된 이름 지정된 데이터 블록입니다. 컨테이너는 *채널* 세트로 그룹화됩니다. 이 정보는 Java 애플리케이션에서 채널과 컨테이너를 사용하는 방법을 설명하고 일부 코드 예제를 제공합니다.

채널 및 컨테이너에 대한 기본 정보와 Java 이외 애플리케이션에서의 채널 사용에 대한 지침은 애플리케이션 개발에서 채널을 사용한 고급 프로그램간 데이터 전송의 내용을 참조하십시오. Java 프로그램이 기존 CICS 애플리케이션 데이터에 액세스할 수 있도록 하는 도구에 대한 정보는 38 페이지의 『Java 구조화 데이터와의 상호작용』의 내용을 참조하십시오.

표 3에는 채널 및 컨테이너에 대한 JCICS 지원을 구현하는 클래스와 메소드가 나열되어 있습니다.

표 3. 채널 및 컨테이너에 대한 JCICS 지원

메소드	JCICS 클래스	EXEC CICS 명령
containerIterator()	Channel	STARTBROWSE CONTAINER
createContainer()	Channel	
deleteContainer()	Channel	DELETE CONTAINER CHANNEL
getContainer()	Channel	
getName()	Channel	
delete()	Container	DELETE CONTAINER CHANNEL

표 3. 채널 및 컨테이너에 대한 JCICS 지원 (계속)

메소드	JCICS 클래스	EXEC CICS 명령
get(), getLength()	Container	GET CONTAINER CHANNEL [NODATA]
getName()	Container	
put()	Container	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()	Task	
getCurrentChannel()	Task	ASSIGN CHANNEL
containerIterator()	Task	STARTBROWSE CONTAINER

CICS 조건 CHANNELERR을 지정하면 ChannelErrorException이 전달됩니다. CONTAINERERR CICS 조건을 지정하면 ContainerErrorException이 전달되고 CCSIDERR CICS조건을 지정하면 CCSIDErrorException이 전달됩니다.

56 페이지의『JCICS에서 채널 및 컨테이너 작성』
채널을 작성하려면 Task 클래스의 createChannel() 메소드를 사용하십시오.

56 페이지의『컨테이너에 데이터 추가』
Container 오브젝트에 데이터를 추가하려면 Container.put() 메소드를 사용하십시오.

57 페이지의『다른 프로그램 또는 태스크로 채널 전달』
프로그램 링크 또는 전송 프로그램 제어(XCTL) 호출에서 채널을 전달하려면 각각 Program 클래스의 link() 및 xctl() 메소드를 사용합니다.

57 페이지의『현재 채널 수신』
프로그램이 현재 채널을 명시적으로 수신할 필요는 없습니다. 그러나 프로그램은 현재 태스크에서 현재 채널을 가져올 수 있습니다.

57 페이지의『컨테이너에서 데이터 가져오기』
Container.get() 메소드를 사용하여 컨테이너의 데이터를 바이트 배열로 읽어들입니다.

57 페이지의『현재 채널 찾아보기』
채널이 전달되는 JCICS 프로그램은 채널을 명시적으로 수신하지 않고 모든 Container 오브젝트에 액세스할 수 있습니다.

58 페이지의『채널 및 컨테이너 예제』
이 예제는 COBOL 서버 프로그램 PAYR을 호출하는 Java 클래스 Payroll의 발췌

본을 보여줍니다. Payroll 클래스는 채널 및 해당 컨테이너를 사용할 수 있도록 JCICS com.ibm.cics.server.Channel 및 com.ibm.cics.server.Container 클래스를 사용합니다.

관련 개념:

43 페이지의『데이터 전달 인수』
채널과 컨테이너를 사용하거나 통신 영역(COMMAREA)을 사용하여 프로그램 간에 데이터를 전달할 수 있습니다.

JCICS에서 채널 및 컨테이너 작성:

채널을 작성하려면 Task 클래스의 createChannel() 메소드를 사용하십시오.

예:

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

createChannel 메소드에 제공되는 문자열은 CICS가 인식하는 Channel 오브젝트의 이름입니다. CICS 이름 지정 규칙을 준수하기 위해 이름이 최대 16자까지 공백으로 채워집니다.

채널에서 새 컨테이너를 작성하려면 Channel의 createContainer() 메소드를 사용합니다. 예:

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

createContainer() 메소드에 제공되는 문자열은 CICS가 인식하는 Container 오브젝트의 이름입니다. 필요한 경우 CICS 이름 지정 규칙을 준수하기 위해 이름이 최대 16자까지 공백으로 채워집니다. 이 채널에 이름이 같은 컨테이너가 이미 존재하면 ContainerErrorException이 전달됩니다.

컨테이너에 데이터 추가:

Container 오브젝트에 데이터를 추가하려면 Container.put() 메소드를 사용하십시오.

Container 오브젝트에 데이터를 추가하려면 Container.put() 메소드를 사용하십시오. 컨테이너에 바이트 배열 또는 문자열로 데이터를 추가할 수 있습니다. 예를 들어, 다음과 같습니다.

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

또는 :

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

다른 프로그램 또는 태스크로 채널 전달:

프로그램 링크 또는 전송 프로그램 제어(XCTL) 호출에서 채널을 전달하려면 각각 Program 클래스의 link() 및 xctl() 메소드를 사용합니다.

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

```
programY.xctl(custData);
```

프로그램 리턴 호출에서 다음 채널을 설정하려면 TerminalPrincipalFacility 클래스의 setNextChannel() 메소드를 사용합니다.

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

START 요청에서 채널을 전달하려면 StartRequest 클래스의 issue 메소드를 사용합니다.

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

현재 채널 수신:

프로그램이 현재 채널을 명시적으로 수신할 필요는 없습니다. 그러나 프로그램은 현재 태스크에서 현재 채널을 가져올 수 있습니다.

프로그램이 현재 태스크에서 현재 채널을 가져오는 경우 태스크가 이름별로 컨테이너를 추출할 수 있습니다.

```
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");
}
```

컨테이너에서 데이터 가져오기:

Container.get() 메소드를 사용하여 컨테이너의 데이터를 바이트 배열로 읽어들입니다.

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

현재 채널 찾아보기:

채널이 전달되는 JCICS 프로그램은 채널을 명시적으로 수신하지 않고 모든 Container 오브젝트에 액세스할 수 있습니다.

이 작업을 수행하기 위해 ContainerIterator 오브젝트를 사용합니다.
ContainerIterator 클래스는 java.util.Iterator 인터페이스를 구현합니다. Task 오브젝트가 현재 태스크에서 인스턴스화되면 해당 containerIterator() 메소드가 Iterator(현재 채널의 경우) 또는 널(null)(현재 채널이 없는 경우)을 리턴합니다. 예를 들어, 다음과 같습니다.

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

채널 및 컨테이너 예제:

이 예제는 COBOL 서버 프로그램 PAYR을 호출하는 Java 클래스 Payroll의 발췌본을 보여줍니다. Payroll 클래스는 채널 및 해당 컨테이너를 사용할 수 있도록 JCICS com.ibm.cics.server.Channel 및 com.ibm.cics.server.Container 클래스를 사용합니다.

```
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.put("John Doe");

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

     // put the wage into the container
     wage.put("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();
     ...
}
```

그림 1. JCICS com.ibm.cics.server.Channel 및 com.ibm.cics.server.Container 클래스를 사용하여 COBOL 서버 프로그램으로 채널을 전달하는 Java 클래스

진단 서비스

JCICS API(Application Programming Interface)는 이러한 CICS 추적 및 덤프 명령을 지원합니다.

메소드	JCICS 클래스	EXEC CICS 명령
	지원되지 않음	DUMP
enterTrace()	EnterRequest	ENTER

문서 서비스

이 섹션에서는 DOCUMENT API(Application Programming Interface)의 명령에 대한 JCICS 지원을 설명합니다.

Document 클래스는 **EXEC CICS DOCUMENT** API에 맵핑됩니다. DocumentLocation 클래스의 생성자는 **EXEC CICS DOCUMENT** API의 AT 및 TO 키워드에 맵핑됩니다. SymbolList 클래스의 Setter 및 Getter는 **EXEC CICS DOCUMENT** API의 SYMBOLLIST, LENGTH, DELIMITER, UNESCAPE 키워드에 맵핑됩니다.

메소드	JCICS 클래스	EXEC CICS 명령
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

환경 서비스

CICS 환경 서비스는 애플리케이션과 관련된 CICS 데이터 영역, 매개변수, 자원 속성에 대한 액세스를 제공합니다.

동등한 JCICS 지원을 하는 **EXEC CICS** 명령 및 선택항목은 다음과 같습니다.

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

60 페이지의『ADDRESS』
ADDRESS API 명령 선택항목에는 다음 지원이 제공됩니다.

61 페이지의『ASSIGN』
ASSIGN API 명령 선택항목에는 다음 지원이 제공됩니다.

62 페이지의『INQUIRE SYSTEM』
INQUIRE SYSTEM SPI 선택항목에 대한 지원이 제공됩니다.

62 페이지의『INQUIRE TASK』
INQUIRE TASK API 명령 선택항목에는 다음과 같은 지원이 제공됩니다.

62 페이지의『INQUIRE TERMINAL 및 INQUIRE NETNAME』
INQUIRE TERMINAL 및 **INQUIRE NETNAME** SPI 선택항목에는 다음과 같은 지원이 제공됩니다.

ADDRESS:

ADDRESS API 명령 선택항목에는 다음 지원이 제공됩니다.

EXEC CICS ADDRESS 명령에 대한 전체 정보는 참조 -> 애플리케이션 개발에서 ADDRESS의 내용을 참조하십시오.

ACEE

ACEE(Access Control Environment Element)는 CICS 사용자가 사인온할 때 외부 보안 관리자가 작성합니다. JCICS에서는 이 선택항목이 지원되지 않습니다.

COMMAREA

COMMAREA에는 명령과 함께 전달되는 사용자 데이터가 포함됩니다. COMMAREA 포인터는 **CommAreaHolder** 인수로 링크된 프로그램으로 자동 전달됩니다. 자세한 정보는 43 페이지의『데이터 전달 인수』의 내용을 참조하십시오.

CWA CWA(Common Work Area)에는 태스크 간에 공유 가능한 글로벌 사용자 데이터가 포함됩니다. CWA 사본은 Region 클래스의 getCWA() 메소드를 사용하여 얻을 수 있습니다.

EIB 에는 마지막으로 실행된 CICS 명령에 대한 정보가 포함됩니다. EIB 값에 대한 액세스 권한은 해당 오브젝트의 메소드로 제공됩니다. 예를 들어, 다음과 같습니다.

eibtrnid

Task 클래스의 getTransactionName() 메소드로 리턴됩니다.

eibaid TerminalPrincipalFacility 클래스의 the getAIDbyte() 메소드로 리턴됩니다.

eibcposn

Cursor 클래스의 getRow() 및 getColumn() 메소드로 리턴됩니다.

TCTUA

TCTUA(Terminal Control Table User Area)에는 CICS 트랜잭션(프린시펄 기능)을 구동하는 단말기와 연관된 사용자 데이터가 포함됩니다. 이 영역은 애플

리케이션 간에 정보를 전달하는 데 사용되지만 이 경우 동일한 단말기가 관련 애플리케이션과 연관되어야 합니다. TCTUA의 내용은 TerminalPrincipalFacility 클래스의 getTCTUA() 메소드를 사용하여 얻을 수 있습니다.

TWA TWA(Transaction Work Area)에는 CICS 태스크와 연관된 사용자 데이터가 포함됩니다. 이 영역은 애플리케이션 간에 정보를 전달하는 데 사용되지만 이 경우 애플리케이션이 동일한 태스크에 있어야 합니다. TWA 사본은 Task 클래스의 getTWA() 메소드를 사용하여 얻을 수 있습니다.

ASSIGN:

ASSIGN API 명령 선택항목에는 다음 지원이 제공됩니다.

이 명령에 대한 자세한 정보는 참조 -> 애플리케이션 개발에서 ASSIGN의 내용을 참조하십시오.

메소드	JCICS 클래스
getABCODE()	AbendException
getApplicationContext()	Task
getAPPLID()	Region
getCurrentChannel()	Task
getCWA()	Region
getName()	TerminalPrincipalFacility 또는 ConversationPrincipalFacility
getFCI()	Task
getNetName()	TerminalPrincipalFacility 또는 ConversationPrincipalFacility
getPrinSysid()	TerminalPrincipalFacility 또는 ConversationPrincipalFacility
getProgramName()	Task
getQNAME()	Task
getSTARTCODE()	Task
getSysid()	Region
getTCTUA()	TerminalPrincipalFacility
getTERMCODE()	TerminalPrincipalFacility
getTWA()	Task
getUSERID(), Task.getUSERID()	Task, TerminalPrincipalFacility 또는 ConversationPrincipalFacility

다른 ASSIGN 선택항목은 지원되지 않습니다.

INQUIRE SYSTEM:

INQUIRE SYSTEM SPI 선택항목에 대한 지원이 제공됩니다.

메소드	JCICS 클래스
getAPPLID()	Region
getSYSID()	Region

다른 **INQUIRE SYSTEM** 선택항목은 지원되지 않습니다.

INQUIRE TASK:

INQUIRE TASK API 명령 선택항목에는 다음과 같은 지원이 제공됩니다.

메소드	JCICS 클래스
getSTARTCODE()	Task
getTransactionName()	Task
getUSERID()	Task

FACILITY

태스크의 프린시펄 기능에서 getName() 메소드를 호출하여 태스크의 프린시펄 기능 이름을 찾을 수 있습니다. 이 기능은 현재 Task 오브젝트에서 getPrincipalFacility() 메소드를 호출하여 찾을 수 있습니다.

FACILITYTYPE

Java instanceof 연산자로 리턴된 오브젝트 참조 클래스를 확인하여 기능 유형을 판별할 수 있습니다.

다른 INQUIRE TASK 선택항목은 지원되지 않습니다.

INQUIRE TERMINAL 및 INQUIRE NETNAME:

INQUIRE TERMINAL 및 **INQUIRE NETNAME** SPI 선택항목에는 다음과 같은 지원이 제공됩니다.

메소드	JCICS 클래스
getUSERID()	Terminal, ConversationalPrincipalFacility
Terminal.getUser()	Terminal, ConversationalPrincipalFacility

현재 Task 오브젝트 또는 태스크의 프린시펄 기능을 나타내는 오브젝트에서 getUSERID() 메소드를 호출하여 USERID 값을 찾을 수도 있습니다.

다른 **INQUIRE TERMINAL** 또는 **INQUIRE NETNAME** 선택사항은 지원되지 않습니다.

파일 서비스

JCICS는 CICS 파일 및 인덱스의 각 유형에 대한 **EXEC CICS** API 명령에 맵핑되는 클래스와 메소드를 제공합니다.

Java 프로그램이 기존 CICS 애플리케이션 데이터에 액세스할 수 있도록 해주는 도구에 대한 정보는 애플리케이션 개발에서 Java의 구조 데이터와 상호 작용의 내용을 참조하십시오.

CICS는 다음과 같은 유형의 파일을 지원합니다.

- KSDS(Key Sequenced Data Set)
- ESDS(Entry Sequenced Data Set)
- RRDS(Relative Record Data Set)

KSDS 및 ESDS 파일은 대체(또는 2차) 색인을 가질 수 있습니다. CICS는 2차 색인을 통한 RRDS 파일 액세스를 지원하지 않습니다. 2차 색인은 CICS에서 별도 KSDS 파일인 것처럼 간주됩니다. 이는 별도 FD 항목이 있음을 의미합니다.

KSDS, ESDS(1차 색인) 및 ESDS(2차 색인) 파일 액세스에는 몇 가지 차이점이 있으므로 항상 공통 인터페이스를 사용할 수는 없습니다.

ESDS 파일에서 삭제할 수 없는 레코드를 제외하고는 모든 유형의 파일에서 레코드를 읽고, 찾아보고, 갱신, 삭제할 수 있습니다.

데이터 세트에 대한 자세한 정보는 VSAM 데이터 세트: KSDS, ESDS, RRDS를 참조하십시오.

데이터를 읽는 Java 명령은 **EXEC CICS** 명령에서 SET 선택항목에 해당하는 항목만 지원합니다. 리턴된 데이터는 CICS 저장영역에서 Java 오브젝트로 자동으로 복사됩니다.

파일 제어와 관련된 Java 인터페이스는 다음 다섯 가지 범주로 분류됩니다.

File 다른 파일 클래스의 수퍼클래스로 모든 파일 클래스의 공통 메소드를 포함합니다.

KeyedFile
1차 색인을 사용하여 액세스하는 KSDS 파일, 2차 색인을 사용하여 액세스하는 KSDS 파일, 2차 색인을 사용하여 액세스하는 ESDS 파일에 공통적인 인터페이스를 포함합니다.

KSDS KSDS 파일에 고유한 인터페이스를 포함합니다.

ESDS 1차 색인인 RBA(Relative Byte Address) 또는 XRBA(Extended Relative Byte Address)를 통해 액세스하는 ESDS 파일에 고유한 인터페이스를 포함합니다. RBA 대신 XRBA를 사용하려면 setXRBA(true) 메소드를 실행하십시오.

RRDS 1차 색인인 RRN(Relative Record Number)을 통해 액세스하는 RRDS 파일에 고유한 인터페이스를 포함합니다.

각 파일마다 두 개 오브젝트(File 오브젝트 및 FileBrowse 오브젝트)가 실행될 수 있습니다. 파일 오브젝트는 파일 자체를 나타내며 다음 API 조작을 수행하기 위해 메소드와 함께 사용할 수 있습니다.

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

File 오브젝트는 필수 파일 클래스를 명시적으로 시작하는 사용자 애플리케이션으로 작성됩니다. FileBrowse 오브젝트는 파일에 대한 찾아보기 조작을 나타냅니다. 언제라도 특정 파일에 대해 여러 활성 찾아보기가 존재할 수 있습니다. 각 찾아보기는 REQID로 구분됩니다. 다음 API 조작을 수행하기 위해 FileBrowse 오브젝트에 대한 메소드가 인스턴스화될 수 있습니다.

- ENDBR
- READNEXT
- READPREV
- RESETBR

FileBrowse 오브젝트는 사용자 애플리케이션이 명시적으로 인스턴스화하지 않으며 STARTBR 조작을 수행하는 메소드로 작성되어 사용자 클래스에 리턴됩니다.

다음 표는 JCICS 클래스와 메소드가 각 유형의 CICS 파일과 색인마다 **EXEC CICS** API 명령에 어떻게 맵핑되는지를 보여줍니다. 이 표에서는 JCICS 클래스와 메소드가 `class.method()` 양식으로 표시됩니다. 예를 들어, KeyedFile.read()는 KeyedFile 클래스에서 read() 메소드를 참조합니다.

첫 번째 표는 키 파일의 클래스와 메소드를 보여줍니다.

표 4. 키 파일의 클래스와 메소드

KSDS 1차 또는 2차 색인 클래스 및 메소드	**ESDS 2차 색인 클래스 및 메소드**	**CICS 파일 API 명령**
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**

표 4. 키 파일의 클래스와 메소드 (계속)

KSDS 1차 또는 2차 색인 클래스 및 메소드	**ESDS 2차 색인 클래스 및 메소드**	**CICS 파일 API 명령**
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**

이 표는 키 파일 이외의 파일에 대한 클래스와 메소드를 보여줍니다. ESDS 및 RRDS는 1차 색인으로 액세스합니다.

ESDS 1차 색인 클래스 및 메소드	**RRDS 1차 색인 클래스 및 메소드**	**CICS 파일 API 명령**
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_Browse.reset()	RRDS_Browse.reset()	**RESET BROWSE**
FileBrowse.end()	FileBrowse.end()	**END BROWSE**
ESDS.setXRBA()		

파일에 쓸 데이터는 Java 바이트 배열에 있어야 합니다.

파일에서 RecordHolder 오브젝트로 데이터를 읽어들입니다. 저장영역은 CICS가 제공하며 프로그램이 끝나면 자동으로 해제됩니다.

File 메소드에서 **KEYLENGTH** 값을 지정하지 않아도 됩니다. 사용된 길이는 전달된 키의 실제 길이입니다. FileBrowse 오브젝트가 작성될 때 startBrowse 메소드에 지정된 키의 길이가 포함되며 해당 오브젝트에 대한 이후 찾아보기 요청에서 해당 길이가 CICS에 전달됩니다.

찾아보기 오퍼레이션에 **REQID**를 제공하지 않아도 됩니다. 각 찾아보기 오브젝트에는 해당 찾아보기 오브젝트에 대한 모든 후속 찾아보기 요청에 대해 자동으로 사용되는 고유 REQID가 포함됩니다.

HTTP 및 TCP/IP 서비스

HttpHeader, NameValueData, FormField 클래스의 Getter는 해당 API 명령에 대한 HTTP 헤더, 이름 및 값 쌍, 양식 필드 값을 리턴합니다.

메소드	JCICS 클래스	EXEC CICS 명령
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
sendDocument()	HttpResponse	WEB SEND

참고: get HttpRequestInstance() 메소드를 사용하여 HttpRequest 오브젝트를 얻습니다.

CICS 웹 지원이 처리하는 수신 HTTP 요청 각각에 HTTP 헤더가 포함됩니다. 요청이 POST HTTP verb를 사용하는 경우에는 문서 데이터도 포함됩니다. CICS 웹 지원으로 생성된 응답 HTTP 요청 각각에는 HTTP 헤더와 문서 데이터가 포함됩니다.

이 JCICS를 처리하기 위해 다음 웹 및 TCP/IP 서비스를 제공합니다.

HTTP 헤더

HttpRequest 클래스를 사용하여 HTTP 헤더를 검사할 수 있습니다. HTTP가 GET 모드인 경우 클라이언트가 HTTP 양식을 작성하고 제출 단추를 선택하면 조회 문자열이 제출됩니다.

SSL CICS 웹 지원은 SSL 지원에 대한 기본 정보와 요청을 제출한 클라이언트에

대한 자세한 정보를 얻기 위해 HttpRequest로 확장되는 TcpipRequest 클래스를 제공합니다. SSL 인증서가 제공되는 경우 CertificateInfo 클래스를 사용하여 세부사항을 검사할 수 있습니다.

문서 서버(HTTP POST)에 문서가 공개되는 경우 CICS 문서로 제공됩니다. 이 문서는 HttpRequest 클래스에서 getDocument() 메소드를 호출하여 액세스할 수 있습니다. 기존 문서 처리에 대한 자세한 정보는 59 페이지의 『문서 서비스』의 내용을 참조하십시오.

요청에 따른 HTTP 클라이언트 웹 내용을 제공하려면 서버 프로그래머가 문서 서비스 API를 사용하여 CICS 문서를 작성하고 sendDocument() 메소드를 호출해야 합니다.

CICS 웹 지원에 대한 자세한 정보는 제품 개요의 인터넷, TCP/IP 및 HTTP 개념의 내용을 참조하십시오. JCICS 웹 클래스에 대한 자세한 정보는 *JCICS 클래스 참조*를 참고하십시오.

JTA(Java Transaction API)

JTA(Java Transaction API)를 사용하여 트랜잭션 갱신을 여러 자원 관리자로 조정할 수 있습니다.

JTA(Java Transaction API)를 사용하여 CICS 자원 및 기타 써드 파티 자원 관리자에 대한 트랜잭션 갱신을 조정할 수 있습니다(예를 들어, Liberty JVM 서버 내의 유형 4 데이터베이스 드라이버 연결). 이 시나리오에서는 트랜잭션이 CICS 시스템 외부에서 시작된 것처럼 Liberty 트랜잭션 관리자가 트랜잭션 조정자이고 CICS 작업 단위가 하위입니다.

CICS 데이터 소스를 사용한 로컬 DB2 데이터베이스에 대한 유형 2 드라이버 연결은 CICS DB2 첨부를 사용하여 액세스합니다. JTA를 사용하여 다른 CICS 자원에 대한 갱신으로 조정할 필요가 없습니다.

JTA에서 여러 자원 관리자에 대한 갱신을 캡슐화하고 조정하기 위한 UserTransaction 오브젝트를 작성합니다. 다음 코드 단편은 사용자 트랜잭션을 작성하고 사용하는 방법을 보여줍니다.

```
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();
}
```

CICS 작업 단위와 달리, UserTransaction은 `begin()` 메소드를 사용하여 명시적으로 시작되어야 합니다. `begin()`을 호출하면 CICS가 UserTransaction을 시작하기 전에 수행되었을 수 있는 모든 갱신을 확약합니다. UserTransaction은 `commit()` 또는 `rollback()` 메소드를 호출하거나 웹 애플리케이션이 종료될 때 웹 컨테이너로 종료됩니다. UserTransaction이 활성화된 상태에서는 프로그램이 JCICS `Task commit()` 또는 `rollback()` 메소드를 호출할 수 없습니다.

JCICS 메소드 Task.commit() 및 Task.rollback()은 JTA 트랜잭션 컨텍스트 내에서 유효하지 않습니다. 어떤 메소드를 시도하더라도 InvalidRequestException이 전달됩니다.

Liberty 기본값은 의심 트랜잭션 복구를 시도하기 전에 첫 번째 UserTransaction이 작성될 때까지 대기하는 것입니다. `<transaction recoverOnStartup="true"/>`가 server.xml에 지정된 경우에도 Liberty 초기화가 완료된 직후 항상 CICS가 복구를 시작합니다. JVM 서버가 사용 불가능하도록 설치된 경우 이를 사용 가능하도록 설정할 때 복구가 실행됩니다.

프로그램 서비스

JCICS는 CICS 프로그램 제어 명령인 LINK, RETURN 및 INVOKE APPLICATION을 지원합니다.

Java 프로그램이 기존 CICS 애플리케이션 데이터에 액세스할 수 있도록 해주는 도구에 대한 정보는 애플리케이션 개발에서 Java의 구조 데이터와 상호 작용의 내용을 참조하십시오.

표 5에는 CICS 프로그램 제어 명령에 맵핑되는 메소드와 JCICS 클래스가 나열되어 있습니다.

표 5. 메소드, JCICS 클래스 및 CICS 명령 간의 관계.

EXEC CICS 명령	**JCICS 클래스**	**JCICS 메소드**
LINK	Program	link()
RETURN	TerminalPrincipalFacility	setNextTransaction(), setNextCOMMAREA(), setNextChannel()
INVOKE APPLICATION	Application	invoke()

LINK

link() 메소드를 사용하여 CICS에 정의된 다른 프로그램으로 제어를 전송할 수 있습니다. 대상 프로그램은 CICS가 지원하는 모든 언어를 사용할 수 있습니다.

RETURN

이 명령의 의사 대화식 측면만 지원됩니다. 리턴할 CICS 호출을 작성하지 않아도 됩니다. 애플리케이션이 정상적으로 종료될 수 있습니다. 의사 대화식 기능은 TerminalPrincipalFacility 클래스에서 메소드로 지원됩니다. setNextTransaction()은 RETURN의 TRANSID 선택항목을 사용하는 것과 같고 setNextCOMMAREA()는 COMMAREA 선택항목을 사용하는 것과 같으며 while setNextChannel()은 CHANNEL 선택항목을 사용하는 것과 같습니다. 이러한 메소드는 프로그램 실행 중에 언제라도 호출될 수 있으며 프로그램이 종료될 때 적용됩니다.

INVOKE

공용 또는 개인용 프로그램에 관계 없이 애플리케이션 시작점 프로그램의 이름을 알지 못하는 상태에서도 프로그램 시작점 중 하나에 해당하는 조작의 이름을 지정하여 애플리케이션을 호출할 수 있습니다.

참고: 제공된 COMMAREA의 길이가 CICS의 LENGTH 값으로 사용됩니다. COMMAREA가 두 CICS 서버(임의로 조합된 제품/버전/릴리스의 경우) 간에 전달되는 경우에는 이 값이 32,500바이트를 초과해서는 안됩니다. 이 한계는 32,500바이트 COMMAREA와 헤더 공간에 허용됩니다.

스케줄링 서비스

JCICS는 태스크에 대해 저장된 데이터를 검색하고 간격 제어 요청을 취소하며 지정된 시간에 태스크를 시작할 수 있는 CICS 스케줄링 서비스를 지원합니다.

메소드	JCICS 클래스	EXEC CICS 명령
cancel()	StartRequest	CANCEL
retrieve()	Task	RETRIEVE
issue()	StartRequest	START

Task.retrieve() 메소드로 검색할 대상을 정의하려면 java.util.BitSet 오브젝트를 사용하십시오. com.ibm.cics.server.RetrieveBits 클래스는 BitSet 오브젝트에 설정될 수 있는 비트를 정의합니다. 예를 들어, 다음과 같습니다.

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

이는 **EXEC CICS** RETRIEVE 명령의 선택항목에 해당됩니다.

Task.retrieve() 메소드는 RetrieveBits 설정에 따라 최대 네 가지 다른 정보를 단일 호출로 검색합니다. DATA, RTRANSID, RTERMID, QUEUE 데이터는 RetrievedDataHolder 오브젝트에 포함된 RetrievedData 오브젝트에 배치됩니다. 다음 예제는 데이터와 transid를 검색합니다.

```
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;
```

직렬화 서비스

JCICS는 자원 사용을 태스크로 예약할 수 있는 CICS 직렬화 서비스를 지원합니다.

메소드	**JCICS** 클래스	**EXEC CICS** 명령
dequeue()	SynchronizationResource	DEQ
enqueue(), tryEnqueue()	SynchronizationResource	ENQ

관련 참조:

44 페이지의 『순차 클래스』
JCICS 순차 클래스의 목록입니다.

저장영역 서비스

CICS 서비스(예: **EXEC CICS** GETMAIN)를 사용한 명시적인 저장영역 관리는 지원되지 않습니다. 표준 Java 저장영역 관리 기능이 태스크 전용 저장영역에 대한 요구를 충족시키는 데 충분한지 확인해야 합니다.

태스크 간의 데이터 공유는 CICS 자원을 사용하여 이루어져야 합니다.

이름은 일반적으로 Java 문자열 또는 바이트 배열로 표시됩니다. 이름이 필수 길이를 충족하는지 확인해야 합니다.

스레드 및 태스크 예제

CICSExecutorService를 사용하여 CICS에서 태스크를 시작하는 스레드를 작성할 수 있습니다. 이 서비스를 사용하여 스레드를 작성하는 경우 JCICS API를 사용하여 CICS 서비스를 액세스할 수 있는 CICS 태스크가 작성됩니다.

CICS 태스크를 시작하는 스레드를 작성하는 데는 두 가지 선택항목이 있습니다. 이식형 코드를 작성하려는 경우 OSGi 프레임워크에서 ExecutorService를 사용할 수 있습니다. 애플리케이션이 JVM 서버에서 실행되는 경우 OSGi 프레임워크는 스레드가 작성될 때 CICS 태스크를 시작하기 위해 자동으로 CICS 구현을 사용합니다. CICS 전용 애플리케이션을 작성하는 경우 JCICS로 CICSExecutorService 클래스를 직접 사용할 수 있습니다.

다음 예제는 임시 저장영역 큐를 작성하고 큐에 데이터를 쓰기 위해 스레드를 시작하는 Java 클래스의 발췌본을 보여줍니다.

```
package com.ibm.cics.executor.test;

import com.ibm.cics.server.CICSExecutorService;
import com.ibm.cics.server.TSQ;
import com.ibm.cics.server.TSQType;

public class ExecutorTest
{
    public static void main(String[] args)
    {
        // Inline the new Runnable class
        class CICSJob implements Runnable
        {
            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();
                }
            }
        }

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

관련 개념:

45 페이지의『스레드』

JVM 서버 환경에서는 OSGi 프레임워크에서 실행되는 애플리케이션이 ExecutorService를 사용하여 CICS 태스크에서 실행되는 스레드를 비동기식으로 작성할 수 있습니다.

임시 저장영역 큐 서비스

JCICS는 CICS 임시 저장영역 명령으로 DELETEQ TS, READQ TS, WRITEQ TS를 지원합니다.

JCICS 메소드와 EXEC CICS 명령 간의 상호작용

Java 프로그램이 기존 CICS 애플리케이션 데이터에 액세스할 수 있도록 해주는 도구에 대한 정보는 애플리케이션 개발에서 Java의 구조 데이터와 상호 작용의 내용을 참조하십시오.

표 6에는 CICS 임시 저장영역 명령으로 맵핑되는 JCICS 클래스와 메소드가 나열되어 있습니다.

표 6. 메소드, JCICS 클래스와 CICS 명령 간의 관계

메소드	JCICS 클래스	EXEC CICS 명령
delete()	TSQ	DELETEQ TS
readItem(), readNextItem()	TSQ	READQ TS
writeItem(), rewriteItem() writeItemConditional() rewriteItemConditional()	TSQ	WRITEQ TS

DELETEQ TS

TSQ 클래스에서 delete() 메소드를 사용하여 임시 저장영역 큐(TSQ)를 삭제할 수 있습니다.

READQ TS

Java 프로그램에서는 CICS INTO 선택항목이 지원되지 않습니다. TSQ 클래스에서 readItem() 및 readNextItem() 메소드를 사용하여 TSQ에서 특정 항목을 읽을 수 있습니다. 이러한 메소드는 ItemHolder 오브젝트를 인수 중 하나로 취하므로 바이트 배열에 데이터 읽기가 포함됩니다. 이 바이트 배열의 저장영역은 CICS로 작성되며 프로그램 끝에서 사용공간을 정리합니다.

WRITEQ TS

임시 저장영역 큐에 기록될 데이터를 Java 바이트 배열에 제공해야 합니다. NOSPACE 조건을 발견하면 writeItem() 및 rewriteItem() 메소드가 일시중단되며 큐에 데이터를 쓸 수 있는 공간이 확보될 때까지 대기합니다. NOSPACE 조건의 경우에는 writeItemConditional() 및 rewriteItemConditional() 메소드가 일시중단되지는 않지만 애플리케이션에 즉시 NoSpaceException으로 조건을 리턴합니다.

단말기 서비스

JCICS는 이러한 CICS 단말기 서비스 명령을 지원합니다.

메소드	JCICS 클래스	EXEC CICS 명령
converse()	TerminalPrincipalFacility	CONVERSE
	지원되지 않음	HANDLE AID
receive()	TerminalPrincipalFacility	RECEIVE
send()	TerminalPrincipalFacility	SEND
	지원되지 않음	WAIT TERMINAL

태스크에 프린시펄 기능으로 할당된 단말기가 있는 경우 CICS는 표준 출력 및 표준 오류 스트림으로 사용될 수 있는 Java PrintWriter 구성요소 두 개를 자동으로 작성합니

다. 이러한 구성요소는 태스크 단말기에 맵핑됩니다. 두 개 스트림(out 및 err)은 Task 오브젝트의 공용 파일이며 System.out 및 System.err과 동일한 방식으로 사용될 수 있습니다.

단말기로 전달될 데이터를 Java 바이트 배열에 제공해야 합니다. 단말기에서 DataHolder 오브젝트로 데이터를 읽어들입니다. CICS는 프로그램이 종료될 때 할당이 취소되는 리턴된 데이터의 저장영역을 제공합니다.

트랜지언트 데이터 큐 서비스

JCICS는 CICS 트랜지언트 데이터 명령인 DELETEQ TD, READQ TD, WRITEQ TD를 지원합니다. INTO 선택항목을 제외한 모든 선택항목이 지원됩니다.

JCICS 메소드와 EXEC CICS 명령 간의 상호작용

Java 프로그램이 기존 CICS 애플리케이션 데이터에 액세스할 수 있도록 해주는 도구에 대한 정보는 애플리케이션 개발에서 Java의 구조 데이터와 상호 작용의 내용을 참조하십시오.

표 7에는 CICS 트랜지언트 데이터 명령에 맵핑되는 JCICS 클래스와 메소드가 나열되어 있습니다.

표 7. 메소드, JCICS 클래스와 CICS 명령 간의 관계

메소드	JCICS 클래스	EXEC CICS 명령
delete()	TDQ	DELETEQ TD
readData(), readDataConditional()	TDQ	READQ TD
writeData()	TDQ	WRITEQ TD

DELETEQ TD

TDQ 클래스에서 delete() 메소드를 사용하여 트랜지언트 데이터 큐(TDQ)를 삭제할 수 있습니다.

READQ TD

Java 프로그램에서는 CICS INTO 선택항목이 지원되지 않습니다. TDQ 클래스의 readData() 또는 readDataConditional() 메소드를 사용하여 TDQ에서 읽을 수 있습니다. 이 메소드는 데이터 읽기를 바이트 배열에 포함하는 DataHolder 오브젝트의 인스턴스를 매개변수로 취합니다. 이 바이트 배열의 저장영역은 CICS로 작성되며 프로그램 끝에서 사용공간을 정리합니다.

readDataConditional() 메소드는 CICS NOSUSPEND 로직을 구동합니다. QBUSY 조건을 발견하면 애플리케이션에 즉시 QueueBusyException으로 리턴됩니다.

readData() 메소드가 다른 태스크가 사용 중인 레코드에 대한 액세스를 시도하고 확약된 레코드가 더 없으면 메소드가 일시중단됩니다.

WRITEQ TD

TDQ에 기록될 데이터를 Java 바이트 배열에 제공해야 합니다.

작업 단위(UOW) 서비스

JCICS는 CICS SYNCPOINT 서비스를 위한 지원을 제공합니다.

표 8. UOW 서비스를 위한 JCICS 명령과 EXEC CICS 명령의 관계

메소드	JCICS 클래스	EXEC CICS 명령
commit(), rollback()	Task	SYNCPOINT

Liberty JVM 서버에서 UOW syncpointing은 JTA(Java Transaction API)를 사용하여 제어할 수 있습니다. 자세한 정보는 67 페이지의 『JTA(Java Transaction API)』의 내용을 참조하십시오.

웹 서비스 예제

JCICS는 애플리케이션에서 웹 서비스 작업을 위해 사용할 수 있는 모든 API 명령을 지원합니다.

메소드	JCICS 클래스	EXEC CICS 명령
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

다음 예제는 JCICS를 사용하여 웹 서비스 요청을 작성할 수 있는 방법을 보여줍니다.

```
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();
```

웹 서비스 요청으로 주고 받는 애플리케이션 데이터를 처리하려면 구조화된 데이터 작업을 수행하는 경우 JZOS와 같은 도구를 사용하여 클래스를 생성하면 됩니다. 자세한

정보는 38 페이지의 『Java 구조화 데이터와의 상호작용』의 내용을 참조하십시오. Java를 사용하여 XML을 직접 생성하여 이용할 수도 있습니다.

JCICS 사용

JCICS 라이브러리에서 일반 Java 클래스와 같은 클래스를 사용합니다. 애플리케이션은 필수 유형의 참조를 선언하며 new 연산자를 사용하여 클래스의 새 인스턴스가 작성됩니다.

이 태스크 정보

setName 메소드로 CICS 자원의 이름을 지정하여 기본 CICS 자원의 이름을 제공합니다. 자원을 작성한 후 표준 Java 구성을 사용하여 오브젝트를 조작할 수 있습니다. 선언된 오브젝트의 메소드는 일반적인 방법으로 호출할 수 있습니다. 각 클래스에 지원되는 메소드에 대한 자세한 내용은 제공되는 Javadoc에 나와 있습니다.

CICS Java 프로그램에서 파이널라이저를 사용하지 마십시오. 파이널라이저를 권장하지 않는 이유에 대한 설명은 IBM SDK for z/OS, Java Technology Edition, 버전 7(문제점 해결 절 참조)의 내용을 참조하십시오.

System.exit() 호출을 실행하여 CICS Java 프로그램을 종료하지 마십시오. Java 애플리케이션이 CICS에서 실행되면 Java 랩퍼라는 다른 Java 프로그램을 사용하여 public static void main() 메소드가 호출됩니다. 랩퍼를 사용하는 경우 CICS는 Java 애플리케이션의 환경을 초기화하며 무엇보다 애플리케이션 수명에서 사용되는 모든 프로세스를 정리합니다. 정리 리턴 코드 0으로 JVM을 종료하면 이 정리 프로세스가 실행되지 않으며 데이터가 불일치할 수 있습니다. JVM 서버에서 애플리케이션이 실행될 때 System.exit()를 사용하면 JVM 서버가 종료되며 CICS 작업을 즉시 거부합니다.

프로시저

1. 기본 메소드를 작성하십시오. CICS는 PROGRAM 자원의 JVMCLASS 속성으로 지정된 클래스에서 main(CommAreaHolder)의 서명을 사용하여 메소드로의 제어 전달을 시도합니다. 이 메소드가 없으면 CICS가 main(String[]) 메소드 호출을 시도합니다.
2. JCICS를 사용하여 오브젝트를 작성하려면 다음 단계를 따르십시오.
   a. 참조를 선언하십시오.

      ```
      TSQ tsq;
      ```

   b. new 연산자를 사용하여 오브젝트를 작성하십시오.

      ```
      tsq = new TSQ()
      ```

   c. setName 메소드를 사용하여 오브젝트 이름을 지정하십시오.

      ```
      tsq.setName("JCICSTSQ");
      ```

3. 오브젝트를 사용하여 CICS와 상호 작용하십시오.

예

이 예제는 TSQ 오브젝트를 작성하고 작성한 임시 저장영역 큐 오브젝트에서 delete 메소드를 호출하고 큐가 비어 있을 때 전달된 예외를 포착하는 방법을 보여줍니다.

```
// 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 제한사항

Java 애플리케이션을 개발할 때 다양한 제한사항이 적용됩니다. CICS에서 애플리케이션이 실행될 때는 다른 문제점이 발생할 수 있습니다.

CICS에서 사용되는 Java 애플리케이션에는 다음 제한사항이 적용됩니다.

- System.exit() 메소드: 이 메소드는 Java 애플리케이션에서 사용할 수 없습니다. 이 메소드를 사용하면 애플리케이션이 비정상적으로 종료되고 JVM 서버가 종료되며 CICS도 종료됩니다. 보안 정책을 사용하여 System.exit()에 대한 지원을 사용 안함으로 설정하십시오. 관련 정보는 192 페이지의 『Java 보안 관리자 사용』의 내용을 참조하십시오.
- JCICS API 호출: 이러한 호출은 OSGi 번들의 활성기 클래스에서 사용할 수 없습니다.
- 번들 활성기에서 사용되는 시작 및 중지 메소드: 이러한 메소드는 적정 시간 내에 리턴되어야 합니다.

Java 애플리케이션에서 데이터 액세스

DB2 및 VSAM의 데이터를 액세스하고 갱신할 수 있는 Java 애플리케이션을 작성할 수 있습니다. 또는 다른 언어의 프로그램에 링크하여 DB2, VSAM, IMS에 액세스할 수 있습니다.

CICS의 데이터에 액세스하기 위해 Java 애플리케이션을 작성하는 경우 다음 기법을 사용할 수 있습니다. CICS 복구 관리자는 데이터 무결성을 유지합니다.

관련 데이터 액세스

다음 방법을 사용하여 DB2의 관련 데이터에 액세스하기 위한 Java 애플리케이션을 작성할 수 있습니다.

- 데이터에 액세스하기 위해 SQL(Structured Query Language) 명령을 사용하는 프로그램에 링크하기 위한 **JCICS LINK** 명령
- 적합한 드라이버를 사용할 수 있는 경우, JDBC(Java Data Base Connectivity) 또는 SQLJ(Structured Query Language for Java) 호출을 사용하여 데이터에 직접 액세스하십시오. DB2에 적합한 JDBC 드라이버를 사용할 수 있습니다. JDBC 및 SQLJ 애플리케이션 프로그래밍 인터페이스 사용에 대한 자세한 정보는 애플리케이션 개발에서 Java 프로그램을 통해 DB2 데이터에 액세스하기 위해 JDBC 및 SQLJ 사용의 내용을 참조하십시오.
- JDBC 또는 SQLJ를 기본 액세스 메커니즘으로 사용하는 JavaBeans. 적합한 Java IDE(Integrated Development Environment)를 사용하여 해당 JavaBeans을 개발할 수 있습니다.

DL/I 데이터 액세스

IMS의 DL/I 데이터에 액세스하려면 Java 애플리케이션이 **JCICS LINK** 명령을 사용하여 데이터 액세스를 위해 EXEC DLI 명령을 실행하는 중간 프로그램으로 링크해야 합니다.

VSAM 데이터 액세스

Java 애플리케이션은 VSAM 데이터에 액세스하기 위해 다음 방법 중 하나를 사용할 수 있습니다.

- VSAM에 직접 액세스하는 JCICS 파일 제어 클래스
- 데이터에 액세스하기 위해 CICS 파일 제어 명령을 실행하는 프로그램에 링크하기 위한 **JCICS LINK** 명령

CICS에서 Java 애플리케이션으로부터의 연결성

CICS 환경의 Java 프로그램은 TCP/IP 소켓을 열고 외부 프로세스와 통신할 수 있습니다. Java 프로그램을 게이트웨이로 사용하여 다른 언어의 CICS 프로그램이 사용할 수 없는 다른 엔터프라이즈 애플리케이션에 연결할 수 있습니다. 예를 들어, 원격 servlet 또는 데이터베이스와 통신할 Java 프로그램을 작성할 수 있습니다.

경우에 따라 이 연결성은 CICS에 통합되어 엔터프라이즈 서비스 품질(QoS)(예: 분산 트랜잭션, ID 전파)을 제공합니다. 다른 경우에는 분산 트랜잭션 및 CICS가 제공하는 다른 서비스 없이 연결성을 사용할 수 있습니다. 필요한 연결 유형에 따라, CICS가 기본적으로 지원하지 않는 엔터프라이즈 애플리케이션과의 연결성을 지원하는 써드 파티 벤더 제품을 사용할 수 있습니다.

일반적으로 CICS 환경의 JVM은 성능 면에서 일괄처리 모드 JVM과 유사합니다. 일괄처리 모드 JVM은 CICS 환경 외부에서 독립형 프로세스로 실행되며 일반적으로 UNIX 시스템 서비스 명령행에서 또는 JCL 작업으로 시작됩니다. 일괄처리 모드 JVM에서 작업을 수행할 수 있는 대부분의 애플리케이션은 CICS의 JVM에서도 동일한 익스텐트까지 실행될 수 있습니다. 예를 들어, 써드 파티 JDBC 드라이버를 사용하여 IBM 이외 데이터베이스와 통신하도록 일괄처리 모드 Java 애플리케이션을 작성하는 경우 동일한 애플리케이션이 CICS의 JVM에서 실행될 수 있습니다. CICS의 JVM에서 IBM 이외 JDBC 드라이버와 같은 벤더 제공 코드를 사용하려면 벤더와 상의하여 해당 코드가 CICS의 JVM에서 실행되도록 지원하는지 여부를 판별하십시오.

CICS의 Java 애플리케이션 작동에 대한 자세한 정보는 33 페이지의 『CICS의 Java 런타임 환경』의 내용을 참조하십시오.

CICS 환경의 JVM에서 실행되는 일괄처리 모드 애플리케이션은 일반적으로 CICS의 기능을 이용하지 않습니다. 예를 들어, CICS의 Java 프로그램이 써드 파티 JDBC 드라이버를 사용하여 IBM 이외 데이터베이스에서 레코드를 갱신하는 경우 CICS가 해당 활동을 인식하지 못하고 현재 CICS 트랜잭션에 갱신을 포함하려고 시도하지 않습니다.

JDBC 및 SQLJ를 사용하여 Java 프로그램에서 DB2 데이터 액세스

CICS에서 실행되는 Java 프로그램은 여러 가지 방법을 사용하여 DB2 데이터베이스에 저장된 데이터에 액세스할 수 있습니다.

Java 프로그램의 기능은 다음과 같습니다.

- JCICS LINK 명령을 사용하여 SQL(Structured Query Language) 명령으로 데이터에 액세스하는 CICS 프로그램에 링크할 수 있습니다.
- JDBC(Java Data Base Connectivity) 및 SQLJ(Structured Query Language for Java) API(Application Programming Interface)를 사용하여 데이터에 직접 액세스할 수 있습니다.

  『CICS DB2 환경에서 JDBC 및 SQLJ 작업 작성』
  CICS용 Java 애플리케이션이 JDBC 및 SQLJ 요청을 작성하는 경우 해당 요청은 DB2가 제공하는 JDBC 드라이버로 처리됩니다.

  80 페이지의 『JVM 서버가 DB2를 지원하도록 구성』
  JVM 서버는 Java 애플리케이션을 위한 런타임 환경입니다. JVM 서버에서 JDBC 및 SQLJ 기반 애플리케이션을 지원하도록 구성할 수 있습니다.

  81 페이지의 『CICS DB2 환경에서 JDBC 및 SQLJ로 프로그래밍』
  CICS용 Java 프로그램은 일반 CICS 프로그래밍 모델보다 제한적인 JDBC API(Application Programming Interface)의 프로그래밍 규칙을 준수해야 합니다.

  81 페이지의 『데이터베이스 연결 획득』
  SQL문을 실행하기 전에 JDBC 또는 SQLJ 애플리케이션이 데이터베이스에 대한 연결 또는 연결 컨텍스트를 획득해야 합니다. 애플리케이션은 두 Java 클래스 중 하나를 사용하여 대상 데이터 소스에 연결합니다.

  84 페이지의 『데이터베이스 연결 닫기』
  데이터베이스에 대한 연결을 닫으면 JDBC 및 SQLJ 자원이 자동으로 해제됩니다. 일반적으로 태스크가 종료되면 데이터베이스에 대한 연결이 닫힙니다.

  84 페이지의 『작업 단위 확약』
  작업 단위를 확약하기 위해 JDBC 및 SQLJ 애플리케이션이 JDBC 및 SQLJ 확약 및 롤백 메소드 호출을 실행할 수 있습니다. DB2 JDBC 드라이버는 이러한 호출을 JCICS 확약 또는 JCICS 롤백 호출로 변환하여 CICS 동기점을 취합니다.

  86 페이지의 『JDBC 또는 SQLJ 요청 시 CICS 이상종료』
  DB2 제공 JDBC 드라이버로 빌드된 EXEC SQL 요청 처리 중에 실행된 CICS 이상 종료는 Java 예외로 변환되지 않으므로 CICS용 Java 애플리케이션으로 캐치할 수 없습니다. CICS 트랜잭션은 이상 종료되고 마지막 동기점으로 롤백됩니다.

CICS DB2 환경에서 JDBC 및 SQLJ 작업 작성

CICS용 Java 애플리케이션이 JDBC 및 SQLJ 요청을 작성하는 경우 해당 요청은 DB2가 제공하는 JDBC 드라이버로 처리됩니다.

이 태스크 정보

CICS 환경에서 DB2 제공 JDBC 드라이버는 CICS DB2 언어 인터페이스(스텁) DSNCLI와 링크 편집됩니다. 드라이버는 JDBC 또는 SQLJ 요청을 해당 EXEC SQL로 변환합니다. DB2 제공 JDBC 드라이버로부터의 변환된 요청은 다른 프로그램(예: COBOL 프로그램)의 EXEC SQL 요청과 동일한 방식으로 CICS DB2 첨부 기능으로 이동합니다. 따라서 CICS DB2용 Java 프로그램과 CICS DB2용 다른 프로그램 간에 운영 차이가 없고 RDO를 사용하여 사용 가능한 모든 사용자 정의 및 조정 선택항목이 CICS DB2용 Java 프로그램에 적용됩니다.

DB2와 함께 분배되는 DB2 Universal JDBC 드라이버를 사용하십시오. DB2가 제공하는 JDBC 드라이버를 사용하려면 JDBC 드라이버의 적절한 레벨을 지원하는 DB2 서브시스템에 CICS가 연결되어야 합니다.

JDBC 및 SQLJ API(Application Programming Interface)를 사용하는 Java 애플리케이션 코딩 및 빌드 방법에 대한 세부사항은 해당 DB2 버전에 적용되는 *DB2 for z/OS: Programming for Java*의 내용을 참조하십시오.

JVM 서버가 DB2를 지원하도록 구성

JVM 서버는 Java 애플리케이션을 위한 런타임 환경입니다. JVM 서버에서 JDBC 및 SQLJ 기반 애플리케이션을 지원하도록 구성할 수 있습니다.

시작하기 전에

DB2에 JVM 서버를 사용하려면 IBM Data Server Driver for JDBC and SQLJ의 최신 버전이 필요합니다. 필요한 APAR에 대한 자세한 정보는 z/OS용 CICS Transaction Server V5.2 세부사항 시스템 요구사항의 내용을 참조하십시오. 또한 CICS STEPLIB 병합(concatenation)에 DB2 SDSNLOD2 라이브러리를 추가해야 합니다.

프로시저

애플리케이션에서 DB2와 함께 제공되는 JDBC 드라이버를 사용하도록 설정하십시오.

1. 올바른 JDBC 드라이버 정보로 JVM 서버를 설정하십시오. Liberty JVM 서버의 경우 구성에서 웹 애플리케이션에 대한 Liberty JVM 서버 구성의 내용을 참조하십시오. OSGi JVM 서버의 경우 구성에서 OSGi 애플리케이션에 대한 JVM 서버 구성의 내용을 참조하십시오.
2. JDBC 드라이버를 CICS 환경에서 실행하는 경우 시스템 등록 정보를 변경할 수 있습니다. JVM 서버의 JVM 프로파일에서 JDBC 드라이버와 관련된 시스템 등록 정보를 설정할 수 있습니다. JDBC 드라이버가 사용할 시스템 등록 정보 파일의 이름을 지정하는 DB2 환경 변수 **DB2SQLJPROPERTIES**는 사용하지 않습니다. 대부분의 DB2 JDBC 드라이버 시스템 등록 정보는 CICS 환경에서 사용하지 않습니다. DB2

JDBC 드라이버 등록 정보와 CICS 환경에서 다른 의미를 갖거나 사용하지 않는 등록 정보에 대한 목록은 *DB2 for z/OS: Programming for Java*의 내용을 참조하십시오.

다음에 수행할 작업

Java 2 보안 정책 메커니즘을 활성화한 OSGi JVM 서버의 Java 애플리케이션에서 JDBC 또는 SQLJ를 사용하려면 전개에서 Java 보안 관리자 사용의 내용을 참조하십시오.

CICS DB2 환경에서 JDBC 및 SQLJ로 프로그래밍

CICS용 Java 프로그램은 일반 CICS 프로그래밍 모델보다 제한적인 JDBC API(Application Programming Interface)의 프로그래밍 규칙을 준수해야 합니다.

IBM Data Server Driver for JDBC and SQLJ는 JDBC 4.0 이하 버전을 지원합니다.

참고: Liberty JVM 서버에서는 SQLJ가 지원되지 않습니다.

JDBC API에 대한 자세한 정보는 JDBC 웹 사이트를 참조하십시오. JDBC 및 SQLJ API를 사용하는 Java 애플리케이션 개발에 대한 정보는 z/OS용 정보 관리 소프트웨어 Solutions Information Center를 참조하십시오.

데이터베이스 연결 획득

SQL문을 실행하기 전에 JDBC 또는 SQLJ 애플리케이션이 데이터베이스에 대한 연결 또는 연결 컨텍스트를 획득해야 합니다. 애플리케이션은 두 Java 클래스 중 하나를 사용하여 대상 데이터 소스에 연결합니다.

이 태스크 정보

두 Java 클래스는 다음과 같습니다.

- DriverManager: 이 클래스가 데이터베이스 URL에 의해 지정되는 데이터베이스에 애플리케이션을 연결합니다.
- DataSource: 이 인터페이스는 DriverManager보다 우선합니다. 기본 데이터베이스에 대한 세부사항이 애플리케이션에 대해 투명하게 처리되므로 애플리케이션의 이동성이 향상됩니다. DataSource 구현은 Java 프로그램 외부에 작성되며 JNDI(Java Naming and Directory Interface) DataSource 이름 찾아보기를 통해 얻을 수 있습니다. DataSource 인터페이스는 Liberty JVM 서버에서만 사용할 수 있습니다.

CICS DB2 환경에서 사용자 ID와 암호를 지정하지 않아도 되며 대신 기존 CICS DB2 보안 프로시저가 사용됩니다. IBM Data Server Driver for JDBC and SQLJ는 JDBC 4 이하를 지원합니다.

CICS Explorer SDK에 제공되는 OSGi 및 Liberty JVM 서버 모두에 대한 JDBC 예제가 있습니다. 자세한 정보는 애플리케이션 개발에서 서블릿 예제 시작하기의 내용을 참조하십시오.

JDBC DriverManager 및 DataSource 인터페이스를 사용하여 연결을 얻는 방법에 대한 자세한 정보는 z/OS용 DB2: DB2 버전에 적합한 Java 프로그래밍을 참조하십시오.

『데이터베이스에 대한 DriverManager 연결 얻기』
DriverManager 클래스를 사용한 데이터베이스로의 연결은 JDBC 드라이버에 제공되는 데이터베이스 URL(Uniform Resource Locator)로 식별합니다.

83 페이지의 『데이터베이스에 대한 데이터 소스 연결 얻기』
데이터 소스 인터페이스를 사용하면 JNDI(Java Naming and Directory Interface) 데이터 소스 이름 찾아보기를 통해 데이터베이스에 대한 연결을 얻을 수 있습니다. JDBC 데이터 소스 구현은 CICS JDBC가 제공합니다.

84 페이지의 『가능한 연결 수』
CICS용 Java 애플리케이션은 한 번에 최대 하나의 JDBC 연결 또는 SQLJ 연결 컨텍스트를 열 수 있습니다.

데이터베이스에 대한 DriverManager 연결 얻기

DriverManager 클래스를 사용한 데이터베이스로의 연결은 JDBC 드라이버에 제공되는 데이터베이스 URL(Uniform Resource Locator)로 식별합니다.

이 태스크 정보

JVM 프로파일 `OSGI_BUNDLES` 및 `LIBPATH_SUFFIX` JVM 프로파일의 DB2 JDBC jar 및 고유 DLL을 지정하여 OSGi JVM 서버에 대해 `DriverManager` 연결이 구성됩니다.

Liberty JVM 서버에서 DriverManager 구성은 CICS JDBC Liberty 특성을 통해 제공됩니다.

JDBC 드라이버는 다음과 같은 두 가지 유형의 URL을 인식합니다.

기본 URL

기본 URL에는 DB2 서브시스템의 위치 이름이 포함되지 않습니다. DB2 for z/OS의 기본 URL은 다음 두 가지 형식 중 하나로 지정될 수 있습니다.

```
jdbc:db2os390sqlj:
```
또는
```
jdbc:default:connection
```

기본 URL이 지정되면 CICS가 연결되는 로컬 DB2에 대한 연결이 애플리케이션에 제공됩니다. 설치 시 DB2 데이터 공유를 사용하는 경우 로컬 DB2에서 sysplex의 모든 데이터에 액세스할 수 있습니다.

명시적 URL

명시적 URL에는 DB2 서브시스템의 위치 이름이 포함됩니다. DB2 for z/OS의 명시적 URL의 기본 구조는 다음과 같습니다.

```
jdbc:db2os390:<location-name>
```
또는
```
jdbc:db2os390sqlj:<location-name>
```

일반적으로 위치 이름은 CICS가 연결된 로컬 DB2의 이름입니다. 그러나 액세스할 원격 DB2의 이름을 지정할 수 있습니다. 이 경우 CICS는 로컬 DB2를 통과 지점으로 사용하며 DB2 분산 데이터 기능을 사용하여 원격 DB2에 액세스합니다.

CICS 환경에서는 기본 URL을 사용하는 것이 좋습니다. 명시적 URL을 사용하면 연결이 닫힐 때 특정 조치가 발생할 수 있습니다. 이는 동일한 애플리케이션 세트에서 여러 프로그램을 사용할 때 편리합니다. 또한 기본 URL을 사용하는 경우에는 연결 작동이 JDBC 드라이버 버전에 따라 영향을 받지 않습니다.

연결을 얻으려면 Java 애플리케이션이 `getConnection()` 메소드를 URL과 함께 호출해야 합니다. 예:

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

데이터베이스에 대한 데이터 소스 연결 얻기

데이터 소스 인터페이스를 사용하면 JNDI(Java Naming and Directory Interface) 데이터 소스 이름 찾아보기를 통해 데이터베이스에 대한 연결을 얻을 수 있습니다. JDBC 데이터 소스 구현은 CICS JDBC가 제공합니다.

이 태스크 정보

데이터 소스 연결은 Liberty JVM 서버에서 CICS JDBC Liberty 특성을 통해 구성됩니다.

데이터 소스를 구성한 후 애플리케이션은 Liberty 서버 구성의 `cicsts_dataSource` 요소에 지정된 `jndiName`의 값을 사용하여 JNDI 이름 지정 서비스에서 해당 `DataSource` 클래스의 인스턴스를 얻을 수 있습니다. 그런 다음 해당 `DataSource` 오브젝트에 대해 `getConnection()` 메소드를 호출하여 연결을 얻을 수 있습니다. 예를 들어, 다음과 같습니다.

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

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

가능한 연결 수

CICS용 Java 애플리케이션은 한 번에 최대 하나의 JDBC 연결 또는 SQLJ 연결 컨텍스트를 열 수 있습니다.

JDBC에서는 애플리케이션에 동시에 여러 연결을 가질 수 있지만 CICS는 이를 허용하지 않습니다. 그러나 애플리케이션이 기존 연결을 닫고 새 DB2 위치에 대한 연결을 열 수 있습니다.

열린 연결이 있는 애플리케이션은 JDBC 또는 SQLJ를 사용하려는 다른 애플리케이션에 링크하기 전에 연결을 닫아야 합니다. 애플리케이션 세트에 포함되는 Java 프로그램의 경우 연결 닫기의 의미를 고려해야 합니다. 명시적 URL을 사용하는 경우 연결을 닫으면 동기점이 적용되기 때문입니다. 기본 URL을 사용하는 경우 연결이 닫힐 때 동기점을 적용하지 않아도 됩니다. 자세한 정보는 『작업 단위 확약』의 내용을 참조하십시오.

데이터베이스 연결 닫기

데이터베이스에 대한 연결을 닫으면 JDBC 및 SQLJ 자원이 자동으로 해제됩니다. 일반적으로 태스크가 종료되면 데이터베이스에 대한 연결이 닫힙니다.

성능상의 이유로 인해 동일한 JVM의 후속 사용자가 사용할 수 있도록 애플리케이션이 JDBC 연결을 열어둘 수 있습니다. JDBC 연결이 계속 열려 있으면 시간 경과에 따라 JDBC 자원이 누수되지 않는지 애플리케이션이 확인해야 합니다.

애플리케이션이 JDBC 또는 SQLJ 연결을 계속 열어두면 애플리케이션이 다음을 수행해야 합니다.

- JDBC 및 SQLJ 자원이 해제되었는지 확인합니다.
- 기본 DB2 연결에 대해 다음 DB2 SIGNON을 복구합니다.
- 캐시된 연결이 올바르지 않게 되면 연결을 재생합니다
  (예: StaleConnection, SQLCODE=4499).

연결이 캐시되는 경우 지정된 횟수의 트랜잭션 이후에 연결을 재생하는 논리를 애플리케이션에 포함해야 합니다. 캐시된 연결을 재생하면 자원 누수를 방지할 수 있습니다.

작업 단위 확약

작업 단위를 확약하기 위해 JDBC 및 SQLJ 애플리케이션이 JDBC 및 SQLJ 확약 및 롤백 메소드 호출을 실행할 수 있습니다. DB2 JDBC 드라이버는 이러한 호출을 JCICS 확약 또는 JCICS 롤백 호출로 변환하여 CICS 동기점을 취합니다.

이 태스크 정보

JDBC 또는 SQLJ 확약은 DB2 갱신뿐 아니라 전체 CICS 작업 단위를 확약시킵니다. CICS는 나머지 CICS 작업 단위와 관계 없이 JDBC 연결을 사용한 작업 완료 확약을 지원하지 않습니다.

JDBC 또는 SQLJ 애플리케이션은 또한 JCICS 확약 또는 롤백을 직접 실행할 수 있으며 이는 JDBC 또는 SQLJ 확약 또는 롤백 메소드 호출 실행과 동일한 결과를 나타냅니다. 전체 작업 단위는 DB2 갱신 및 CICS 제어 자원 갱신 모두 함께 확약 또는 롤백됩니다.

JDBC 연결 작업을 수행하는 경우 동기점 획득을 방지하지 못하거나, DB2 데이터베이스에 대한 연결이 닫힐 때 작업 단위가 확약되는 것을 방지하지 못할 수 있습니다. 이러한 상황은 다음과 같은 경우에 해당됩니다.

- JDBC 연결의 자동 확약 등록 정보를 사용했습니다.
- 명시적 URL을 사용하여 DriverManager 연결을 획득했습니다.

독립형 애플리케이션의 경우, CICS에서 연결이 닫힐 때 취하는 동기점뿐 아니라 태스크 동기점 끝을 취할 수 있으므로 문제점을 야기하지 않습니다. 그러나 JDBC 및 SQLJ API(Application Programming Interface)는 각 작업 단위에 복수 애플리케이션 개념을 지원하지 않습니다. 애플리케이션을 구성하는 프로그램이 여러 개 있는 경우 하나의 프로그램이 단일 작업 단위 과정에서 DB2에 액세스한 다음 역시 DB2에 액세스하는 다른 프로그램을 호출할 수 있습니다. 이러한 프로그램을 JDBC 또는 SQLJ를 사용하는 Java 프로그램으로 지정하려면 DB2에 대한 연결이 닫힐 때 작업 단위가 확약되지 않는지 확인해야 합니다. 그렇지 않으면 애플리케이션이 계획대로 작동하지 않습니다. 특히 기존 애플리케이션의 프로그램을 JDBC 또는 SQLJ를 사용하는 Java 프로그램으로 바꾸고 해당 프로그램 간에 동일한 CICS-DB2 스레드를 공유하려면 이 요구사항에 유의해야 합니다. 이 문제를 해결하려면 DriverManager를 기본 URL 또는 DataSource 연결과 함께 사용하십시오.

86 페이지의 『autocommit』
JDBC 애플리케이션은 JDBC 연결의 autocommit 등록 정보를 사용할 수 있습니다. autocommit 등록 정보는 DB2로 갱신할 때마다 확약을 실행합니다. 이 확약은 CICS 확약이므로 전체 작업 단위가 확약됩니다.

86 페이지의 『명시적 URL과 기본 URL이 있는 DriverManager에 대한 동기점 이슈』
JDBC 또는 SQLJ를 사용하는 CICS용 Java 애플리케이션이 명시적 URL을 사용하여 연결을 획득하는 경우, SYNCONRETURN 속성을 사용하여 링크된 DPL 서버 프로그램의 환경과 유사한 환경에서 애플리케이션이 실행됩니다.

autocommit

JDBC 애플리케이션은 JDBC 연결의 autocommit 등록 정보를 사용할 수 있습니다. autocommit 등록 정보는 DB2로 갱신할 때마다 확약을 실행합니다. 이 확약은 CICS 확약이므로 전체 작업 단위가 확약됩니다.

또한 autocommit 등록 정보를 사용하면 명시적 URL 또는 기본 URL을 사용하여 얻은 DriverManager 연결과 type-2 DataSource 연결의 경우 연결이 닫힐 때 확약이 실행됩니다.

CICS 환경에서 autocommit을 사용하는 것은 권장되지 않습니다. 따라서 DB2 JDBC 드라이버는 CICS 환경에서 실행될 때 기본값 `autocommit(false)`를 설정합니다. CICS 이외의 환경에서 기본값은 `autocommit(true)`입니다.

명시적 URL과 기본 URL이 있는 DriverManager에 대한 동기점 이슈

JDBC 또는 SQLJ를 사용하는 CICS용 Java 애플리케이션이 명시적 URL을 사용하여 연결을 획득하는 경우, SYNCONRETURN 속성을 사용하여 링크된 DPL 서버 프로그램의 환경과 유사한 환경에서 애플리케이션이 실행됩니다.

JDBC 또는 SQLJ를 사용하는 애플리케이션이 명시적 URL 연결을 닫을 때 CICS가 IBM Data Server Driver for JDBC and SQLJ를 사용하는 경우에는 암시적 동기점을 취하지 않습니다.

그러나 명시적 URL 연결 닫기는 작업 단위 경계에서만 성공합니다. 따라서 애플리케이션은 연결을 닫기 전에 JDBC 또는 SQLJ 확약 메소드 호출이나 JCICS 확약을 실행하여 동기점을 취해야 합니다. 애플리케이션은 autocommit(true)를 사용하여 동기점을 취했는지 확인할 수 있지만 CICS 환경에서는 이 등록 정보 사용이 권장되지 않습니다. 애플리케이션이 명시적 URL 연결을 닫으면 작업 단위가 끝난 것입니다.

이 제한사항은 명시적 URL 대신 **기본** URL을 사용하여 연결을 획득하거나 기본 URL 연결을 제공하는 데이터 소스를 사용하여 극복할 수 있습니다(81 페이지의 『데이터베이스 연결 획득』 참조). 기본 URL을 사용하는 경우, Java 애플리케이션이 작업 단위 경계에서 연결을 닫지 않아도 되므로 연결이 닫힐 때 동기점을 취하지 않습니다(autocommit(true)이 지정되지 않은 경우).

CICS 환경에서는 항상 기본 URL 연결을 사용해야 합니다.

JDBC 또는 SQLJ 요청 시 CICS 이상종료

DB2 제공 JDBC 드라이버로 빌드된 EXEC SQL 요청 처리 중에 실행된 CICS 이상종료는 Java 예외로 변환되지 않으므로 CICS용 Java 애플리케이션으로 캐치할 수 없습니다. CICS 트랜잭션은 이상 종료되고 마지막 동기점으로 롤백됩니다.

Liberty JVM 서버에서 실행할 Java 웹 애플리케이션 개발

Java servlet 및 JSP 페이지를 사용하는 Java™ 웹 애플리케이션을 전개하려는 경우 Liberty JVM 서버에서 웹 컨테이너를 실행하도록 구성합니다.

CICS Explorer SDK 다운로드 및 설치 방법을 알아보려면 CICS Explorer(r) SDK 설치를 참조하십시오.

『개발 환경 설정』
Liberty JVM 서버에서 실행할 웹 애플리케이션을 개발하려면 CICS Explorer SDK와 함께 제공되는 servlet 및 JSP 기능을 설치해야 합니다.

88 페이지의 『servlet 및 JSP 애플리케이션 개발』
CICS 애플리케이션에 최신 인터페이스를 제공하기 위해 Java servlet 및 JSP 기술을 사용하는 프리젠테이션 계층을 개발할 수 있습니다. Eclipse 기반 웹 개발 도구는 해당 애플리케이션을 작성하기 위한 개발 플랫폼을 제공하고 CICS Explorer SDK는 애플리케이션을 빌드, 패키지, 전개하여 CICS에서 실행하기 위한 지원을 제공합니다.

92 페이지의 『Liberty JVM 서버에서 실행할 Java 웹 애플리케이션 이주』
Java 웹 애플리케이션이 네트워크를 통해 CICS에 액세스하는 Liberty 프로파일 인스턴스에서 실행되는 경우 Liberty JVM 서버에서 애플리케이션을 실행하여 성능을 최적화할 수 있습니다.

개발 환경 설정

Liberty JVM 서버에서 실행할 웹 애플리케이션을 개발하려면 CICS Explorer SDK와 함께 제공되는 servlet 및 JSP 기능을 설치해야 합니다.

이 태스크 정보

Liberty 프로파일의 개발자 도구를 사용하여 Liberty 프로파일 서버의 인스턴스에서 실행될 수 있는 servlet과 JSP 애플리케이션을 개발할 수 있습니다. Liberty 프로파일은 단순 웹 애플리케이션을 실행하기 위해 WAS(WebSphere Application Server)와 함께 제공되는 애플리케이션 서버 인스턴스로 빠르게 구성, 전개, 시작할 수 있습니다.

Liberty JVM 서버에 웹 애플리케이션을 설치하여 CICS 환경에서 servlet과 JSP 페이지를 실행할 수도 있습니다. CICS Explorer SDK는 CICS용 Java 애플리케이션을 개발 및 전개할 수 있는 도구를 제공합니다. 이 도구와 개발자 도구를 함께 사용하여 웹 인터페이스로 Java 애플리케이션을 CICS에 전개할 수 있습니다.

Liberty로 대상 플랫폼을 CICS TS V5.x Runtime으로 설정하십시오. 그렇지 않으면 작업 공간에서 예제를 작성할 때 컴파일 오류가 발생할 수 있습니다. 자세한 정보는 servlet 예제 작성의 내용을 참조하십시오.

프로시저

1. CICS Explorer SDK를 설치했지만 servlet 및 JSP 기능을 설치하지 않은 경우 CICS Explorer SDK 다운로드 아카이브에서 이 기능을 설치할 수 있습니다.
   a. Eclipse IDE에서 **도움말 > 새 소프트웨어 설치**를 누르십시오.

      참고: 설치 중에 모든 갱신 사이트를 확인하여 필수 소프트웨어 찾기 선택항목을 선택했는지 확인하십시오. Liberty 프로파일 기능은 다른 Eclipse 구성요소에 대한 종속 항목이 있으므로 갱신 관리자가 해당 종속 항목을 다운로드 및 설치하는지 확인해야 합니다.
   b. 목록에서 CICS Explorer SDK 다운로드 파일을 선택하십시오. **추가**를 누르십시오. "사이트 추가" 대화 상자에서 **아카이브**를 누르십시오.
   c. CICS Explorer SDK 다운로드 파일을 찾아보고 **열기**를 누르십시오.
   d. **IBM CICS Explorer**를 펼치십시오.
   e. **IBM CICS SDK for Servlet 및 JSP 지원 v5.1**을 선택하고 **다음**을 누르십시오.
   f. 라이센스 정보를 읽고 승인한 후 **완료**를 눌러 기능을 설치하십시오. Eclipse IDE에 아직 설치되지 않은 모든 종속 항목 또한 설치됩니다.
2. 옵션: EBA를 전개하려는 경우 WebSphere Application Server Developer Tools for Eclipse를 설치해야 합니다. 개발자 도구를 사용하여 웹 애플리케이션을 로컬로 테스트할 수도 있습니다. 개발자 도구 다운로드 및 설치에 대한 자세한 정보는 Eclipse용 WAS(Websphere Application Server) 개발자 도구 설치의 내용을 참조하십시오.

결과

Eclipse IDE를 다시 시작하면 Liberty JVM 서버를 실행하는 데 적합한 웹 애플리케이션을 개발, 전개, 테스트할 수 있는 도구가 포함됩니다.

다음에 수행할 작업

Liberty JVM 서버에서 실행할 웹 애플리케이션 개발을 시작할 수 있습니다. CICS Explorer SDK가 제공하는 예제를 사용하여 시작할 수 있습니다. 시작하기에 대한 자세한 정보는 topics/gettingstarted_liberty.dita의 내용을 참조하십시오.

servlet 및 JSP 애플리케이션 개발

CICS 애플리케이션에 최신 인터페이스를 제공하기 위해 Java servlet 및 JSP 기술을 사용하는 프리젠테이션 계층을 개발할 수 있습니다. Eclipse 기반 웹 개발 도구는 해당 애플리케이션을 작성하기 위한 개발 플랫폼을 제공하고 CICS Explorer SDK는 애플리케이션을 빌드, 패키지, 전개하여 CICS에서 실행하기 위한 지원을 제공합니다.

이 태스크 정보

Liberty 서버, 동적 웹 프로젝트 및 OSGi 번들 프로젝트에 전개할 수 있는 servlet 및 JSP를 포함하는 두 가지 유형의 프로젝트가 있습니다.

『동적 웹 프로젝트 작성』
Java 애플리케이션에 대한 웹 프리젠테이션 계층을 개발하기 위해 동적 웹 프로젝트를 작성할 수 있습니다.

90 페이지의『OSGi 애플리케이션 프로젝트 작성』
OSGi 애플리케이션 프로젝트(EBA)는 번들 세트를 함께 그룹화합니다. 애플리케이션은 웹 사용 번들 및 OSGi 번들과 같은 여러 번들 유형으로 구성될 수 있습니다.

관련 태스크:

topics/gettingstarted_liberty.dita
CICS Explorer SDK에는 CICS의 Liberty JVM 서버에서 실행될 수 있는 servlet 및 JSP 페이지 개발을 시작하는 데 유용한 예제가 포함되어 있습니다.

92 페이지의『Liberty JVM 서버에서 실행할 Java 웹 애플리케이션 이주』
Java 웹 애플리케이션이 네트워크를 통해 CICS에 액세스하는 Liberty 프로파일 인스턴스에서 실행되는 경우 Liberty JVM 서버에서 애플리케이션을 실행하여 성능을 최적화할 수 있습니다.

동적 웹 프로젝트 작성

Java 애플리케이션에 대한 웹 프리젠테이션 계층을 개발하기 위해 동적 웹 프로젝트를 작성할 수 있습니다.

시작하기 전에

웹 개발 도구가 Eclipse IDE에 설치되었는지 확인하십시오. 자세한 정보는 87 페이지의『개발 환경 설정』의 내용을 참조하십시오.

이 태스크 정보

CICS Explorer SDK 도움말은 웹 애플리케이션 개발 및 패키지화를 위해 다음 단계 각각을 완료할 수 있는 방법에 대한 전체 세부사항을 제공합니다.

프로시저

1. 애플리케이션의 동적 웹 프로젝트를 작성하십시오. 빌드 경로를 갱신하여 Liberty 라이브러리를 추가해야 합니다. Eclipse에서는 다른 유형의 웹 프로젝트를 작성할 수 있지만 CICS는 OSGi 번들 프로젝트와 동적 웹 프로젝트만 지원합니다.
   a. 동적 웹 프로젝트를 마우스 오른쪽 단추로 누르고 **빌드 경로 > 빌드 경로 구성**을 누르십시오. 프로젝트에 대한 등록 정보 대화 상자가 열립니다.
   b. Java 빌드 경로에서 **라이브러리** 탭을 누르십시오.
   c. **라이브러리 추가**를 누르고 Liberty JVM 서버 라이브러리를 선택하십시오.

d. 라이브러리 추가를 완료하려면 **다음**을 누르고 CICS 버전을 선택하고 **완료**를 누르십시오.

e. 변경사항을 저장하려면 **확인**을 누르십시오.

2. 웹 애플리케이션을 개발하십시오. JCICS API를 사용하여 CICS 서비스에 액세스하고 DB2에 연결할 수 있습니다. CICS Explorer SDK에는 JCICS 및 DB2를 사용하는 OSGi 번들과 웹 구성요소의 예제가 포함됩니다.
3. 옵션: CICS 보안으로 애플리케이션을 보호하려면 동적 웹 프로젝트에서 CICS 보안 제한조건을 포함할 web.xml 파일을 작성하십시오. CICS Explorer SDK에는 CICS에 대한 올바른 정보를 포함하는 이 파일의 템플리트가 포함됩니다. 자세한 정보는 187 페이지의 『Liberty JVM 서버에서 사용자 인증』의 내용을 참조하십시오. CICS 보안은 기본 인증을 사용하여 애플리케이션 요청에서 사용자 ID와 암호를 확인합니다. 대신 Liberty 보안을 사용할 수 있지만 보안 역할과 기본 사용자 레지스트리를 제공해야 합니다. 경고: RequestDispatcher.forward() 메소드를 사용하여 특정 servlet에서 다른 servlet으로 요청을 전달하는 경우 클라이언트에서 요청되는 첫 번째 servlet에서만 보안 검사가 수행됩니다.
4. 하나 이상의 CICS 번들 프로젝트를 작성하여 애플리케이션을 패키지화하십시오. OSGi 애플리케이션 프로젝트, 동적 웹 프로젝트, OSGi 번들 프로젝트에 참조를 추가하고 CICS 자원에 대한 정의와 가져오기를 추가할 수 있습니다. 각 CICS 번들에 ID 및 버전을 포함시켜서 변경사항을 세부적으로 관리할 수 있습니다.
5. 옵션: 특정 트랜잭션에서 실행할 URI로부터의 인바운드 웹 요청을 맵핑하려는 경우 CICS 번들에 URIMAP 및 TRANSACTION 자원을 추가하십시오.이러한 자원을 정의하지 않으면 모든 작업이 제공된 트랜잭션(CJSA)에서 실행됩니다. 이러한 자원은 동적으로 설치되며 CICS에서 번들의 일부로 관리됩니다.

결과

개발 환경을 설정하고 동적 웹 프로젝트에서 웹 애플리케이션을 작성했으며 전개 대상으로 패키지화했습니다.

다음에 수행할 작업

애플리케이션을 전개할 수 있는 준비가 되면 CICS 번들 프로젝트를 zFS로 내보내십시오. 참조된 프로젝트가 빌드되어 zFS로의 전송에 포함됩니다. 또는 애플리케이션을 WAR로 내보내고 실행 Liberty JVM 서버의 dropins 디렉토리에 전개하여 Liberty 전개 모델을 따를 수 있습니다.

OSGi 애플리케이션 프로젝트 작성

OSGi 애플리케이션 프로젝트(EBA)는 번들 세트를 함께 그룹화합니다. 애플리케이션은 웹 사용 번들 및 OSGi 번들과 같은 여러 번들 유형으로 구성될 수 있습니다.

시작하기 전에

웹 개발 도구가 Eclipse IDE에 설치되었는지 확인하십시오. 자세한 정보는 87 페이지의 『개발 환경 설정』의 내용을 참조하십시오.

이 태스크 정보

CICS Explorer SDK 도움말은 웹 애플리케이션 개발 및 패키지화를 위해 다음 단계 각각을 완료할 수 있는 방법에 대한 전체 세부사항을 제공합니다.

프로시저

1. **CICS TS V5.2 with Liberty and PHP** 템플리트를 사용하여 Java 개발에 맞게 대상 플랫폼을 설정하십시오. 대상이 현재 Eclipse 설치보다 최신 버전임을 나타내는 경고를 수신하지만 이 경고 메시지는 무시할 수 있습니다.
2. 애플리케이션의 OSGi 번들 프로젝트를 작성하십시오. 대상 플랫폼은 라이브러리를 사용할 수 있도록 효과적으로 지정할 수 있으므로 번들에 적절한 Import문을 포함해야 합니다. Eclipse에서는 다른 유형의 웹 프로젝트를 작성할 수 있지만 CICS는 OSGi 번들 프로젝트와 동적 웹 프로젝트만 지원합니다. 웹 사용 OSGi 번들 프로젝트는 동적 웹 프로젝트의 해당 번들입니다. 웹 사용 OSGi 번들 프로젝트를 사용하여 애플리케이션을 OSGi 애플리케이션 프로젝트(엔터프라이즈 번들 아카이브 또는 EBA 파일)로 전개하거나 Java 코드를 다른 OSGi 번들에서 사용할 수 있습니다. 웹 사용 OSGi 번들 프로젝트(WAB 파일)와 웹을 사용하지 않는 OSGi 번들 프로젝트를 OSGi 애플리케이션 프로젝트에서 혼합할 수 있습니다. 웹 사용 OSGi 번들 프로젝트는 일반적으로 애플리케이션의 프론트 엔드를 구현하며 비즈니스 로직을 포함하는 웹 이외 OSGi 번들과 상호작용합니다.
3. 웹 애플리케이션을 개발하십시오. JCICS API를 사용하여 CICS 서비스에 액세스하고 DB2에 연결할 수 있습니다. CICS Explorer SDK에는 JCICS 및 DB2를 사용하는 OSGi 번들과 웹 구성요소의 예제가 포함됩니다. JCICS를 사용하여 표시 논리와 비즈니스를 구분하는 OSGi 번들을 작성하십시오. OSGi 번들의 시맨틱 버전화를 사용하여 애플리케이션의 비즈니스 로직에 대한 갱신을 관리할 수도 있습니다. DB2를 사용하는 각 WAB 또는 OSGi 번들의 경우 번들 Manifest에 `com.ibm.db2.jcc`의 Import-Package 헤더를 포함하십시오.
4. 옵션: CICS 보안으로 애플리케이션을 보호하려면 동적 웹 프로젝트에서 CICS 보안 제한조건을 포함할 web.xml 파일을 작성하십시오. CICS Explorer SDK에는 CICS에 대한 올바른 정보를 포함하는 이 파일의 템플리트가 포함됩니다. 자세한 정보는 187 페이지의 『Liberty JVM 서버에서 사용자 인증』의 내용을 참조하십시오. CICS 보안은 기본 인증을 사용하여 애플리케이션 요청에서 사용자 ID와 암호를 확인합니다. 대신 Liberty 보안을 사용할 수 있지만 보안 역할과 기본 사용자 레지스트리를 제공해야 합니다. 경고: RequestDispatcher.forward() 메소드를 사용하여 특정 servlet에서 다른 servlet으로 요청을 전달하는 경우 클라이언트에서 요청되는 첫 번째 servlet에서만 보안 검사가 수행됩니다.

5. OSGi 번들을 참조하는 OSGi 애플리케이션 프로젝트를 작성할 수 있습니다.
6. 하나 이상의 CICS 번들 프로젝트를 작성하여 애플리케이션을 패키지화하십시오. OSGi 애플리케이션 프로젝트, 동적 웹 프로젝트, OSGi 번들 프로젝트에 참조를 추가하고 CICS 자원에 대한 정의와 가져오기를 추가할 수 있습니다. 각 CICS 번들에 ID 및 버전을 포함하여 변경사항을 세부적으로 관리할 수 있습니다.
7. 옵션: 특정 트랜잭션에서 실행할 URI로부터의 인바운드 웹 요청을 맵핑하려는 경우 CICS 번들에 URIMAP 및 TRANSACTION 자원을 추가하십시오.이러한 자원을 정의하지 않으면 모든 작업이 제공된 트랜잭션(CJSA)에서 실행됩니다. 이러한 자원은 동적으로 설치되며 CICS에서 번들의 일부로 관리됩니다.

결과

개발 환경을 설정하고 OSGi 웹 애플리케이션을 작성했으며 전개 대상으로 패키지화했습니다.

다음에 수행할 작업

애플리케이션을 전개할 수 있는 준비가 되면 CICS 번들 프로젝트를 zFS로 내보내십시오. 참조된 프로젝트가 빌드되어 zFS로의 전송에 포함됩니다. 또는 애플리케이션을 EBA 파일로 내보내고 실행 Liberty JVM 서버의 dropins 디렉토리에 전개하여 Liberty 전개 모델을 따를 수 있습니다.

Liberty JVM 서버에서 실행할 Java 웹 애플리케이션 이주

Java 웹 애플리케이션이 네트워크를 통해 CICS에 액세스하는 Liberty 프로파일 인스턴스에서 실행되는 경우 Liberty JVM 서버에서 애플리케이션을 실행하여 성능을 최적화할 수 있습니다.

이 태스크 정보

CICS는 Liberty 프로파일에서 사용 가능한 특성의 서브세트를 지원합니다. 모든 코어 웹 컨테이너 API(예: JSP 및 servlet)가 여러 공통 Java EE API 및 기능과 함께 지원됩니다. 지원되는 특성 목록은 topics/liberty_features.dita의 내용을 참조하십시오.

애플리케이션이 보안을 사용하는 경우 계속해서 Liberty 보안 특성을 사용할 수 있지만 추가적인 조치가 없으면 CICS가 CJSA 트랜잭션에서 실행되고 CICS에서 일치하는 URIMAP을 사용할 수 없으며 자원 액세스가 CICS 기본 사용자 ID로 수행될 수 있습니다. Liberty에서 판별된 것과 동일한 사용자 ID로 CICS 태스크가 실행되도록 하여 CICS와 보안 솔루션을 잘 통합하려면 189 페이지의 『Liberty JVM 서버에서 애플리케이션을 실행하도록 사용자에게 권한 부여』의 내용을 참조하십시오.

애플리케이션이 RequestDispatcher.forward() 호출을 사용하는 경우에는 클라이언트가 호출하는 첫 번째 servlet에만 보안이 적용됩니다. 다른 servlet으로 전달되는 모든 요

청은 CICS에서 동일한 태스크 및 트랜잭션 ID로 실행되므로 추가적인 보안 검사가 적용되지 않습니다.

프로시저

1. JCICS API를 사용하여 CICS 서비스에 직접 액세스하도록 애플리케이션을 갱신하십시오. 애플리케이션이 CICS에서 또는 CICS로 데이터를 전달할 때 올바른 JCICS 인코딩을 사용해야 합니다. 인코딩에 대한 자세한 정보는 46 페이지의 『데이터 인코딩』의 내용을 참조하십시오.
2. 기본 인증에 CICS 보안을 사용하려면 인증 시 CICS 역할을 사용하도록 동적 웹 프로젝트의 `web.xml` 파일에서 보안 제한조건을 갱신하십시오.

```
<auth-constraint>
     <description>All authenticated users of my application</description>
     <role-name>cicsAllAuthenticated</role-name>
   </auth-constraint>
```

3. 애플리케이션을 WAR(동적 웹 프로젝트) 또는 EBA(OSGi 애플리케이션 프로젝트) 파일로 CICS 번들에 패키지화하십시오. CICS 번들은 애플리케이션의 전개 단위입니다. 번들의 모든 CICS 자원은 동적으로 함께 설치 및 관리됩니다. 함께 관리하려는 애플리케이션 구성요소의 CICS 번들 프로젝트를 작성하십시오.
4. CICS 번들 프로젝트를 zFS에 전개하고 Liberty JVM 서버에서 CICS 번들을 설치하십시오.

결과

애플리케이션이 JVM 서버에서 실행됩니다.

관련 태스크:

88 페이지의 『servlet 및 JSP 애플리케이션 개발』
CICS 애플리케이션에 최신 인터페이스를 제공하기 위해 Java servlet 및 JSP 기술을 사용하는 프리젠테이션 계층을 개발할 수 있습니다. Eclipse 기반 웹 개발 도구는 해당 애플리케이션을 작성하기 위한 개발 플랫폼을 제공하고 CICS Explorer SDK는 애플리케이션을 빌드, 패키지, 전개하여 CICS에서 실행하기 위한 지원을 제공합니다.

제 3 장 Java 지원 설정

CICS 영역에서 Java를 지원하기 위한 기본 설정 태스크를 수행하고 JVM 서버가 Java 애플리케이션을 실행하도록 구성합니다.

시작하기 전에

CICS에 필요한 Java 구성요소는 제품을 설치하는 동안 설정됩니다. 설치의 Java 컴포넌트 설치 확인의 정보를 사용하여 Java 구성요소가 올바르게 설치되었는지 확인해야 합니다.

이 태스크 정보

CICS는 z/OS UNIX의 파일을 사용하여 JVM을 시작합니다. CICS 영역이 올바른 zFS 디렉토리를 사용하도록 구성되고 해당 디렉토리가 올바른 권한을 갖는지 확인해야 합니다. CICS를 구성하고 zFS를 설정한 후 JVM 서버가 Java 애플리케이션을 실행하도록 구성할 수 있습니다.

프로시저

1. JVMPROFILEDIR 시스템 초기화 매개변수를 CICS 영역이 사용하는 JVM 프로파일을 저장하려는 z/OS UNIX의 적합한 디렉토리로 설정하십시오. 자세한 정보는 96 페이지의 『JVM 프로파일에 대한 위치 설정』의 내용을 참조하십시오.
2. CICS 영역에 Java 애플리케이션을 실행할 수 있는 충분한 메모리가 있는지 확인하십시오. 자세한 정보는 98 페이지의 『Java에 대한 메모리 한계 설정』의 내용을 참조하십시오.
3. CICS 영역에 z/OS UNIX에 저장된 자원(JVM을 작성하는 데 필요한 JVM 프로파일, 디렉토리, 파일)에 대한 액세스 권한을 부여하십시오. 자세한 정보는 98 페이지의 『CICS 영역에 z/OS UNIX 디렉토리 및 파일에 대한 액세스 권한 제공』의 내용을 참조하십시오.
4. JVM 서버를 설정하십시오. JVM 서버가 다른 워크로드를 실행하도록 구성할 수 있습니다. 자세한 정보는 101 페이지의 『JVM 서버 설정』의 내용을 참조하십시오.
5. 옵션: Java 보안 관리자로 하여금 Java 애플리케이션이 잠재적으로 안전하지 않은 조치를 수행하지 못하게 방지하도록 설정하십시오. 자세한 정보는 192 페이지의 『Java 보안 관리자 사용』의 내용을 참조하십시오.

결과

CICS 영역이 Java를 지원하도록 설정했으며 Java 애플리케이션을 실행할 JVM 서버를 작성했습니다.

다음에 수행할 작업

기존 Java 애플리케이션을 업그레이드하는 경우 업그레이드의 지침을 따르십시오. JVM 서버에서 Java 애플리케이션을 실행하려면 139 페이지의 제 4 장 『JVM 서버에 애플리케이션 전개』의 내용을 참조하십시오.

『JVM 프로파일에 대한 위치 설정』
CICS는 **JVMPROFILEDIR** 시스템 초기화 매개변수로 지정된 z/OS UNIX 디렉토리에서 JVM 프로파일을 로드합니다. **JVMPROFILEDIR** 매개변수 값을 새 위치로 변경하고 제공된 샘플 JVM 프로파일을 이 디렉토리에 복사해야 설치를 확인하는 데 사용할 수 있습니다.

98 페이지의 『Java에 대한 메모리 한계 설정』
Java 애플리케이션에는 다른 언어로 작성된 프로그램보다 많은 메모리가 필요합니다. CICS 및 Java에 Java 애플리케이션을 실행할 수 있는 충분한 저장영역과 메모리가 있는지 확인해야 합니다.

98 페이지의 『CICS 영역에 z/OS UNIX 디렉토리 및 파일에 대한 액세스 권한 제공』
CICS에는 z/OS UNIX의 디렉토리 및 파일에 대한 액세스 권한이 필요합니다. 설치 중에 각 CICS 영역에 z/OS UNIX 사용자 ID(UID)가 지정됩니다. 이 영역은 z/OS UNIX 그룹 ID(GID)가 지정된 RACF® 그룹에 연결됩니다. UID 및 GID를 사용하여 CICS 영역이 z/OS UNIX의 디렉토리와 파일에 액세스할 수 있는 권한을 부여할 수 있습니다.

101 페이지의 『JVM 서버 설정』
Java 애플리케이션, 웹 애플리케이션, Axis2 또는 CICS 보안 토큰 서비스를 JVM 서버에서 실행하려면 CICS 자원을 설정하고 JVM으로 선택항목을 전달하는 JVM 프로파일을 작성해야 합니다.

JVM 프로파일에 대한 위치 설정

CICS는 **JVMPROFILEDIR** 시스템 초기화 매개변수로 지정된 z/OS UNIX 디렉토리에서 JVM 프로파일을 로드합니다. **JVMPROFILEDIR** 매개변수 값을 새 위치로 변경하고 제공된 샘플 JVM 프로파일을 이 디렉토리에 복사해야 설치를 확인하는 데 사용할 수 있습니다.

시작하기 전에

USSHOME 시스템 초기화 매개변수는 z/OS UNIX에서 CICS 파일의 루트 디렉토리를 지정해야 합니다.

이 태스크 정보

CICS 제공 샘플 JVM 프로파일은 CICS 설치 프로세스에서 시스템에 맞게 사용자 정의되므로 설치를 검증하기 위해 즉시 사용할 수 있습니다. 사용자 Java 애플리케이션에 맞게 이러한 파일의 사본을 사용자 정의할 수 있습니다.

JVM 프로파일에서 사용하기에 적합한 설정은 특정 CICS 릴리스에서 다른 릴리스로 변경될 수 있으므로 간편한 문제점 판별을 위해서는 CICS 제공 샘플을 모든 프로파일의 기반으로 사용하십시오. 업그레이드 정보를 보고 JVM 프로파일에서 변경된 선택항목 또는 새 선택항목을 확인하십시오.

프로시저

1. JVMPROFILEDIR 시스템 초기화 매개변수를 CICS 영역이 사용하는 JVM 프로파일을 저장하려는 z/OS UNIX의 위치로 설정하십시오. 지정하는 값의 최대 *크기*는 240자입니다.

   JVMPROFILEDIR 시스템 초기화 매개변수에 제공되는 설정은 `/usr/lpp/cicsts/cicsts52/JVMProfiles`(샘플 JVM 프로파일의 설치 위치)입니다. 이 디렉토리는 사용자 정의된 JVM 프로파일을 저장하기에 안전한 위치가 아닙니다. 프로그램 유지보수가 적용되는 경우 샘플 JVM 프로파일을 겹쳐쓸 때 변경사항이 유실될 위험이 있기 때문입니다. 따라서 항상 **JVMPROFILEDIR**을 변경하여 JVM 프로파일을 저장할 수 있는 다른 z/OS UNIX 디렉토리를 지정해야 합니다. JVM 프로파일을 사용자 정의해야 하는 사용자에게 적절한 권한을 부여할 수 있는 디렉토리를 선택하십시오.
2. 제공된 샘플 JVM 프로파일을 설치 위치에서 z/OS UNIX 디렉토리에 복사하십시오.

   CICS를 설치하는 경우 샘플 JVM 프로파일이 zFS 디렉토리에 배치됩니다. 이 디렉토리는 DFHISTAR 설치 작업에서 **USSDIR** 매개변수로 지정됩니다. 기본 설치 디렉토리는 `/usr/lpp/cicsts/cicsts52/JVMProfiles`입니다.

결과

샘플 JVM 프로파일을 zFS 디렉토리에 복사하고 CICS에서 해당 디렉토리를 사용하도록 구성했습니다. 샘플 JVM 프로파일에는 JVM 서버를 설정하기 위해 즉시 사용할 수 있는 기본값이 포함됩니다.

다음에 수행할 작업

CICS와 Java에 Java 애플리케이션을 실행할 수 있는 메모리가 충분한지 확인하십시오 (98 페이지의 『Java에 대한 메모리 한계 설정』의 설명 참조). 또한 CICS 영역에 Java가 설치되고 Java 애플리케이션이 전개된 z/OS UNIX 디렉토리에 대한 액세스 권한이 있는지 확인하십시오. 자세한 정보는 98 페이지의 『CICS 영역에 z/OS UNIX 디렉토

리 및 파일에 대한 액세스 권한 제공』의 내용을 참조하십시오.

Java에 대한 메모리 한계 설정

Java 애플리케이션에는 다른 언어로 작성된 프로그램보다 많은 메모리가 필요합니다. CICS 및 Java에 Java 애플리케이션을 실행할 수 있는 충분한 저장영역과 메모리가 있는지 확인해야 합니다.

이 태스크 정보

Java는 16MB 행, 31비트 저장영역, 64비트 저장영역 미만의 저장영역을 사용합니다. JVM 힙에 필요한 저장영역은 CICS DSA가 아닌 MVS의 CICS 영역에서 가져옵니다.

프로시저

1. z/OS **MEMLIMIT** 매개변수가 적절한 값으로 설정되었는지 확인하십시오. 이 매개변수는 CICS 주소 공간이 사용할 수 있는 64비트 저장영역의 *크기*를 제한합니다. CICS는 64비트 버전 Java를 사용하므로 **MEMLIMIT**가 CICS 영역에서 64비트 저장영역의 이 용도와 다른 용도에 모두 충분한 값으로 설정되었는지 확인해야 합니다.

   다음 주제를 참조하십시오.

   - 171 페이지의 『JVM 서버에 대한 저장영역 요구사항 계산』
   - 성능 개선에서 MEMLIMIT 추정, 검사 및 설정

2. 시동 작업 스트림의 **REGION** 매개변수가 Java를 실행할 수 있을 정도로 충분한지 확인하십시오. 각 JVM마다 애플리케이션을 실행하기 위해 16MB 행 아래 일부 저장영역이 필요합니다(JIT(Just-In-Time) 컴파일 코드, CICS로 매개변수를 전달하기 위한 작업 저장영역 포함).

CICS 영역에 z/OS UNIX 디렉토리 및 파일에 대한 액세스 권한 제공

CICS에는 z/OS UNIX의 디렉토리 및 파일에 대한 액세스 권한이 필요합니다. 설치 중에 각 CICS 영역에 z/OS UNIX 사용자 ID(UID)가 지정됩니다. 이 영역은 z/OS UNIX 그룹 ID(GID)가 지정된 RACF 그룹에 연결됩니다. UID 및 GID를 사용하여 CICS 영역이 z/OS UNIX의 디렉토리와 파일에 액세스할 수 있는 권한을 부여할 수 있습니다.

시작하기 전에

본인이 z/OS UNIX의 수퍼유저 또는 디렉토리와 파일의 소유자인지 확인하십시오. 디렉토리와 파일의 소유자는 처음에 제품을 설치하는 시스템 프로그래머의 UID로 설정됩니다. 디렉토리와 파일의 소유자는 설치 중에 GID가 지정된 RACF 그룹에 연결되어야

합니다. 소유자는 RACF 그룹을 기본 그룹(DFLTGRP)으로 갖거나 보조 그룹 중 하나로 그룹에 연결될 수 있습니다.

이 태스크 정보

z/OS UNIX 시스템 서비스는 각 CICS 영역을 UNIX 사용자로 간주합니다. 다양한 방법으로 사용자에게 z/OS UNIX 디렉토리 및 파일에 대한 액세스 권한을 부여할 수 있습니다. 예를 들어, 디렉토리 또는 파일에 적절한 그룹 권한을 CICS 영역이 연결되는 RACF 그룹에 제공할 수 있습니다. 이 선택항목은 프로덕션 환경에 가장 적합하며 다음 단계에서 설명합니다.

프로시저

1. z/OS UNIX에서 CICS 영역에 액세스 권한이 필요한 디렉토리와 파일을 식별하십시오.

JVM 서버 등록 정보	**기본 디렉토리**	**권한**	**설명**
JAVA_HOME	`/usr/lpp/java/J7.0_64/bin`	읽기 및 실행	IBM 64-bit SDK for z/OS, Java Technology Edition 디렉토리
USSHOME	`/usr/lpp/cicsts/cicsts52`	읽기 및 실행	z/OS UNIX에서 CICS 파일의 설치 디렉토리입니다. 이 디렉토리의 파일에는 샘플 프로파일과 CICS 제공 JAR 파일이 포함됩니다.
WORK_DIR	*/u/CICS region userid*	읽기, 쓰기 및 실행	CICS 영역의 작업 디렉토리입니다. 이 디렉토리에는 JVM의 입력, 출력, 메시지가 포함됩니다.
JVMPROFILEDIR	*USSHOME*/JVMProfiles/	읽기 및 실행	CICS 영역에 대한 JVM 프로파일을 포함하는 디렉토리(**JVMPROFILEDIR** 시스템 초기화 매개변수에 지정됨)입니다.
WLP_USER_DIR	`WORK_DIR/APPLID/JVMSERVER/wlp/usr/`	읽기 및 실행	Liberty JVM 서버의 구성 파일을 포함하는 디렉토리를 지정합니다.
WLP_OUTPUT_DIR	`WLP_USER_DIR/servers/server_name`	읽기, 쓰기 및 실행	Liberty JVM 서버의 출력 디렉토리를 지정합니다.

참고: Liberty JVM 서버 자동 구성이 사용되는 경우 CICS가 server.xml에 쓸 수 있어야 하므로 WLP_USER_DIR은 추가 x 권한(읽기, 쓰기, 실행)이 필요합니다.

2. 권한을 표시하기 위해 디렉토리와 파일을 나열합니다. 시작하려는 디렉토리로 이동하고 다음 UNIX 명령을 실행하십시오.

   ```
   ls -la
   ```

   현재 디렉토리가 `CICSHT##`의 홈 디렉토리일 때 z/OS UNIX 시스템 서비스 쉘 환경에서 이 명령이 실행되면 다음 예와 같은 목록이 나타날 수 있습니다.

```
/u/cicsht##:>ls -la
total 256
drwxr-xr-x   2 CICSHT## CICSTS52     8192 Mar 15  2008 .
drwx------   4 CICSHT## CICSTS52     8192 Jul  4 16:14 ..
-rw-------   1 CICSHT## CICSTS52     2976 Dec  5  2010 Snap0001.trc
-rw-r--r--   1 CICSHT## CICSTS52     1626 Jul 16 11:15 dfhjvmerr
-rw-r--r--   1 CICSHT## CICSTS52        0 Mar 15  2010 dfhjvmin
-rw-r--r--   1 CICSHT## CICSTS52      458 Oct  9 14:28 dfhjvmout
/u/cicsht##:>
```

3. 그룹 권한을 사용하여 액세스 권한을 제공하는 경우, 각 디렉토리 및 파일에 대한 그룹 권한이 자원에 대해 CICS에 필요한 액세스 레벨을 제공하는지 확인하십시오. 권한은 문자 `r`, `w`, `x`, `-`로 세 개 세트에 표시됩니다. 이러한 문자는 읽기, 쓰기, 실행, 없음을 나타내며 명령행 왼쪽 열에 두 번째 문자부터 표시됩니다. 첫 번째 세트는 소유자 권한이고 두 번째 세트는 그룹 권한이며 세 번째 세트는 기타 권한입니다. 이전 예제에서는 소유자가 `dfhjvmerr`, `dfhjvmin`, `dfhjvmout`에 대해 읽기 및 쓰기 권한을 갖지만 그룹과 다른 모두는 읽기 권한만 갖습니다.
4. 자원에 대한 그룹 권한을 변경하려면 UNIX 명령 `chmod`를 사용하십시오. 다음 예제는 읽기, 쓰기, 실행을 수행할 이름 지정된 디렉토리와 그 서브디렉토리, 파일에 대한 그룹 권한을 설정합니다. `-R`은 모든 서브디렉토리와 파일에 반복적으로 권한을 적용합니다.

   ```
   chmod -R g=rwx directory
   ```

   다음 예제는 읽고 실행할 이름 지정된 파일에 대해 그룹 권한을 설정합니다.

   ```
   chmod g+rx filename
   ```

   다음 예제는 이름 지정된 두 개 파일에서 그룹에 대한 쓰기 권한을 끕니다.

   ```
   chmod g-w filename filename
   ```

   이들 예제 모두에서 `g`는 그룹 권한을 지정합니다. 다른 권한을 정정하려는 경우 `u`가 사용자(소유자) 권한을 지정하고 `o`가 다른 권한을 지정합니다.
5. 각 자원에 대한 그룹 권한을 CICS 영역이 z/OS UNIX에 액세스하기 위해 선택한 RACF 그룹에 지정하십시오. 각 디렉토리와 해당 서브디렉토리 및 디렉토리 내 파일에 대한 그룹 권한을 지정해야 합니다. 다음 UNIX 명령을 입력하십시오.

   ```
   chgrp -R GID directory
   ```

   *GID*는 RACF 그룹의 숫자 GID이고 *directory*는 CICS 영역 권한을 지정할 디렉토리의 전체 경로입니다. 예를 들어, `/usr/lpp/cicsts/cicsts52` 디렉토리에 대한 그룹 권한을 지정하려면 다음 명령을 사용하십시오.

   ```
   chgrp -R GID /usr/lpp/cicsts/cicsts52
   ```

   CICS 영역 사용자 ID는 RACF 그룹에 연결되므로 CICS 영역은 이들 모든 디렉토리와 파일에 적절한 권한을 갖습니다.

결과

CICS에 Java 애플리케이션을 실행하기 위해 z/OS UNIX의 디렉토리와 파일에 액세스할 수 있는 적절한 권한이 있음을 확인했습니다.

파일 이동 또는 새 파일 작성과 같이 설정하려는 CICS 기능을 변경하는 경우, 이 프로시저를 반복하여 CICS 영역에 새 파일 또는 이동한 파일에 액세스할 수 있는 권한이 있는지 확인해야 합니다.

다음에 수행할 작업

샘플 프로그램과 프로파일을 사용하여 Java 지원이 올바르게 설정되었는지 확인하십시오.

JVM 서버 설정

Java 애플리케이션, 웹 애플리케이션, Axis2 또는 CICS 보안 토큰 서비스를 JVM 서버에서 실행하려면 CICS 자원을 설정하고 JVM으로 선택항목을 전달하는 JVM 프로파일을 작성해야 합니다.

이 태스크 정보

JVM 서버는 다른 Java 애플리케이션에 대한 여러 동시 요청을 단일 JVM에서 처리할 수 있습니다. JVMSERVER 자원은 CICS의 JVM 서버를 나타냅니다. 이 자원은 JVM, Language Environment 인클레이브에 값을 제공하는 프로그램 및 스레드 한계에 대한 선택항목을 지정하는 JVM 프로파일을 정의합니다. JVM 서버는 다른 유형의 워크로드를 실행할 수 있으며 각 유형마다 JVM 프로파일을 제공합니다.

- OSGi 번들로 패키지화된 애플리케이션을 실행하려면 DFHOSGI.jvmprofile을 사용하여 JVM 서버를 구성하십시오. 이 프로파일에는 JVM 서버에서 OSGi 프레임워크를 실행하기 위한 선택항목이 포함됩니다.
- JSP 페이지 및 servlet을 포함하는 애플리케이션을 실행하려면 DFHWLP.jvmprofile을 사용하여 JVM 서버를 구성하십시오. 이 프로파일에는 Liberty 프로파일 기술을 기반으로 하는 웹 컨테이너를 실행하기 위한 선택항목이 포함됩니다. 또한 웹 컨테이너에는 OSGi 프레임워크가 포함되므로 OSGi 번들로 패키지화되는 애플리케이션을 실행할 수 있습니다.
- Axis2 SOAP 엔진을 사용하여 웹 서비스에 대한 SOAP 처리를 실행하려면 DFHJVMAX.jvmprofile을 사용하여 JVM 서버를 구성하십시오. 이 프로파일에는 JVM 서버에서 Axis2를 실행하기 위한 선택항목이 포함됩니다.
- CICS 보안 토큰 서비스(STS)를 실행하려면 DFHJVMST.jvmprofile을 사용하여 JVM 서버를 구성하십시오. 이 프로파일에는 STS를 실행하기 위한 선택항목이 포함됩니다.

프로파일 변경사항은 프로파일을 사용하는 모든 JVM 서버에 적용됩니다. 각 프로파일을 사용자 정의할 때 변경사항이 JVM 서버를 사용하는 모든 Java 애플리케이션에 적합한지 확인하십시오.

CICS 온라인 자원 정의로 JVM 서버 및 JVM 프로파일을 구성하거나 CICS Explorer를 사용하여 CICS 번들에서 JVMSERVER 자원과 JVM 프로파일을 정의하고 패키지화할 수 있습니다. 자세한 정보는 *CICS Explorer 사용자 안내서*의 번들 작업을 참조하십시오.

『OSGi 애플리케이션에 대한 JVM 서버 구성』
OSGi 번들에 패키지화되는 Java 애플리케이션을 전개하려는 경우 JVM 서버에서 OSGi 프레임워크를 실행하도록 구성합니다.

105 페이지의 『웹 애플리케이션에 대한 Liberty JVM 서버 구성』
Java servlet 및 JSP 페이지를 사용하는 Java 웹 애플리케이션을 전개하려는 경우 Liberty JVM 서버에서 웹 컨테이너를 실행하도록 구성합니다. 웹 컨테이너를 Liberty 프로파일 기술을 기반으로 합니다.

117 페이지의 『Axis2에 대한 JVM 서버 구성』
파이프라인에서 SOAP 요청을 처리하거나 Java 웹 서비스를 실행하려는 경우 JVM 서버에서 Axis2를 실행하도록 구성합니다.

120 페이지의 『CICS 보안 토큰 서비스에 대한 JVM 서버 구성』
SAML 토큰을 유효성 검증하고 처리하려는 경우 CICS 보안 토큰 서비스를 실행하도록 JVM 서버를 구성합니다.

121 페이지의 『JVM 프로파일 유효성 검사 및 등록 정보』
JVM 프로파일에는 시작 시 JVM으로 전달되는 선택항목 및 시스템 등록 정보 세트가 포함됩니다. 일부 JVM 프로파일 선택항목은 CICS 환경에 고유하므로 다른 환경에서는 JVM에 사용하지 않습니다. CICS는 JVM 서버를 시작할 때 JVM 프로파일이 올바르게 코딩되었는지 검사합니다.

결과

JVM 서버가 구성되고 Java 워크로드를 실행할 준비가 되었습니다.

다음에 수행할 작업

Java 환경에 맞게 보안을 구성하십시오. 애플리케이션 개발자에게 Java 애플리케이션을 전개하고 설치할 수 있는 적절한 액세스 권한을 부여하고 애플리케이션 사용자에게는 CICS에서 Java 프로그램과 트랜잭션을 실행할 수 있는 권한을 부여하십시오. 자세한 정보는 Java 애플리케이션에 대한 보안을 참조하십시오.

OSGi 애플리케이션에 대한 JVM 서버 구성

OSGi 번들에 패키지화되는 Java 애플리케이션을 전개하려는 경우 JVM 서버에서 OSGi 프레임워크를 실행하도록 구성합니다.

이 태스크 정보

JVM 서버에는 클래스 로딩을 자동으로 처리하는 OSGi 프레임워크가 포함되므로 JVM 프로파일에 표준 클래스 경로 선택항목을 추가할 수 없습니다. 제공되는 샘플 DFHOSGI.jvmprofile은 OSGi JVM 서버에 적합합니다. 이 태스크는 이 샘플 프로파일에서 OSGi 애플리케이션에 대한 JVM 서버를 정의하는 방법을 보여줍니다.

CICS 온라인 자원 정의를 사용하거나 CICS Explorer의 CICS 번들에서 JVM 서버를 정의할 수 있습니다.

프로시저

1. JVM 서버에 대한 JVMSERVER 자원을 작성하십시오.
   a. JVM 서버의 JVM 프로파일 이름을 지정하십시오. JVMSERVER의 JVMPROFILE 속성에 1 - 8자 이름을 지정하십시오. 이 이름은 JVM 서버의 구성 선택항목을 보유하는 파일인 JVM 프로파일의 접두부에 사용됩니다. 여기에 접미부 .jvmprofile을 지정할 필요가 없습니다.
   b. JVM 서버의 스레드 한계를 지정하십시오. JVMSERVER의 THREADLIMIT 속성에서 JVM 서버의 Language Environment 인클레이브에 허용되는 최대 스레드 수를 지정하십시오. 스레드 수는 JVM 서버에서 실행하려는 워크로드에 따라 다릅니다. 시작하려면 기본값을 승인하고 나중에 환경을 조정하면 됩니다. JVM 서버에서 최대 256개 스레드를 설정할 수 있습니다.
2. JVM 프로파일을 작성하여 JVM 서버의 구성 선택항목을 정의하십시오. 샘플 프로파일 DFHOSGI.jvmprofile을 기초로 사용할 수 있습니다. 이 프로파일에는 JVM 서버를 시작하는 데 적합한 선택항목의 서브세트가 포함되어 있습니다. JVM 프로파일에 대한 모든 선택항목 및 값은 121 페이지의 『JVM 프로파일 유효성 검사 및 등록 정보』에 설명되어 있습니다. 122 페이지의 『JVM 프로파일 코딩 규칙』에 있는 코딩 규칙(프로파일 이름에 대한 코딩 규칙 포함)을 따르십시오.
   a. JVM 프로파일의 위치를 설정하십시오. 시스템 초기화 매개변수 JVMPROFILEDIR에 지정한 디렉토리에 JVM 프로파일이 있어야 합니다. 자세한 정보는 96 페이지의 『JVM 프로파일에 대한 위치 설정』의 내용을 참조하십시오.
   b. 샘플 프로파일을 다음과 같이 변경하십시오.
      - JAVA_HOME을 설치된 IBM Java SDK의 위치로 설정하십시오.
      - WORK_DIR을 JVM 서버의 메시지, 추적 및 출력에 대한 대상 디렉토리 선택사항으로 설정하십시오.
      - JVM 서버의 메시지에 대한 시간 소인의 시간대를 지정하도록 TZ를 설정하십시오. 예를 들어, 영국의 경우 TZ=GMT0BST,M3.5.0,M10.4.0입니다.
   c. JVM 프로파일에 변경사항을 저장하십시오. JVM 프로파일은 z/OS UNIX 시스템 서비스 파일 시스템에 EBCDIC으로 저장되어야 합니다.

3. JVMSERVER 자원을 설치하고 사용 가능으로 설정하십시오.

결과

CICS가 Language Environment 인클레이브를 작성하며 JVM 프로파일의 선택항목을 JVM 서버로 전달합니다. JVM 서버가 시작되고 OSGi 프레임워크가 OSGi 미들웨어 번들을 분석합니다. JVM 서버 시동이 완료되면 JVMSERVER 자원이 ENABLED 상태로 설치됩니다.

오류가 발생하면(예를 들어, CICS가 JVM 프로파일을 찾거나 읽을 수 없는 경우) JVM 서버가 시작되지 않습니다. JVMSERVER 자원은 DISABLED 상태로 설치되며 CICS가 시스템 로그에 오류 메시지를 발행합니다. 도움말은 195 페이지의 제 8 장 『Java 애플리케이션 문제점 해결』의 내용을 참조하십시오.

다음에 수행할 작업

- 140 페이지의 『JVM 서버에 OSGi 번들 전개』에 설명된 대로 JVM 서버의 OSGi 프레임워크에 애플리케이션에 대한 OSGi 번들을 설치하십시오.
- DB2 또는 WebSphere MQ와 같은 고유 C 동적 링크 라이브러리(DLL) 파일을 포함하는 디렉토리를 지정하십시오. JVM 프로파일의 LIBPATH_SUFFIX 선택항목에 이러한 디렉토리를 지정합니다.
- OSGi 프레임워크에서 실행하려는 미들웨어 번들을 지정하십시오. 미들웨어 번들은 공유 서비스 구현(예를 들어, WebSphere MQ 및 DB2에 연결)을 위한 Java 클래스를 포함하는 OSGi 번들의 한 유형입니다. JVM 프로파일의 OSGI_BUNDLES 선택항목에 이러한 번들을 지정합니다.

  『JVM 프로파일 예제』
  OSGI 애플리케이션을 위한 JVM 프로파일 예제

JVM 프로파일 예제

OSGI 애플리케이션을 위한 JVM 프로파일 예제

다음 발췌본은 DB2 버전 11을 사용하는 OSGi 프레임워크 및 JDBC 4.0 OSGi 미들웨어 번들을 시작하도록 구성된 JVM 프로파일 예제를 보여줍니다.

```
#*********************************************************************
#
#                          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)"
#
#
```

웹 애플리케이션에 대한 Liberty JVM 서버 구성

Java servlet 및 JSP 페이지를 사용하는 Java 웹 애플리케이션을 전개하려는 경우 Liberty JVM 서버에서 웹 컨테이너를 실행하도록 구성합니다. 웹 컨테이너를 Liberty 프로파일 기술을 기반으로 합니다.

이 태스크 정보

Liberty 프로파일 기술은 JVM 서버에서 웹 컨테이너로 실행되도록 CICS와 함께 설치됩니다. Liberty JVM 서버는 Liberty 프로파일에서 사용 가능한 특성의 서브세트를 지원합니다.

Liberty JVM 서버를 구성하는 두 가지 방법이 있습니다.

- 자동 구성. CICS가 CICS 설치 디렉토리에 제공되는 템플리트에서 Liberty 프로파일의 구성 파일인 `server.xml`을 자동으로 작성하고 갱신합니다. 자동 구성은 Liberty 프로파일의 최소 구성 값 세트를 빠르게 시작하므로 개발 환경에서 유용할 수 있습니다. 자동 구성을 사용하려면 JVM 시스템 등록 정보인 **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure** 등록 정보를 `true`로 설정하십시오. CICS 번들에 JVM 서버를 정의하는 경우 이 선택항목을 설정하십시오.
- 수동 구성. 이것이 기본 설정입니다. 구성 파일 및 모든 값을 제공합니다. 수동 구성은 프로덕션 환경에 적합합니다.

CICS 온라인 자원 정의를 사용하거나 CICS 번들에서 JVM 서버를 정의할 수 있습니다.

프로시저

1. JVM 서버에 대한 JVMSERVER 자원을 작성하십시오.
   a. JVM 서버의 JVM 프로파일 이름을 지정하십시오. JVMSERVER의 JVMPROFILE 속성에 1 - 8자 이름을 지정하십시오. 이 이름은 JVM 서버의 구성 선택항목을 보유하는 파일인 JVM 프로파일의 접두부에 사용됩니다. 여기에 접미부 `.jvmprofile`을 지정할 필요가 없습니다.
   b. JVM 서버의 스레드 한계를 지정하십시오. JVMSERVER의 THREADLIMIT 속성에서 JVM 서버의 Language Environment 인클레이브에 허용되는 최대 스레드 수를 지정하십시오. 스레드 수는 JVM 서버에서 실행하려는 워크로드에 따라 다릅니다. 시작하려면 기본값을 승인하고 나중에 환경을 조정하면 됩니다. JVM 서버에서 최대 256개 스레드를 설정할 수 있습니다.
2. JVM 프로파일을 작성하여 JVM 서버의 구성 선택항목을 정의하십시오. 샘플 프로파일 `DFHWLP.jvmprofile`을 기초로 사용할 수 있습니다. 이 프로파일에는 JVM 서버를 시작하는 데 적합한 선택항목의 서브세트가 포함되어 있습니다. JVM 프로파일에 대한 모든 선택항목 및 값은 121 페이지의 『JVM 프로파일 유효성 검사 및 등록 정보』에 설명되어 있습니다. 122 페이지의 『JVM 프로파일 코딩 규칙』에 있는 코딩 규칙(프로파일 이름에 대한 코딩 규칙 포함)을 따르십시오.
   a. JVM 프로파일의 위치를 설정하십시오. 시스템 초기화 매개변수 **JVMPROFILEDIR**에 지정한 디렉토리에 JVM 프로파일이 있어야 합니다. 자세한 정보는 96 페이지의 『JVM 프로파일에 대한 위치 설정』의 내용을 참조하십시오.
   b. 샘플 프로파일을 다음과 같이 변경하십시오.
      - **JAVA_HOME**을 설치된 IBM Java SDK의 위치로 설정하십시오.
      - **WORK_DIR**을 JVM 서버의 메시지, 추적 및 출력에 대한 대상 디렉토리 선택사항으로 설정하십시오.
      - **WLP_INSTALL_DIR**을 `&USSHOME;/wlp`에 설정하십시오.
      - JVM 서버의 메시지에 대한 시간 소인의 시간대를 지정하도록 TZ를 설정하십시오. 예를 들어, 영국의 경우 TZ=GMT0BST,M3.5.0,M10.4.0입니다.
   c. JVM 프로파일에 변경사항을 저장하십시오. JVM 프로파일은 UNIX 시스템 서비스의 EBCDIC 파일 인코딩으로 저장되어야 하며 파일 접미부는 `.jvmprofile`이어야 합니다.
3. Liberty 프로파일 서버 구성을 작성하십시오. 자동 구성을 사용할 경우 JVM 서버 시작 시 자동 구성이 수행됩니다. Liberty JVM 서버 자동 구성을 사용 가능하게 설정하려면 JVM 프로파일에서 다음 JVM 시스템 등록 정보를 설정해야 합니다.
   a. **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true**를 설정하여 자동 구성을 사용 가능하게 설정하십시오.

b. **-Dcom.ibm.cics.jvmserver.wlp.server.http.port**를 사용하여 HTTP 서버 리스너 포트를 설정하십시오.

c. Liberty HTTP 서버가 **-Dcom.ibm.cics.jvmserver.wlp.server.host** 사용을 청취하는 호스트를 설정하십시오.

d. **-Dcom.ibm.cics.jvmserver.wlp.server.name**을 사용하여 Liberty 프로파일 서버 이름을 설정하십시오.

또는 Liberty 프로파일 디렉토리 구조 및 server.xml 구성 파일을 수동으로 구성할 수 있습니다. 이는 기본 설정이며 구성 파일을 주의 깊게 제어해야 하는 프로덕션 환경에 적절합니다. 자세한 정보는 109 페이지의『구성 디렉토리 구조 작성』 및 server.xml 수동 조정을 참조하십시오. server.xml 구성 파일을 CICS 번들의 JVM 프로파일에 포함할 수 없으므로 CICS 번들의 JVM 서버를 정의하는 경우 자동 구성을 사용해야 합니다.

4. JVMSERVER 자원을 설치하고 사용 가능으로 설정하십시오.

결과

CICS는 Language Environment 인클레이브를 작성하며 JVM 프로파일의 선택항목을 JVM 서버에 전달합니다. JVM 서버는 Liberty 서버를 시작 및 초기화하고 Liberty HTTP 리스너는 실행을 시작합니다. JVM 서버 시동이 완료되면 JVMSERVER 자원이 ENABLED 상태로 설치됩니다.

오류가 발생하면(예를 들어, CICS가 JVM 프로파일을 찾거나 읽을 수 없는 경우) JVM 서버가 초기화되지 않습니다. JVM 서버는 DISABLED 상태로 설치되며 CICS가 시스템 로그에 오류 메시지를 표시합니다. 자세한 정보는 Liberty JVM 서버 및 Java 웹 애플리케이션 문제점 해결을 참조하십시오. JVM 서버에서 Liberty 프로파일이 성공적으로 시작되었는지 확인하려면 zFS의 WLP_USER_DIR 출력 디렉토리에 있는 message.log 파일을 참조하십시오.

참고: JVM 서버를 사용하도록 설정하면 Liberty 프로파일 서버가 시작되고 JVM 서버를 사용 안함으로 설정하면 Liberty 프로파일 서버가 중지됩니다. JVM 서버에서 실행되는 Liberty 프로파일 서버를 시작하거나 중지하려면 Liberty 프로파일 bin/server 스크립트를 사용하지 마십시오.

다음에 수행할 작업

- WebSphere MQ와 같은 고유 C 동적 링크 라이브러리(DLL) 파일을 포함하는 디렉토리를 지정하십시오. IBM 또는 벤더가 제공하는 미들웨어와 도구에서 라이브러리 경로에 DLL 파일을 추가해야 할 수 있습니다.
- JVM 프로파일에서 WLP로 시작하는 선택항목을 다시 설정하여 Liberty JVM 서버에 대한 환경 변수를 덮어쓰십시오.

- CICS 영역에서 여러 JVM 서버를 실행하려는 경우 각 추가 Liberty JVM 서버의 JVM 프로파일에 WLP_ZOS_PLATFORM=FALSE를 추가하십시오. CICS 영역당 하나의 '보안 사용' Liberty JVM 서버만 허용됩니다. 이는 CICS가 인증 및 권한 부여를 위해 Liberty의 z/OS 플랫폼 확장을 사용하기 때문입니다. 하지만 Liberty의 이러한 z/OS 플랫폼 측면은 주소 공간 내의 단일 Liberty 서버로 제한됩니다. 기본적으로 Liberty JVM 서버가 z/OS 플랫폼 확장을 시작하지만 CICS 영역 내의 첫 번째 Liberty JVM 서버만 보안 설정된 상태로 성공적으로 사용 가능하게 됩니다.
- 보안 지원을 추가하십시오. 182 페이지의 『Liberty JVM 서버에 대한 보안 구성』의 내용을 참조하십시오.
- 웹 애플리케이션(WAR 파일 또는 WAB 및 EBA 파일)을 설치하십시오(142 페이지의 『Liberty JVM 서버에 CICS 번들의 웹 애플리케이션 전개』의 설명 참조).

  『JVM 프로파일 예제』
  Liberty 서버의 JVM 프로파일 예제입니다.

  109 페이지의 『구성 디렉토리 구조 작성』
  zFS에서 JVM 서버용으로 구성 디렉토리 구조를 작성합니다.

  110 페이지의 『server.xml 수동 조정』
  자동 구성을 사용하지 않는 경우, server.xml 파일을 다음과 같이 변경하여 환경에 맞게 조정하십시오.

  112 페이지의 『Liberty용 기본 CICS DB2 JDBC 유형 2 드라이버 DataSource 작성』
  자동 구성 등록 정보를 사용하여 기본 CICS DB2 JDBC 유형 2 드라이버 DataSource를 작성하십시오.

  113 페이지의 『수동으로 Liberty용 CICS DB2 JDBC 유형 2 드라이버 데이터 소스 구성』
  CICS의 Liberty 프로파일 사용자는 JDBC 유형 2 드라이버 데이터 소스 정의를 사용하여 Java 애플리케이션에서 DB2 데이터베이스에 액세스할 수 있습니다.

  115 페이지의 『수동으로 Liberty에 대한 DB2 JDBC 유형 4 드라이버 DataSource 구성』
  CICS의 Liberty 프로파일 사용자는 Liberty 제공 DataSource 정의를 통해 Java 애플리케이션의 JDBC 유형 4를 사용하여 DB2 데이터베이스에 액세스할 수 있습니다.

관련 개념:

보안에서 Liberty JVM 서버에 대한 보안 구성

JVM 프로파일 예제

Liberty 서버의 JVM 프로파일 예제입니다.

다음 발췌본은 필수 구성 파일과 디렉토리 구조를 자동으로 작성하도록 구성된 JVM 프로파일 예제를 보여줍니다. 여기서는 DB2 버전 11을 사용합니다.

```
#*********************************************************************
# 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-8859-1WLP_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
```

구성 디렉토리 구조 작성

zFS에서 JVM 서버용으로 구성 디렉토리 구조를 작성합니다.

프로시저

1. JVM 서버에 대한 zFS의 구성 디렉토리 구조를 작성하십시오. JVM 서버는 구성 파일이 작업 디렉토리의 *WLP_USER_DIR*/servers/*server_name* 디렉토리에 있는 것으로 가정합니다. 여기서 *WLP_USER_DIR*은 WLP_USER_DIR 선택항목의 값이고 *server_name*은 **-Dcom.ibm.cics.jvmserver.wlp.server.name** 등록 정보의 값입니다.
2. *server_name* 디렉토리에서 Liberty 프로파일 구성을 작성하십시오. 최소한 server.xml 파일을 작성해야 합니다. 이 파일은 Liberty 프로파일의 설치 디렉토리에서 wlp/templates/config/cics/server.xml로 제공되는 템플리트를 기반으로 할 수 있습니다. 이 파일은 ISO-8859-1 또는 UTF-8의 파일 인코딩으로 저장해야 합니다.
3. 설치에 따라 server.xml 파일을 편집하십시오. 호스트 이름과 포트 번호로 <httpEndpoint>를 갱신하십시오. JVM 서버에서 server.xml 구성에 대한 정보는 110 페이지의 『server.xml 수동 조정』의 내용을 참조하십시오. 보안을 사용하려면 Liberty JVM 서버에 대한 보안 구성을 참조하십시오.

server.xml 수동 조정

자동 구성을 사용하지 않는 경우, server.xml 파일을 다음과 같이 변경하여 환경에 맞게 조정하십시오.

HTTP 엔드 포인트 구성

애플리케이션에 웹으로 액세스하려면 필요한 호스트 이름과 포트 번호로 httpEndpoint 속성을 갱신하십시오. 예를 들어, 다음과 같습니다.

```
<httpEndpoint host="winmvs2c.example.com" httpPort="28216" httpsPort="28217"
 id="defaultHttpEndpoint"/>
```

다른 곳(예를 들어, CICS의 TCPIPSERVICE)에서 사용하지 않는 포트 번호를 사용하십시오.

HTTPS는 SSL이 구성된 경우에만 사용할 수 있습니다(182 페이지의 『Liberty JVM 서버에 대한 보안 구성』 참조).

자세한 정보는 WAS(WebSphere Application Server) Liberty 프로파일 문서를 참조하십시오.

특성 추가

특성의 <featureManager> 목록에 다음 특성을 추가합니다.

- CICS 특성 cicsts:core-1.0은 CICS 시스템 OSGi 번들을 Liberty 프레임워크에 설치합니다. 이 특성은 JVM 서버를 시작하는 데 필요합니다.
- CICS 보안 특성 cicsts:security-1.0은 CICS Liberty 보안에 필요한 CICS 시스템 OSGi 번들을 Liberty 프레임워크에 설치합니다. 이 특성은 CICS 외부 보안을 사용하고(SIT의 **SEC**=YES) Liberty 서버에서 보안을 사용하려는 경우에 필요합니다.
- jsp-2.2 특성은 servlet 및 JSP(JavaServer Pages) 애플리케이션에 대한 지원을 제공합니다. 이 특성은 동적 웹 프로젝트(WAR 파일)와, CICS 번들로 설치되는 OSGi Bundle Projects with Web Support를 포함하는 OSGi 애플리케이션 프로젝트에 필요합니다.
- wab-1.0 특성을 통해 엔터프라이즈 번들(EBA) 내부에 있는 웹 아카이브 번들(WAB)을 지원할 수 있습니다. 이 특성은 CICS 번들로 설치되는 OSGi Bundle Projects with Web Support를 포함하는 OSGi 애플리케이션 프로젝트에 필요합니다.
- cicsts:jdbc-1.0 특성을 사용하면 애플리케이션이 JDBC DriverManager 또는 DataSource 인터페이스를 통해 DB2에 액세스할 수 있습니다.

예제:

```
<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>
```

CICS 번들 전개 애플리케이션

CICS 번들을 사용하는 Liberty 애플리케이션을 전개하려면 server.xml 파일에 다음 항목이 포함되어야 합니다.

```
<include location="${server.output.dir}/installedApps.xml"/>
```

포함된 파일은 CICS 번들 전개 애플리케이션을 정의하는 데 사용됩니다.

번들 저장소

공통 OSGi 번들을 디렉토리에 배치하고 해당 디렉토리를 bundleRepository 요소에 표시하여 공유합니다. 예를 들어, 다음과 같습니다.

```
<bundleRepository>
    <fileset dir="directory_path" include="*.jar"/>
</bundleRepository>
```

글로벌 라이브러리

공통 JAR 파일을 디렉토리에 배치하고 글로벌 라이브러리 정의에서 해당 디렉토리를 참조하여 웹 애플리케이션 간에 공유합니다.

```
<library id="global">
<fileset dir="directory_path" include="*.jar"/>
</library>
```

글로벌 라이브러리는 EBA의 OSGi 애플리케이션에서 사용할 수 없으며 번들 저장소를 사용해야 합니다.

Liberty 서버 애플리케이션 및 구성 갱신 모니터링

Liberty JVM 서버는 server.xml 파일에서 갱신을 스캔합니다. 기본적으로 500밀리초 간격으로 스캔합니다. 이 값을 다양화하려면 다음과 같은 항목을 추가하십시오.

```
<config monitorInterval="5s" updateTrigger="polled"/>
```

또한 dropins 디렉토리를 스캔하여 애플리케이션 추가, 갱신 또는 제거를 발견합니다. 웹 애플리케이션을 CICS 번들에 설치하는 경우에는 다음과 같이 dropins 디렉토리를 사용 안함으로 설정하십시오.

```
<applicationMonitor updateTrigger="disabled" dropins="dropins"
dropinsEnabled="false" pollingRate="5s"/>.
```

JTA 트랜잭션 로그

JTA(Java Transaction API)를 사용할 때 Liberty 트랜잭션 관리자가 zFS 파일링 시스템에 복구 가능한 로그 파일을 저장합니다. 트랜잭션 로그의 기본 위치는 `${WLP_USER_DIR}/tranlog/`입니다. 이 위치는 트랜잭션 요소를 server.xml (예:

`<transaction transactionLogDirectory="/u/cics/CICSPRD/DFHWLP/tranlog/"/>`)에 추가하여 덮어쓸 수 있습니다.

관련 태스크:

109 페이지의『구성 디렉토리 구조 작성』
zFS에서 JVM 서버용으로 구성 디렉토리 구조를 작성합니다.

Liberty용 기본 CICS DB2 JDBC 유형 2 드라이버 DataSource 작성

자동 구성 등록 정보를 사용하여 기본 CICS DB2 JDBC 유형 2 드라이버 DataSource를 작성하십시오.

시작하기 전에

DB2에 연결하려면 CICS 영역을 구성해야 합니다. 자세한 정보는 구성에서 CICS DB2 연결 정의의 내용을 참조하십시오.

이 태스크 정보

Liberty server.xml에서 JVM 프로파일 자동 구성 등록 정보를 사용하여 기본 CICS DB2 JDBC 유형 2 드라이버 DataSource 자원을 작성할 수 있습니다. 기본 구성은 CICS Explorer SDK CICS JDBC(Java DataBase Connectivity) Servlet 및 JSP 예제에서 사용될 수 있습니다. JNDI 이름은 `jdbc/defaultCICSDataSource`입니다.

프로시저

1. JVM 프로파일에서 **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true**를 설정하여 자동 구성을 사용 가능하게 설정하십시오.
2. JVM 프로파일에서 **-Dcom.ibm.cics.jvmserver.wlp.jdbc.driver.location**을 설정하여 기본 CICS DB2 JDBC 유형 2 드라이버 DataSource의 자동 구성을 사용 가능하게 설정하십시오. DB2 JDBC 라이브러리의 위치 (예: **-Dcom.ibm.cics.jvmserver.wlp.jdbc.driver.location=/usr/lpp/db2v11/jdbc**)
3. JVMSERVER 자원을 설치하고 사용 가능으로 설정하십시오.

결과

기본 CICS DB2 JDBC 유형 2 드라이버 DataSource가 Liberty 서버 구성 파일 `server.xml`에 추가됩니다.

```
<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>
```

관련 개념:

보안에서 Liberty JVM 서버에 대한 보안 구성

관련 태스크:

115 페이지의『수동으로 Liberty에 대한 DB2 JDBC 유형 4 드라이버 DataSource 구성』

CICS의 Liberty 프로파일 사용자는 Liberty 제공 DataSource 정의를 통해 Java 애플리케이션의 JDBC 유형 4를 사용하여 DB2 데이터베이스에 액세스할 수 있습니다.

수동으로 Liberty용 CICS DB2 JDBC 유형 2 드라이버 데이터 소스 구성

CICS의 Liberty 프로파일 사용자는 JDBC 유형 2 드라이버 데이터 소스 정의를 사용하여 Java 애플리케이션에서 DB2 데이터베이스에 액세스할 수 있습니다.

시작하기 전에

DB2에 연결하려면 CICS 영역을 구성해야 합니다. 자세한 정보는 구성에서 CICS DB2 연결 정의의 내용을 참조하십시오.

이 태스크 정보

이 태스크는 server.xml 구성 파일에 필요한 요소를 정의하여 JDBC 유형 2 드라이버 연결을 사용 가능하게 설정하는 방법에 대해 설명합니다. 이는 로컬 DB2 데이터베이스와의 연결을 설정할 수 있도록 합니다.

참고:

-Dcom.ibm.cics.jvmserver.wlp.autoconfigure가 true로 설정되어 있고 **-Dcom.ibm.cics.jvmserver.wlp.jdbc.driver.location**가 JVM 프로파일에 설정되어 있는 경우 JVM 서버가 사용 가능하게 설정되면 server.xml 파일의 기존 예제 JDBC 구성이 기본 구성으로 바뀌고 모든 사용자 갱신이 손실됩니다.

프로시저

1. **cicsts:jdbc-1.0** 특성을 featureManager 요소에 추가하십시오. 이렇게 하면 나중에 server.xml 파일에서 사용되는 **cicsts_jdbcDriver** 및 **cicsts_dataSource** 요소를 사용할 수 있습니다.

   ```
   <featureManager>
       <feature>cicsts:jdbc-1.0</feature>
   </featureManager>
   ```

2. **cicsts_jdbcDriver** 요소를 추가하십시오. 이는 **java.sql.DriverManager**를 사용하거나 **javax.sql.DataSource**를 통해 JDBC 유형 2 드라이버 데이터 소스에 대한 액세스 지원을 사용 가능하게 합니다. **cicsts_jdbcDriver** 요소는 JDBC 드라이버 구성요소(DB2 JDBC jar 및 고유 dll 파일)가 로드될 라이브러리를 지정하는 라이브러리 정의를 나타내야 합니다. 일반적인 정의는 다음과 같습니다.

```
  <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>
```

   참고: 하나의 **cicsts_jdbcDriver** 요소만 필요합니다. 둘 이상이 지정된 경우에는 server.xml 파일의 마지막 요소만 사용되고 다른 요소는 무시됩니다.

3. **java.sql.DriverManager** 지원만 필요한 경우에는 선행 단계로 충분합니다. 데이터 소스 정의를 통해 DB2 연결에 액세스하려면 액세스가 정의될 각 데이터 소스에 **cicsts_dataSource** 요소가 필요합니다. **cicsts_dataSource**는 jndiName 속성을 지정합니다. 이는 해당 데이터 소스에 대한 연결을 설정할 때 애플리케이션에서 사용되는 참조인 JNDI 이름을 정의합니다. 정의는 다음과 같습니다.

```
<cicsts_dataSource id="defaultCICSDataSource" jndiName="jdbc/defaultCICSDataSource"/>
```

   참고: 사용된 DataSource 클래스는 **javax.sql.DataSource**를 구현하는 **com.ibm.db2.jcc.DB2SimpleDataSource**입니다.

4. 옵션: **properties.db2.jcc** 요소를 사용하여 **cicsts_dataSource**에 대한 등록 정보를 설정할 수 있습니다. 아래 예제는 이를 수행하는 방법을 보여줍니다.

```
<cicsts_dataSource id="defaultCICSDataSource" jndiName="jdbc/defaultCICSDataSource">
      <properties.db2.jcc currentSchema="DB2USER" fullyMaterializeLobData="true" />
</cicsts_dataSource>
```

   properties.db2.jcc 요소에 허용되는 일부 등록 정보는 JDBC 유형 2 드라이버 DataSource에 적합하지 않습니다. 부적절한 등록 정보는 무시됩니다.

   참고: 인식되지 않은 등록 정보는 무시됩니다. 다음과 같은 등록 정보는 CICS JDBC 유형 2 드라이버 지원과 관련되지 않습니다. 지정된 등록 정보도 무시되며 "driverType","serverName","portNumber","user","password","databaseName"이라는 경고 메시지가 표시됩니다.

결과

Liberty 서버가 시작될 때 JDBC 유형 2 드라이버 연결을 통해 DB2 데이터베이스에 액세스할 수 있도록 구성됩니다.

참고: CICS 데이터 소스 및 해당 구성요소의 동적 갱신은 지원하지 않습니다. Liberty 서버가 실행 중인 동안 구성을 갱신하면 DB2 애플리케이션 실패를 야기할 수 있습니다. 임의의 변경사항을 활성화하려면 서버를 다시 실행해야 합니다.

관련 개념:

보안에서 Liberty JVM 서버에 대한 보안 구성

관련 태스크:

『수동으로 Liberty에 대한 DB2 JDBC 유형 4 드라이버 DataSource 구성』
CICS의 Liberty 프로파일 사용자는 Liberty 제공 DataSource 정의를 통해 Java 애플리케이션의 JDBC 유형 4를 사용하여 DB2 데이터베이스에 액세스할 수 있습니다.

수동으로 Liberty에 대한 DB2 JDBC 유형 4 드라이버 DataSource 구성

CICS의 Liberty 프로파일 사용자는 Liberty 제공 DataSource 정의를 통해 Java 애플리케이션의 JDBC 유형 4를 사용하여 DB2 데이터베이스에 액세스할 수 있습니다.

시작하기 전에

Liberty DB2 JDBC 유형 2 DataSource는 CICS DB2 연결 자원을 사용하지 않지만 DB2 SDSNLOAD 및 SDSNLOD2 라이브러리를 CICS STEPLIB 병합에 추가해야 합니다.

이 태스크 정보

이 태스크는 `server.xml` 구성 파일에 필요한 요소를 수동으로 정의하여 JDBC 유형 4 드라이버 연결을 사용 가능하게 설정하는 방법에 대해 설명합니다. 이는 원격 DB2 데이터베이스와의 연결을 설정할 수 있도록 합니다. JDBC 유형 4 드라이버를 사용한 DB2 데이터베이스로의 갱신에서는 CICS DB2 연결 자원을 사용하지 않으므로 이러한 갱신이 JTA 사용자 트랜잭션 내에서 이루어지지 않는 한 CICS 작업 단위에 포함되지 않습니다. 자세한 정보는 67 페이지의 『JTA(Java Transaction API)』의 내용을 참조하십시오.

프로시저

1. featureManager 요소에 **jdbc-4.0** 특성을 추가하십시오. 이렇게 하면 나중에 `server.xml` 파일에서 사용되는 **dataSource** 및 **jdbcDriver** 요소를 사용할 수 있습니다.

   ```
   <featureManager>
       <feature>jdbc-4.0</feature>
   </featureManager>
   ```

2. **dataSource** 및 **jdbcDriver** 요소를 추가하십시오. **dataSource** 요소는 JDBC 드라이버 구성요소(DB2 JDBC jar 및 고유 DLL 파일)가 로드될 라이브러리를 지정하는 라이브러리 정의를 나타내야 합니다. 일반적인 정의는 다음과 같습니다.

   ```
   <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" />
    <fileset dir="/usr/lpp/db2v11/jdbc/lib" />
</library>
```

dataSource는 **jndiName** 속성을 지정합니다. 이는 해당 데이터 소스에 대한 연결을 설정할 때 애플리케이션 프로그램에서 사용되는 참조인 JNDI 이름을 정의합니다. 필요한 등록 정보는 다음과 같이 **properties.db2.jcc** 요소에 설정됩니다.

driverType

설명: 데이터베이스 드라이버 유형으로, 순수한 Java 드라이버를 사용하려면 4로 설정해야 합니다.

기본값: 4

필수: false

데이터 유형: int

serverName

설명: 데이터베이스가 실행하고 있는 서버의 호스트 이름입니다. 이는 DB2 DISPLAY DDF 명령의 SQL DOMAIN 값입니다.

기본값: localhost

필수: false

데이터 유형: string

portNumber

설명: 데이터베이스 연결을 얻는 데 사용할 포트입니다. 이는 DB2 DISPLAY DDF 명령의 TCPPORT 값입니다.

기본값: 50000

필수: false

데이터 유형: int

databaseName

설명: DataSource의 이름을 지정합니다. 이는 DB2 DISPLAY DDF 명령의 LOCATION 값입니다.

필수: true

데이터 유형: string

user 설명: 데이터베이스에 연결하는 데 사용되는 사용자 ID입니다.

필수: true

데이터 유형: string

password

설명: 데이터베이스에 연결하는 데 사용되는 사용자 ID의 암호입니다. 이 값은 일반 텍스트 또는 인코딩된 양식으로 저장할 수 있습니다. 암호는 암호화하는 것이 좋습니다. 이를 위해서는 securityUtility 도구를 인코드 선택 항목과 함께 사용하십시오. 자세한 정보는 Liberty 프로파일: securityUtility 명령의 내용을 참조하십시오.

필수: true

데이터 유형: string

결과

Liberty 서버가 시작될 때 JDBC 유형 4 드라이버 연결을 통해 DB2 데이터베이스에 액세스할 수 있도록 구성됩니다. 자세한 정보는 Liberty 프로파일: server.xml 파일의 구성 요소의 내용을 참조하십시오.

관련 개념:

보안에서 Liberty JVM 서버에 대한 보안 구성

관련 태스크:

112 페이지의『Liberty용 기본 CICS DB2 JDBC 유형 2 드라이버 DataSource 작성』

자동 구성 등록 정보를 사용하여 기본 CICS DB2 JDBC 유형 2 드라이버 DataSource를 작성하십시오.

113 페이지의『수동으로 Liberty용 CICS DB2 JDBC 유형 2 드라이버 데이터 소스 구성』

CICS의 Liberty 프로파일 사용자는 JDBC 유형 2 드라이버 데이터 소스 정의를 사용하여 Java 애플리케이션에서 DB2 데이터베이스에 액세스할 수 있습니다.

Axis2에 대한 JVM 서버 구성

파이프라인에서 SOAP 요청을 처리하거나 Java 웹 서비스를 실행하려는 경우 JVM 서버에서 Axis2를 실행하도록 구성합니다.

이 태스크 정보

Axis2는 제공자 및 요청자 파이프라인에서 웹 서비스 요청을 처리할 수 있는 Java SOAP 엔진입니다. JVM 서버에서 Axis2를 실행하도록 구성하면 CICS가 필수 JAR 파일을 클래스 경로에 자동으로 추가합니다.

CICS 온라인 자원 정의를 사용하거나 CICS 번들에서 JVM 서버를 정의할 수 있습니다.

프로시저

1. JVM 서버에 대한 JVMSERVER 자원을 작성하십시오.
   a. JVM 서버의 JVM 프로파일 이름을 지정하십시오. JVMSERVER의 JVMPROFILE 속성에 1 - 8자 이름을 지정하십시오. 이 이름은 JVM 서버의 구성 선택항목을 보유하는 파일인 JVM 프로파일의 접두부에 사용됩니다. 여기에 접미부 .jvmprofile을 지정할 필요가 없습니다.
   b. JVM 서버의 스레드 한계를 지정하십시오. JVMSERVER의 THREADLIMIT 속성에서 JVM 서버의 Language Environment 인클레이브에 허용되는 최대 스레드 수를 지정하십시오. 필요한 스레드 수는 JVM 서버에서 실행하려는 워크로드에 따라 다릅니다. 시작하려면 기본값을 승인한 다음 환경을 조정하면 됩니다. JVM 서버에서 최대 256개 스레드를 설정할 수 있습니다.
2. JVM 프로파일을 작성하여 JVM 서버의 구성 선택항목을 정의하십시오. 샘플 프로파일 DFHJVMAX.jvmprofile을 기초로 사용할 수 있습니다. 이 프로파일에는 JVM 서버를 시작하는 데 적합한 선택항목의 서브세트가 포함되어 있습니다. JVM 프로파일에 대한 모든 선택항목 및 값은 121 페이지의 『JVM 프로파일 유효성 검사 및 등록 정보』에 설명되어 있습니다. 122 페이지의 『JVM 프로파일 코딩 규칙』에 있는 코딩 규칙(프로파일 이름에 대한 코딩 규칙 포함)을 따르십시오.
   a. JVM 프로파일의 위치를 설정하십시오. 시스템 초기화 매개변수 JVMPROFILEDIR에 지정한 디렉토리에 JVM 프로파일이 있어야 합니다. 자세한 정보는 96 페이지의 『JVM 프로파일에 대한 위치 설정』의 내용을 참조하십시오.
   b. 샘플 프로파일을 다음과 같이 변경하십시오.
      - JAVA_HOME을 설치된 IBM Java SDK의 위치로 설정하십시오.
      - Axis2를 실행하도록 JAVA_PIPELINE을 설정하십시오.
      - Java로 작성된 SOAP 핸들러와 Axis2 애플리케이션에 대한 클래스를 지정하도록 CLASSPATH_SUFFIX를 설정하십시오.
      - WORK_DIR을 JVM 서버의 메시지, 추적 및 출력에 대한 대상 디렉토리 선택사항으로 설정하십시오.
      - JVM 서버의 메시지에 대한 시간 소인의 시간대를 지정하도록 TZ를 설정하십시오. 영국의 경우 예제는 TZ=GMT0BST,M3.5.0,M10.4.0입니다.
   c. JVM 프로파일에 변경사항을 저장하십시오. JVM 프로파일은 z/OS UNIX 시스템 서비스 파일 시스템에 EBCDIC으로 저장되어야 합니다.
3. JVMSERVER 자원을 설치하고 사용 가능으로 설정하십시오.

결과

CICS가 Language Environment 인클레이브를 작성하며 JVM 프로파일의 선택항목을 JVM 서버로 전달합니다. JVM 서버가 시작되고 Axis2 JAR 파일을 로드합니다. JVM 서버 시동이 완료되면 JVMSERVER 자원이 ENABLED 상태로 설치됩니다.

오류가 발생하면(예를 들어, CICS가 JVM 프로파일을 찾거나 읽을 수 없는 경우) JVM 서버가 시작되지 않습니다. JVMSERVER 자원은 DISABLED 상태로 설치되며 CICS가 시스템 로그에 오류 메시지를 실행합니다. 도움말은 195 페이지의 제 8 장 『Java 애플리케이션 문제점 해결』의 내용을 참조하십시오.

다음에 수행할 작업

- DB2 또는 WebSphere MQ와 같은 고유 C 동적 링크 라이브러리(DLL) 파일을 포함하는 디렉토리를 지정하십시오. JVM 프로파일의 LIBPATH_SUFFIX 선택항목에 이러한 디렉토리를 지정합니다.
- CICS가 JVM 서버에서 웹 서비스 요청을 실행하도록 구성하십시오(애플리케이션 개발에 웹 서비스와 Java 사용의 설명 참조).

  『JVM 프로파일 예제』

  Axis2를 시작하도록 구성된 JVM 프로파일 예제입니다.

JVM 프로파일 예제

Axis2를 시작하도록 구성된 JVM 프로파일 예제입니다.

다음 발췌본은 Axis2를 시작하도록 구성된 JVM 프로파일 예제를 보여줍니다.

```
#*********************************************************************
#
#                          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 보안 토큰 서비스에 대한 JVM 서버 구성

SAML 토큰을 유효성 검증하고 처리하려는 경우 CICS 보안 토큰 서비스를 실행하도록 JVM 서버를 구성합니다.

이 태스크 정보

제공된 샘플 DFHJVMST.jvmprofile은 CICS 보안 토큰 서비스를 실행하는 JVM 서버에 적합합니다.

CICS 온라인 자원 정의를 사용하거나 CICS 번들에서 JVM 서버를 정의할 수 있습니다. CICS Explorer를 사용하여 CICS 번들에서 자원을 작성 및 편집하는 작업에 대한 자세한 도움말은 *CICS Explorer 사용자 안내서*의 번들 작업 섹션을 참조하십시오.

프로시저

JVM 서버에 대한 JVMSERVER 자원을 작성하십시오.

1. JVM 서버의 JVM 프로파일 이름을 지정하십시오. JVMPROFILE 속성에 1 - 8자 이름을 지정하십시오. 이 이름은 JVM 서버의 구성 선택항목을 보유하는 파일인 JVM 프로파일의 접두부에 사용됩니다. 접미부 .jvmprofile을 지정할 필요가 없습니다.
2. JVM 서버의 스레드 한계를 지정하십시오. 스레드 수는 JVM 서버에서 실행하려는 워크로드에 따라 다릅니다. 시작하려면 기본값을 승인한 후 나중에 환경을 조정하면 됩니다. JVM 서버에서 최대 256개 스레드를 설정할 수 있습니다.
3. JVM 프로파일을 작성하여 JVM 서버의 구성 선택항목을 정의하십시오. 시스템 초기화 매개변수 JVMPROFILEDIR에 지정한 디렉토리에 JVM 프로파일이 있어야 합니다. 샘플 프로파일 DFHJVMST.jvmprofile을 기초로 사용할 수 있습니다. 이 프로파일에는 JVM 서버를 시작하는 데 적합한 선택항목의 서브세트가 포함되어 있습니다. DFHJVMST.jvmprofile을 설치 디렉토리에서 JVMPROFILEDIR에 지정한 디렉토리로 복사하거나 CICS Explorer에서 선택한 후 대상 디렉토리에 저장할 수 있습니다.

JVM 프로파일에 대한 모든 선택항목 및 값은 『JVM 프로파일 유효성 검사 및 등록 정보』에 설명되어 있습니다. 122 페이지의 『JVM 프로파일 코딩 규칙』의 코딩 규칙을 따르십시오.

샘플 프로파일을 다음과 같이 변경하십시오.

- JAVA_HOME을 설치된 IBM Java SDK의 위치로 설정하십시오.
- WORK_DIR을 JVM 서버의 메시지, 추적 및 출력에 대한 대상 디렉토리 선택사항으로 설정하십시오.
- SECURITY_TOKEN_SERVICE를 YES로 설정하십시오.
- JVM 서버의 메시지에 대한 시간 소인의 시간대를 지정하도록 TZ를 설정하십시오. 예를 들어, 영국의 경우 TZ=GMT0BST,M3.5.0,M10.4.0입니다.

4. JVM 프로파일에 변경사항을 저장하십시오. JVM 프로파일은 USS 파일 시스템에 EBCDIC으로 저장되어야 합니다.

결과

JVMSERVER 자원을 설치하고 사용 가능으로 설정하면 CICS가 Language Environment 인클레이브를 작성하며 JVM 프로파일의 선택항목을 JVM 서버로 전달합니다. JVM이 시동되고 OSGi 프레임워크가 OSGi 미들웨어 번들을 해결합니다. JVM 서버 시동이 완료되면 JVMSERVER 자원이 ENABLED 상태로 설치됩니다.

오류가 발생하면(예를 들어, CICS가 JVM 프로파일을 찾거나 읽을 수 없는 경우) JVM 서버가 초기화되지 않습니다. JVMSERVER 자원은 DISABLED 상태로 설치되며 CICS가 오류 메시지를 발행합니다. 195 페이지의 제 8 장 『Java 애플리케이션 문제점 해결』의 내용을 참조하십시오.

다음에 수행할 작업

다음과 같이 JVM 서버를 추가로 사용자 정의할 수 있습니다.

- DB2 또는 WebSphere MQ와 같은 고유 C 동적 링크 라이브러리(DLL) 파일을 포함하는 디렉토리를 지정하십시오. LIBPATH_SUFFIX 선택항목에 이러한 디렉토리를 지정합니다.
- 자세한 정보는 CICS 보안 토큰 서비스 구성의 내용을 참조하십시오.

JVM 프로파일 유효성 검사 및 등록 정보

JVM 프로파일에는 시작 시 JVM으로 전달되는 선택항목 및 시스템 등록 정보 세트가 포함됩니다. 일부 JVM 프로파일 선택항목은 CICS 환경에 고유하므로 다른 환경에서는 JVM에 사용하지 않습니다. CICS는 JVM 서버를 시작할 때 JVM 프로파일이 올바르게 코딩되었는지 검사합니다.

JVM 선택항목은 126 페이지의 『CICS 환경의 JVM에 대한 선택항목』에서 설명합니다. CICS는 CICS가 지원하는 각 JVM 서버 구성의 샘플 프로파일을 제공합니다. 이러한 샘플 프로파일은 가장 일반적인 JVM 선택항목의 기본값을 갖습니다. 샘플 프로파일은 zFS에서 `/usr/lpp/cicsts/cicsts52/JVMProfiles/`에 저장됩니다.

z/OS UNIX 시스템 서비스 환경 변수를 JVM 프로파일에 지정할 수도 있습니다. 올바른 JVM 선택항목에 없는 이름 및 값 쌍은 z/OS UNIX 시스템 서비스 환경 변수로 간주되어 내보냅니다. JVM 프로파일에 지정된 z/OS UNIX 시스템 서비스 환경 변수는 해당 프로파일로 작성된 JVM에만 적용됩니다.

환경 변수 예에는 Liberty 프로파일의 `WLP_INSTALL_DIR` 변수와 JVM 시간대 변경을 위한 `TZ` 변수가 포함됩니다.

Java 클래스 라이브러리에는 JVM 프로파일에서 설정할 수 있는 다른 시스템 등록 정보가 포함됩니다. 예를 들어, 애플리케이션은 또한 고유한 시스템 등록 정보를 가질 수 있습니다. IBM Java 문서는 Java 정보의 기본 소스입니다. JVM 시스템 등록 정보에 대한 자세한 정보는 IBM SDK for z/OS, Java Technology Edition, 버전 7 사용자 안내서의 내용을 참조하십시오.

『JVM 프로파일 코딩 규칙』
JVM 프로파일은 표준 텍스트 편집기를 사용하여 편집할 수 있습니다. JVM 프로파일을 코딩할 때 이 규칙을 따르십시오.

125 페이지의 『JVM 프로파일 선택항목의 유효성 검사』
CICS는 JVM을 시작할 때마다 JVM 프로파일에 지정된 키 선택항목에 대해 여러 가지 확인을 수행합니다. 이러한 확인을 통해 JVM 설정에서 문제점을 쉽게 판별할 수 있습니다.

126 페이지의 『CICS 환경의 JVM에 대한 선택항목』
JVM 프로파일의 선택항목은 CICS에서 JVM 서버를 시작하는 데 사용됩니다. 일부 선택항목은 CICS에만 해당되지만 환경 변수와 Java 시스템 등록 정보를 지정할 수도 있습니다.

134 페이지의 『JVM 시스템 등록 정보』
JVM 시스템 등록 정보는 JVM 및 해당 런타임 환경에 대한 구성 정보를 제공합니다. JVM 프로파일에 JVM 시스템 등록 정보를 추가하여 제공합니다. CICS는 런타임에 JVM 프로파일의 등록 정보를 읽고 JVM로 전달합니다.

JVM 프로파일 코딩 규칙

JVM 프로파일은 표준 텍스트 편집기를 사용하여 편집할 수 있습니다. JVM 프로파일을 코딩할 때 이 규칙을 따르십시오.

JVM 프로파일 이름

- 이름은 최대 8자까지 가능합니다.

- 이름은 z/OS UNIX 시스템 서비스에서 파일에 유효한 모든 이름이 가능합니다. DFH로 시작되는 이름은 사용하지 마십시오. 이러한 문자는 CICS에서 사용하도록 예약되어 있기 때문입니다.
- JVM 프로파일은 UNIX 파일이므로 대소문자 구분이 중요합니다. CICS에서 이름을 지정하는 경우 z/OS UNIX 파일 이름과 동일한 대소문자 조합을 사용하여 입력해야 합니다.
- 파일 시스템의 JVM 프로파일은 파일 확장자 `.jvmprofile`을 가져야 합니다. 파일 확장자는 소문자로 설정되며 변경해서는 안됩니다.

디렉토리

JVM 프로파일에서 디렉토리 값을 지정할 때 따옴표를 사용하지 마십시오.

CEDA

CEDA 패널은 단말기 UCTRAN 설정에 관계 없이 JVMPROFILE 필드에 대한 대소문자 혼합 입력을 승인합니다. 그러나 명령행에서 CEDA를 사용하거나 다른 CICS 트랜잭션을 사용하는 경우에는 대소문자가 혼합된 JVM 프로파일의 이름을 입력해야 합니다. 단말기가 대문자 변환을 비활성화하여 올바르게 구성되었는지 확인하십시오. 현재 세션의 경우에만, 제공되는 CEOT 트랜잭션을 사용하여 단말기의 대문자 변환 상태(UCTRAN)를 변경할 수 있습니다.

대소문자 구분

모든 매개변수 키워드와 피연산자는 대소문자를 구분하므로 126 페이지의 『CICS 환경의 JVM에 대한 선택항목』 및 134 페이지의 『JVM 시스템 등록 정보』에 표시된 대로 정확하게 지정해야 합니다.

클래스 경로 분리 문자

:(콜론) 문자를 사용하여 클래스 경로 선택항목에 지정하는 디렉토리 경로를 구분할 수 있습니다(예: `CLASSPATH_SUFFIX`).

연속

JVM 선택항목의 경우에는 텍스트 파일에서 행 끝으로 값이 구분됩니다. 입력 또는 편집하는 값이 편집기 창에 너무 긴 경우 화면 이동 방지를 위해 행을 구분할 수 있습니다. 다음 행에서 계속 진행하려면 다음 예제와 같이 백슬래시 문자와 공백 연속 문자로 현재 행을 종료하십시오.

```
CLASSPATH_SUFFIX=/u/example/pathToJarOrZipFile/jarfile.jar:\
/u/example/pathToRootDirectoryForClasses
```

동일한 행에 여러 JVM 선택항목을 삽입하지 마십시오.

설명

설명을 추가하거나 선택항목을 삭제하지 않고 주석 처리하려면 설명의 각 행을 # 기호로 시작하십시오. JVM 실행 프로그램이 파일을 읽을 때 설명 행이 무시됩니다.

빈 행 또한 무시됩니다. 빈 행을 선택항목 또는 선택항목 그룹 간의 분리 문자로 사용할 수 있습니다.

프로파일 구문 분석 코드는 UNIX 유사 쉘 처리를 기반으로 인라인 설명을 제거하므로 샘플 JVM 프로파일의 문서화 프로세스가 개선됩니다. 인라인 설명은 다음과 같이 정의됩니다.

- 설명은 # 기호로 시작됩니다.
- 하나 이상의 공백(탭)이 앞에 추가됩니다.
- 따옴표로 묶은 텍스트에 포함되지 않습니다.

표 9. 인라인 설명 예제

코드	결과
`MYVAR=myValue  # Comment`	MYVAR=myValue
`MYVAR=#myValue  # Comment`	MYVAR=#myValue
`MYVAR=myValue  "# Quoted comment" # Comment`	MYVAR=myValue "# Quoted comment"

문자 이스케이프 순서

표 10에 표시되는 이스케이프 순서를 코딩할 수 있습니다.

표 10. 이스케이프 순서

이스케이프 순서	문자 값
\b	백스페이스
\t	가로 탭
\n	줄 바꾸기
\r	캐리지 리턴
\"	큰따옴표 표시
\'	작은따옴표 표시
\\	백슬래시
xxx	8진 값 *xxx*에 해당하는 문자. 여기서 *xxx*는 000과 377 사이의 값입니다.
\uxxxx	*xxxx* 인코딩을 사용하는 유니코드 문자. 여기서 *xxxx*는 1자에서 4자의 16진수입니다. 자세한 정보는 참고사항을 참조하십시오.

참고: 유니코드 \u 이스케이프는 다른 이스케이프 유형과 다릅니다. 유니코드 이스케이프 순서는 표 10에서 설명하는 나머지 이스케이프 순서보다 먼저 처리됩니다. 유니코드 이스케이프는 유니코드 이외 시스템에서 표시될 수 없는 문자를 나타내는 대체 방법입니다. 그러나 문자 이스케이프는 해당 문자의 일반적인 해석을 방지하는 방식으로 특수 문자를 나타낼 수 있습니다.

선택항목의 복수 인스턴스

동일한 선택항목의 여러 인스턴스가 JVM 프로파일에 포함되는 경우 마지막으로 찾은 선택항목의 값을 사용하고 이전 값은 무시됩니다.

저장영역 크기

JVM 프로파일에서 저장영역 관련 선택항목을 지정하는 경우 저장영역 *크기*는 1024바이트의 배수로 지정합니다. KB를 나타내려면 문자 K, MB를 나타내려면 문자 M, GB를 나타내려면 문자 G를 사용하십시오. 예를 들어, 6291456 바이트를 힙의 *초기 크기*로 지정하려면 다음 중 한 가지 방법으로 **-Xms**를 코딩하십시오.

```
-Xms6144K
-Xms6M
```

JVM 프로파일 선택항목의 유효성 검사

CICS는 JVM을 시작할 때마다 JVM 프로파일에 지정된 키 선택항목에 대해 여러 가지 확인을 수행합니다. 이러한 확인을 통해 JVM 설정에서 문제점을 쉽게 판별할 수 있습니다.

CICS는 다음 JVM 프로파일 선택항목과 관련된 확인을 수행합니다.

CLASSPATH_PREFIX, CLASSPATH_SUFFIX

JVM 프로파일에 이러한 선택항목이 없으면 OSGi 프레임워크 없이 JVM 서버가 시작됩니다.

JAVA_HOME

CICS는 디렉토리에 대한 다음 사항을 확인합니다.

- 이 디렉토리는 z/OS UNIX에 있습니다.
- CICS에 최소한 읽기 권한이 있어 디렉토리에 액세스할 수 있습니다.
- JDK_INSTALL_OK 파일이 디렉토리에 있으므로 이 위치에서 IBM 64-bit SDK for z/OS, Java Technology Edition 7 파일 설치를 완료했음을 나타냅니다.
- JDK_INSTALL_OK 파일의 Java 릴리스 번호는 CICS가 지원하는 버전입니다.

문제점이 발견되면 CICS가 오류 메시지를 실행하고 JVM을 시작하지 않습니다.

OSGI_BUNDLES

JVM 서버 프로파일의 경우 CICS가 지정된 JAR 파일이 OSGi 번들인지 확인합니다. CICS는 또한 미들웨어 번들이 올바르게 구분되고 올바른 분리 문자가 있는지 확인합니다.

더 이상 사용되지 않는 클래스 경로 선택항목: LIBPATH, CLASSPATH

JVM 프로파일에 이러한 선택항목 중 하나 이상이 있으면 JVM 시동 시 경고 메시지가 실행됩니다. JVM 프로파일에서 이러한 선택항목을 사용하지 마십시오. 대신 메시지가 사용할 올바른 선택항목을 알려줍니다.

더 이상 사용되지 않는 클래스 경로 선택항목: CICS_HOME, TMPREFIX, TMSUFFIX

이러한 선택항목은 무시됩니다.

CICS 환경의 JVM에 대한 선택항목

JVM 프로파일의 선택항목은 CICS에서 JVM 서버를 시작하는 데 사용됩니다. 일부 선택항목은 CICS에만 해당되지만 환경 변수와 Java 시스템 등록 정보를 지정할 수도 있습니다.

코딩 규칙

JVM 선택항목을 지정할 때 코딩 규칙을 따르는지 확인하십시오. 자세한 정보는 122 페이지의 『JVM 프로파일 코딩 규칙』의 내용을 참조하십시오.

형식

선택항목 형식은 다음가 같이 다양할 수 있습니다.

- JVM 프로파일의 선택항목은 등호 부호(=)로 구분되는 키워드와 값의 양식을 취하거나(예: `JAVA_PIPELINE=YES`) 하이픈으로 시작됩니다(예: `-Xmx16M`).
- 키워드 값 쌍은 CICS 변수(예: JAVA_PIPELINE=YES)이거나, CICS 선택항목으로 인식되지 않는 경우 z/OS UNIX 시스템 서비스 환경 변수로 간주되어 내보냅니다.
- JVM 프로파일에서 `-D`로 시작되는 선택항목은 JVM 시스템 등록 정보입니다. `-X`로 시작되는 선택항목은 JVM 명령행 선택항목으로 간주됩니다. `-`로 시작되는 선택항목은 CICS가 구문을 분석하지 않고 JVM으로 전달됩니다. 자세한 정보는 134 페이지의 『JVM 시스템 등록 정보』의 내용을 참조하십시오.

기호

지원되는 기호는 다음과 같습니다.

`&APPLID;`

이 기호를 사용하면 런타임에 CICS 영역의 APPLID가 대체됩니다. 이러한 방식으로 모든 영역에 동일한 프로파일을 사용할 수 있으며 영역에 고유한 작업 디렉토리 또는 출력 대상을 계속 갖습니다. APPLID는 항상 대문자입니다.

`&CONFIGROOT;`

이 기호를 사용하면 JVM 프로파일이 있는 디렉토리의 절대 경로가 런타임에 대체됩니다. CICS 번들에 정의된 JVM 서버의 경우 JVM 프로파일의 기본 위치는 번들의 루트 디렉토리입니다. 다른 메소드로 정의되는 JVM 서버의 경우 JVM 프로파일은 JVMPROFILEDIR 시스템 초기화 매개변수로 지정되는 디렉토리에 있습니다.

&DATE;

이 기호를 사용하는 경우 기호는 런타임에 *Dyymmdd* 형식의 현재 날짜로 바뀝니다.

&JVMSERVER;

이 기호를 사용하는 경우 런타임에 JVMSERVER 자원의 이름이 대체됩니다. 각 JVM 서버에 고유한 출력 또는 덤프 파일을 작성하려면 이 기호를 사용하십시오.

&TIME;

이 기호를 사용하는 경우 기호는 런타임에 *Thhmmss* 형식의 JVM 시작 시간으로 바뀝니다.

&USSHOME;

이 기호를 사용하는 경우 USSHOME 시스템 초기화 매개변수의 값으로 기호가 대체됩니다. 이 기호를 지정하여 CICS가 Java용 라이브러리와 Liberty 프로파일을 제공하는 z/OS UNIX용 홈 디렉토리를 자동으로 선택할 수 있습니다.

선택항목 목록

해당되는 경우 기본값은 선택항목이 지정되지 않을 때 CICS가 사용하는 값입니다. 일부 또는 모든 샘플 JVM 프로파일이 기본값과 다른 값을 지정할 수 있습니다.

CLASSPATH_PREFIX, CLASSPATH_SUFFIX=*class_pathnames*

이 선택항목을 사용하면 OSGi를 사용하지 않는 JVM이 검색할 디렉토리 경로, Java 아카이브 파일, 압축 파일을 지정할 수 있습니다. 예를 들어, Java 웹 서비스에 사용됩니다. OSGi 프레임워크가 자동으로 클래스 로딩을 처리하므로 OSGi 프레임워크를 사용하려는 경우 클래스 경로를 설정하지 마십시오. Axis2용 표준 클래스 경로를 지정하기 위해 이러한 선택항목을 사용하는 경우에는 또한 JAVA_PIPELINE=YES를 지정하여 Axis2 엔진을 시작해야 합니다.

CLASSPATH_PREFIX는 표준 클래스 경로 처음에 클래스 경로 항목을 추가하고 CLASSPATH_SUFFIX는 표준 클래스 경로 끝에 추가합니다. 계속될 각 행 끝에 \(백슬래시)를 사용하여 개별 행에서 항목을 지정할 수 있습니다.

CLASSPATH_PREFIX 선택항목은 신중하게 사용하십시오. CLASSPATH_PREFIX의 클래스는 CICS 및 Java 런타임이 제공하는 동일한 이름을 사용하는 클래스에 우선하며 잘못된 클래스가 로드될 수 있습니다.

CICS는 JVM 프로파일에서 JAVA_HOME 선택항목과 **USSHOME** 시스템 초기화 매개변수로 지정된 디렉토리의 /lib 서브디렉토리를 사용하여 JVM에 대한 기본 클래스 경로를 빌드합니다. 이 기본 클래스 경로에는 CICS 및 JVM이 제공하는 Java 아카이브 파일이 포함됩니다. JVM 프로파일에는 이 경로가 표시되지 않습니다. JVM 프로파일의 클래스 경로에서 이러한 파일을 다시 지정하지 마십시오.

JAVA_DUMP_TDUMP_PATTERN=

JVM으로부터의 트랜잭션 덤프(TDUMP)에 사용될 파일 이름을 지정하는 z/OS

UNIX 시스템 서비스 환경 변수입니다. JVM이 이상 종료되는 경우 Java TDUMP는 데이터 세트 대상에 기록됩니다. 이 값에서 기호 &APPLID;(CICS 영역 APPLID) 및 &JVMSERVER;를 사용할 수 있습니다.

JAVA_HOME=/usr/lpp/java/J7.0_64/

z/OS UNIX에서 IBM 64-bit SDK for z/OS, Java Technology Edition의 설치 위치를 지정합니다. 이 위치에는 Java 지원에 필요한 Java 아카이브 파일과 서브디렉토리가 포함됩니다.

제공된 샘플 JVM 프로파일에는 DFHISTAR CICS 설치 작업에서 **JAVADIR** 매개변수로 생성된 경로가 포함됩니다. **JAVADIR** 매개변수의 기본값은 IBM 64-bit SDK for z/OS, Java Technology Edition의 기본 설치 위치인 java/J7.0_64/입니다. 이 값은 /usr/lpp/java/J7.0_64/의 JVM 프로파일에서 JAVA_HOME 설정을 생성합니다.

JAVA_PIPELINE={YES,NO}

클래스 경로에 필수 Java 아카이브 파일을 추가하면 JVM 서버가 Java 표준 SOAP 파이프라인에서 웹 서비스 처리를 지원할 수 있습니다. 기본값은 NO입니다. 이 값을 설정하면 JVM 서버가 OSGi 대신 Axis2를 지원하도록 구성됩니다. CLASSPATH 선택항목을 사용하여 클래스 경로에 JAR 파일을 더 추가할 수 있습니다.

참고: JAVA_PIPELINE=YES 및 SECURITY_TOKEN_SERVICE=YES 선택항목은 호환되지 않습니다.

JNDI_REGISTRATION=YES|NO

Java 애플리케이션의 JNDI 사용을 지원하기 위해 JNDI 유효성 검사 JAR 파일이 JVM 런타임 환경에 자동으로 추가되도록 지정합니다. Liberty JVM 서버에는 이 선택항목이 무시됩니다. JNDI_REGISTRATION=NO를 설정하여 이러한 파일을 자동 추가하는 기능을 선택 취소할 수 있습니다. 이 기능이 필요하지 않으면 선택 취소하여 최신 JAR과의 충돌을 방지하고 JVM 영역을 더 작게 유지하며 불필요한 클래스 로드를 방지할 수 있습니다.

JVMTRACE={&APPLID;.&JVMSERVER;.Dyyyymmdd.Thhmmss.dfhjvmtrc|*file_name*}

JVM 서버 시작 및 종료 시 Java 추적이 기록되는 z/OS UNIX 파일의 이름을 지정합니다. 이 선택항목의 값을 설정하지 않으면 CICS가 각 JVM 서버에 고유한 추적 파일을 자동으로 작성합니다. CICS는 &APPLID; 및 &JVMSERVER; 기호와 JVM 서버가 시작되는 날짜 및 시간을 사용합니다. 이 파일은 WORK_DIR 선택항목으로 지정되는 디렉토리에서 작성됩니다.

LIBPATH_PREFIX, LIBPATH_SUFFIX=*pathnames*

JVM이 사용하고 z/OS UNIX에서 확장자 .so를 갖는 고유 C 동적 링크 라이브

러리(DLL) 파일을 검색할 디렉토리 경로를 지정합니다. 여기에는 JVM을 실행하는 데 필요한 파일과 애플리케이션 코드 또는 서비스로 로드되는 추가 고유 라이브러리가 포함됩니다.

JVM의 기본 라이브러리 경로는 JVM 프로파일에서 **USSHOME** 시스템 초기화 매개변수 및 JAVA_HOME 선택항목으로 지정된 디렉토리를 사용하여 자동으로 빌드됩니다. 기본 라이브러리 경로는 JVM 프로파일에 표시되지 않습니다. 이 경로에는 CICS가 사용하는 고유 라이브러리와 JVM을 실행하는 데 필요한 모든 DLL 파일이 포함됩니다.

LIBPATH_SUFFIX 선택항목을 사용하여 라이브러리 경로를 확장할 수 있습니다. 이 선택항목은 라이브러리 경로 끝에서 기본 라이브러리 경로 다음에 디렉토리를 추가합니다. 이 선택항목을 사용하면 애플리케이션이 사용하는 추가 고유 라이브러리를 포함하는 디렉토리를 지정할 수 있습니다. 또한 이 선택항목을 사용하면 CICS에 대한 표준 JVM 설정에 포함되지 않은 서비스가 사용하는 디렉토리를 지정할 수 있습니다. 예를 들어, 추가 고유 라이브러리에는 DB2 JDBC 드라이버를 사용하는 데 필요한 DLL 파일이 포함될 수 있습니다.

LIBPATH_PREFIX 선택항목은 라이브러리 경로 시작에서 기본 라이브러리 경로 이전에 디렉토리를 추가합니다. 이 선택항목은 신중하게 사용해야 합니다. 지정된 디렉토리의 DLL 파일이 기본 라이브러리 경로에서 DLL 파일과 이름이 같은 경우 제공된 파일 대신 해당 파일이 로드됩니다.

LIBPATH_PREFIX 또는 LIBPATH_SUFFIX 선택항목을 사용하여 지정하는 여러 항목을 구분하려면 쉼표가 아닌 콜론을 사용하십시오.

애플리케이션이 사용하기 위해 라이브러리 경로에 있는 DLL 파일은 XPLink 선택항목으로 컴파일 및 링크되어야 합니다. XPLink 선택항목으로 컴파일 및 링크하면 최적의 성능을 제공할 수 있습니다. 기본 라이브러리 경로에서 제공되는 DLL 파일과 DB2 JDBC 드라이버와 같은 서비스가 사용하는 DLL 파일은 XPLink 선택항목으로 빌드됩니다.

OSGI_BUNDLES=*pathnames*

OSGi JVM 서버의 OSGi 프레임워크에서 사용 가능하게 설정되는 미들웨어 번들의 디렉토리 경로를 지정합니다. 이러한 OSGi 번들에는 프레임워크에서 시스템 기능을 구현하기 위한 클래스가 포함됩니다(예를 들어, WebSphere MQ 또는 DB2에 연결). 여러 OSGi 번들을 지정하는 경우에는 쉼표를 사용하여 구분하십시오.

OSGI_FRAMEWORK_TIMEOUT=60|*number*

시간이 종료되기 전에 OSGi 프레임워크가 초기화 또는 종료될 때까지 CICS가 대기하는 시간(초)을 지정합니다. 1초 - 60000초 범위로 값을 설정할 수 있습니다. 기본값은 60초입니다. OSGi 프레임워크를 시작하는 데 지정된 시간(초)보다 많은 시간이 소요되면 JVM 서버가 초기화되지 않고 CICS에서 DFHSJ0215 메시지가 실

행됩니다. zFS의 JVM 서버 로그 파일에도 오류 메시지가 기록됩니다. OSGi 프레임워크를 종료하는 데 지정된 시간(초)보다 많은 시간이 소요되면 JVM 서버가 정상적으로 종료되지 않습니다.

PRINT_JVM_OPTIONS={YES|NO}

이 선택항목이 YES로 설정되면 JVM이 시작될 때마다 시동 시 JVM으로 전달되는 선택항목이 SYSPRINT에도 인쇄됩니다. 해당 프로파일의 이 선택항목으로 JVM이 시작될 때마다 출력이 생성됩니다. 이 선택항목을 사용하면 기본 라이브러리 경로와 JVM 프로파일에서 표시되지 않으며 CICS로 빌드된 기본 클래스 경로를 포함하는 특정 JVM 프로파일에 대한 클래스 경로의 내용을 확인할 수 있습니다.

SECURITY_TOKEN_SERVICE=YES|NO

이 선택항목이 YES로 설정되면 JVM 서버가 보안 토큰을 사용할 수 있습니다. 이 선택항목이 NO로 설정되면 JVM 서버에 대해 보안 토큰 서비스 지원을 사용할 수 없습니다.

참고: SECURITY_TOKEN_SERVICE=YES 및 JAVA_PIPELINE=YES 선택항목은 호환되지 않습니다.

STDERR={&APPLID;.&JVMSERVER;.Dyyyymmdd.Thhmmss.dfhjvmerr|*file_name*}

stderr에 사용될 z/OS UNIX 파일의 이름을 지정합니다. 이 파일이 없으면 WORK_DIR 선택항목으로 지정된 디렉토리에서 작성됩니다. 이 파일이 있으면 파일 끝에 출력이 추가됩니다. 이 선택항목의 값을 설정하지 않으면 CICS가 각 JVM 서버에 고유한 출력 파일을 자동으로 작성합니다. CICS는 &APPLID; 및 &JVMSERVER; 기호와 JVM 서버가 시작되는 날짜 및 시간을 사용합니다. 각 JVM 서버의 고유한 출력 파일을 작성하려면 파일 이름에 &JVMSERVER; 및 &APPLID; 기호를 사용하십시오(샘플 JVM 프로파일 참조).

USEROUTPUTCLASS 선택항목을 JVM 프로파일에서 지정하면 해당 선택항목에서 이름이 지정되는 Java 클래스가 대신 System.err 요청을 처리합니다. USEROUTPUTCLASS 선택항목으로 이름이 지정되는 클래스가 지정된 대상에 데이터를 쓸 수 없는 경우, STDERR 선택항목으로 이름이 지정된 z/OS UNIX 파일을 계속 사용할 수 있습니다. 이는 제공된 샘플 클래스 com.ibm.cics.samples.SJMergedStream을 사용하는 경우입니다. 다른 이유로 인해 USEROUTPUTCLASS 선택항목으로 이름이 지정되는 클래스에 의해 출력 방향이 지정되는 경우에도 이 파일을 사용할 수 있습니다.

STDIN={*file_name*}

stdin에 사용될 z/OS UNIX 파일의 이름을 지정합니다. 이 선택항목의 값을 지정하지 않으면 CICS가 이 파일을 작성하지 않습니다.

STDOUT={&APPLID;.&JVMSERVER;.Dyyyymmdd.Thhmmss.dfhjvmout|*file_name*}

stdout 파일에 대한 출력에 사용될 z/OS UNIX 파일의 이름을 지정합니다. 이 파일이 없으면 WORK_DIR 선택항목으로 지정된 디렉토리에서 작성됩니다. 이 파일이 있으면 파일 끝에 출력이 추가됩니다. 이 선택항목의 값을 설정하지 않으면 CICS

가 각 JVM 서버에 고유한 출력 파일을 자동으로 작성합니다. CICS는 &APPLID; 및 &JVMSERVER; 기호와 JVM 서버가 시작되는 날짜 및 시간을 사용합니다.

USEROUTPUTCLASS 선택항목을 JVM 프로파일에서 지정하면 해당 선택항목에서 이름이 지정되는 Java 클래스가 대신 System.out 요청을 처리합니다. USEROUTPUTCLASS 선택항목으로 이름을 지정한 클래스가 지정된 대상에 데이터를 쓰지 못하면 STDOUT 선택항목으로 이름이 지정되는 z/OS UNIX 파일을 계속 사용할 수 있습니다(예를 들어, 동일한 클래스 com.ibm.cics.samples.SJMergedStream을 사용하는 경우). 다른 이유로 인해 USEROUTPUTCLASS 선택항목으로 이름이 지정되는 클래스에 의해 출력 방향이 지정되는 경우에도 이 파일을 사용할 수 있습니다.

USEROUTPUTCLASS={*classname***}**

JVM의 출력과 JVM 내부의 메시지를 인터셉트하는 Java 클래스의 완전한 이름을 지정합니다. 이 Java 클래스를 사용하여 JVM의 출력과 메시지의 경로를 재지정할 수 있으므로 출력 레코드에 시간 소인과 헤더를 추가할 수 있습니다. Java 클래스가 지정된 대상에 데이터를 쓸 수 없는 경우에는 STDOUT 및 STDERR 선택항목에서 이름이 지정되는 파일을 계속 사용할 수 있습니다.

USEROUTPUTCLASS 선택항목을 지정하면 JVM 성능에 부정적인 영향을 미칩니다. 프로덕션 환경에서 최대 성과를 얻으려면 이 선택항목을 사용하지 마십시오. 그러나 동일한 CICS 영역을 사용하는 애플리케이션 개발자에게는 이 선택항목이 유용할 수 있습니다. JVM 출력 방향은 식별 가능한 대상으로 지정될 수 있기 때문입니다.

이 클래스 및 제공된 샘플에 대한 자세한 정보는 201 페이지의 『JVM stdout, stderr, JVMTRACE 및 덤프 출력의 위치 제어』의 내용을 참조하십시오.

WLP_INSTALL_DIR={$USSHOME;/wlp|*directory_path***/wlp}**

Liberty 프로파일 기술의 설치 디렉토리를 지정합니다. Liberty 프로파일은 CICS용 z/OS UNIX 홈에서 wlp 서브디렉토리에 설치됩니다. 기본 설치 디렉토리는 /usr/lpp/cicsts/cicsts52/wlp입니다. 항상 &USSHOME; 기호를 사용하여 올바른 파일 경로를 설정하고 wlp 디렉토리를 추가하십시오.

이 환경 변수는 Liberty JVM 서버를 시작하려는 경우 필요합니다. 이 환경 변수를 설정하는 경우 다른 환경 변수와 시스템 등록 정보를 제공하여 Liberty JVM 서버를 구성할 수도 있습니다. 환경 변수에는 접두부 WLP가 추가됩니다. 시스템 등록 정보는 134 페이지의 『JVM 시스템 등록 정보』의 내용을 참조하십시오.

WLP_OUTPUT_DIR={*WLP_USER_DIR***/servers}**

Liberty 프로파일의 출력 파일을 포함하는 디렉토리를 지정합니다. 기본적으로 Liberty 프로파일은 서버를 따라 이름이 지정되는 디렉토리에 서버의 로그, 작업 영역, 구성 파일, 애플리케이션을 저장합니다.

이 환경 변수는 선택적입니다. 이 환경 변수를 지정하지 않는 경우 CICS의 기본값은 *$WORK_DIR*/*$APPLID*/*$JVMSERVER*/wlp/usr/servers/*server_name*이며 기호는 런타임 값으로 대체됩니다.

이 환경 변수가 설정되는 경우 출력 로그와 작업 영역은 ${WLP_OUTPUT_DIR}/*server_name*에 저장됩니다.

WLP_USER_DIR={$APPLID;/&JVMSERVER;/wlp/usr/|*directory_path*}

Liberty JVM 서버의 구성 파일을 포함하는 디렉토리를 지정합니다. 이 환경 변수는 선택적입니다. 이 환경 변수를 지정하지 않는 경우 CICS는 작업 디렉토리에서 *$APPLID*/*$JVMSERVER*/wlp/usr/을 사용합니다. 기호는 런타임 값으로 대체됩니다. 구성 파일은 servers/*server_name*에 작성됩니다.

WLP_ZOS_PLATFORM=FALSE

Liberty JVM 서버에서 z/OS 플랫폼 확장자를 사용 안함으로 설정합니다. 이 경우 cicsts:security-1.0 특성을 사용하지 않고 여러 Liberty JVM 서버를 동일한 영역에서 시작할 수 있습니다.

WORK_DIR={.|*directory_name*}

CICS 영역이 Java와 관련된 활동을 위해 사용하는 작업 디렉토리를 z/OS UNIX에서 지정합니다. CICS JVM 인터페이스는 stdin, stdout, stderr 파일을 작성할 때 이 디렉토리를 사용합니다. 마침표(.)가 제공된 JVM 프로파일에 정의되며 이는 CICS 영역 사용자 ID의 홈 디렉토리가 작업 디렉토리로 사용됨을 나타냅니다. 이 디렉토리는 CICS 설치 중에 작성될 수 있습니다. 이 디렉토리가 없거나 WORK_DIR이 생략되면 /tmp가 z/OS UNIX 디렉토리 이름으로 사용됩니다.

작업 디렉토리의 절대 경로 또는 상대 경로를 지정할 수 있습니다. 상대 작업 디렉토리는 CICS 영역 사용자 ID의 홈 디렉토리에 상대적입니다. 홈 디렉토리를 Java와 관련된 활동의 작업 디렉토리로 사용하지 않거나 CICS 영역이 z/OS 사용자 ID(UID)를 공유해서 동일한 홈 디렉토리를 갖는 경우 각 CICS 영역마다 다른 작업 디렉토리를 작성할 수 있습니다. CICS가 실제 CICS 영역 APPLID를 대체하는 &APPLID; 기호를 사용하는 디렉토리 이름을 지정합니다. 따라서 모든 CICS 영역이 JVM 프로파일 세트를 공유하더라도 각 영역에 고유한 작업 디렉토리를 가질 수 있습니다. 예를 들어, 다음을 지정하는 경우

```
WORK_DIR=/u/&APPLID;/javaoutput
```

해당 JVM 프로파일을 사용하는 각 CICS 영역이 고유한 작업 디렉토리를 갖습니다. 관련 디렉토리가 z/OS UNIX에서 작성되고 CICS 영역에 읽기, 쓰기, 실행 액세스 권한이 부여되었는지 확인하십시오.

작업 디렉토리에 고정된 이름을 지정할 수도 있습니다. 이러한 경우 또한 관련 디렉토리가 z/OS UNIX에서 작성되고 올바른 CICS 영역에 액세스 권한이 부여되었는지 확인해야 합니다. 작업 디렉토리에 고정된 이름을 사용하는 경우 CICS 영역에서 JVM 프로파일을 공유하는 모든 JVM 서버의 출력 파일이 해당 디렉토리에

서 작성됩니다. 출력 파일에 고정된 파일 이름을 사용하는 경우 해당 CICS 영역에 있는 모든 JVM 서버의 출력이 동일한 z/OS UNIX 파일에 추가됩니다. 동일한 파일에 추가되지 않도록 하려면 &JVMSERVER; 기호와 &APPLID; 기호를 사용하여 각 JVM 서버에 고유한 출력 및 덤프 파일을 생성하십시오.

z/OS UNIX의 경우 **USSHOME** 시스템 초기화 매개변수로 정의된 CICS 파일의 홈 디렉토리인 CICS 디렉토리에 작업 디렉토리를 정의하지 마십시오.

또한 USEROUTPUTCLASS 선택항목을 사용하여 JVM에서 stderr 및 stdout 출력을 인터셉트하고 경로를 재지정하며 형식화하는 Java 클래스의 이름을 지정할 수도 있습니다. 출력 방향 전환을 위해 제공된 샘플 클래스는 경우에 따라 WORK_DIR로 지정된 디렉토리를 사용합니다.

WSDL_VALIDATOR=YES|NO

해당 정의와 스키마에 대해 SOAP 요청 및 응답의 유효성을 검사합니다. Liberty JVM 서버에는 이 선택항목이 무시됩니다.자세한 정보는 SOAP 메시지 유효성 검사의 내용을 참조하십시오. WSDL_VALIDATOR=NO를 설정하여 이 선택항목을 설정할 수 있습니다. 이 선택항목을 선택하지 않으면 최신 JAR과의 충돌, 저장영역 낭비 및 시작 지연을 방지할 수 있습니다.

-agentlib

JVM에서 디버깅 지원을 사용하는지 여부를 지정합니다.

자세한 정보는 208 페이지의 『Java 애플리케이션 디버깅』의 내용을 참조하십시오. JPDA(Java Platform Debugger Architecture)에 대한 자세한 정보는 Oracle Technology Network Java 웹 사이트의 내용을 참조하십시오.

-Xms

힙의 초기 크기를 지정합니다. 저장영역 크기는 1024바이트의 배수로 지정하십시오. KB를 나타내려면 문자 K, MB를 나타내려면 문자 M, GB를 나타내려면 문자 G를 사용하십시오. 예를 들어, 6,291,456바이트를 힙의 초기 크기로 지정하려면 다음 중 한 가지 방법으로 **-Xms**를 코딩하십시오.

```
-Xms6144K
-Xms6M
```

*size*는 숫자(KB 또는 MB)로 지정하십시오. 자세한 정보는 JVM 명령행 옵션의 내용을 참조하십시오.

-Xmx

힙의 최대 크기를 지정합니다. 이 고정 크기의 저장영역은 JVM 초기화 중에 JVM에 의해 할당됩니다.

*size*는 숫자(KB 또는 MB)로 지정하십시오.

-Xscmx

공유 클래스 캐시의 크기를 지정합니다. 최소 크기는 4KB입니다. 최대 크기와 기본 크기는 플랫폼에 따라 다릅니다.

*size*는 숫자(KB 또는 MB)로 지정하십시오. 자세한 정보는 JVM 명령행 옵션의 내용을 참조하십시오.

-Xshareclasses
공유 클래스 캐시에서 클래스 데이터 공유를 사용하려면 이 선택항목을 지정하십시오. JVM은 기존 캐시에 연결되거나 캐시가 없는 경우 캐시를 작성합니다. 여러 캐시를 가질 수 있으며 **-Xshareclasses** 선택항목에 하위 선택항목을 추가하여 올바른 캐시를 지정할 수 있습니다. 자세한 정보는 JVM 간 클래스 데이터 공유의 내용을 참조하십시오.

-Xmso JVM 선택항목 및 기본값에 대한 정보는 JVM 명령행 옵션의 내용을 참조하십시오.

JVM 시스템 등록 정보

JVM 시스템 등록 정보는 JVM 및 해당 런타임 환경에 대한 구성 정보를 제공합니다. JVM 프로파일에 JVM 시스템 등록 정보를 추가하여 제공합니다. CICS는 런타임에 JVM 프로파일의 등록 정보를 읽고 JVM로 전달합니다.

등록 정보 접두부

-Dcom.ibm.cics는 등록 정보가 CICS 환경의 IBM JVM에 고유함을 나타냅니다.

-Dcom.ibm은 보다 널리 사용되는 일반 JVM 등록 정보를 나타냅니다.

-Djava.ibm은 또한 보다 널리 사용되는 일반 JVM 등록 정보를 나타냅니다.

일반 등록 정보에 대한 정보는 121 페이지의 『JVM 프로파일 유효성 검사 및 등록 정보』의 내용을 참조하십시오.

등록 정보 코딩 규칙

등록 정보는 코딩 규칙 세트에 따라 지정해야 합니다. 규칙에 대한 자세한 정보는 122 페이지의 『JVM 프로파일 코딩 규칙』의 내용을 참조하십시오.

Liberty 사용 JVM 서버에만 관련된 등록 정보

-Dcom.ibm.cics.jvmserver.wlp.autoconfigure={false|true}
CICS가 Liberty JVM 서버의 `server.xml` 파일을 자동으로 작성 및 갱신하는지 여부를 지정합니다. 이 등록 정보를 true로 설정하면 CICS가 zFS에서 디렉토리 구조 및 구성 파일을 작성합니다.다른 Java 등록 정보(예: HTTP 포트 번호)의 값을 제공하는 경우 CICS는 또한 `server.xml` 파일을 갱신합니다.

-Dcom.ibm.cics.jvmserver.wlp.server.http.port={9080|*port_number*}
Java 웹 애플리케이션에 대한 HTTP 요청을 승인하기 위한 포트를 지정합니다. CICS는 Liberty 프로파일이 제공하는 기본값을 사용합니다. Liberty JVM 서버는 TCPIPSERVICE 자원을 사용하지 않으므로 포트 번호를 z/OS 시스템에서 사용할

수 있거나 공유해야 합니다. 이 등록 정보는 선택적이며 **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure** 등록 정보가 true로 설정된 경우에만 사용합니다.

-Dcom.ibm.cics.jvmserver.wlp.server.https.port={9443|*port_number*}
Java 웹 애플리케이션에 대한 HTTPS 요청을 승인하기 위한 포트를 지정합니다. CICS는 Liberty 프로파일이 제공하는 기본값을 사용합니다. Liberty JVM 서버는 TCPIPSERVICE 자원을 사용하지 않으므로 포트 번호를 z/OS 시스템에서 사용할 수 있거나 공유해야 합니다. 이 등록 정보는 선택적이며 **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure** 등록 정보가 true로 설정된 경우에만 사용합니다.

-Dcom.ibm.cics.jvmserver.wlp.server.host={*|*hostname*|*IP_address*}
웹 애플리케이션에 액세스하기 위한 HTTP 요청에 대해 호스트의 이름 또는 IP 주소(IPv4 또는 IPv6 형식)를 지정합니다. Liberty JVM 서버는 *를 기본값으로 사용합니다. 이 값은 CICS에서 웹 애플리케이션을 실행하는 데 적합하지 않을 수 있으므로 이 등록 정보를 사용하여 다른 값을 제공하거나 server.xml 파일을 갱신하십시오. 이 등록 정보는 선택적이며 **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure** 등록 정보가 true로 설정된 경우에만 사용합니다.

-Dcom.ibm.cics.jvmserver.wlp.args=
이 등록 정보는 IBM 서비스 지침이 있는 경우에만 사용합니다.

-Dcom.ibm.cics.jvmserver.wlp.jdbc.driver.location=
DB2 JDBC 드라이버를 포함하는 zFS의 디렉토리 위치를 지정합니다. 이 위치에는 DB2 JDBC 드라이버 classes 및 lib 디렉토리가 포함되어야 합니다. 자동 구성 등록 정보 **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure**가 true로 설정되어 있는 경우 JVM 서버가 사용으로 설정되면 server.xml의 기존 예제 구성이 기본 구성으로 바뀌고 사용자 갱신이 손실됩니다.

-Dcom.ibm.cics.jvmserver.wlp.optimize.static.resources={true|false}
정적 자원 최적화를 사용하여 CICS에서 요청을 충족시키기 위해 트랜잭션을 적게 사용할 수 있도록 설정합니다. .css, .gif, .ico, .jpg, .jpeg, .png 파일 유형은 정적인 것으로 인식됩니다.

-Dcom.ibm.ws.logging.console.log.level={INFO|AUDIT|WARNING|ERROR|OFF}
Liberty가 JVM 서버 stdout 파일에 쓰는 메시지를 제어합니다. 이 등록 정보 설정에 관계 없이 Liberty 콘솔 메시지 또한 Liberty messages.log 파일에 기록됩니다.

-Dcom.ibm.cics.jvmserver.wlp.server.name={defaultServer|*server_name*}

Liberty 프로파일 서버의 이름을 지정합니다. 이 등록 정보의 기본값은 defaultServer입니다. 이 등록 정보는 선택항목이며 zFS 시스템에 있는 Liberty 서버 구성 및 출력 파일과 디렉토리의 위치에 영향을 미치기 때문에 지정할 필요가 없습니다.

OSGi 사용 JVM 서버 및 Liberty 사용 JVM 서버 모두와 관련된 등록 정보

-Dcom.ibm.cics.jvmserver.name=

JVM 서버의 이름입니다. 읽기 전용 등록 정보입니다. 해당 값을 보고 애플리케이션에서 사용할 수 있지만 변경할 수는 없습니다.

-Dcom.ibm.cics.jvmserver.applid=

CICS 영역 애플리케이션 ID(APPLID)입니다. 읽기 전용 등록 정보입니다. 해당 값을 보고 애플리케이션에서 사용할 수 있지만 변경할 수는 없습니다.

-Dcom.ibm.cics.jvmserver.supplied.ccsid=

로컬 영역의 기본 CCSID입니다. 읽기 전용 등록 정보입니다. 해당 값을 보고 애플리케이션에서 사용할 수 있지만 변경할 수는 없습니다.

-Dcom.ibm.cics.jvmserver.trace.filename=

JVM 서버 추적 파일의 이름입니다. 읽기 전용 등록 정보입니다. 해당 값을 보고 애플리케이션에서 사용할 수 있지만 변경할 수는 없습니다.

-Dcom.ibm.cics.jvmserver.local.ccsid=

JCICS API를 사용할 때 파일 인코딩을 위한 코드 페이지를 지정합니다. 읽기 전용 등록 정보입니다. 해당 값을 보고 애플리케이션에서 사용할 수 있지만 변경할 수는 없습니다.

-Dcom.ibm.cics.jvmserver.override.ccsid=

이 등록 정보는 고급 사용자용입니다. JCICS API를 사용하는 경우 파일 인코딩을 위해 코드 페이지를 덮어씁니다. 기본적으로 JCICS는 **LOCALCCSID** 시스템 초기화 매개변수의 값을 파일 인코딩으로 사용합니다. 이 값을 덮어쓰도록 선택하면 이 등록 정보에서 코드 페이지를 설정하십시오. EBCDIC 코드 페이지를 사용하십시오. 애플리케이션이 새 코드 페이지와 일치하는지 확인해야 합니다. 일치하지 않으면 오류가 발생할 수 있습니다. 올바른 CCSID에 대한 자세한 정보는 참조에서 LOCALCCSID 시스템 초기화 매개변수 -> 시스템 정의의 내용을 참조하십시오.

-Dcom.ibm.cics.jvmserver.trace.format={FULL|SHORT|ABBREV}

추적 형식을 제어합니다. 목적에 따라 추적 형식을 달리할 수 있지만 IBM 서비스로 진단 정보를 전송할 때는 FULL로 설정해야 합니다.

-Dcom.ibm.cics.jvmserver.controller.timeout={*time*|60000ms}

이 등록 정보는 IBM 서비스 지침이 있는 경우에만 사용합니다. 언제라도 변경될 수 있습니다.

-Dcom.ibm.cics.jvmserver.threadjoin.timeout={*time*|30000ms}

이 등록 정보는 IBM 서비스 지침이 있는 경우에만 사용합니다. 언제라도 변경될 수 있습니다.

-Dcom.ibm.cics.jvmserver.trigger.timeout={*time*|500ms}

이 등록 정보는 IBM 서비스 지침이 있는 경우에만 사용합니다. 언제라도 변경될 수 있습니다.

-Dcom.ibm.cics.jvmserver.configroot=*zFS_directory*

JVM 서버의 JVM 프로파일과 같은 구성 파일이 있는 zFS 위치. 읽기 전용 등록 정보입니다. 해당 값을 보고 애플리케이션에서 사용할 수 있지만 변경할 수는 없습니다.

CICS 보안 토큰 서비스에 사용되는 JVM 서버 관련 등록 정보

-Dcom.ibm.cics.sts.config=*path*

STS 구성 파일의 위치와 이름입니다.

JVM 서버 관련 등록 정보

-Dconsole.encoding=

JVM 서버 출력 파일의 인코딩을 지정합니다.

-Dfile.encoding=

JVM의 문자 읽기 및 쓰기를 위한 코드 페이지를 지정합니다. 기본적으로 z/OS의 JVM은 EBCDIC 코드 페이지 IBM1047(또는 cp1047)을 사용합니다.

- OSGi용으로 구성된 프로파일에서 JVM이 지원하는 코드 페이지를 지정할 수 있습니다. JCICS는 문자 인코딩을 위해 CICS 영역의 로컬 CCSID를 사용하므로 CICS는 모든 코드 페이지를 허용합니다.
- Liberty JVM 서버에 대해 구성된 프로파일의 경우 제공된 기본값은 ISO-8859-1입니다. UTF-8을 사용할 수도 있습니다. 다른 코드 페이지는 파일 인코딩에 대해 지원되지 않습니다.
- Axis2용으로 구성된 프로파일에서는 EBCDIC 코드 페이지를 지정해야 합니다.

-Djava.security.manager={default| "" | |*other_security_manager*}

JVM에 사용할 Java 보안 관리자를 지정합니다. 기본 Java 보안 관리자를 사용하려면 이 시스템 등록 정보를 다음 형식 중 하나로 포함하십시오.

- `-Djava.security.manager=default`
- `-Djava.security.manager=""`
- `-Djava.security.manager=`

이들 명령문은 모두 기본 보안 관리자를 사용하도록 설정합니다. JVM 프로파일에 **-Djava.security.manager** 시스템 등록 정보를 포함하지 않으면 Java 보안을 사

용하지 않는 상태로 JVM이 실행됩니다. JVM에 대해 Java 보안을 사용 안함으로 설정하려면 이 시스템 등록 정보를 주석 처리하십시오.

-Djava.security.policy=

보안 관리자가 JVM에 대한 보안 정책을 결정하기 위해 사용하는 예비 정책 파일의 위치를 설명하십시오. 기본 정책 파일은 `/usr/lpp/java/J7.0_64/lib/security/java.policy`에서 JVM과 같이 제공됩니다. 여기서 `java/J7.0_64` 서브디렉토리 이름은 IBM 64-bit SDK for z/OS, Java Technology Edition을 설치할 때 기본값입니다. 기본 보안 관리자는 항상 이 기본 정책 파일을 사용하여 JVM에 대한 보안 정책을 판별하므로 **-Djava.security.policy** 시스템 등록 정보를 사용하여 기본 정책 파일뿐 아니라 보안 관리자가 고려할 정책 파일을 지정할 수 있습니다.

Java 보안이 활성화된 상태에서 CICS Java 애플리케이션이 실행될 수 있도록 설정하려면 최소한 CICS가 애플리케이션을 실행하기 위해 필요한 권한을 제공하는 특별 정책 파일을 지정하십시오.

Java 보안 사용에 대한 정보는 192 페이지의 『Java 보안 관리자 사용』의 내용을 참조하십시오.

제 4 장 JVM 서버에 애플리케이션 전개

Java 애플리케이션을 JVM 서버에 전개하려면 애플리케이션이 성공적으로 설치 및 실행될 수 있도록 올바르게 패키지화되어야 합니다. CICS Explorer SDK를 사용하여 애플리케이션을 패키지화하고 전개할 수 있습니다.

JVM 서버에서 실행되는 모든 Java 애플리케이션은 스레드세이프(threadsafe) 상태이고 OSGi 스펙을 준수해야 하므로 애플리케이션을 전개하기 전에 이 요구사항을 충족시키는지 확인하십시오. Java 애플리케이션은 다음과 같은 여러 가지 방법으로 전개할 수 있습니다.

- 애플리케이션의 OSGi 번들을 포함하는 하나 이상의 CICS 번들을 OSGi 프레임워크를 실행하는 JVM 서버에 전개합니다.
- 하나 이상의 WAR 파일을 포함하는 하나 이상의 CICS 번들을 Liberty JVM 서버에 전개합니다.
- 엔터프라이즈 번들 아카이브(EBA) 파일을 포함하는 하나 이상의 CICS 번들을 Liberty JVM 서버에 전개합니다.
- CICS 번들과 OSGi 번들을 구성하는 애플리케이션 번들을 플랫폼에 전개합니다.

  140 페이지의『JVM 서버에 OSGi 번들 전개』
  Java 애플리케이션을 JVM 서버에 전개하려면 대상 JVM 서버의 OSGi 프레임워크에 애플리케이션의 OSGi 번들을 설치해야 합니다.

  142 페이지의『Liberty JVM 서버에 CICS 번들의 웹 애플리케이션 전개』
  CICS 번들로 패키지화되는 Java 웹 애플리케이션을 Liberty JVM 서버에 전개할 수 있습니다.

  144 페이지의『Liberty JVM 서버에 직접 드롭인으로 웹 애플리케이션 전개』
  Liberty 프로파일 전개 모델에 따라 WAR 또는 EBA 파일을 Liberty JVM 서버로 내보내거나 전개할 수 있습니다.

  145 페이지의『JVM 서버에서 Java 애플리케이션 호출』
  JVM 서버에서 실행 중인 Java 애플리케이션을 호출하기 위한 여러 가지 방법이 있습니다. 사용되는 방법은 JVM 서버의 특성에 따라 다릅니다. 예를 들어, 클라이언트 브라우저로부터의 HTTP 요청은 Liberty JVM 서버의 웹 애플리케이션을 호출하는 데 사용됩니다. EXEC CICS LINK 또는 EXEC CICS START는 OSGi JVM 서버의 OSGi 번들 애플리케이션을 구동하는 데 사용됩니다. 반면에 벤더 인터페이스 DFHSJJI는 비OSGi JVM 서버의 클래스 경로 Java 함수에 링크하는 데 사용될 수 있습니다.

JVM 서버에 OSGi 번들 전개

Java 애플리케이션을 JVM 서버에 전개하려면 대상 JVM 서버의 OSGi 프레임워크에 애플리케이션의 OSGi 번들을 설치해야 합니다.

시작하기 전에

애플리케이션의 OSGi 번들을 포함하는 CICS 번들을 zFS에 전개해야 합니다. 대상 JVM 서버는 CICS 영역에서 사용하도록 설정해야 합니다.

이 태스크 정보

하나의 CICS 번들에 하나 이상의 OSGi 번들이 포함될 수 있습니다. CICS 번들은 전개 단위이므로 모든 OSGi 번들은 BUNDLE 자원의 일부로 함께 관리됩니다. OSGi 프레임워크는 또한 종속 항목 및 버전화 관리를 포함하여 OSGi 번들의 라이프 사이클을 관리합니다.

Java 애플리케이션 구성요소를 구성하는 모든 OSGi 번들이 동일한 CICS 번들에 전개되는지 확인하십시오. OSGi 번들 간에 종속 항목이 있는 경우 동일한 CICS 번들에 전개하십시오. CICS BUNDLE 자원을 설치하는 경우 CICS에서는 OSGi 번들 간의 모든 종속 항목이 해결됩니다.

공통 코드 라이브러리를 포함하는 OSGi 번들의 종속 항목이 있는 경우 라이브러리에 별도 CICS 번들을 작성하십시오. 이러한 경우 라이브러리를 포함하는 CICS BUNDLE 자원을 먼저 설치해야 합니다. 종속되는 CICS 번들보다 먼저 Java 애플리케이션을 설치하면 OSGi 프레임워크가 Java 애플리케이션의 종속 항목을 분석할 수 없습니다.

OSGi 번들을 포함하는 CICS 번들을 Liberty JVM 서버에 설치하지 마십시오. 이 구성은 지원되지 않습니다. 대신 OSGi 번들과 웹 애플리케이션을 엔터프라이즈 번들 아카이브(EBA)에 함께 패키지화하거나 WebSphere Liberty Profile 번들 저장소를 사용하여 Liberty JVM 서버의 모든 웹 애플리케이션이 OSGi 번들을 사용할 수 있도록 허용할 수 있습니다.

프로시저

1. zFS의 번들 디렉토리를 지정하는 BUNDLE 자원을 작성하십시오.
   a. CICS SM 퍼스펙티브의 CICS Explorer 메뉴 표시줄에서 **정의 > 번들 정의**를 눌러 번들 정의 보기를 여십시오.
   b. 보기 내 임의 위치에서 마우스 오른쪽 단추를 누르고 **새로 작성**을 눌러 새 번들 정의 마법사를 여십시오. 마법사 필드에 BUNDLE 자원에 대한 세부사항을 입력하십시오.
   c. BUNDLE 자원을 설치하십시오. 자원은 다음과 같이 사용 또는 사용 불가능 상태로 설치할 수 있습니다.

- 자원을 DISABLED 상태로 설치하는 경우 CICS는 OSGi 번들을 프레임워크에 설치하고 종속 항목을 분석하지만 번들을 시작하려고 시도하지는 않습니다.
- 자원을 ENABLED 상태로 설치하는 경우 CICS는 OSGi 번들을 설치하고 종속 항목을 분석한 후 OSGi 번들을 시작합니다. OSGi 번들에 지연 번들 활성기가 포함되면 다른 OSGi 번들이 처음 호출할 때까지 OSGi 프레임워크가 번들을 시작하려고 시도하지 않습니다.

2. 옵션: 자원이 아직 사용 상태가 아닌 경우 BUNDLE 자원을 사용하여 프레임워크에서 OSGi 번들을 시작하십시오.
3. CICS Explorer 메뉴 표시줄에서 **운영 > 번들**을 눌러 번들 보기를 여십시오. BUNDLE 자원 상태를 확인하십시오.
   - BUNDLE 자원이 ENABLED 상태인 경우 CICS가 모든 자원을 번들에 성공적으로 설치한 것입니다.
   - BUNDLE 자원이 DISABLED 상태인 경우 CICS가 하나 이상의 자원을 번들에 설치하지 못한 것입니다.

   BUNDLE 자원이 사용 상태로 설치하지 못한 경우 BUNDLE 자원의 번들 파트를 확인하십시오. 번들 파트가 UNUSABLE 상태인 경우 CICS가 OSGi 번들을 작성하지 못한 것입니다. 일반적으로 이 상태는 zFS에 CICS 번들 관련 문제가 있음을 나타냅니다. BUNDLE 자원을 버리고 문제를 수정한 다음 BUNDLE 자원을 다시 설치해야 합니다.
4. CICS Explorer 메뉴 표시줄에서 **운영 > Java > OSGi 번들**을 눌러 OSGi 번들 보기를 여십시오. OSGi 프레임워크에서 설치된 OSGi 번들과 서비스의 상태를 확인하십시오.
   - OSGi 번들이 STARTING 상태인 경우 번들 활성기가 호출되었지만 아직 리턴되지 않은 것입니다. OSGi 번들이 지연 활성화 정책을 갖는 경우 OSGi 프레임워크에서 호출될 때까지 번들이 이 상태를 유지합니다.
   - OSGi 번들과 OSGi 서비스가 활성화되면 Java 애플리케이션 준비가 완료된 것입니다.
   - OSGi 서비스가 비활성화된 경우 CICS가 해당 이름을 사용하는 OSGi 서비스가 OSGi 프레임워크에 이미 있음을 발견했을 수 있습니다.
   - BUNDLE 자원을 사용 불가능으로 설정하면 OSGi 번들이 RESOLVED 상태로 전환됩니다.
   - OSGi 번들이 INSTALLED 상태이면 시작되지 않았거나 OSGi 번들의 종속 항목을 분석할 수 없어 시작하지 못한 것입니다.

결과

BUNDLE을 사용하고 OSGi 번들이 OSGi 프레임워크에 설치되며 OSGi 서비스가 활성화됩니다. OSGi 번들은 프레임워크의 다른 번들이 사용할 수 있습니다.

다음에 수행할 작업

OSGi 프레임워크 외부의 다른 CICS 애플리케이션이 Java 애플리케이션을 사용할 수 있도록 허용할 수 있습니다(145 페이지의 『JVM 서버에서 Java 애플리케이션 호출』의 설명 참조). 애플리케이션을 갱신하거나 제거하려면 149 페이지의 제 5 장 『Java 애플리케이션 관리』의 내용을 참조하십시오.

Liberty JVM 서버에 CICS 번들의 웹 애플리케이션 전개

CICS 번들로 패키지화되는 Java 웹 애플리케이션을 Liberty JVM 서버에 전개할 수 있습니다.

시작하기 전에

WAR 파일 또는 EBA 파일 양식의 Java 웹 애플리케이션을 zFS의 CICS 번들로 전개해야 합니다. 대상 JVM 서버는 CICS 영역에서 사용하도록 설정해야 합니다.

Java 애플리케이션을 작성 및 다시 패키지화하는 방법에 대한 일반 정보는 34 페이지의 『CICS Explorer SDK를 사용하여 애플리케이션 개발』의 내용을 참조하십시오.

공통 코드 라이브러리를 포함하는 OSGi 번들에 대한 종속 항목이 있는 경우 Liberty 번들 저장소에 번들을 설치하십시오. 140 페이지의 『JVM 서버에 OSGi 번들 전개』의 내용을 참조하십시오.

이 태스크 정보

CICS 애플리케이션 모델은 Java 애플리케이션 구성요소를 CICS 번들에 패키지화하며 zFS에 전개합니다. CICS 번들을 설치하여 애플리케이션 구성요소의 라이프 사이클을 관리할 수 있습니다. Java 웹 애플리케이션에는 애플리케이션의 프리젠테이션 계층과 비즈니스 로직을 제공하는 하나 이상의 WAR 파일 또는 EBA 파일로 내보내는 OSGi 애플리케이션 프로젝트가 포함됩니다. OSGi 애플리케이션 프로젝트에는 프리젠테이션 계층을 제공하기 위한 웹 사용 OSGi 번들 프로젝트와 비즈니스 로직을 제공하는 추가적인 OSGi 번들 세트가 포함됩니다.

프로시저

1. zFS의 번들 디렉토리를 지정하는 BUNDLE 자원을 작성하십시오.
   a. CICS SM 퍼스펙티브의 CICS Explorer 메뉴 표시줄에서 **정의 > 번들 정의**를 눌러 번들 정의 보기를 여십시오.

b. 보기 내 임의 위치에서 마우스 오른쪽 단추를 누르고 **새로 작성**을 눌러 새 번들 정의 마법사를 여십시오. 마법사 필드에 BUNDLE 자원에 대한 세부사항을 입력하십시오.

c. BUNDLE 자원을 설치하십시오. 자원은 다음과 같이 사용 또는 사용 불가능 상태로 설치할 수 있습니다.

- 자원을 DISABLED 상태로 설치하는 경우 CICS가 Liberty 서버에 웹 애플리케이션을 설치하려고 시도하지 않습니다.
- 자원을 ENABLED 상태로 설치하는 경우 CICS가 웹 애플리케이션(WAR, EBA 파일)을 `${server.output.dir}/installedApps` 디렉토리에 설치하고 `${server.output.dir}/installedApps.xml`에 <application> 항목을 추가합니다.

2. 옵션: 자원이 아직 ENABLED 상태가 아닌 경우 BUNDLE 자원을 사용하여 Liberty 서버에서 웹 애플리케이션을 시작하십시오.
3. CICS Explorer 메뉴 표시줄에서 **운영 > 번들**을 눌러 번들 보기를 여십시오. BUNDLE 자원 상태를 확인하십시오.
   - BUNDLE 자원이 ENABLED 상태인 경우 CICS가 모든 자원을 번들에 성공적으로 설치한 것입니다.
   - BUNDLE 자원이 DISABLED 상태인 경우 CICS가 하나 이상의 자원을 번들에 설치하지 못한 것입니다.

   BUNDLE 자원이 사용 상태로 설치하지 못한 경우 BUNDLE 자원의 번들 파트를 확인하십시오. 번들 파트가 UNUSABLE 상태인 경우 CICS가 해당 번들 파트의 자원을 작성하지 못한 것입니다. 일반적으로 이 상태는 zFS에 CICS 번들 관련 문제가 있음을 나타냅니다. BUNDLE 자원을 버리고 문제를 수정한 다음 BUNDLE 자원을 다시 설치해야 합니다.
4. 옵션: 애플리케이션 트랜잭션에 대한 웹 애플리케이션 요청을 실행하려면 URIMAP 및 TRANSACTION 자원을 작성하면 됩니다. 애플리케이션에 대한 보안을 제어하려는 경우 URI 맵을 정의하는 것이 좋습니다. 해당 URI를 특정 트랜잭션에 맵핑하고 트랜잭션 보안을 사용할 수 있기 때문입니다. 일반적으로 이러한 자원은 CICS 번들의 일부로 작성되며 애플리케이션으로 관리됩니다. 그러나 필요한 경우 해당 자원을 개별적으로 정의할 수 있습니다.

   a. PROGRAM 속성을 DFHSJTHP로 설정하는 애플리케이션에 대해 TRANSACTION 자원을 작성하십시오. 이 CICS 프로그램은 Liberty JVM 서버로의 인바운드 웹 요청에 대한 보안 검사를 처리합니다. 원격 속성을 설정하는 경우 CICS에서는 무시됩니다. 트랜잭션은 항상 로컬 CICS 영역에서 연결되어야 하기 때문입니다.

   b. USAGE 유형이 JVMSERVER인 URIMAP 자원을 작성하십시오. TRANSACTION 속성을 애플리케이션 트랜잭션으로 설정하고 스키마를 HTTP 또는 HTTPS로 설정하십시오. **USERID** 속성을 사용하여 사용자 ID를 설정할 수

도 있습니다. 애플리케이션이 기본 인증을 사용하는 경우에는 이 값이 무시됩니다. 기본 인증을 사용하지 않거나 URI 맵에서 사용자 ID를 설정하는 경우에는 CICS 기본 사용자 ID로 작업이 실행됩니다.

결과

CICS 자원을 사용 가능하게 설정하면 웹 애플리케이션이 Liberty JVM 서버에 설치됩니다.

다음에 수행할 작업

웹 클라이언트를 통해 Java 애플리케이션을 사용할 수 있는지 테스트할 수 있습니다. 애플리케이션을 갱신하거나 제거하려면 Java 애플리케이션 관리를 참조하십시오.

Liberty JVM 서버에 직접 드롭인으로 웹 애플리케이션 전개

Liberty 프로파일 전개 모델에 따라 WAR 또는 EBA 파일을 Liberty JVM 서버로 내보내거나 전개할 수 있습니다.

시작하기 전에

JVM 서버는 Liberty 프로파일 기술을 사용하도록 구성해야 하며 CICS 영역에서 사용하도록 설정해야 합니다.

이 태스크 정보

Liberty 프로파일 전개 모델은 애플리케이션 서버에서 웹 구성요소를 빠르게 전개하고 실행할 수 있는 방법을 제공합니다. 웹 애플리케이션을 웹 아카이브(WAR) 또는 엔터프라이즈 번들 아카이브(EBA) 파일로서 Liberty JVM 서버의 `dropins` 디렉토리에 전개하여 이 모델을 따를 수 있습니다. 이 모델을 사용하는 경우 CICS가 JVM 서버에서 실행되는 웹 애플리케이션을 인식하지 못합니다. 즉, 오류가 발생할 수 있으므로 두 모델을 모두 사용하는 동일한 애플리케이션을 동일한 JVM 서버에 전개하지 마십시오. 이 방법으로 전개되는 애플리케이션은 보안과 같은 추가적인 서비스 품질(QoS)에 따른 이점이 없으므로 항상 트랜잭션 CJSA로 실행됩니다.

참고: CICS 자동 구성에서 제공되는 기본값을 승인하면 `dropins` 디렉토리가 자동으로 작성되지 않습니다.

프로시저

1. Eclipse 웹 프로젝트를 WAR 또는 EBA 파일로서 로컬 워크스테이션으로 내보내십시오.
2. CICS Liberty 구성의 `server.xml`에 다음 행을 추가하십시오.

```
<applicationMonitor dropins="dropins" dropinsEnabled="true" pollingRate="5s" updateTrigger="disabled"/>
```

3. FTP를 사용하여 2진 모드로 내보낸 파일을 Liberty 프로파일 서버의 dropins 디렉토리로 전송하십시오. 디렉토리 경로는 WLP_USER_DIR/servers/*server_name*/dropins입니다. 여기서 *server_name*은 **com.ibm.cics.jvmserver.wlp.server.name** 등록 정보의 값입니다.

결과

Liberty JVM 서버는 전개된 WAR 또는 EBA 파일을 자동으로 발견하고 설치합니다.

다음에 수행할 작업

웹 브라우저에서 웹 애플리케이션에 액세스하여 올바르게 실행되는지 확인하십시오. 애플리케이션 파일을 제거하려면 dropins 디렉토리에서 WAR 또는 EBA 파일을 삭제하십시오.

CICS Explorer SDK를 사용하여 애플리케이션을 패키지화하고 전개할 수 있습니다. 자세한 정보는 139 페이지의 제 4 장 『JVM 서버에 애플리케이션 전개』의 내용을 참조하십시오.

JVM 서버에서 Java 애플리케이션 호출

JVM 서버에서 실행 중인 Java 애플리케이션을 호출하기 위한 여러 가지 방법이 있습니다. 사용되는 방법은 JVM 서버의 특성에 따라 다릅니다. 예를 들어, 클라이언트 브라우저로부터의 HTTP 요청은 Liberty JVM 서버의 웹 애플리케이션을 호출하는 데 사용됩니다. EXEC CICS LINK 또는 EXEC CICS START는 OSGi JVM 서버의 OSGi 번들 애플리케이션을 구동하는 데 사용됩니다. 반면에 벤더 인터페이스 DFHSJJI는 비 OSGi JVM 서버의 클래스 경로 Java 함수에 링크하는 데 사용될 수 있습니다.

이 태스크 정보

특정 URL을 포함하는 HTTP 요청을 사용하여 Liberty JVM 서버에서 실행되는 웹 애플리케이션을 호출할 수 있습니다. EXEC CICS LINK 또는 EXEC CICS START에서 직접 웹 애플리케이션을 구동할 수 없습니다. Java로 정의된 프로그램에 대해 EXEC CICS LINK하거나 Java로 정의된 대상 프로그램이 있는 트랜잭션을 EXEC CICS START하여 OSGi JVM 서버에서 실행 중인 Java 애플리케이션을 호출할 수 있습니다. PROGRAM 정의는 JVMSERVER 및 호출하려는 CICS 생성 OSGi 서비스의 이름을 지정합니다. 이러한 '링크 가능' OSGi 서비스는 해당 Manifest에 CICS-MainClass 헤더를 포함하는 OSGi 번들을 설치할 때 CICS에서 작성됩니다. CICS-MainClass 헤더는 OSGi 번들에서 애플리케이션에 대한 시작점 역할을 하게 될 Java 클래스의 기본 메소드를 식별합니다.

OSGi 서비스는 OSGi 프레임워크에 등록되는 잘 정의된 인터페이스입니다. OSGi 번들과 원격 애플리케이션은 OSGi 서비스를 사용하여 OSGi 번들에 패키지화된 애플리

케이션 코드를 호출합니다. 하나의 OSGi 번들이 둘 이상의 OSGi 서비스를 내보낼 수 있습니다. 자세한 정보는 151 페이지의 『OSGi 번들 갱신』의 내용을 참조하십시오.

참고: Liberty JVM 서버에서 엔터프라이즈 번들 아카이브(EBA) 또는 EBA(웹 애플리케이션)를 정의하는 OSGi 애플리케이션 프로젝트 내에 OSGi 서비스를 작성할 수도 있습니다. 그러나 이러한 OSGi 서비스는 CICS PROGRAM 정의의 대상이 될 수 없으며 동일한 EBA의 외부에 패키지화된 구성요소에 표시되지 않습니다.

클래스 경로 기반의 JVM 서버에서 Java 함수를 호출하면 일반적으로 특정 JVM 서버 기능 중 일부로 수행됩니다(예: 일괄처리, Axis2 및 SAML). 이러한 기능을 위해 DFHSJJI 벤더 인터페이스가 제공됩니다.

프로시저

- WAB 파일을 포함하고 Liberty JVM 서버에서 실행되는 EBA 파일 또는 WAR 파일로 개발되는 웹 애플리케이션의 경우 URL을 사용하여 클라이언트 브라우저에서 애플리케이션을 호출하십시오.
- OSGi JVM 서버에 전개된 OSGi 번들의 경우 다음 단계를 따르십시오.
  1. OSGi 프레임워크에서 사용하려는 활성 OSGi 서비스의 기호 이름을 결정하십시오. 활성화된 OSGi 서비스를 나열하려면 CICS Explorer에서 **운영 > Java > OSGi 서비스**를 누르십시오.
  2. 다른 CICS 애플리케이션에 OSGi 서비스를 나타내는 PROGRAM 자원을 작성하십시오.
     - JVM 속성에서 `YES`를 지정하여 프로그램이 Java 프로그램임을 나타내십시오.
     - JVMCLASS 속성에서 OSGi 서비스의 기호 이름을 지정하십시오. 이 값은 대소문자를 구분합니다.
     - JVMSERVER 속성에서 OSGi 서비스가 실행 중인 JVMSERVER 자원의 이름을 지정하십시오.
  3. Java 애플리케이션은 다음 중 한 가지 방법으로 호출할 수 있습니다.
     - 트랜잭션 ID를 지정하는 3270 또는 **EXEC CICS START** 요청을 사용하십시오. OSGi 서비스를 위한 PROGRAM 자원을 정의하는 TRANSACTION 자원을 작성하십시오.
     - **EXEC CICS LINK** 요청, ECI 호출 또는 EXCI 호출을 사용하십시오. 요청을 코딩할 때 OSGi 서비스를 위한 PROGRAM 자원의 이름을 지정하십시오.
- Axis, 일괄처리 또는 SAML 기능의 경우 117 페이지의 『Axis2에 대한 JVM 서버 구성』, 및 의 내용을 참조하십시오.

결과

Java 애플리케이션을 다른 구성요소가 사용할 수 있는 정의를 작성했습니다. CICS가 대상 JVM 서버에서 요청을 수신하면 새 CICS Java 스레드에서 지정된 Java 클래스

또는 웹 애플리케이션을 호출합니다. 연관된 OSGi 서비스 또는 웹 애플리케이션이 등록되지 않거나 비활성 상태이면 호출 프로그램으로 오류가 리턴됩니다.

제 5 장 Java 애플리케이션 관리

Java 애플리케이션을 사용하도록 설정한 후 CICS 영역을 모니터하여 애플리케이션 성능을 이해할 수 있습니다. 환경을 조정하여 애플리케이션 성능을 최적화하라 수 있습니다.

이 태스크 정보

통계 및 모니터링을 사용하여 CICS 영역에서의 Java 애플리케이션 성능에 대한 정보를 수집할 수 있습니다. 특히 JVM의 성능을 확인할 수 있습니다. 정보를 수집한 후 JVM 또는 Language Environment 인클레이브를 변경하여 성능을 향상시킬 수 있습니다.

150 페이지의『JVM 서버에서 OSGi 번들 갱신』
OSGi 프레임워크에서 OSGi 번들을 갱신하는 프로세스는 번들 유형과 종속 항목에 따라 다릅니다. JVM 서버를 다시 시작하지 않고 애플리케이션에 맞게 OSGi 번들을 갱신할 수 있습니다. 그러나 미들웨어 번들을 갱신하려면 JVM 서버를 다시 시작해야 합니다.

155 페이지의『JVM 서버에서 OSGi 번들 제거』
JVM 서버에서 OSGi 번들을 제거하려면 CICS Explorer를 사용하여 BUNDLE 자원을 사용 안함으로 설정하고 버리십시오.

155 페이지의『JVM 서버로 애플리케이션 이동』
Java 애플리케이션을 풀 JVM에서 실행하는 경우 JVM 서버에서 실행하려면 애플리케이션을 이동해야 합니다. JVM 서버는 Java 애플리케이션에 대한 여러 요청을 동일한 JVM에서 처리할 수 있으므로 동일한 워크로드를 실행하는 데 필요한 JVM 수를 줄일 수 있습니다.

157 페이지의『JVM 서버의 스레드 한계 관리』
JVM 서버는 Java 애플리케이션을 실행하기 위해 사용할 수 있는 스레드 수가 제한적입니다. CICS 영역에도 스레드 수에 대한 한계가 있습니다. 각 스레드가 T8 TCB를 사용하기 때문입니다. CICS 통계를 사용하여 각 JVM 서버에서 실행되는 애플리케이션의 성능에 대해 영역 내 JVM 서버 수의 균형을 맞추어 스레드 한계를 조정할 수 있습니다.

158 페이지의『CICS 다시 시작 시 OSGi 번들 복구』
OSGi 번들을 포함하는 CICS 영역을 다시 시작하면 CICS가 BUNDLE 자원을 복구하고 JVM 서버의 프레임워크에 OSGi 번들을 설치합니다.

159 페이지의『JVM stdout 및 stderr 출력 경로 재지정을 위한 Java 클래스 작성』
JVM 프로파일에서 USEROUTPUTCLASS 선택항목을 사용하여 JVM에서 stdout 및

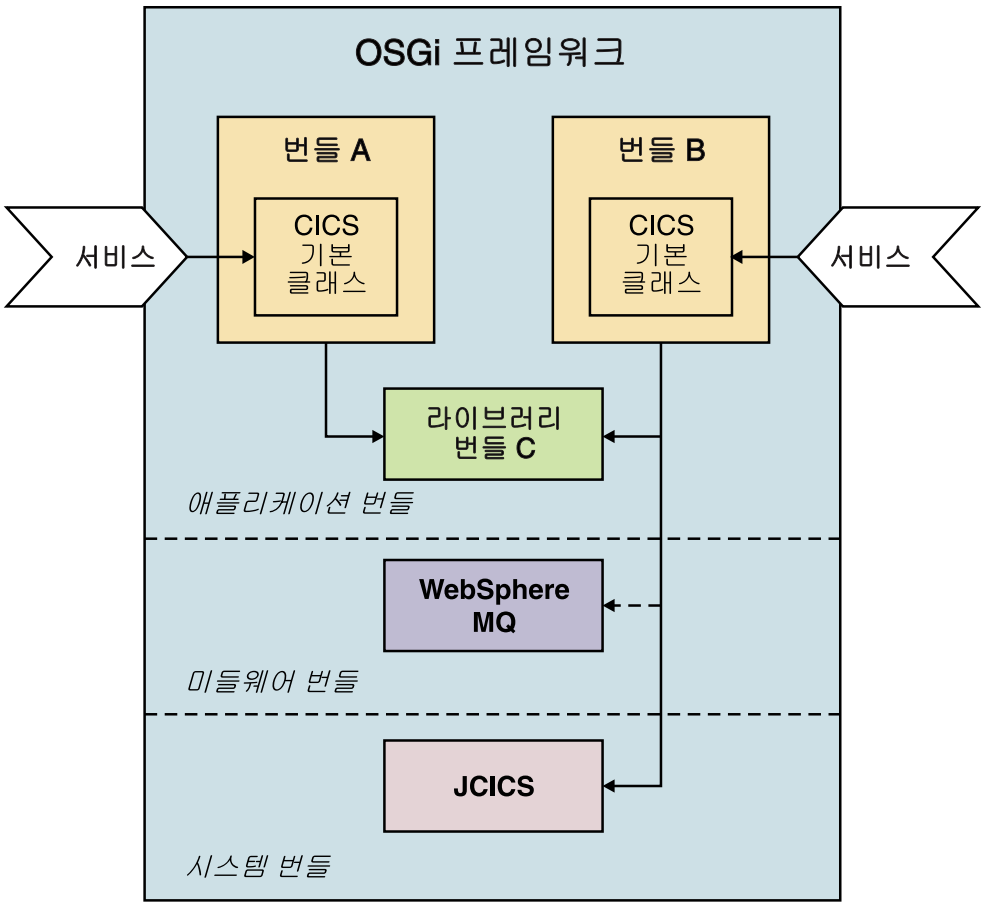
stderr 출력을 인터셉트하는 Java 클래스의 이름을 지정합니다. 이 클래스를 갱신하여 선택한 시간 소인 및 레코드 헤더를 지정하고 출력 경로를 재지정할 수 있습니다.

JVM 서버에서 OSGi 번들 갱신

OSGi 프레임워크에서 OSGi 번들을 갱신하는 프로세스는 번들 유형과 종속 항목에 따라 다릅니다. JVM 서버를 다시 시작하지 않고 애플리케이션에 맞게 OSGi 번들을 갱신할 수 있습니다. 그러나 미들웨어 번들을 갱신하려면 JVM 서버를 다시 시작해야 합니다.

이 태스크 정보

일반 JVM 서버의 경우 OSGi 프레임워크에는 다음 다이어그램에 표시된 OSGi 번들의 혼합 구성이 포함됩니다.



번들 A와 번들 B는 별도 CICS 번들에서 OSGi 번들로 패키지화되는 별도 Java 애플리케이션입니다. 두 애플리케이션 모두 번들 C에 패키지화되는 공통 라이브러리에 종속

됩니다. 번들 C는 별도로 관리 및 갱신됩니다. 또한 번들 B는 WebSphere MQ 미들웨어 번들과 JCICS 시스템 번들에 종속됩니다.

번들 A, B 모두 프레임워크의 다른 번들에 영향을 주지 않고 독립적으로 갱신될 수 있습니다. 그러나 번들 C를 갱신하면 종속되는 두 번들에 모두 영향을 줄 수 있습니다. 번들 C의 내보낸 패키지는 OSGi 프레임워크에서 메모리에 남아 있으므로 번들 C의 변경사항을 적용하려면 프레임워크에서 번들 A, B도 갱신되어야 합니다.

미들웨어 번들에는 프레임워크 서비스가 포함되며 JVM 서버의 라이프 사이클로 관리됩니다. 예를 들어, 프레임워크에서 한 번 로드하려는 고유 코드가 있거나 WebSphere MQ와 같은 다른 제품에 액세스하기 위해 드라이버를 추가할 수 있습니다.

시스템 번들은 CICS가 OSGi 프레임워크와의 상호작용을 관리하기 위해 제공됩니다. IBM은 이러한 번들을 제품의 일부로 지원합니다. 시스템 번들의 한 예로 CICS 서비스에 액세스하기 위해 JCICS API의 대부분을 제공하는 `com.ibm.cics.server.jar` 파일이 있습니다.

『OSGi 번들 갱신』
Java 개발자가 CICS 번들의 갱신된 버전을 제공하는 경우 CICS 번들을 완전히 바꾸거나 새 버전을 단계적으로 진행한 다음 이전 버전을 제거할 수 있습니다.

152 페이지의『공통 라이브러리를 포함하는 번들 갱신』
다른 OSGi 번들이 사용할 공통 라이브러리를 포함하는 OSGi 번들은 특정 순서로 갱신해야 합니다.

154 페이지의『OSGi 미들웨어 번들 갱신』
OSGi 프레임워크에서 실행되는 미들웨어 번들을 갱신하려면 JVM 서버를 중지한 후 다시 시작해야 합니다.

OSGi 번들 갱신

Java 개발자가 CICS 번들의 갱신된 버전을 제공하는 경우 CICS 번들을 완전히 바꾸거나 새 버전을 단계적으로 진행한 다음 이전 버전을 제거할 수 있습니다.

시작하기 전에

OSGi 번들의 새 버전을 포함하는 갱신된 CICS 번들이 zFS에 있어야 합니다.

프로시저

- OSGi JVM 서버에서 기존 OSGi 번들을 바꾸려면 다음을 수행하십시오.
  1. 갱신하려는 CICS 번들에 대해 BUNDLE 자원을 사용 안함으로 설정하고 버리십시오. 해당 CICS 번들에 포함되는 OSGi 번들과 서비스는 OSGi 프레임워크에서 제거됩니다.
  2. 옵션: 갱신된 CICS 번들이 다른 디렉토리에 전개되는 경우에는 BUNDLE 자원 정의를 편집하십시오.

3. BUNDLE 자원 정의를 설치하여 변경된 OSGi 번들을 선택하십시오. CICS 번들의 OSGi 번들과 서비스는 OSGi 프레임워크에 설치됩니다.
4. CICS Explorer의 **운영** > **Java** 보기에서 OSGi 번들과 서비스의 상태를 확인하십시오.

- OSGi 서버에서 새 버전을 단계적으로 진행하고 두 번들을 프레임워크에서 동시에 실행하려면 OSGi 서비스가 별명을 지정해야 합니다. 별명을 지정하지 않으면, 서비스가 이미 실행 중인 서비스의 복제본으로 간주되므로 프레임워크에서 비활성으로 나열됩니다.
  1. BUNDLE 자원을 작성하여 변경된 CICS 번들을 선택하십시오. CICS 번들의 OSGi 번들과 서비스는 OSGi 프레임워크에 설치됩니다. 번들 Manifest에 별명이 지정되지 않은 경우 OSGi 서비스는 비활성 상태입니다.
  2. CICS Explorer의 **운영** > **Java** 보기에서 OSGi 번들과 서비스의 상태를 확인하십시오. OSGi 번들 보기에 OSGi 번들의 두 가지 버전이 나열됩니다. 별명이 지정되지 않은 경우 번들의 OSGi 서비스는 비활성 상태입니다. 별명이 지정되면 두 가지 OSGi 서비스가 모두 활성화됩니다.
  3. OSGi 번들의 이전 버전을 가리키는 BUNDLE 자원을 사용 안함으로 설정하십시오. CICS는 번들과 연관된 OSGi 서비스를 제거하고 OSGi 번들을 해결된 상태로 설정합니다. 결과적으로 변경된 OSGi 번들에 대한 OSGi 서비스가 비활성 상태에서 활성 상태로 이동합니다.
  4. OSGi 서비스에 대한 별명이 있는 경우 PROGRAM 자원에 별명을 지정하여 JVM 서버 외부에서 갱신된 애플리케이션을 호출할 수 있습니다.

결과

OSGi JVM 서버를 사용하는 경우 OSGi 번들의 기호 버전이 증가하여 Java 코드가 갱신되었음을 나타냅니다. 갱신된 OSGi 번들은 OSGi 프레임워크에서 사용할 수 있으며 JVM 서버 외부에서 호출될 수 있습니다.

Liberty JVM 서버에서 EBA를 사용하고 있는 경우 번들이 사용 가능하게 되면 OSGi 패키지가 새로 고쳐지고 EBA 내의 웹 애플리케이션에서 사용할 수 있게 됩니다.

공통 라이브러리를 포함하는 번들 갱신

다른 OSGi 번들이 사용할 공통 라이브러리를 포함하는 OSGi 번들은 특정 순서로 갱신해야 합니다.

시작하기 전에

OSGi 번들의 새 버전을 포함하는 갱신된 CICS 번들이 zFS에 있어야 합니다. 공통 라이브러리를 별도의 CICS 번들에서 관리하는 경우 이러한 라이브러리의 라이프 사이클을 종속되는 애플리케이션과 별도로 관리할 수 있습니다.

이 태스크 정보

일반적으로 OSGi 번들은 다른 OSGi 번들에 대한 종속 항목에서 지원되는 버전의 범위를 지정합니다. 범위를 사용하면 프레임워크에서 호환되는 변경사항을 보다 유연하게 작성할 수 있습니다. 공통 라이브러리를 포함하는 번들을 갱신하는 경우 OSGi 번들의 버전 번호가 증가합니다. 그러나 실행 애플리케이션은 종속 항목을 충족하는 번들 버전을 이미 사용하고 있습니다. 라이브러리의 최신 버전을 선택하려면 애플리케이션에 맞게 OSGi 번들을 새로 고쳐야 합니다. 따라서 특정 애플리케이션을 갱신하여 라이브러리의 다른 버전을 사용하고 다른 애플리케이션은 이전 버전에서 계속 실행할 수 있습니다.

공통 라이브러리를 포함하는 OSGi 번들을 갱신하는 경우 CICS 번들을 완전히 바꿀 수 있습니다. 그러나 라이브러리에서 클래스가 로드되지 않은 경우 종속 번들이 오류를 수신할 수 있습니다. 라이브러리의 새 버전을 단계적으로 진행하고 프레임워크에서 원래 버전과 함께 실행할 수 있습니다. OSGi 번들의 버전 번호가 여러 가지인 경우 OSGi 프레임워크가 두 번들을 동시에 실행할 수 있습니다.

프로시저

1. OSGi 번들의 새 버전을 가리키는 BUNDLE 자원을 작성하십시오. CICS는 OSGi 프레임워크에서 OSGi 번들의 새 버전을 작성합니다. 기존 OSGi 번들은 라이브러리의 이전 버전을 계속 사용합니다.
2. CICS Explorer에서 OSGi 번들 보기를 확인하십시오. 프레임워크에서 실행되는 다른 버전과 함께 동일한 OSGi 번들 기호 이름에 대한 두 개 항목이 목록에 표시됩니다.
3. 종속 Java 애플리케이션에서 라이브러리의 최신 버전을 선택하려면 다음을 수행하십시오.
   a. Java 애플리케이션에 대해 BUNDLE 자원을 사용 안함으로 설정하고 버리십시오. 또는 Java 개발자에게 OSGi 번들에 대한 버전 정보를 갱신하고 CICS 번들의 새 버전을 전개하도록 요청하여 애플리케이션의 사용 가능성을 유지할 수 있습니다.
   b. BUNDLE 자원을 설치하십시오. OSGi 번들이 프레임워크에서 로드되면 공통 라이브러리의 최신 버전을 선택합니다.
4. CICS Explorer의 번들 보기에서 BUNDLE 자원의 상태를 확인하십시오.

결과

공통 라이브러리를 포함하는 OSGi 번들을 갱신했으며 라이브러리의 최신 버전을 사용하도록 Java 애플리케이션을 갱신했습니다.

OSGi 미들웨어 번들 갱신

OSGi 프레임워크에서 실행되는 미들웨어 번들을 갱신하려면 JVM 서버를 중지한 후 다시 시작해야 합니다.

이 태스크 정보

OSGi 미들웨어 번들은 JVM 서버를 초기화하는 중 OSGi 프레임워크에 설치됩니다. 예를 들어, 패치를 적용하거나 새 버전을 사용하기 위해 미들웨어 번들을 갱신하려면 JVM 서버를 중지한 후 다시 시작하여 변경된 번들을 선택해야 합니다.

CICS Explorer를 사용하여 JVM 서버의 라이프 사이클을 관리하고 JVM 프로파일을 편집할 수 있습니다.

프로시저

1. 미들웨어 번들의 새 버전이 CICS에 읽기 및 실행 액세스 권한이 있는 zFS의 디렉토리에 있는지 확인하십시오. CICS에는 또한 파일에 대한 읽기 액세스 권한이 필요합니다.
2. zFS 디렉토리 또는 파일 이름이 JVM 프로파일에 지정된 값과 다른 경우 JVM 서버의 JVM 프로파일에서 `OSGI_BUNDLES` 선택항목을 편집하십시오.
   a. CICS Explorer에서 JVM 서버 보기를 열어 zFS의 JVM 프로파일 이름과 위치를 확인하십시오. JVMSERVER 자원을 보려면 선택된 CICSplex 또는 영역과 연결되어야 합니다.
   b. z/OS UNIX 파일 보기를 열고 JVM 프로파일이 있는 디렉토리를 찾아보십시오.
   c. JVM 프로파일을 편집하여 `OSGI_BUNDLES` 선택항목을 갱신하십시오.
3. JVMSERVER 자원을 사용 안함으로 설정하여 JVM 서버를 종료하십시오. JVMSERVER를 사용 안함으로 설정하면 해당 JVM에 설치된 OSGi 번들을 포함하는 BUNDLE 자원도 모두 사용하지 않습니다.
4. JVMSERVER 자원을 사용하여 갱신된 JVM 프로파일로 JVM 서버를 시작하십시오. JVM 서버가 시동되고 OSGi 프레임워크에서 미들웨어 번들의 새 버전을 설치합니다. CICS는 또한 사용 불가능한 BUNDLE 자원을 사용 가능하게 하고 갱신된 프레임워크에 OSGi 번들과 서비스를 설치합니다.

결과

OSGi 프레임워크에는 갱신된 미들웨이 번들과, JVM 서버를 종료하기 전에 설치된 Java 애플리케이션의 OSGi 번들 및 서비스가 포함됩니다.

JVM 서버에서 OSGi 번들 제거

JVM 서버에서 OSGi 번들을 제거하려면 CICS Explorer를 사용하여 BUNDLE 자원을 사용 안함으로 설정하고 버리십시오.

이 태스크 정보

BUNDLE 자원은 CICS 번들에서 정의되는 OSGi 서비스와 OSGi 번들의 콜렉션에 대한 라이프 사이클 관리를 제공합니다. OSGi 프레임워크에서 OSGi 번들을 제거해도 설치된 다른 OSGi 번들 및 서비스의 상태에 자동으로 영향을 주지 않습니다. 다른 번들의 전제조건인 번들을 제거하면 해당 번들을 명시적으로 새로 고칠 때까지 종속된 번들의 상태가 변경되지 않습니다. 단, 싱글톤 번들을 사용하는 경우는 예외입니다. 다른 번들이 종속되어 있는 싱글톤 번들을 설치 제거하면 종속 번들이 설치 제거된 번들의 서비스를 사용할 수 없습니다. CICS BUNDLE 자원의 보고된 상태는 OSGi 번들의 상태를 정확히 반영하지 못할 수 있습니다.

프로시저

1. **운영 > Java > OSGi 번들**을 눌러 OSGi 번들을 포함하는 BUNDLE 자원을 찾으십시오.
2. **운영 > 번들**을 눌러 BUNDLE 자원을 사용 불가능으로 설정하십시오. CICS는 CICS 번들에 정의된 각 자원을 사용 안함으로 설정합니다. OSGi 번들 및 서비스의 경우 CICS는 JVM 서버의 OSGi 프레임워크로 요청을 전송하여 OSGi 서비스를 등록 취소하고 OSGi 번들을 해결됨 상태로 이동합니다. 모든 인플라이트 트랜잭션이 완료되지만 CICS 애플리케이션에서 OSGi 서비스로의 새 링크는 오류와 함께 리턴됩니다.
3. BUNDLE 자원을 버리십시오. CICS가 JVM 서버에서 OSGi 번들을 제거하는 요청을 OSGi 프레임워크로 전송합니다.

결과

OSGi 프레임워크에서 OSGi 번들과 서비스를 제거했습니다.

다음에 수행할 작업

더 이상 OSGi 프레임워크에 없는 OSGi 서비스를 가리키는 PROGRAM 자원이 있는 경우 PROGRAM 자원을 사용하지 않고 버릴 수 있습니다.

JVM 서버로 애플리케이션 이동

Java 애플리케이션을 풀 JVM에서 실행하는 경우 JVM 서버에서 실행하려면 애플리케이션을 이동해야 합니다. JVM 서버는 Java 애플리케이션에 대한 여러 요청을 동일한 JVM에서 처리할 수 있으므로 동일한 워크로드를 실행하는 데 필요한 JVM 수를 줄일 수 있습니다.

시작하기 전에

애플리케이션이 스레드세이프(threadsafe) 상태이고 하나 이상의 OSGi 번들로 패키지화되는지 확인하십시오. OSGi 번들은 CICS 번들에서 zFS에 전개되어야 하며 올바른 대상 JVMSERVER 자원을 지정해야 합니다.

Java 개발자는 CICS Explorer SDK를 사용하여 OSGi로 Java 애플리케이션을 다시 패키지화할 수 있습니다. 써드 파티 JAR을 사용하는 애플리케이션을 이주하는 방법에 대한 자세한 정보는 의 내용을 참조하십시오.

이 태스크 정보

기존 JVM 서버를 사용하거나 애플리케이션 전용 JVM 서버를 작성할 수 있습니다. 이미 스레드 한계가 높고 사용량이 많은 JVM 서버로는 애플리케이션을 이동하지 마십시오. JVM 서버에서 잠금 경합을 야기할 수 있기 때문입니다.

프로시저

1. JVM 서버를 작성 또는 갱신하십시오.
   - JVM 서버를 작성하려면 101 페이지의 『JVM 서버 설정』의 내용을 참조하십시오. 풀 JVM에 대한 JVM 프로파일의 많은 설정은 JVM 서버에 적용되지 않습니다. 풀 JVM 프로파일에서 DFHOSGI 프로파일에 복사할 유일한 선택항목은 `LIBPATH_SUFFIX` 선택항목입니다.
   - 기존 JVM 서버를 사용하는 경우 추가 애플리케이션을 처리하거나 JVM 서버 프로파일에서 선택항목을 갱신하기 위해 JVMSERVER 자원에서 THREADLIMIT 속성을 늘려야 할 수 있습니다. JVM 프로파일을 변경하는 경우 JVM 서버를 다시 시작하여 변경사항을 선택하십시오.
2. zFS에서 전개된 번들을 가리키는 BUNDLE 자원을 작성하십시오. BUNDLE 자원을 설치할 때 CICS는 JVM 서버의 OSGi 프레임워크에 OSGi 번들을 로드합니다. OSGi 프레임워크는 OSGi 번들을 해결하고 OSGi 서비스를 등록합니다. CICS Explorer를 사용하여 BUNDLE 자원을 사용할 수 있는지 확인하십시오. OSGi 번들 및 OSGi 서비스 보기를 사용하여 OSGi 번들과 서비스의 상태를 확인할 수도 있습니다.
3. 애플리케이션에 대한 PROGRAM 자원을 갱신하십시오.
   a. EXECKEY 속성이 CICS로 설정되었는지 확인하십시오. 모든 JVM 서버 작업은 CICS 키에서 실행됩니다.
   b. JVM 프로파일 이름을 제거하고 JVMSERVER 자원의 이름을 입력하십시오.
   c. JVMCLASS 속성이 Java 애플리케이션의 OSGi 서비스와 일치하는지 확인하십시오.
   d. 애플리케이션에 대한 PROGRAM 자원을 다시 설치하십시오.

PROGRAM 자원은 OSGi 서비스를 사용하여 JVM 서버 외부의 다른 CICS 애플리케이션이 OSGi 번들을 사용할 수 있도록 지정할 수 있습니다.

결과

Java 애플리케이션은 호출될 때 JVM 서버에서 실행됩니다.

다음에 수행할 작업

CICS Explorer 및 CICS 통계에서 JVM 서버 보기를 사용하여 JVM 서버를 모니터할 수 있습니다. 성능이 최적 상태가 아니면 스레드 한계를 조정하십시오.

JVM 서버의 스레드 한계 관리

JVM 서버는 Java 애플리케이션을 실행하기 위해 사용할 수 있는 스레드 수가 제한적입니다. CICS 영역에도 스레드 수에 대한 한계가 있습니다. 각 스레드가 T8 TCB를 사용하기 때문입니다. CICS 통계를 사용하여 각 JVM 서버에서 실행되는 애플리케이션의 성능에 대해 영역 내 JVM 서버 수의 균형을 맞추어 스레드 한계를 조정할 수 있습니다.

이 태스크 정보

각 JVM 서버는 Java 애플리케이션을 실행하기 위해 최대 256개 스레드를 가질 수 있습니다. CICS 영역에서는 최대 2000개 스레드를 가질 수 있습니다. CICS 영역에서 실행되는 JVM 서버가 많은 경우(예를 들어, 여덟 개 이상) 모든 JVM 서버에 대해 최대값을 설정할 수 없습니다. 각 JVM 서버의 스레드 한계를 조정하여 Java 애플리케이션의 성능에 대해 CICS 영역 내 JVM 서버 수의 균형을 맞출 수 있습니다.

스레드 한계는 JVMSERVER 자원에서 설정되므로 초기값을 설정하고 CICS 통계를 사용하여 Java 워크로드를 테스트할 때 스레드 수를 조정하십시오.

프로시저

1. JVMSERVER 자원을 사용 가능하게 설정하고 Java 애플리케이션 워크로드를 실행하십시오.
2. 적절한 통계 간격을 사용하여 JVMSERVER 자원 통계를 수집하십시오. CICS Explorer에서 **운영 > Java > JVM 서버** 보기를 사용하거나 DFH0STAT 통계 프로그램을 사용할 수 있습니다.
3. 태스크가 스레드를 대기하는 횟수와 시간을 확인하십시오. 『JVMSERVER 스레드 한계 대기 횟수』 및 『JVMSERVER 스레드 한계 대기 시간』 필드에 이 정보가 포함됩니다.

- 이러한 필드의 값이 크고 JVMTHRD 대기로 많은 태스크가 일시중단되는 경우에는 JVM 서버에 사용 가능한 스레드가 충분하지 않기 때문입니다. 스레드 수를 늘리면 프로세서 사용이 증가할 수 있으므로 사용 가능한 MVS 자원이 충분한지 확인하십시오.
- 이러한 필드의 값이 작고 최대 태스크 수가 사용 가능한 최대 스레드 수보다 적은 경우에는 스레드 한계를 줄여 다른 JVM 서버를 위한 스레드를 확보할 수 있습니다.

4. MVS 자원의 사용가능성을 확인하려면 디스패처 TCB 풀 및 TCB 모드 통계를 사용하여 CICS 영역에서의 T8 TCB 사용량을 평가하십시오. JVM 서버의 각 스레드는 T8 TCB를 사용하며 한 영역에서 2000으로 제한됩니다. 모든 TCB가 THRD TCB 풀에 있더라도 T8 TCB는 JVM 서버 간에 공유할 수 없습니다. 대기 TCB 수와 프로세서 사용이 적은 경우, 사용 가능한 MVS 자원이 충분함을 나타냅니다.
5. JVM 서버에서 실행될 수 있는 스레드 수를 조정하려면 JVMSERVER 자원의 THREADLIMIT 값을 변경하십시오.
6. Java 애플리케이션 워크로드를 다시 실행하고 통계를 사용하여 대기 태스크 수가 감소했는지 확인하십시오.

다음에 수행할 작업

JVM 서버 성능을 조정하려면 169 페이지의 『JVM 서버 성능 개선』의 내용을 참조하십시오.

CICS 다시 시작 시 OSGi 번들 복구

OSGi 번들을 포함하는 CICS 영역을 다시 시작하면 CICS가 BUNDLE 자원을 복구하고 JVM 서버의 프레임워크에 OSGi 번들을 설치합니다.

CICS 번들에서 패키지화되는 OSGi 번들은 CSD에 저장되지 않습니다. BUNDLE 자원 자체는 카탈로그에 저장되므로 CICS 영역을 다시 시작하면 BUNDLE 자원이 복원될 때 OSGi 번들이 동적으로 다시 작성됩니다.

CICS의 콜드, 웜 또는 긴급 재시작이 수행되면 JVM 서버가 BUNDLE 자원 복구에 비동기식으로 시작됩니다. CICS가 다시 시작될 때 OSGi 번들을 성공적으로 복원하려면 JVM 서버를 완전히 사용할 수 있어야 합니다. 따라서 BUNDLE 자원은 CICS 시동 마지막 단계에서 복구되지만 OSGi 번들은 JVM 서버가 시동을 완료한 경우에만 설치됩니다.

BUNDLE 자원과 자원에 포함된 OSGi 번들은 CICS 번들과 OSGi 번들 간의 종속 항목이 프레임워크에서 분석될 수 있도록 올바른 순서로 설치됩니다. CICS가 OSGi 번

들을 설치하지 못하면 BUNDLE 자원이 사용 불가능 상태로 설치됩니다. IBM CICS Explorer를 사용하여 BUNDLE 자원, OSGi 번들, OSGi 서비스의 상태를 볼 수 있습니다.

JVM stdout 및 stderr 출력 경로 재지정을 위한 Java 클래스 작성

JVM 프로파일에서 USEROUTPUTCLASS 선택항목을 사용하여 JVM에서 stdout 및 stderr 출력을 인터셉트하는 Java 클래스의 이름을 지정합니다. 이 클래스를 갱신하여 선택한 시간 소인 및 레코드 헤더를 지정하고 출력 경로를 재지정할 수 있습니다.

CICS는 이 목적으로 사용할 수 있는 샘플 Java 클래스, com.ibm.cics.samples.SJMergedStream 및 com.ibm.cics.samples.SJTaskStream을 제공합니다. 이러한 두 클래스 모두에 대한 샘플 소스가 /usr/lpp/cicsts/cicsts52/samples/com.ibm.cics.samples 디렉토리에 제공됩니다. /usr/lpp/cicsts/cicsts52 디렉토리는 z/OS UNIX에서 CICS 파일의 설치 디렉토리입니다. 이 디렉토리는 DFHISTAR 설치 작업에서 **USSDIR** 매개변수로 지정됩니다. 샘플 클래스는 클래스 파일, com.ibm.cics.samples.jar(/usr/lpp/cicsts/cicsts52/lib 디렉토리에 있음)로도 제공됩니다. 이러한 클래스를 수정하거나 샘플을 기반으로 직접 클래스를 작성할 수 있습니다.

201 페이지의 『JVM stdout, stderr, JVMTRACE 및 덤프 출력의 위치 제어』에서 다음 정보를 제공합니다.

- USEROUTPUTCLASS 선택항목으로 이름이 지정된 클래스로 인터셉트되지 않는 JVM의 출력 유형. 사용하는 클래스는 인터셉트할 수 있는 모든 출력 유형을 처리할 수 있어야 합니다.
- 제공된 샘플 클래스의 작동. com.ibm.cics.samples.SJMergedStream 클래스는 JVM 출력과 오류 메시지를 위한 두 개의 병합 로그 파일을 작성합니다. 각 레코드의 헤더에는 APPLID, 날짜, 시간, 트랜잭션 ID, 태스크 번호, 프로그램 이름이 포함됩니다. 사용 가능한 경우 트랜지언트 데이터 큐를 사용하여 로그 파일이 작성됩니다. 트랜지언트 데이터 큐가 없거나 Java 애플리케이션이 사용할 수 없는 경우 z/OS UNIX 파일이 작성됩니다. com.ibm.cics.samples.SJTaskStream 클래스는 단일 태스크의 출력 방향을 z/OS UNIX 파일로 지정하고 시간소인과 헤더를 추가하여 단일 태스크에 고유한 출력 스트림을 제공합니다.

JVM 서버가 출력 방향 전환 클래스를 사용하려면 출력 방향 전환 클래스를 포함하는 OSGi 번들을 작성해야 합니다. 번들 활성기가 클래스 인스턴스를 프레임워크에서 서비스로 등록하고 com.ibm.cics.server.outputredirectionplugin.name=*class_name* 등록 정보를 설정하는지 확인해야 합니다.
com.ibm.cics.server.Constants.CICS_USER_OUTPUT_CLASSNAME_PROPERTY 상수를 사용하여 등록 정보 이름을 가져올 수 있습니다. 다음 코드 발췌본은 번들 활성기에서 서비스를 등록하는 방법을 보여줍니다.

```
Properties serviceProperties = new Properties();
    serviceProperties.put(Constants.CICS_USER_OUTPUT_CLASSNAME_PROPERTY, MyOwnStreamPlugin.class.getName());
    context.registerService(OutputRedirectionPlugin.class.getName(), new MyOwnStreamPlugin(), serviceProperties);
```

JVM 프로파일에서 OSGI_BUNDLES 선택항목에 OSGi 번들을 추가하거나 첫 번째 태스크가 실행될 때 번들이 프레임워크에서 설치되었는지 확인할 수 있습니다. 어떤 방법을 사용하더라도 USEROUTPUTCLASS 선택항목에서 클래스를 지정해야 합니다.

자체 클래스를 작성하려면 다음 정보가 필요합니다.

- OutputRedirectionPlugin 인터페이스
- 가능한 출력 대상
- 출력 방향 전환 오류 및 내부 오류 처리

  『출력 방향 전환 인터페이스』
  CICS는 JVM에서 stdout 및 stderr 출력을 인터셉트하는 클래스로 구현될 수 있는 com.ibm.cics.server.jar의 com.ibm.cics.server.OutputRedirectionPlugin 인터페이스를 제공합니다. 제공된 샘플은 이 인터페이스를 구현합니다.

  161 페이지의『가능한 출력 대상』
  CICS에서 제공하는 샘플 클래스는 JVM의 출력 방향을 CICS 영역의 고유한 디렉토리로 지정합니다. 디렉토리 이름은 CICS 영역과 연관된 applid를 사용하여 작성됩니다. 원하는 경우 고유한 클래스를 작성할 때 여러 CICS 영역의 출력을 동일한 z/OS UNIX 디렉토리 또는 파일로 전송할 수 있습니다.

  162 페이지의『출력 방향 전환 오류 및 내부 오류 처리』
  클래스가 CICS 기능을 사용하여 출력 방향을 전환하는 경우 해당 기능 사용에 따른 오류를 처리하기 위해 적절한 예외 처리가 포함되어야 합니다.

출력 방향 전환 인터페이스

CICS는 JVM에서 stdout 및 stderr 출력을 인터셉트하는 클래스로 구현될 수 있는 com.ibm.cics.server.jar의 com.ibm.cics.server.OutputRedirectionPlugin 인터페이스를 제공합니다. 제공된 샘플은 이 인터페이스를 구현합니다.

다음 샘플 클래스가 제공됩니다.

- 이 인터페이스를 구현하는 com.ibm.cics.samples.SJStream 수퍼클래스
- JVM 프로파일에서 이름이 지정된 클래스인 com.ibm.cics.samples.SJMergedStream 및 com.ibm.cics.samples.SJTaskStream 서브클래스

샘플 클래스와 마찬가지로 클래스가 OutputRedirectionPlugin 인터페이스를 직접 구현하거나 인터페이스를 구현하는 클래스를 확장하는지 확인하십시오.
com.ibm.cics.samples.SJStream 수퍼클래스에서 상속하거나 동일한 인터페이스로 클래스 구조를 구현할 수 있습니다. 사용하는 방법에 관계 없이 클래스는 java.io.OutputStream을 확장해야 합니다.

initRedirect() 메소드는 하나 이상의 출력 방향 전환 클래스가 사용하는 매개변수 세트를 받습니다. 다음 코드는 인터페이스를 보여줍니다.

```
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
                              );
  }
```

com.ibm.cics.samples.SJStream 수퍼클래스에는 com.ibm.cics.samples.SJMergedStream 및 com.ibm.cics.samples.SJTaskStream의 공통 구성요소가 포함됩니다. 이 클래스에는 `false`를 리턴하는 initRedirect() 메소드가 포함되므로 이 메소드를 서브클래스의 다른 메소드로 덮어쓰지 않는 한 출력 방향 전환을 효과적으로 사용 불가능하게 합니다. writeRecord() 메소드는 구현하지 않으며 그러한 메소드는 서브클래스가 출력 방향 전환 프로세스를 제어하기 위해 제공해야 합니다. 이 메소드는 고유한 클래스 구조에서 사용할 수 있습니다. 출력 방향 전환 초기화는 또한 initRedirect() 메소드가 아닌 생성자를 사용하여 수행할 수 있습니다.

inPS 매개변수에는 원래 System.out 인쇄 스트림 또는 JVM의 원래 System.err 인쇄 스트림이 포함됩니다. 이러한 기본 로깅 대상에 로깅을 기록할 수 있습니다. 이러한 인쇄 스트림은 영구적으로 닫혀 향후 사용할 수 없으므로 close() 메소드를 호출해서는 안 됩니다.

가능한 출력 대상

CICS에서 제공하는 샘플 클래스는 JVM의 출력 방향을 CICS 영역의 고유한 디렉토리로 지정합니다. 디렉토리 이름은 CICS 영역과 연관된 applid를 사용하여 작성됩니다. 원하는 경우 고유한 클래스를 작성할 때 여러 CICS 영역의 출력을 동일한 z/OS UNIX 디렉토리 또는 파일로 전송할 수 있습니다.

예를 들어, 여러 다른 CICS 영역에서 실행되는 특정 애플리케이션과 연관된 출력을 포함하는 단일 파일을 작성할 수 있습니다.

Thread.start()를 사용하여 프로그래밍 방식으로 시작되는 스레드는 CICS 요청을 작성할 수 없습니다. 이러한 애플리케이션의 경우 JVM으로부터의 출력은 USEROUTPUTCLASS에 지정한 클래스가 인터셉트하지만 CICS 기능(예: 트랜지언트 데이터 큐)을 사용하여 경로를 재지정할 수 없습니다. 이러한 애플리케이션의 출력 방향을 제공된 샘플 클래스와 마찬가지로 z/OS UNIX 파일로 지정할 수 있습니다.

출력 방향 전환 오류 및 내부 오류 처리

클래스가 CICS 기능을 사용하여 출력 방향을 전환하는 경우 해당 기능 사용에 따른 오류를 처리하기 위해 적절한 예외 처리가 포함되어야 합니다.

예를 들어, 트랜지언트 데이터 큐 CSJO 및 CSJE에 기록하고 이러한 큐에 CICS 제공 정의를 사용하는 경우 TDQ.writeData로 다음 예외가 발생할 수 있습니다.

- IOErrorException
- LengthErrorException
- NoSpaceException
- NotOpenException

클래스가 출력 방향을 z/OS UNIX 파일로 지정하는 경우 z/OS UNIX에 쓸 때 발생하는 오류 처리를 위해 적합한 예외 처리를 포함해야 합니다. 이러한 오류의 가장 일반적인 원인은 보안 예외입니다.

USEROUTPUTCLASS 선택항목에서 클래스 이름을 지정하는 JVM에서 실행될 Java 프로그램에는 클래스로 처리될 수 있는 예외 처리를 위해 적합한 예외 처리가 포함됩니다. CICS 제공 샘플 클래스는 Try/Catch 블록으로 전달 가능한 모든 예외를 발견하고 문제점 보고를 위해 하나 이상의 오류 메시지를 쓰는 방식으로 예외를 내부적으로 처리합니다. 출력 메시지의 경로를 재지정할 때 오류가 발견되면 해당 오류 메시지가 `System.err`에 기록되어 경로 재지정에 사용할 수 있습니다. 그러나 오류 메시지 경로를 재지정할 때 오류가 발견되면 이 문제점을 보고하는 메시지가 요청을 처리하는 JVM이 사용하는 JVM 프로파일에서 STDERR 선택항목으로 표시되는 파일에 기록됩니다. 샘플 클래스는 모든 오류를 이러한 방식으로 처리하므로 프로그램을 호출하여 출력 방향 전환 클래스로 전달된 예외를 처리하지 않아도 됩니다. 이 방법을 사용하면 호출 프로그램이 변경되지 않도록 방지할 수 있습니다. 클래스로 실행된 오류 메시지의 방향을 실패한 대상으로 전환하도록 시도하여 출력 방향 전환 클래스를 루프로 전송하지 않도록 유의하십시오.

제 6 장 Java 성능 개선

Java 애플리케이션과 애플리케이션이 실행되는 JVM의 성능을 개선하기 위해 다양한 조치를 수행할 수 있습니다.

이 태스크 정보

CICS가 올바르게 조정되더라도 비효율적으로 작성된 애플리케이션은 올바르게 작성된 애플리케이션과 비교하여 성능이 저하될 수 있습니다. Java 성능을 개선하는 한 가지 방법으로 사용공간이 적게 생성되도록 애플리케이션을 수정하는 방법이 있습니다. 이 방법을 사용하면 사용공간이 적게 생성되어 사용공간 정리 처리에 적은 시간이 소요되므로 사용공간 정리 비용을 크게 절감할 수 있습니다. 성능 개선을 위해서는 항상 Java 애플리케이션이 효율적으로 작성되고 Java 환경이 올바르게 조정되었는지 확인하십시오.

프로시저

1. Java 워크로드에 대한 성능 목표를 판별하십시오. 가장 일반적인 목표에는 프로세서 사용 또는 애플리케이션 응답 시간 최소화가 포함됩니다. 목표를 설정한 후 Java 환경을 조정할 수 있습니다.
2. Java 애플리케이션을 분석하여 애플리케이션이 효율적으로 실행되고 너무 많은 사용공간을 생성하지 않는지 확인하십시오. IBM은 특정 메소드 및 애플리케이션의 효율성과 성능을 전체적으로 개선하기 위해 Java 애플리케이션을 분석하는 데 도움을 줄 수 있는 도구를 제공합니다.
3. JVM 서버를 조정하십시오. 통계와 IBM 도구로 저장영역 설정, 사용공간 정리, 태스크 대기, 기타 정보를 분석하여 JVM 성능을 조정할 수 있습니다.
4. JVM가 실행되는 Language Environment 인클레이브를 조정하십시오. JVM은 MVS Language Environment 서비스에 대한 호출로 획득한 MVS 저장영역을 사용합니다. Language Environment에 대한 런타임 선택항목을 수정하여 MVS가 할당한 저장영역을 조정할 수 있습니다.
5. 옵션: z/OS 공유 라이브러리 영역을 사용하여 다른 CICS 영역의 JVM 간에 DLL을 공유하는 경우 저장영역 설정을 조정할 수 있습니다.

   164 페이지의『Java 워크로드에 대한 성능 목표 판별』
   지정된 애플리케이션 워크로드에 가장 적합한 전체 성능을 달성하도록 CICS JVM을 조정하는 데는 여러 가지 요인이 존재합니다. Java 워크로드의 원하는 성능 특성을 결정해야 합니다. 이러한 특성을 설정할 때 변경할 매개변수와 변경 방법을 결정할 수 있습니다.

   165 페이지의『IBM Health Center를 사용하여 Java 애플리케이션 분석』
   Java 애플리케이션의 성능을 향상시키기 위해 IBM Health Center를 사용하여 애

플리케이션을 분석할 수 있습니다. 이 도구는 애플리케이션의 성능과 효율성을 향상시키는 데 도움이 되는 권장사항을 제공합니다.

166 페이지의 『사용공간 정리 및 힙 확장』
사용공간 정리와 힙 확장은 JVM 조작의 본질적인 부분입니다. JVM의 사용공간 정리 빈도는 사용공간 *크기* 또는 오브젝트의 영향을 받으며 JVM에서 실행되는 애플리케이션으로 작성됩니다.

169 페이지의 『JVM 서버 성능 개선』
JVM 서버에서 실행되는 애플리케이션의 성능을 개선하기 위해 사용공간 정리 및 힙 *크기*를 포함하는 환경의 여러 파트를 조정할 수 있습니다.

174 페이지의 『JVM용 Language Environment 인클레이브 저장영역』
JVM은 Language Environment 사전 *초기*화 모듈, CELQPIPI를 사용하여 작성되는 Language Environment 인클레이브에서 z/OS UNIX 시스템 서비스 프로세스로 실행됩니다. 인클레이브에 대한 런타임 선택항목을 수정하여 MVS가 할당하는 저장영역을 조정할 수 있습니다.

179 페이지의 『z/OS 공유 라이브러리 영역 조정』
공유 라이브러리 영역은 주소 공간이 동적 링크 라이브러리(DLL) 파일을 공유할 수 있도록 해주는 z/OS 특성입니다. 이 특성을 사용하면 각 CICS 영역이 JVM에 필요한 DLL을 개별적으로 로드하지 않고 공유할 수 있습니다. 따라서 MVS가 사용하는 실제 저장영역의 *크기*와 영역이 파일을 로드하는 데 소요되는 시간을 *크게* 줄일 수 있습니다.

Java 워크로드에 대한 성능 목표 판별

지정된 애플리케이션 워크로드에 가장 적합한 전체 성능을 달성하도록 CICS JVM을 조정하는 데는 여러 가지 요인이 존재합니다. Java 워크로드의 원하는 성능 특성을 결정해야 합니다. 이러한 특성을 설정할 때 변경할 매개변수와 변경 방법을 결정할 수 있습니다.

Java 워크로드에 대한 가장 일반적인 성능 목표는 다음과 같습니다.

최소 전체 프로세서 사용

이 목표는 사용 가능한 프로세서 자원을 가장 효율적으로 사용하기 위한 것입니다. 이 목표를 달성하기 위해 워크로드가 조정되면 전체 워크로드에서의 총 프로세서 사용은 최소화되지만 개별 태스크는 프로세서 이용 증가를 경험할 수 있습니다. 전체 프로세서 사용을 최소화하기 위한 조정에는 사용공간 정리 횟수 최소화를 위한 JVM에 대형 저장영역 힙 *크기* 지정이 포함됩니다.

최소 애플리케이션 응답 시간

이 목표는 애플리케이션 *태스크*가 최대한 빠르게 호출자에게 돌아가도록 하기 위한 것입니다. 이 목표는 특히 서비스 수준 계약을 준수해야 하는 경우 관련이 있습니다. 이 목표를 달성하기 위해 워크로드가 조정되면 애플리케이션이 지

속적이고 빠르게 응답하지만 사용공간 정리를 위해 프로세서 사용이 증가할 수 있습니다. 최소 애플리케이션 응답 시간을 위한 조정에는 힙 *크기*를 작게 유지하는 것과 gencon 사용공간 정리 정책 사용이 포함됩니다.

최소 JVM 저장영역 힙 *크기*

이 목표는 JVM이 사용하는 저장영역 *크기* 감소를 우선시합니다. JVM은 64비트 저장영역을 사용하므로 많은 풀링 JVM 및 JVM 서버를 CICS 영역에서 실행할 수 있습니다. 풀링 JVM이 보다 작은 저장영역 힙을 사용하는 경우 CICS 영역에서 더 많이 실행할 수 있습니다. 그러나 이 목표를 선택하면 프로세서 비용이 증가할 수 있습니다. 저장영역 힙 *크기*를 최소화하기 위해 JVM을 조정하면 사용공간 정리 이벤트 빈도가 증가합니다.

애플리케이션의 응답 시간에는 기타 요인이 영향을 줄 수 있습니다. 가장 중요한 요인은 JIT(Just In Time) 컴파일러입니다. JIT 컴파일러는 런타임에 애플리케이션 코드를 동적으로 최적화하여 많은 이점을 제공하지만 이를 위해서는 특정 *크기*의 프로세서 자원이 필요합니다.

IBM Health Center를 사용하여 Java 애플리케이션 분석

Java 애플리케이션의 성능을 향상시키기 위해 IBM Health Center를 사용하여 애플리케이션을 분석할 수 있습니다. 이 도구는 애플리케이션의 성능과 효율성을 향상시키는 데 도움이 되는 권장사항을 제공합니다.

이 태스크 정보

IBM Health Center는 IBM Support Assistant Workbench에서 사용할 수 있습니다. 이러한 도구는 IBM에서 무료로 다운로드할 수 있습니다(시작하기 안내서의 설명 참조). 애플리케이션은 JVM에서 직접 실행하십시오. JVM 서버에서 혼합 워크로드를 실행하는 경우에는 특정 애플리케이션을 분석하기가 더 어려울 수 있습니다.

프로시저

1. JVM 서버의 JVM 프로파일에 필수 연결 선택항목을 추가하십시오. IBM Health Center 문서는 도구에서 JVM에 연결하기 위해 추가해야 하는 선택항목을 설명합니다.
2. IBM Health Center를 시작하고 실행 중인 JVM에 연결하십시오. IBM Health Center는 JVM 활동을 실시간으로 보고하므로 JVM을 모니터할 때까지 몇 분 동안 기다리십시오.
3. **프로파일링** 링크를 선택하여 애플리케이션을 프로파일링하십시오. 다른 메소드에서 소요되는 시간을 확인할 수 있습니다. 사용량이 가장 많은 메소드에서 잠재적인 문제점을 확인하십시오.

팁: 분석 및 권장사항 탭은 올바른 최적화 후보가 될 수 있는 특정 메소드를 식별할 수 있습니다.

4. **잠금** 링크를 선택하여 애플리케이션의 잠금 경합을 확인하십시오. Java 워크로드가 사용 가능한 프로세서 중 일부만 사용할 수 있는 경우에는 잠금이 원인이 될 수 있습니다. 애플리케이션에서의 잠금은 실행될 수 있는 병렬 스레드 수를 줄일 수 있습니다.
5. **사용공간 정리** 링크를 선택하여 힙 사용 및 사용공간 정리를 확인하십시오. **사용공간 정리** 탭에서는 사용하는 힙의 *크기*와 사용공간 정리를 수행하기 위해 JVM이 일시정지되는 빈도를 알 수 있습니다.
   a. 사용공간 정리에서 소요되는 시간의 비율을 확인하십시오. 이 정보는 요약 섹션에 나타납니다. 사용공간 정리에 소요되는 시간이 2%보다 많은 경우에는 사용공간 정리를 조정해야 합니다.
   b. 사용공간 정리를 위한 일시정지 시간을 확인하십시오. 일시정지 시간이 10밀리초보다 긴 경우에는 사용공간 정리로 인해 애플리케이션 응답 시간에 영향을 줄 수 있습니다.
   c. 각 트랜잭션으로 생성된 대략적인 사용공간 *크기*를 확인하려면 사용공간 정리 비율을 트랜잭션 수로 나누십시오. 사용공간 *크기*가 애플리케이션에 비해 큰 것처럼 보이는 경우에는 애플리케이션을 더 조사할 수 있습니다.

다음에 수행할 작업

애플리케이션을 분석한 후 Java 워크로드에 맞게 Java 환경을 조정할 수 있습니다.

사용공간 정리 및 힙 확장

사용공간 정리와 힙 확장은 JVM 조작의 본질적인 부분입니다. JVM의 사용공간 정리 빈도는 사용공간 *크기* 또는 오브젝트의 영향을 받으며 JVM에서 실행되는 애플리케이션으로 작성됩니다.

할당 실패

JVM이 저장영역 힙에서 공간이 부족하고 오브젝트를 더 할당할 수 없는 경우(할당 실패) 사용공간 정리가 트리거됩니다. 사용공간 정리기는 저장영역 힙에서 더 이상 애플리케이션이 참조하지 않는 오브젝트를 정리하여 사용 가능한 공간을 확보합니다. 사용공간 정리는 사용공간 정리 기간 동안 다른 모든 프로세스가 JVM에서 실행되지 못하게 하므로 사용공간 정리에 소요되는 시간은 애플리케이션을 실행하기 위해 사용하지 않는 시간입니다. JVM 사용공간 정리 프로세스에 대한 자세한 설명은 IBM SDK for z/OS, Java Technology Edition, 버전 7 사용자 안내서의 내용을 참조하십시오.

사용공간 정리가 할당 실패로 인해 트리거되지만 사용공간 정리로 충분한 공간을 확보하지 못하면 사용공간 정리기가 저장영역 힙을 확장합니다. 힙 확장 중에 사용공간 정

리기가 힙에 예약된 최대 저장영역 *크기*(-Xmx 선택항목으로 지정된 *크기*)에서 저장영역을 가져오고 힙의 활성 파트(-Xms 선택항목으로 지정된 *크기*로 시작됨)에 추가합니다. 시동 시 -Xmx 선택항목으로 지정된 최대 저장영역 *크기*가 JVM에 이미 할당되었으므로 힙 확장으로도 JVM에 필요한 저장영역 *크기*가 증가하지 않습니다. -Xms 선택항목의 값이 애플리케이션에 대한 힙의 활성 파트에 충분한 저장영역을 제공하는 경우 사용공간 정리기가 힙 확장을 전혀 수행하지 않아도 됩니다.

JVM 수명 중 특정 시점에 사용공간 정리기가 저장영역 힙 확장을 중지합니다. 힙이 사용공간 정리기가 사용공간 정리 빈도와 프로세스에서 확보한 공간의 *크기*에 만족하는 상태에 도달했기 때문입니다. 사용공간 정리기는 할당 실패를 제거하기 위한 것이 아니므로 사용공간 정리기가 저장영역 힙 확장을 중지한 후에도 할당 실패로 인해 일부 사용공간 정리가 계속 트리거될 수 있습니다. 성능 목표에 따라 이 사용공간 정리 빈도가 과도한 것으로 판단할 수 있습니다.

사용공간 정리 선택항목

애플리케이션 및 전체 시스템의 처리량과 사용공간 정리로 인한 일시정지 시간의 균형을 조율하는 다양한 사용공간 정리 정책을 사용할 수 있습니다. 사용공간 정리는 -Xgcpolicy 선택항목으로 제어됩니다.

-Xgcpolicy:optthruput

이 정책은 애플리케이션에 높은 처리량을 제공하지만 사용공간 정리가 수행될 때 일시정지의 가능성이 있습니다.

-Xgcpolicy:gencon

이 정책은 사용공간 정리 일시정지에 소요되는 시간을 최소화하는 데 도움을 줍니다. 이 사용공간 정리 정책은 JVM 서버에 사용합니다. JVMSERVER 자원에 대한 조회를 수행하여 JVM 서버가 사용하는 정책을 확인할 수 있습니다. JVM 서버 통계에는 발생하는 기본 및 보조 사용공간 정리 이벤트의 수와 사용공간 정리에 소요되는 프로세서 시간을 알려주는 필드가 있습니다.

JVM 프로파일을 갱신하여 사용공간 정리 정책을 변경할 수 있습니다. 모든 사용공간 정리 선택항목에 대한 세부사항은 z/OS에서 Java 버전 7용 사용자 안내서의 가비지 콜렉션 정책 지정의 내용을 참조하십시오.

예제: 대량 사용공간을 생성하는 멀티스레드된 애플리케이션

168 페이지의 그림 2는 gencon 사용공간 정리 정책에 있어 JVM 서버의 저장영역 힙을 다양한 단계로 보여줍니다. 하나의 JVM 서버가 동일한 애플리케이션에 대한 여러 동시 요청을 실행할 수 있으므로 애플리케이션이 보다 많은 사용공간을 생성할 수 있습니다. JVM 서버의 사용공간 정리는 JVM이 자동으로 처리합니다.

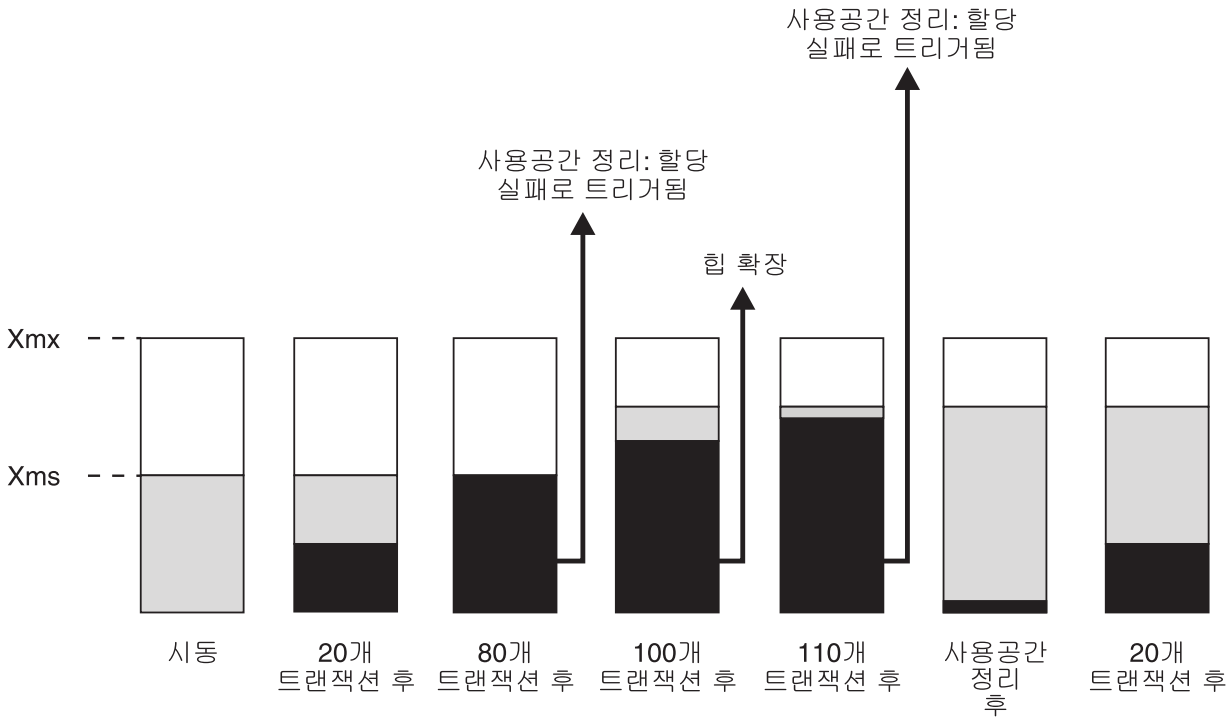


그림 2. 사용공간이 많은 JVM 서버의 저장영역 힙

처음 20개 트랜잭션에서는 저장영역 힙의 활성 파트를 채우기 시작합니다. 80개 트랜잭션 이후에는 힙이 가득차고 할당 실패가 발생하여 JVM에서 보조 사용공간 정리가 트리거됩니다. 사용공간 정리는 단기 오브젝트를 정리합니다. 그러나 애플리케이션 요청은 계속 실행되어 일부 오브젝트는 계속 참조되므로 사용공간 정리 대상이 될 수 없습니다.

100개 트랜잭션 이후에는 사용공간 정리기가 현재 필요한 모든 오브젝트에 충분한 공간을 찾을 수 없어 저장영역 힙을 확장합니다. 저장영역 힙에 예약된 최대 저장영역 *크기*(`-Xmx` 선택항목으로 지정된 *크기*)의 일부 저장영역이 힙의 활성 파트에 추가됩니다. 애플리케이션은 오브젝트를 계속 생성하지만 힙 확장으로 충분한 공간이 작성되어 현재 트랜잭션을 완료할 수 있습니다.

110개 트랜잭션 이후에는 저장영역 힙이 대부분 점유됩니다. 기본 사용공간 정리를 트리거하는 다른 할당 실패가 발생합니다. JVM은 이전 트랜잭션이 사용하는 장기 오브젝트를 다수 정리합니다. 또 다른 20개 트랜잭션 이후에는 힙이 다시 채워지기 시작합니다.

`gencon` 정책을 사용하는 경우, 기본 사용공간 정리가 발생하기 전에 힙 *크기*를 관리하기 위해 다수의 보조 사용공간 정리가 발생할 수 있습니다. JVM 서버 통계를 사용하면 발생한 사용공간 정리 횟수, 힙 점유도, 기타 정보를 얻을 수 있습니다.

JVM 서버 성능 개선

JVM 서버에서 실행되는 애플리케이션의 성능을 개선하기 위해 사용공간 정리 및 힙 크기를 포함하는 환경의 여러 파트를 조정할 수 있습니다.

이 태스크 정보

CICS는 태스크가 스레드, 힙 *크기*, 사용공간 정리 빈도, 프로세서 사용을 대기하는 시간에 대한 세부사항을 포함하는 JVM 서버에 대한 통계 보고서를 제공합니다. JVM을 직접 모니터하고 분석하는 추가 IBM 도구를 사용하여 JVM 서버를 조정하고 문제점 진단에 도움을 줄 수도 있습니다. 통계를 사용하면 JVM이 효율적으로 실행되는지, 특히 힙 *크기*가 적절하고 사용공간 정리가 최적화되었는지 확인할 수 있습니다.

프로시저

1. JVM 서버가 사용하는 프로세서 시간을 확인하십시오. 디스패처 통계를 통해 T8 TCB가 사용하는 프로세서 시간을 알 수 있습니다. JVM 서버 통계는 JVM이 사용공간 정리에서 소비하는 시간과 발생한 사용공간 정리 횟수를 알려줍니다. JVM 사용공간 정리는 애플리케이션 응답 시간 및 프로세서 사용에 부정적인 영향을 줄 수 있습니다.
2. JVM 서버에 필요한 힙 *크기*를 조정하기 위해 사용할 수 있는 저장영역이 충분한지 확인하십시오.
3. JVM에서 사용공간 정리와 힙을 조정하십시오. 작은 힙으로도 사용공간 정리가 자주 수행될 수 있지만 힙이 너무 크면 MVS 저장영역 사용 효율이 낮아질 수 있습니다. IBM Health Center를 사용하면 사용공간 정리를 시각화 및 조정하고 그에 따라 힙을 조정할 수 있습니다.

다음에 수행할 작업

메모리 사용 및 힙 *크기*에 대한 자세한 분석을 위해 IBM Support Assistant의 메모리 분석기 도구와 Java 프로세스의 시스템 덤프 또는 힙 덤프 스냅샷을 사용하여 Java 힙 메모리를 분석할 수 있습니다.

170 페이지의『JVM 서버의 프로세서 사용 검사』
CICS 모니터링 기능을 사용하여 JVM 서버에서 실행되는 트랜잭션이 사용하는 프로세서 시간을 모니터할 수 있습니다. JVM 서버의 모든 스레드는 T8 TCB에서 실행됩니다.

171 페이지의『JVM 서버에 대한 저장영역 요구사항 계산』
CICS 영역에서 JVM 서버 수를 늘리려면 CICS에 사용 가능한 저장영역이 충분한지 확인해야 합니다.

172 페이지의『JVM 서버 힙 및 사용공간 정리 조정』
JVM 서버의 사용공간 정리는 JVM이 자동으로 처리합니다. 애플리케이션 응답 시간과 프로세서 사용이 최적화되도록 사용공간 정리 프로세스와 힙 크기를 조정할 수 있습니다.

174 페이지의『JVM 서버 시작 환경 조정』
여러 JVM 서버를 실행하는 경우 JVM 시동 환경을 조정하여 성능을 개선할 수 있습니다.

JVM 서버의 프로세서 사용 검사

CICS 모니터링 기능을 사용하여 JVM 서버에서 실행되는 트랜잭션이 사용하는 프로세서 시간을 모니터할 수 있습니다. JVM 서버의 모든 스레드는 T8 TCB에서 실행됩니다.

이 태스크 정보

DFH$MOLS 유틸리티를 사용하여 SMF 레코드를 인쇄하거나 CICS 성능 분석기와 같은 도구를 사용하여 SMF 레코드를 분석할 수 있습니다.

프로시저

1. CICS 영역에서 모니터링을 켜고 모니터링 데이터의 성능 클래스를 수집하십시오.
2. 성능 데이터 그룹 DFHTASK를 확인하십시오. 특히 다음 필드를 확인하십시오.

필드 ID	필드 이름	설명
283	MAXTTDLY	CICS 영역이 사용 가능 스레드 한계에 도달하여 사용자 태스크가 T8 TCB 획득을 위해 대기한 경과 시간입니다. 각 CICS 영역에 대한 스레드 한계는 2000이며 각 JVM 서버가 최대 256개 스레드를 가질 수 있습니다.
400	T8CPUT	CICS T8 모드 TCB에서 CICS 디스패처 도메인이 사용자 태스크를 디스패치한 프로세서 시간입니다. 스레드에 T8 TCB가 할당되면 처리가 완료될 때까지 동일한 TCB가 계속 스레드와 연관됩니다.
401	JVMTHDWT	CICS 시스템이 CICS 영역에서 JVM 서버에 대한 스레드 한계에 도달하여 사용자 태스크가 JVM 서버 스레드 획득을 위해 대기한 경과 시간입니다.

3. 프로세서 사용을 개선하려면 가능한 경우 추적 사용을 줄이거나 사용하지 마십시오.
   a. 프로덕션 환경의 경우 CICS 마스터 시스템 추적 플래그 설정을 끄고 CICS 영역을 실행할 수 있습니다. 이 플래그를 켜면 Java 프로그램 실행을 위한 프로세서 비용이 크게 증가합니다. SYSTR=OFF로 CICS를 초기화하거나 CETR 트랜잭션을 사용하여 플래그 설정을 끌 수 있습니다.

b. JVM 추적을 특수 트랜잭션에만 활성화하는지 확인하십시오. JVM 추적은 짧은 시간 안에 많은 양의 출력을 생성할 수 있으므로 프로세서 비용이 증가합니다. JVM 추적 제어에 대한 자세한 정보는 199 페이지의 『Java에 대한 진단』의 내용을 참조하십시오.

4. 프로덕션 환경에서는 JVM 프로파일에서 USEROUTPUTCLASS 선택항목을 사용하지 마십시오. 이 선택항목을 지정하면 JVM 성능에 부정적인 영향을 미칩니다. USEROUTPUTCLASS 선택항목을 지정하면 동일한 CICS 영역을 사용하는 개발자가 JVM 출력을 구분하여 적합한 대상에 지정할 수 있지만 추가 클래스 인스턴스 빌드 및 호출이 필요합니다.

JVM 서버에 대한 저장영역 요구사항 계산

CICS 영역에서 JVM 서버 수를 늘리려면 CICS에 사용 가능한 저장영역이 충분한지 확인해야 합니다.

이 태스크 정보

JVM은 16MB 행, 31비트 저장영역, 64비트 저장영역 미만의 저장영역을 사용합니다. JVM 서버를 실행하면 CICS 영역에서 실행되는 JVM 수에 관계 없이 단 한번의 저장영역 비용이 발생합니다. 각 JVM 서버와 해당 Language Environment 인클레이브에는 또한 특정 *크기*의 31비트 및 64비트 저장영역이 필요합니다. JVM 힙 *크기*는 JVM으로 관리하고 CICS는 기본값을 사용합니다. 필요한 경우 환경 조정의 일부로 힙 *크기*를 조정할 수 있습니다.

JVM 힙에 필요한 저장영역은 CICS 영역 저장영역에서 가져옵니다(EDSA 저장영역이 아닌 MVS 저장영역에서). 더 큰 JVM 힙은 CICS 영역에 존재할 수 있는 JVM 수를 줄이고 힙을 지원할 수 있도록 영역 *크기*를 늘립니다. 그러나 힙 *크기*가 너무 작게 설정되면 과도한 사용공간 정리가 수행되어 성능에 영향을 줍니다. JVM 저장영역 선택항목을 조정하여 Java 워크로드에 최대 성능을 제공할 수 있습니다. JVM 저장영역 선택항목은 Java 애플리케이션에 대한 프로세서 사용, 저장영역 사용, 태스크 응답 시간을 판별하는 데 도움을 줍니다.

프로시저

1. 샘플 통계 프로그램 DFH0STAT를 사용하여 막대 아래 사용 가능한 저장영역의 크기를 판별하십시오. 저장영역 보고서에는 31비트 저장영역에 할당된 16MB 행 아래 사용자 저장영역의 *크기*가 포함됩니다.
   - JVM 서버가 실행되지 않는 경우에는 CICS 주소 공간 내 사용 가능한 총 저장영역 *크기*에서 z/OS 공유 라이브러리 영역에 예약된 저장영역을 빼십시오. 저장영역은 MVS의 **SHRLIBRGNSIZE** 매개변수가 제어하며 영역에서 첫 번째 JVM이 시작될 때 한 번 할당됩니다.
   - JVM 서버가 실행 중인 경우 총 사용 가능 저장영역 *크기*에서 **SHRLIBRGNSIZE** 매개변수 값을 빼십시오. 실행되는 각 JVM은 16MB 행 아래의 12KB 저장영

역을 사용합니다. 각 JVM의 Language Environment 인클레이브는 힙과 라이브러리 힙에 31비트 저장영역을 사용합니다. 할당된 31비트 저장영역의 *크기*는 DFHOSGI의 HEAP64 및 LIBHEAP64 선택항목으로 설정됩니다. 또한 현재 사용 가능한 저장영역의 *크기*를 확인하려면 총 사용 가능 저장영역 *크기*에서 이들 값을 빼야 합니다.

31비트 저장영역 설정을 변경하려는 경우 **SHRLIBRGNSIZE** 매개변수와 Language Environment 선택항목을 조정할 수 있습니다. 179 페이지의 『z/OS 공유 라이브러리 영역 조정』 및 177 페이지의 『DFHAXRO로 JVM 서버 인클레이브 수정』의 내용을 참조하십시오.

2. 추가 JVM 서버마다 필요한 64비트 저장영역을 계산하십시오. 다음 저장영역 요구사항을 더하여 JVM 서버에 대한 64비트 저장영역 요구사항을 계산할 수 있습니다.
   - **-Xmx** 값. 이 매개변수의 기본값은 JVM이 설정하므로 Java Information Center의 문서를 참조하십시오.
   - DFHAXRO의 HEAP64 선택항목으로 할당되는 64비트 저장영역의 값
   - DFHAXRO의 LIBHEAP64 선택항목으로 할당되는 64비트 저장영역의 값
   - DFHAXRO의 STACK64 선택항목으로 할당되는 64비트 저장영역의 값. JVM 서버에서 허용되는 스레드 수를 이 값에 곱하십시오. 허용되는 스레드 수를 계산하려면 JVMSERVER 자원의 THREADLIMIT 속성 값을 **-Xgcthreads** 매개변수 값에 더하십시오. 이 Java 선택항목은 JVM의 사용공간 정리 헬퍼 스레드 수를 제어합니다.
3. **MEMLIMIT** 값을 확인하여 추가 JVM 서버를 실행할 수 있는 충분한 64비트 저장영역이 있는지 여부를 판별하십시오. 64비트 저장영역을 사용하는 다른 CICS 기능을 고려해야 합니다.

   z/OS **MEMLIMIT** 매개변수는 CICS 영역에 대한 64비트(막대 위) 저장영역의 *크기*를 제한합니다. 64비트 저장영역을 사용하는 CICS 기능과 이 매개변수를 확인 및 조정하는 방법에 대한 정보는 성능 개선에서 MEMLIMIT 추정, 검사 및 설정의 내용을 참조하십시오.

JVM 서버 힙 및 사용공간 정리 조정

JVM 서버의 사용공간 정리는 JVM이 자동으로 처리합니다. 애플리케이션 응답 시간과 프로세서 사용이 최적화되도록 사용공간 정리 프로세스와 힙 *크기*를 조정할 수 있습니다.

이 태스크 정보

사용공간 정리 프로세스는 애플리케이션 응답 시간과 프로세서 사용에 영향을 미칩니다. 사용공간 정리는 JVM에서 모든 작업을 일시적으로 중지하므로 애플리케이션 응답

시간에 영향을 줄 수 있습니다. 작은 힙 *크기*를 설정하면 메모리를 절약할 수 있지만 사용공간 정리가 보다 자주 실행되어 사용공간 정리에 프로세서 시간이 많이 소요될 수 있습니다. 힙 *크기*를 너무 *크*게 설정하면 JVM이 MVS 저장영역을 비효율적으로 사용하여 잠재적으로 데이터 캐시 누락은 물론 페이징까지 야기할 수 있습니다. CICS는 JVM 서버 분석을 위해 사용할 수 있는 통계를 제공합니다. 또한 데이터 분석과 조정 선택항목 권장에 따른 이점을 제공하는 IBM Health Center를 사용할 수 있습니다.

프로시저

1. JVM 서버 및 디스패처 통계를 적절한 간격으로 수집하십시오. JVM 서버 통계는 수행되는 기본 및 보조 사용공간 정리 횟수와 사용공간 정리 수행을 위해 소요되는 시간을 알려줍니다. 디스패처 통계는 CICS 영역에서 T8 TCB용 프로세서 사용에 대한 정보를 알려줍니다.
2. T8 TCB에 대한 디스패처 TCB 모드 통계를 사용하여 JVM 서버 스레드에서 소요되는 프로세서 시간을 확인하십시오. 『누적 CPU 시간 / TCB』 필드에는 이 TCB 모드에서 첨부되거나 첨부된 모든 TCB에 대해 소요된 누적 프로세서 시간이 표시됩니다. 『TCB 접속』 필드에는 통계 간격에서 사용된 T8 TCB 수가 표시됩니다. 이 수치를 사용하면 각 T8 TCB가 사용한 대략적인 프로세서 시간을 알 수 있습니다.
3. JVM 서버 통계를 사용하여 사용공간 정리에 소요된 시간의 백분율을 확인하십시오. 사용공간 정리에 소요된 경과 시간으로 통계 간격 시간을 나누십시오. 사용공간 정리에 따른 프로세서 사용 목표를 2% 미만으로 지정하십시오. 이 백분율이 더 높은 경우에는 사용공간 정리가 보다 적게 발생하도록 힙 *크기*를 늘릴 수 있습니다.
4. 간격에서 실행된 트랜잭션 수로 힙 사용 가능 값을 나누어 트랜잭션당 정리되는 사용공간을 확인하십시오. T8 TCB에 대한 디스패처 통계를 보고 실행된 트랜잭션 수를 확인할 수 있습니다. JVM 서버의 각 스레드가 T8 TCB를 사용합니다.
5. 옵션: verbosegc 로그 데이터를 파일에 기록하십시오(**-Xverbosegclog:***path_to_file* 매개변수로 수행 가능). 이 데이터는 다른 ISA 도구 - 사용공간 정리 및 메모리 Visualizer가 분석할 수 있습니다. JVM은 XML의 사용공간 정리 메시지를 JVM 프로파일에서 STDERR 선택항목에 지정된 파일에 씁니다. 메시지 예제 및 해당 설명은 IBM SDK for z/OS, Java Technology Edition, 버전 7(문제점 해결 절 참조)의 내용을 참조하십시오.

   팁: 메모리 분석기 도구에서 파일을 사용하여 자세한 분석을 수행할 수 있습니다.

결과

조정 결과는 Java 워크로드, CICS 및 IBM SDK for z/OS의 유지보수 레벨, 기타 요인에 따라 다를 수 있습니다. 저장영역 및 사용공간 정리 설정과 JVM의 정리 가능성에 대한 자세한 정보는 IBM SDK for z/OS, Java Technology Edition, 버전 7(문제점 해결 절 참조)의 내용을 참조하십시오.

IBM Health Center와 메모리 분석기는 IBM Support Assistant Workbench가 제공하는 두 가지 Java용 IBM 모니터링 및 진단 도구입니다. 이러한 도구는 사이트에서 무료로 다운로드할 수 있습니다.

JVM 서버 시작 환경 조정

여러 JVM 서버를 실행하는 경우 JVM 시동 환경을 조정하여 성능을 개선할 수 있습니다.

이 태스크 정보

JVM 서버가 시작되면 서버가 `/usr/lpp/cicsts/cicsts52/lib` 디렉토리에서 라이브러리 세트를 로드해야 합니다. 동시에 여러 JVM 서버를 시작하는 경우, 필수 라이브러리를 로드하는 데 소요되는 시간으로 인해 일부 JVM 서버에서 시간이 종료되거나 JVM 서버가 시작되는 데 과도한 시간이 소요될 수 있습니다. JVM 서버 시작 시간을 줄이려면 JVM 시작 환경을 조정해야 합니다.

프로시저

1. 라이브러리를 한 번 로드하려면 JVM 서버의 공유 클래스 캐시를 작성하십시오. 공유 클래스 캐시를 사용하려면 각 JVM 서버의 JVM 프로파일에 **-Xshareclasses** 선택항목을 추가하십시오. 자세한 정보는 JVM 간 클래스 데이터 공유의 내용을 참조하십시오.
2. OSGi 프레임워크의 제한시간 값을 늘리십시오. DFHOSGI.jvmprofile에는 JVM 서버가 시작 및 종료될 때까지 CICS가 대기하는 시간을 지정하는 `OSGI_FRAMEWORK_TIMEOUT` 선택항목이 포함됩니다. 값이 초과되면 JVM 서버가 올바르게 초기화 또는 종료되지 않습니다. 기본값은 60초이므로 사용자 환경에 맞게 이 값을 늘려야 합니다.

JVM용 Language Environment 인클레이브 저장영역

JVM은 Language Environment 사전 초기화 모듈, CELQPIPI를 사용하여 작성되는 Language Environment 인클레이브에서 z/OS UNIX 시스템 서비스 프로세스로 실행됩니다. 인클레이브에 대한 런타임 선택항목을 수정하여 MVS가 할당하는 저장영역을 조정할 수 있습니다.

JVM은 CICS Language Environment 서비스 대신 MVS Language Environment 서비스를 사용합니다. 결과적으로 JVM이 획득한 모든 저장영역은 MVS Language Environment 서비스에 대한 호출로 획득한 MVS 저장영역입니다. 이 저장영역은 CICS 주소 공간에 있지만 CICS 동적 저장영역(DSA)에는 포함되지 않습니다. CICS에서 실행되는 모든 JVM은 64비트 저장영역을 사용합니다.

각 JVM의 Language Environment 인클레이브에는 JVM 저장영역 힙뿐 아니라 각 JVM의 기본 저장영역 크기가 포함되어야 합니다. 이 기본 저장영역 비용은 JVM 구조에 사

용되는 Language Environment 인클레이브의 저장영역 크기를 나타냅니다. JVM의 총 크기를 계산할 때 기본 저장영역 비용을 저장영역 힙에 사용되는 저장영역에 추가해야 합니다.

Language Environment 런타임 선택항목은 DFHAXRO로 설정됩니다. 이러한 프로그램이 JVM 인클레이브에 제공하는 기본값이 표 11에 표시되어 있습니다.

표 11. JVM 인클레이브에 대해 CICS가 사용하는 Language Environment 런타임 선택항목

Language Environment 런타임 선택항목	**JVM 서버 값**
힙 저장영역	`HEAP64(100M,4M,KEEP,4M,512K, KEEP,1K,1K,KEEP)`
라이브러리 힙 저장영역	`LIBHEAP64(3M,3M)`
저장영역 내 임의 위치에 상주할 수 있는 라이브러리 루틴 스택 프레임	`STACK64(1M,1M,32M)`
멀티스레드 애플리케이션에 대한 선택적 사용자 힙 저장영역 관리	`HEAPPOOLS64(ALIGN)`
멀티스레드 애플리케이션에 대한 선택적 힙 저장영역 관리	`HEAPPOOLS(ALIGN)`
할당 및 사용 가능한 경우 저장영역의 초기 내용과 저장영역 부족 조건을 위해 예약된 저장영역의 크기	`STORAGE(NONE,NONE,NONE)`

Language Environment 런타임 선택항목에 대한 정보는 *z/OS Language Environment Customization*의 내용을 참조하십시오.

샘플 프로그램 DFHAXRO를 수정하고 재컴파일하여 Language Environment 런타임 선택항목을 덮어쓸 수 있습니다(177 페이지의 『DFHAXRO로 JVM 서버 인클레이브 수정』의 설명 참조). 이 프로그램은 JVMSERVER 자원에서 설정되므로 필요한 경우 개별 JVM 서버에 다른 선택항목 Language Environment 인클레이브를 사용할 수 있습니다. Language Environment 인클레이브 힙 저장영역의 초기 크기와 증분을 제어하는 기본 Language Environment 저장영역 설정에 따라 MVS 저장영역을 비효율적으로 사용할 수 있습니다. CICS가 제공하는 저장영역 설정이 보다 효율적입니다. JVM의 저장영역 사용과 보다 일치하도록 이러한 설정을 수정할 수도 있습니다. 힙 크기가 많은 세그먼트 할당 및 사용 가능을 방지하도록 설정되었는지 확인하십시오.

Language Environment 인클레이브에서 JVM에 필요한 저장영역 크기로 인해 **REGION** 및 **MEMLIMIT** 크기를 제한하기 위해 사용하는 설치 종료, IEALIMIT 또는 IEFUSI를 변경해야 할 수 있습니다. 가능한 접근 방법은 모든 Java 프로그램 요청이 라우팅되는 Java 소유 영역(JOR)을 갖는 것입니다. 이러한 영역은 Java 워크로드만 실행하므로 필요한 CICS DSA 저장영역의 크기가 최소화되고 MVS 저장영역의 최대 크기를 JVM에 할당할 수 있습니다.

176 페이지의 『JVM 서버의 Language Environment 저장영역 요구 식별』
저장영역 보고서를 생성하여 JVM 서버 내 Language Environment 인클레이브 힙

저장영역의 초기 할당에 적합한 값을 식별할 수 있습니다. 저장영역 보고서를 생성하면 프로세서 비용이 증가하므로 프로덕션 환경의 경우 적절한 시간에 보고서를 실행하십시오.

177 페이지의『DFHAXRO로 JVM 서버 인클레이브 수정』
DFHAXRO는 JVM 서버가 실행되는 Language Environment 인클레이브에 기본 런타임 선택항목 세트를 제공하는 샘플 프로그램입니다. 예를 들어, JVM 힙 및 스택에 대한 저장영역 할당 매개변수를 정의합니다. CICS의 경우 DFHAXRO에 제공되는 저장영역 설정이 기본 Language Environment 저장영역 설정보다 적절합니다.

JVM 서버의 Language Environment 저장영역 요구 식별

저장영역 보고서를 생성하여 JVM 서버 내 Language Environment 인클레이브 힙 저장영역의 초기 할당에 적합한 값을 식별할 수 있습니다. 저장영역 보고서를 생성하면 프로세서 비용이 증가하므로 프로덕션 환경의 경우 적절한 시간에 보고서를 실행하십시오.

이 태스크 정보

DFHAXRO의 HEAP64 런타임 선택항목은 JVM 서버에 대한 Language Environment 인클레이브의 힙 *크기*를 제어합니다. 이 선택항목에는 64비트 및 31비트 저장영역에 대한 설정이 포함됩니다. 필요한 경우 DFHAXRO 대신 고유 프로그램을 사용할 수 있습니다. 프로그램은 JVMSERVER 자원에서 지정되어야 합니다.

프로시저

1. DFHAXRO에서 RPTO(ON) 및 RPTS(ON) 선택항목을 설정하십시오. 이러한 선택항목은 DFHAXRO의 제공 소스에서 설명에 나타납니다. 이러한 선택항목을 지정하면 Language Environment가 저장영역 선택항목을 보고하고 사용된 실제 저장영역을 보여주는 저장영역 보고서를 작성합니다.
2. JVMSERVER 자원을 사용 안함으로 설정하십시오. JVM 서버가 종료되고 Language Environment 인클레이브가 제거됩니다.
3. JVMSERVER 자원을 사용 가능으로 설정하십시오. CICS가 DFHAXRO의 Language Environment 런타임 선택항목을 사용하여 JVM 서버의 인클레이브를 작성합니다. JVM 또한 시작됩니다.
4. JVM 서버에서 Java 워크로드를 실행하여 Language Environment 인클레이브가 사용하는 저장영역에 대한 데이터를 수집하십시오.
5. DFHAXRO에서 RPTO(ON) 및 RPTS(ON) 선택항목을 제거하십시오.
6. JVMSERVER 자원을 사용 안함으로 설정하여 저장영역 보고서를 생성하십시오. 저장영역 보고서에는 초기 Language Environment 인클레이브 힙 저장영역에 대한 제안이 포함됩니다. 64비트 사용자 힙 통계의 "제안되는 초기 *크기*" 항목에는 제안되는 값이 포함되며 JVM 서버가 사용한 Language Environment 인클레이브 힙 저장영역의 총 *크기*와 같습니다.

결과

저장영역 보고서는 z/OS UNIX의 stderr 파일에 저장됩니다. 디렉토리는 JVM 프로파일에서 JVM에 대한 출력 방향을 전환했는지 여부에 따라 다릅니다. 방향 전환이 존재하지 않는 경우 JVM의 작업 디렉토리에 파일이 저장됩니다. 프로파일에서 WORK_DIR에 대한 값이 설정되지 않으면 /tmp 디렉토리에 파일이 저장됩니다.

DFHAXRO의 초기 Language Environment 인클레이브 힙 저장영역에 적합한 값을 선택하려면 저장영역 보고서의 정보를 사용하십시오. Language Environment는 힙 저장영역에 추가 항목을 작성할 수 있지만 초기 할당에서 제공된 원하지 않는 저장영역을 제거할 수는 없습니다. 할당되고 사용 가능한 세그먼트 수가 최소화되도록 충분한 저장영역을 할당하십시오.

이 기술을 사용하여 초기 크기를 설정하고 LIBHEAP64 및 STACK64 런타임 선택항목에 대한 값을 늘릴 수도 있습니다.

예

다음 예제는 Language Environment의 저장영역 보고서입니다.

```
64비트 사용자 힙 통계:
      초기 크기:                                    50M
      증분 크기:                                     4M
      사용된 힙 저장영역 총계:                    91977408
      제안된 초기 크기:                             88M
      성공적으로 가져온 힙 요청 수:                2439
      성공적인 사용 가능 힙 요청 수:               1619
      할당된 세그먼트 수:                             1
      사용 가능한 세그먼트 수:                        0
31비트 사용자 힙 통계:
      초기 크기:                                  524288
      증분 크기:                                  524288
      사용된 힙 저장영역 총계(제안된 초기 크기):  8440784
      성공적으로 가져온 힙 요청 수:                1965
      성공적인 사용 가능 힙 요청 수:               1904
      할당된 세그먼트 수:                             2
      사용 가능한 세그먼트 수:                        0
```

예제에서 Language Environment 인클레이브 힙 저장영역의 값을 기반으로 DFHAXRO에서 힙 저장영역에 대해 이러한 값을 설정할 수 있습니다.

```
HEAP64(88M,4M,KEEP,10M,512K,KEEP,1K,1K,KEEP)
```

DFHAXRO로 JVM 서버 인클레이브 수정

DFHAXRO는 JVM 서버가 실행되는 Language Environment 인클레이브에 기본 런타임 선택항목 세트를 제공하는 샘플 프로그램입니다. 예를 들어, JVM 힙 및 스택에 대한 저장영역 할당 매개변수를 정의합니다. CICS의 경우 DFHAXRO에 제공되는 저장영역 설정이 기본 Language Environment 저장영역 설정보다 적절합니다.

이 태스크 정보

샘플 프로그램을 갱신하여 Language Environment 인클레이브를 조정하거나 샘플을 기반으로 고유 프로그램을 만들 수 있습니다. 프로그램은 JVMSERVER 자원에서 정의되며 JVM 서버에 대해 작성되는 Language Environment 인클레이브의 CELQPIPI 사전 초기화 단계에서 호출됩니다.

프로그램은 어셈블리 언어로 작성해야 하며 CICS 변환기로 변환되어서는 안됩니다. 선택항목은 2바이트 문자열 길이와 런타임 선택항목 순서로 구성되는 문자열로 지정됩니다. 모든 Language Environment 런타임 선택항목의 최대 길이는 255바이트이므로 각 선택항목의 축약 버전을 사용하고 변경사항을 총 200바이트 미만으로 제한하십시오.

프로시저

1. DFHAXRO 프로그램을 새 위치에 복사하여 런타임 선택항목을 편집하십시오. CICS 영역에 유지보수가 적용되는 경우 프로그램에 변경사항을 반영할 수 있습니다. DFHAXRO의 소스는 CICSTS52.CICS.SDFHSAMP 라이브러리에 있습니다.
2. 각 선택항목의 약어를 사용하여 런타임 선택항목을 편집하십시오. *z/OS Language Environment Programming Guide*는 Language Environment 런타임 선택항목에 대한 모든 정보를 제공합니다.
   - CICS는 이 목록에 일부 선택항목을 추가하므로 빠른 처리를 위해 선택항목 목록의 크기를 최소값으로 유지하십시오.
   - HEAP64 선택항목을 사용하여 초기 힙 할당을 지정하십시오.
   - ALL31 선택항목, POSIX 선택항목, XPLINK 선택항목은 CICS에서 강제 실행됩니다. CICS에서는 ABTERMENC 선택항목이 (ABEND)로 설정되고 TRAP 선택항목은 (ON,NOSPIE)로 설정됩니다.
   - JVM 서버에서 작성된 파일에 대한 권한을 제어하려면 다음 행을 추가하십시오.

     ```
     DC  C'ENVAR("_EDC_UMASK_DFLT=nnn")'
     ```

     여기서 *nnn*은 필요한 마스크 값입니다.
   - RPTO 및 RPTS 선택항목으로 생성되는 출력은 CESE 트랜지언트 데이터 큐에 기록됩니다.
   - 선택항목은 JVM이 종료될 때마다 출력을 생성합니다. 생성되어 CESE로 방향이 지정될 수 있는 출력 볼륨을 고려하십시오.
3. DFHASMVS 프로시저를 사용하여 프로그램을 컴파일하십시오.

결과

JVMSERVER 자원을 사용하는 경우 CICS는 DFHAXRO 프로그램에 지정한 런타임 선택항목을 사용하여 Language Environment 인클레이브를 작성합니다. CICS는 런타임 선택항목을 Language Environment로 전달하기 전에 그 길이를 확인합니다. 길이

가 255바이트보다 길면 CICS가 JVM 서버 시작을 시도하지 않고 CSMT에 오류 메시지를 씁니다. 사용자가 지정하는 값은 Language Environment로 전달되기 전에 CICS가 확인하지 않습니다.

z/OS 공유 라이브러리 영역 조정

공유 라이브러리 영역은 주소 공간이 동적 링크 라이브러리(DLL) 파일을 공유할 수 있도록 해주는 z/OS 특성입니다. 이 특성을 사용하면 각 CICS 영역이 JVM에 필요한 DLL을 개별적으로 로드하지 않고 공유할 수 있습니다. 따라서 MVS가 사용하는 실제 저장영역의 *크기*와 영역이 파일을 로드하는 데 소요되는 시간을 크게 줄일 수 있습니다.

공유 라이브러리 영역에 대해 예약되는 저장영역은 첫 번째 JVM이 영역에서 시작될 때 각 CICS 영역에 할당됩니다. 할당되는 저장영역의 *크기*는 z/OS의 **SHRLIBRGNSIZE** 매개변수(SYS1.PARMLIB의 BPXPRMxx 멤버에 포함)로 제어됩니다. 최소값은 16MB이고 z/OS 기본값은 64MB입니다. CICS 이외 다른 애플리케이션이 공유 라이브러리 영역을 사용한다는 점을 고려하여 필요한 공간의 *크기*를 조사하고 그에 따라 **SHRLIBRGNSIZE** 매개변수를 조정하여 공유 라이브러리 영역에 할당된 저장영역의 크기를 조정할 수 있습니다.

공유 라이브러리 영역에 할당된 저장영역의 *크기*를 줄이려면 먼저 공유 라이브러리 영역에서 공간을 낭비하지 않는지 확인하십시오. 일반 워크로드를 z/OS 시스템에서 실행한 다음 **D OMVS,L** 명령을 실행하여 라이브러리 통계를 표시하십시오. 공유 라이브러리 영역에 사용하지 않은 공간이 있는 경우 **SHRLIBRGNSIZE**에 대한 설정을 줄여 이 공간을 제거할 수 있습니다. CICS가 공유 라이브러리 영역의 유일한 사용자인 경우 **SHRLIBRGNSIZE**를 최소값 16MB로 줄일 수 있습니다. JVM에 필요한 DLL은 영역의 약 10MB만 사용하기 때문입니다.

공유 라이브러리 영역의 모든 공간을 사용하지만 CICS 영역에서 이 저장영역 할당을 줄이려는 경우 다음과 같은 세 가지 가능한 조치를 고려할 수 있습니다.

1. 파일에 필요한 저장영역 *크기*보다 공유 라이브러리 영역 *크기*를 작게 설정할 수 있습니다. 공유 라이브러리 영역이 가득 찬 경우 대신 개인용 저장영역에 파일이 로드되어 공유 기능의 이점을 활용할 수 없습니다. 이 조치를 선택하는 경우, 보다 중요한 애플리케이션을 먼저 실행하여 공유 라이브러리 영역을 사용할 수 있도록 해야 합니다. 이 조치는 공유 라이브러리 영역에서 대부분의 공간을 중요하지 않은 애플리케이션이 사용하는 경우 가장 적합합니다.
2. 공유 라이브러리 영역에 배치되는 DLL에는 확장 속성 +l이 표시됩니다. 이 속성을 일부 파일에서 제거하면 파일이 공유 라이브러리 영역으로 이동하는 것을 방지할 수 있으므로 공유 라이브러리 영역에 필요한 저장영역의 *크기*를 줄일 수 있습니다. 이 조치를 선택하는 경우, 자주 공유되지 않는 파일을 선택하고 확장자 .so가 있는 파일을 선택하지 마십시오. 확장자가 .so인 파일은(공유 라이브러리 영역에 배치되지

않는 경우) 사용자 공유 라이브러리를 사용하여 공유되므로 이 기능은 공유 라이브러리 영역을 사용하는 것보다 비효율적입니다. 이 조치는 확장자가 .so가 아닌 대형 파일이 공유 라이브러리 영역에서 대부분의 공간을 사용하는 경우 가장 적합합니다.

3. CICS와 관련된 모든 파일에서 확장 속성 +l을 제거하면 CICS 영역이 공유 라이브러리 영역을 전혀 사용하지 않으므로 CICS 영역 내에서 해당 저장영역이 할당되지 않습니다. 이 조치를 선택하는 경우에는 공유 라이브러리 영역의 공유 기능에 따른 이점을 얻을 수 없습니다. 이 조치는 z/OS 시스템의 다른 애플리케이션에 대형 공유 라이브러리 영역이 필요하고 CICS 영역에서 이 크기의 저장영역을 할당하지 않으려는 경우에 가장 적합합니다.

특정 파일을 새 버전으로 바꿀 때(예를 들어, 소프트웨어 업그레이드 중에) 해당 파일에서 확장 속성 +l을 제거하도록 선택하는 경우에는 파일의 새 버전에 이 속성이 없는지 확인해야 합니다.

z/OS UNIX on the z/OS UNIX 시스템 서비스 웹 사이트(http://www.ibm.com/servers/eserver/zseries/zos/unix/perform/sharelib.html)에서 공유 라이브러리에 대한 자세한 정보를 찾을 수 있습니다.

제 7 장 Java 애플리케이션을 위한 보안

권한을 부여 받은 사용자만 애플리케이션을 전개 및 설치하고 웹에서 또는 CICS를 통해 해당 애플리케이션에 액세스할 수 있도록 Java 애플리케이션을 보호할 수 있습니다. Java 보안 관리자를 사용하여 Java 애플리케이션이 잠재적으로 안전하지 않은 조치를 수행하지 못하게 방지할 수도 있습니다.

Java 애플리케이션 라이프 사이클의 다른 지점에서 보안을 추가할 수 있습니다.

- Java 애플리케이션 자원 정의 및 설치를 위한 보안 검사를 구현하십시오. Java 애플리케이션은 CICS 번들에 패키지화되므로 JVM 서버에서 애플리케이션을 설치할 수 있는 사용자가 이 유형의 자원을 설치할 수 있는지 확인해야 합니다.
- 권한을 부여 받은 사용자만 애플리케이션에 액세스할 수 있도록 애플리케이션 사용자에 대한 보안 검사를 구현하십시오.
- CICSExecutorService를 사용하여 시작되는 CICS Java 태스크에 대한 보안 검사를 구현합니다. 모든 해당 CICS 태스크는 CJSA 트랜잭션 및 기본 사용자 ID로 실행됩니다.
- Java 보안 관리자를 사용하여 Java API에 대한 보안 제한사항을 구현합니다.

Java 애플리케이션은 OSGi 프레임워크 또는 Liberty 프로파일 서버에서 실행될 수 있습니다. Liberty 프로파일은 웹 애플리케이션을 호스팅하도록 설계되며 OSGi 프레임워크를 포함합니다. Liberty 프로파일은 고유한 보안 모델을 가지므로 Liberty 프로파일 서버에 대한 보안 구성은 다릅니다.

182 페이지의『OSGi 애플리케이션을 위한 보안 구성』
CICS 보안을 사용하여 해당 사용자가 JVM 서버와 Java 애플리케이션의 라이프 사이클을 관리하고 CICS를 통해 애플리케이션에 액세스할 수 있는 권한을 부여합니다.

182 페이지의『Liberty JVM 서버에 대한 보안 구성』
CICS Liberty 보안 특성을 사용하여 사용자를 인증하고 CICS 트랜잭션 및 자원 보안과의 통합을 제공하여 Java Platform, Enterprise Edition 역할을 통해 웹 애플리케이션에 액세스할 수 있도록 권한을 부여할 수 있습니다. 또한 CICS 자원 보안을 사용하여 CICS BUNDLE 자원에 전개된 Java 웹 애플리케이션과 JVMSERVER 자원 모두의 라이프 사이클을 관리할 수 있는 권한을 적절한 사용자에게 부여할 수 있습니다.

191 페이지의『Liberty JVM 서버에 대한 SSL 구성』
통신은 SSL(Secure Sockets Layer) 프로토콜을 사용하여 보안 설정됩니다. SSL 프로토콜은 인증, 데이터 서명 및 데이터 암호화를 포함한 전송 계층 보안을 제공하여 클라이언트와 서버 간의 보안 통신을 보장합니다. SSL을 데이터 암호화에 사

용하고 선택적으로 클라이언트 인증서를 사용하여 서버를 인증하도록 Liberty JVM 서버를 구성할 수 있습니다. 클라이언트 인증서는 Java 키 저장소 또는 SAF Keyring에 저장될 수 있습니다.

192 페이지의『Java 보안 관리자 사용』
Java 애플리케이션은 기본적으로 Java API에 대해 요청되는 활동에 대한 보안 제한사항을 적용하지 않습니다. Java 애플리케이션이 잠재적으로 안전하지 않은 조치를 수행하지 못하도록 보호하기 위해 Java 보안을 사용하려면 애플리케이션이 실행되는 JVM에 대해 보안 관리자를 사용 가능하도록 설정하면 됩니다.

OSGi 애플리케이션을 위한 보안 구성

CICS 보안을 사용하여 해당 사용자가 JVM 서버와 Java 애플리케이션의 라이프 사이클을 관리하고 CICS를 통해 애플리케이션에 액세스할 수 있는 권한을 부여합니다.

프로시저

애플리케이션 개발자와 시스템 관리자에게 JVMSERVER 및 BUNDLE 자원을 작성, 보기, 갱신, 제거할 수 있는 권한을 적절하게 부여하십시오. JVMSERVER 자원은 JVM 서버의 사용가능성을 제어합니다. BUNDLE 자원은 Java 애플리케이션의 전개 단위이며 애플리케이션의 사용가능성을 제어합니다.

결과

OSGi 프레임워크에서 실행되는 Java 애플리케이션에 대한 보안 구성을 완료했습니다.

Liberty JVM 서버에 대한 보안 구성

CICS Liberty 보안 특성을 사용하여 사용자를 인증하고 CICS 트랜잭션 및 자원 보안과의 통합을 제공하여 Java Platform, Enterprise Edition 역할을 통해 웹 애플리케이션에 액세스할 수 있도록 권한을 부여할 수 있습니다. 또한 CICS 자원 보안을 사용하여 CICS BUNDLE 자원에 전개된 Java 웹 애플리케이션과 JVMSERVER 자원 모두의 라이프 사이클을 관리할 수 있는 권한을 적절한 사용자에게 부여할 수 있습니다.

시작하기 전에

CICS 영역이 SAF 보안을 사용하도록 구성되고 SIT=YES가 시스템 초기화 매개변수로 정의되었는지 확인하십시오. 그런 다음 애플리케이션 개발자와 시스템 관리자에게 JVMSERVER 및 BUNDLE 자원을 작성, 보기, 갱신, 제거할 수 있는 권한을 적절하게 부여하여 웹 애플리케이션이 Liberty JVM 서버에 전개되도록 하십시오. JVMSERVER 자원은 JVM 서버의 사용가능성을 제어하고 BUNDLE 자원은 Java 애플리케이션의 전개 단위이며 애플리케이션의 사용가능성을 제어합니다.

이 태스크 정보

이 태스크는 Liberty JVM 서버를 위한 보안을 구성하고 Liberty 보안을 CICS 보안과 통합하는 방법을 설명합니다.

웹 요청 실행을 위한 기본 트랜잭션 ID는 CJSA입니다. 하지만 JVMSERVER 유형의 URIMAP을 사용하여 다른 트랜잭션 ID로 웹 요청을 실행하도록 CICS를 구성할 수 있습니다. 일반적으로 애플리케이션을 구성하는 servlet 세트로 트랜잭션 ID의 범위를 지정하기 위해 웹 애플리케이션의 일반 컨텍스트 루트(URI)와 일치하도록 URIMAP을 지정할 수 있습니다. 또는 보다 정확한 URI를 사용하는 다른 트랜잭션에서 개별 servlet을 실행하도록 선택할 수 있습니다.

프로시저

1. 인증 및 권한 서비스를 Liberty JVM 서버에 제공하도록 WebSphere Liberty 프로파일 엔젤 프로세스를 구성하십시오. 자세한 정보는 Liberty 서버 엔젤 프로세스를 참조하십시오.
2. `cicsts:security-1.0` 특성을 `server.xml`의 featuremanager 목록에 추가하십시오.

   ```
   <featureManager>
      ...
      <feature>cicsts:security-1.0</feature>
   </featureManager>
   ...
   ```

3. `server.xml`의 변경사항을 저장하십시오.

   참고: 또는 CICS 영역에서 Liberty JVM 서버를 자동으로 구성하고 **SEC** 시스템 초기화 매개변수가 YES로 설정된 경우 JVM 서버를 다시 시작하면 Liberty JVM 보안을 지원하도록 Liberty JVM 서버가 동적으로 구성됩니다. 자세한 정보는 105 페이지의 『웹 애플리케이션에 대한 Liberty JVM 서버 구성』의 내용을 참조하십시오.

   SEC 시스템 초기화 매개변수가 NO로 설정되면 인증 또는 SSL 지원을 위해 계속 Liberty 보안을 사용할 수 있습니다. CICS 보안이 꺼져 있는 상태에서 Liberty 보안을 사용하려면 수동으로 `server.xml` 파일을 구성해야 합니다.

   a. featuremanager 목록에 appSecurity 특성을 추가하십시오.
   b. 사용자를 인증하기 위해 사용자 레지스트리를 추가하십시오. Liberty 보안은 SAF, LDAP 및 기본 사용자 레지스트리를 지원합니다. 자세한 정보는 라이브러리 프로파일용 사용자 레지스트리 구성의 내용을 참조하십시오.
   c. 애플리케이션 자원에 대한 액세스 권한을 부여하려면 security-role 정의를 추가하십시오. 자세한 정보는 189 페이지의 『Liberty JVM 서버에서 애플리케이션을 실행하도록 사용자에게 권한 부여』의 내용을 참조하십시오.

결과

`cicsts:security-1.0` 특성이 사용되면 웹 컨테이너가 자동으로 Liberty의 z/OS 보안 특성을 사용하도록 구성됩니다. 또한 인증을 위해 SAF 레지스트리가 사용되고 권한 부여를 위해 <application-bnd> 요소의 Java Platform, Enterprise Edition 역할이 중요합니다.

다음에 수행할 작업

- Liberty 애플리케이션 보안 인증 규칙을 구성하십시오(187 페이지의 『Liberty JVM 서버에서 사용자 인증』 참조).
- 웹 애플리케이션에 대한 인증 규칙을 정의하십시오(189 페이지의 『Liberty JVM 서버에서 애플리케이션을 실행하도록 사용자에게 권한 부여』 및 190 페이지의 『JEE 애플리케이션 역할 보안』 참조).
- Liberty 인증 캐시를 수정하십시오.

SSL(Secure Sockets Layer) 사용에 대한 자세한 정보는 191 페이지의 『Liberty JVM 서버에 대한 SSL 구성』의 내용을 참조하십시오.

185 페이지의 『Liberty 서버 엔젤 프로세스』
엔젤 프로세스는 WAS(WebSphere Application Server) Liberty 프로파일 서버에 권한 부여된 서비스를 제공합니다.

187 페이지의 『Liberty JVM 서버에서 사용자 인증』
Liberty JVM 서버에서 실행되는 모든 웹 애플리케이션에 대해 CICS 보안을 구성할 수 있지만 이러한 웹 애플리케이션은 보안 제한조건이 포함되어 있는 경우에만 사용자를 인증합니다. 애플리케이션 개발자가 동적 웹 프로젝트 또는 OSGi 애플리케이션 프로젝트의 전개 설명자(web.xml)에 보안 제한조건을 정의합니다. 보안 제한조건은 보호될 대상(URL) 및 그 역할을 정의합니다.

189 페이지의 『Liberty JVM 서버에서 애플리케이션을 실행하도록 사용자에게 권한 부여』
사용자 ID에게 Liberty JVM 서버의 트랜잭션을 실행하도록 권한을 부여하기 위해 CICS 트랜잭션 및 자원 보안을 사용하거나 JEE 애플리케이션 보안 역할을 사용하여 JEE 애플리케이션에 대한 액세스를 부여할 수 있습니다.

190 페이지의 『JEE 애플리케이션 역할 보안』
JEE 애플리케이션 역할 보안은 사용하려는 권한 유형에 따라 다양한 방식으로 구성될 수 있습니다. 분산 시스템의 경우 기본 레지스트리 또는 LDAP 레지스트리는 일반적으로 애플리케이션 특정 `<application-bnd>` 요소와 함께 사용되어 이러한 레지스트리의 사용자를 '역할'에 맵핑합니다. 애플리케이션의 전개 설명자에 따라 어떤 역할이 어떤 애플리케이션의 파트에 액세스할 수 있는지가 결정됩니다.

Liberty 서버 엔젤 프로세스

엔젤 프로세스는 WAS(WebSphere Application Server) Liberty 프로파일 서버에 권한 부여된 서비스를 제공합니다.

엔젤 프로세스는 MVS 콘솔에서 시작됩니다. z/OS 이미지에서 실행 중인 모든 WAS(WebSphere Application Server) Liberty 프로파일 서버는 서버가 실행 중인 코드의 레벨 또는 CICS JVM 서버에서 실행 중인지 여부와 관계 없이 단일 엔젤을 공유할 수 있습니다.

CICS Liberty 보안 특성을 사용하지 않는 경우 엔젤 프로세스를 실행하거나 연관된 보안 규칙을 구현할 필요가 없습니다.

엔젤 프로세스 시작된 태스크

엔젤 프로세스에서 시작된 태스크 JCL 프로시저는 USSHOME 디렉토리에 CICS와 함께 제공됩니다(예:

```
/usr/lpp/cicsts52/wlp/templates/zos/procs/bbgzangl.jcl.
```

JCL이 JES 프로시저 라이브러리로 복사되고 수정되어야 합니다. 루트가 USSHOME/wlp 값으로 설정되어야 합니다(예:

```
/usr/lpp/cicsts52/wlp.
```

Liberty JVM 서버가 시작되기 전에 엔젤 프로세스가 실행 중이어야 합니다. 엔젤 프로세스를 시작하거나 중지하려면 다음 운영자 명령을 실행하십시오.

```
START BBGZANGL
```

```
STOP BBGZANGL
```

엔젤 프로세스에 연결된 Liberty JVM 서버를 표시하려면 다음 운영자 명령을 실행하십시오.

```
MODIFY BBGZANGL,DISPLAY,SERVERS
```

리턴된 메시지에는 엔젤 프로세스에 연결된 활성 Liberty JVM 서버 및 기타 비CICS Liberty JVM 서버가 포함된 CICS 영역이 나열됩니다.

엔젤 프로세스 시작된 태스크 SAF 규칙

엔젤 프로세스가 실행되는 사용자 ID에는 SAF STARTED 프로파일이 필요합니다 (예:

```
RDEF STARTED BBGZANGL.* UACC(NONE) STDATA(USER(WLPUSER))
SETROPTS RACLIST(STARTED) REFRESH
```

CICS Liberty JVM 서버가 실행되는 사용자 ID는 CICS 영역 사용자 ID입니다. 이 사용자 ID는 권한 부여된 서비스를 사용하기 위해 엔젤 프로세스에 연결할 수 있어야

합니다. CICS JVM 서버에서 실행 중일 때 지원되는 권한 부여된 서비스는 CICS Liberty 보안 특성으로 구현된 z/OS 사용자 레지스트리 서비스 및 SAF 권한 부여 서비스(SAFCRED)뿐입니다. 이 특성을 사용하지 않는 경우 엔젤 프로세스를 실행할 필요가 없습니다.

엔젤 프로세스에 대한 SERVER 프로파일을 작성하고 CICS 영역 사용자 ID로부터의 액세스를 허용하십시오. 이렇게 하면 Liberty JVM 서버가 엔젤 프로세스에 연결할 수 있습니다. 영역 사용자 ID가 REGION1인 CICS 영역이 엔젤에 연결할 수 있도록 허용하려면 다음을 수행하십시오.

```
RDEF SERVER BBG.ANGEL UACC(NONE)
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(REGION1)
```

SAF 권한이 부여된 사용자 레지스트리 서비스 및 SAF 권한 부여 서비스(SAFCRED)에 대한 SERVER 프로파일을 작성하고 CICS 영역 사용자 ID로부터의 액세스를 허용하십시오. 이렇게 하면 Liberty JVM 서버가 CICS Liberty 보안 특성에 필요한 권한 부여된 서비스에 액세스할 수 있습니다. 영역 사용자 ID가 REGION1인 CICS 영역이 CICS Liberty 보안 특성을 사용할 수 있도록 허용하려면 다음을 수행하십시오.

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER) ACCESS(READ) ID(REGION1)
```

SERVER 자원을 새로 고치십시오.

```
SETROPTS RACLIST(SERVER) REFRESH
```

권한 부여된 모듈 BBGZSAFM에 대한 SERVER 프로파일을 작성하고 CICS 영역 사용자 ID(REGION1)가 프로파일에 액세스할 수 있도록 허용하십시오. 이 조치를 통해 Liberty 서버가 z/OS의 권한 부여된 서비스를 사용할 수 있습니다. 영역 사용자 ID가 REGION1인 CICS 영역이 권한 부여된 모듈에 액세스할 수 있도록 허용하려면 다음을 수행하십시오.

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(REGION1)
```

IFAUSAGE 서비스(PRODMGR)에 대한 SERVER 프로파일을 작성하고 CICS 영역 사용자 ID로부터의 액세스를 허용하십시오. 이렇게 하면 CICS JVM 서버가 사용 가능/사용 불가능하게 될 때 Liberty JVM 서버가 IFAUSAGE에서 등록 및 등록 취소됩니다. 영역 사용자 ID가 REGION1인 CICS 영역이 IFAUSAGE에서 등록 및 등록 취소되도록 허용하려면 다음을 수행하십시오.

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.PRODMGR UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.PRODMGR CLASS(SERVER) ACCESS(READ) ID(REGION1)
```

자세한 정보는 Liberty 프로파일: z/OS의 프로세스 유형의 내용을 참조하십시오.

Liberty JVM 서버에서 사용자 인증

Liberty JVM 서버에서 실행되는 모든 웹 애플리케이션에 대해 CICS 보안을 구성할 수 있지만 이러한 웹 애플리케이션은 보안 제한조건이 포함되어 있는 경우에만 사용자를 인증합니다. 애플리케이션 개발자가 동적 웹 프로젝트 또는 OSGi 애플리케이션 프로젝트의 전개 설명자(web.xml)에 보안 제한조건을 정의합니다. 보안 제한조건은 보호될 대상(URL) 및 그 역할을 정의합니다.

<login-config> 요소는 사용자가 웹 컨테이너에 액세스하는 방법 및 인증에 사용되는 방법을 정의합니다. 지원되는 방법은 HTTP 기본 인증, 양식 기반 인증 또는 SSL 클라이언트 인증입니다. CICS의 애플리케이션 보안을 정의하는 방법에 대한 세부사항은 CICS Explorer SDK에 설명되어 있습니다. 다음은 web.xml 요소의 예제입니다.

```
<!-- 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>
```

Liberty 보안을 사용하여 CICS에서 인증된 태스크는 Liberty 애플리케이션 보안 메커니즘에서 파생된 사용자 ID를 사용하여 CICS의 트랜잭션 및 자원 보안 검사에 권한을 부여할 수 있습니다. CICS 사용자 ID는 다음 기준에 따라 결정됩니다.

1. Liberty 애플리케이션 보안 인증.

   SAF 사용자 레지스트리와의 통합은 CICS Liberty 보안 특성의 일부이며 WebSphere Application Server Liberty 프로파일에서 지원되는 모든 애플리케이션 보안 메커니즘은 CICS에서 지원됩니다. 여기에는 HTTP 기본 인증, 양식 로그인, SSL 클라이언트 인증서 인증 또는 사용자 정의 로그인 모듈 또는 TAI(Trust Association Interceptor)를 사용한 ID 어설션이 포함됩니다. Liberty에서 인증된 모든 SAF 사용자 ID에는 APPL 클래스의 Liberty JVM 서버 APPLID에 대한 읽기 액세스 권한이 부여되어야 합니다. 해당 이름은 Liberty 서버 구성 파일 server.xml에 있는 safCredentials 요소의 profilePrefix 설정에 따라 결정됩니다.

   ```
   <safCredentials profilePrefix="BBGZDFLT"/>
   ```

APPL 클래스는 CICS 터미널 사용자가 특정 CICS 영역에 대한 액세스를 제어하는 데 사용되며 Liberty JVM 서버는 보안 요구사항에 따라 CICS APPLID와 동일한 프로파일을 사용할 수 있습니다. 이 요소를 지정하지 않은 경우 BBGZDFLT의 기본 profilePrefix가 사용됩니다.

APPLID를 정의해야 하고 사용자는 해당 APPLID에 액세스할 수 있어야 합니다. APPLID를 BBGZDFLT로 구성하고 활성화하려면 다음을 수행하십시오.

```
RDEFINE APPL BBGZDFLT UACC(NONE)
SETROPTS CLASSACT(APPL)
```

사용자에게는 인증을 위해 APPLID에 대한 읽기 액세스 권한이 부여되어야 합니다. 사용자 AUSER가 BBGZDFLT APPLID에서 인증을 받을 수 있도록 허용하려면 다음을 수행하십시오.

```
PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(AUSER)
```

Liberty SAF의 인증되지 않은 사용자 ID에게는 APPLID에 대한 읽기 액세스 권한이 부여되어야 합니다. SAF의 인증되지 않은 사용자 ID는 Liberty 서버 구성 파일 server.xml에 있는 safCredentials 요소에서 지정할 수 있습니다.

```
<safCredentials unauthenticatedUser="WSGUEST"/>
```

이 요소를 지정하지 않은 경우 기본 unauthenticatedUser는 WSGUEST입니다. SAF의 인증되지 않은 사용자 ID WSGUEST가 BBGZDFLT APPLID에 대한 읽기 액세스가 가능하도록 하려면 다음을 수행하십시오.

```
PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(WSGUEST)
```

자세한 정보는 Liberty 프로파일: WZSSAD를 사용하여 z/OS 보안 자원 액세스의 내용을 참조하십시오.

2. 인증되지 않은 대상이 Liberty에서 제공되면 URIMAP에 정의된 USERID가 사용됩니다.
3. USERID가 URIMAP에 정의되어 있지 않으면 요청이 CICS 기본 사용자 ID 아래에서 실행됩니다.

참고: 그러나 Liberty에서 인증을 얻을 때까지 CICS 트랜잭션 첨부 처리 중에 Liberty 트랜잭션에 대한 보안 처리가 지연되는 방식으로 인해 CICS 모니터링 기능(CMF) 레코드, z/OS WLM(Workload Manager) 등급, 태스크 연관 데이터 및 XAPADMGR 종료에 대한 UEPUSID 글로벌 사용자 종료 필드에서 사용되는 사용자 ID가 항상 영역 사용자 ID가 됩니다.

Liberty는 인증된 사용자 ID를 캐시하고 CICS와 달리 캐시 기간 내에 만기 사용자 ID를 확인하지 않습니다. 표준 Liberty 구성 프로세스를 사용하여 캐시 제한시간을 구성할 수 있습니다. Liberty 프로파일에서 인증 캐시 구성을 참조하십시오.

Liberty JVM 서버에서 애플리케이션을 실행하도록 사용자에게 권한 부여

사용자 ID에게 Liberty JVM 서버의 트랜잭션을 실행하도록 권한을 부여하기 위해 CICS 트랜잭션 및 자원 보안을 사용하거나 JEE 애플리케이션 보안 역할을 사용하여 JEE 애플리케이션에 대한 액세스를 부여할 수 있습니다.

이 태스크 정보

CICS 보안을 사용하면 기존 보안 프로시저를 사용할 수 있지만 개별 웹 애플리케이션이 다른 URIMAP에서 액세스되는 반면에 역할 기반 보안을 사용하면 다른 JEE 애플리케이션 서버에서 기존 표준 JEE 보안 정의를 재사용할 수 있습니다. 187 페이지의 『Liberty JVM 서버에서 사용자 인증』의 내용을 참조하십시오.CICS 권한을 단독으로 사용하려면 server.xml의 애플리케이션 정의에서 특별 대상 ALL_AUTHENTICATED_USERS를 사용하여 추가 Liberty 역할 기반 확인을 사용하지 않도록 선택할 수 있습니다. CICS 번들에 Liberty 애플리케이션을 전개하는 경우 CICS가 자동으로 이 애플리케이션을 구성합니다.

```
<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>
```

이 특별 대상을 사용하여 웹 애플리케이션 전개 설명자(web.xml)의 모든 URL에 cicsAllAuthenticated 역할 액세스를 부여하면 인증된 사용자 ID로 웹 애플리케이션에 액세스할 수 있으며 트랜잭션에 대한 권한이 CICS 트랜잭션 보안을 통해 제어되어야 합니다. 애플리케이션을 dropins 디렉토리에 바로 전개할 경우 dropins가 보안을 지원하지 않으므로 CICS 보안을 사용하도록 구성되지 않습니다.

CICS 트랜잭션 또는 자원 보안을 사용하려면 다음 단계를 수행해야 합니다.

프로시저

1. 각 웹 애플리케이션에 대해 JVMSERVER 유형의 URIMAP을 정의하십시오. 일반적으로 애플리케이션을 구성하는 servlet 세트로 트랜잭션 ID의 범위를 지정하기 위해 웹 애플리케이션의 일반 컨텍스트 루트(URI)와 일치하도록 URIMAP을 지정할 수 있습니다. 또는 보다 정확한 URI를 사용하는 다른 트랜잭션에서 개별 servlet을 실행하도록 선택할 수 있습니다.
2. 웹 애플리케이션의 모든 사용자에게 CICS 트랜잭션 또는 자원 보안 프로파일을 사용하여 URIMAP에 지정된 트랜잭션을 사용할 수 있는 권한을 부여하십시오.

JEE 애플리케이션 역할 보안

JEE 애플리케이션 역할 보안은 사용하려는 권한 유형에 따라 다양한 방식으로 구성될 수 있습니다. 분산 시스템의 경우 기본 레지스트리 또는 LDAP 레지스트리는 일반적으로 애플리케이션 특정 <application-bnd> 요소와 함께 사용되어 이러한 레지스트리의 사용자를 '역할'에 맵핑합니다. 애플리케이션의 전개 설명자에 따라 어떤 역할이 어떤 애플리케이션의 파트에 액세스할 수 있는지가 결정됩니다.

이 태스크 정보

z/OS에는 추가 레지스트리 유형, 즉 SAF 레지스트리가 있습니다.
cicsts:security-1.0 특성이 설치된 경우 Liberty JVM 서버는 암시적으로 이 유형을 인증에 사용합니다. 필요한 경우 이 유형을 인증에 사용할 수도 있습니다. SAF 레지스트리 유형에는 "사용자 대 역할" 맵핑(EJB 역할)에 대한 지원이 포함됩니다. 따라서 맵핑 정보를 SAF 레지스트리 자체에서 직접 가져올 수 있습니다.

Liberty JVM 서버에서 SAF 권한 없이 JEE 역할을 사용하려는 경우 CICS 번들을 사용하여 애플리케이션을 설치할 수 없습니다. CICS 번들이 설치된 애플리케이션이 자동으로 <application-bnd> 요소를 작성하고 특별 대상 ALL_AUTHENTICATED_USERS를 사용하므로 사용자가 직접 요소를 정의할 수 없게 됩니다. 대신 server.xml에서 직접 <application> 요소를 작성하고 필요한 역할 및 사용자로 <application-bnd>를 구성해야 합니다.

하지만 JEE 역할 및 SAF 권한을 사용하는 경우 계속해서 CICS 번들을 사용하여 웹 애플리케이션의 라이프 사이클을 지정할 수 있습니다. Liberty는 SAF 레지스트리에서 판별된 역할 맵핑을 사용하여 <application-bnd>를 무시합니다. 역할 맵핑은 EJB 역할에 속하는 '사용자'를 통해 결정됩니다.

프로시저

1. <safAuthorization id="saf"/> 요소를 server.xml에 추가하십시오.
2. 설명된 접두부 체계를 참조하여 필요한 EJB 역할을 작성하십시오.
3. 사용자를 해당 EJB 역할에 추가하십시오.

   기본적으로 SAF 권한이 사용되는 경우 애플리케이션이 <profile_prefix>.<resource>.<role> 패턴을 사용하여 사용자가 역할에 포함되어 있는지 확인합니다. profile_prefix는 BBGZDFLT로 기본 설정되지만 <safCredential> 요소를 사용하여 수정할 수 있습니다. 자세한 정보는 의 내용을 참조하십시오.

   역할 맵핑 환경 설정은 <safRoleMapper profilePattern="myprofile.%resource%.%role%" toUpperCase="true"/>로 기본 설정되는 server.xml의 <safRoleMapper> 요소를 사용하여 수정할 수 있습니다.

그런 다음 사용자는 다음 RACF 명령을 사용하여 특정 EJB 역할에 권한을 부여할 수 있습니다. 여기서 WEBUSER는 인증된 사용자 ID입니다.

```
RDEFINE EJBROLE BBGZDFLT.MYAPP.ROLE UACC(NONE)
PERMIT BBGZDFLT.MYAPP.ROLE CLASS(EJBROLE) ACCESS(READ) ID(WEBUSER)
```

결과

역할 및 해당 역할 내의 사용자를 정의하여 CICS 보안 및/또는 JEE 역할 보안을 통해 웹 애플리케이션에 대한 액세스 권한을 부여할 수 있습니다.

Liberty JVM 서버에 대한 SSL 구성

통신은 SSL(Secure Sockets Layer) 프로토콜을 사용하여 보안 설정됩니다. SSL 프로토콜은 인증, 데이터 서명 및 데이터 암호화를 포함한 전송 계층 보안을 제공하여 클라이언트와 서버 간의 보안 통신을 보장합니다. SSL을 데이터 암호화에 사용하고 선택적으로 클라이언트 인증서를 사용하여 서버를 인증하도록 Liberty JVM 서버를 구성할 수 있습니다. 클라이언트 인증서는 Java 키 저장소 또는 SAF Keyring에 저장될 수 있습니다.

이 태스크 정보

Liberty JVM 서버에서 SSL을 사용하려면 ssl-1.0 Liberty 특성, 키 저장소 및 HTTPS 포트가 필요합니다. SSL을 사용 가능하게 하는 두 가지 대체 방법이 있습니다.

- Liberty 프로파일의 구성 파일인 server.xml 자동 구성. CICS는 `server.xml` 파일을 자동으로 작성하고 갱신합니다. 자동으로 구성하면 항상 Java 키 저장소가 작성됩니다.
- 수동으로 server.xml 구성. server.xml 파일을 편집하여 필수 요소 및 값을 추가합니다. SAF Keyring을 사용하려는 경우 수동 프로시저에 따라야 합니다.

Liberty JVM 서버에 대한 모든 웹 요청은 CICS 소켓 도메인이 아니라 TCP/IP 소켓 및 SSL 처리에 대한 JVM 지원을 사용함을 이해하는 것이 중요합니다.

프로시저

1. 자동 구성을 사용하여 SSL을 구성하려면 다음 단계를 수행하십시오.
   a. JVM 프로파일에서 JVM 시스템 등록 정보 **-Dcom.ibm.cics.jvmserver.wlp.autoconfigure**를 사용하여 자동 구성이 사용으로 설정되었는지 확인하십시오.
   b. JVM 프로파일에서 **-Dcom.ibm.cics.jvmserver.wlp.server.https.port**를 사용하여 SSL 포트를 설정하십시오.
   c. JVM 서버를 다시 시작하십시오.
2. SSL을 수동으로 구성하려면 다음을 수행하십시오.

a. server.xml 파일을 편집하여 SSL 특성과 키 저장소를 추가하고 HTTPS 포트를 설정하십시오. Liberty 프로파일과 함께 제공되는 securityUtility 도구를 사용하여 Java 키 저장소 암호를 암호화할 수 있습니다. SSL에 대한 지원 추가에 대한 자세한 정보는 라이브러리 프로파일용 SSL 통신 사용의 내용을 참조하십시오.

b. SAF Keyring을 수동으로 추가하려면 먼저 SAF Keyring을 작성하고 필요한 인증서를 가져오십시오. server.xml 파일을 편집하여 SSL 특성과 키 저장소를 추가하고 HTTPS 포트를 설정하십시오. 암호 필드는 SAF Keyring 액세스에 사용되지 않으며 "password"로 설정되어야 합니다. SAF Keyring은 URL safkeyring://USERID/KeyringName에 지정되어야 합니다. 여기서 사용자 ID는 "선택항목"입니다. 다음은 SAF Keyring 구성의 예제입니다.

```
<featureManager>
  ...
  <feature>ssl-1.0ssl-1.0</feature>
</featureManager>
...
<httpEndpoint hosthost="*" httpPort="9080" httpsPort="9443"
id="defaultHttpEndpoint"/>
...
<keyStore fileBased="false" id="defaultKeyStore"
location="safkeyring:///CICSKeyring" password="password" readOnly="true" type="JCERACFKS"/>
```

결과

Liberty JVM 서버에 대한 SSL을 구성했습니다.

Java 보안 관리자 사용

Java 애플리케이션은 기본적으로 Java API에 대해 요청되는 활동에 대한 보안 제한사항을 적용하지 않습니다. Java 애플리케이션이 잠재적으로 안전하지 않은 조치를 수행하지 못하도록 보호하기 위해 Java 보안을 사용하려면 애플리케이션이 실행되는 JVM에 대해 보안 관리자를 사용 가능하도록 설정하면 됩니다.

이 태스크 정보

보안 관리자는 코드 소스에 지정된 권한(시스템 액세스 권한) 세트인 보안 정책을 강제 실행합니다. 기본 정책 파일은 Java 플랫폼과 함께 제공됩니다. 그러나 Java 보안이 활성화될 때 Java 애플리케이션을 CICS에서 실행하려면 애플리케이션을 실행하는 데 필요한 권한을 CICS에 제공하는 추가 정책 파일을 지정해야 합니다.

보안 관리자를 사용하는 각 JVM 유형에 이 추가 정책 파일을 지정해야 합니다. CICS는 고유한 정책을 작성하기 위해 사용할 수 있는 몇 가지 예제를 제공합니다.

참고: Liberty JVM 서버에서는 Java 보안 관리자 사용이 지원되지 않습니다.

- OSGi 보안 에이전트 예제는 보안 프로파일을 포함하는 프로젝트에 OSGi 미들웨어 번들 com.ibm.cics.server.examples.security를 작성합니다. 이 프로파일은 설치된 프레임워크의 모든 OSGi 번들에 적용됩니다.
- example.permissions 파일에는 애플리케이션이 System.exit() 메소드를 사용하지 않는지에 대한 확인을 포함하여 JVM 서버의 실행 애플리케이션에 고유한 권한이 포함됩니다.
- OSGi 번들을 전개하는 경우 zFS의 디렉토리에 대한 읽기 및 실행 액세스 권한이 CICS에 필요합니다.

JVM 서버의 OSGi 프레임워크에서 실행되는 애플리케이션의 경우 다음을 수행하십시오.

프로시저

1. CICS Explorer SDK에서 플러그인 프로젝트를 작성하고 제공된 OSGi 보안 에이전트 예제를 선택하십시오.
2. 프로젝트에서 example.permissions 파일을 선택하여 보안 정책에 대한 권한을 편집하십시오.
   a. CICS zFS 및 DB2 설치 디렉토리가 올바르게 지정되었는지 검사하십시오.
   b. 필요에 따라 다른 권한을 추가하십시오.
3. OSGi 번들을 zFS의 적합한 디렉토리(예: /u/bundles)에 전개하십시오.
4. JVM 서버의 JVM 프로파일을 편집하여 OSGi 번들을 다른 번들보다 먼저 OSGI_BUNDLES 선택항목에 추가하십시오.

   ```
   OSGI_BUNDLES=/u/bundles/
   com.ibm.cics.server.examples.security_1.0.0.jar
   ```

5. JVM 프로파일에 다음 Java 등록 정보를 추가하여 보안을 사용 가능하게 설정하십시오.

   ```
   -Djava.security.policy=all.policy
   ```

6. JVM 프로파일에 다음 Java 환경 변수를 추가하여 OSGi 프레임워크에서 보안을 사용 가능하게 설정하십시오.

   ```
   org.osgi.framework.security=osgi
   ```

7. OSGi 프레임워크가 Java 2 보안을 시작하도록 허용하려면 다음 정책을 추가하십시오.

   ```
   grant { permission java.security.AllPermission; };
   ```

8. 변경사항을 저장하고 JVMSERVER 자원으로 JVM 서버에 미들웨어 번들을 설치하십시오.
9. 옵션: Java 2 보안을 활성화하십시오.

a. Java 2 보안 정책 메커니즘을 활성화하려면 해당 JVM 프로파일에 추가하십시오. 또한 Java 2 보안 정책을 편집하여 적절한 권한을 부여해야 합니다.

b. Java 2 보안 정책 메커니즘을 활성화하여 Java 애플리케이션에서 JDBC 또는 SQLJ를 사용하려면 IBM Data Server Driver for JDBC and SQLJ를 사용하십시오.

c. Java 2 보안 정책 메커니즘을 활성화하려면 JVM 프로파일을 편집하십시오. 전개에서 Java 보안 관리자 사용에서는 Java 2 보안 정책을 설정하는 방법을 설명합니다.

d. 예제 1에 표시되는 행을 추가하여, JDBC 드라이버에 권한을 부여하도록 Java 2 보안 정책을 편집하십시오. db2*xxx* 대신 모든 DB2 라이브러리가 있는 디렉토리를 지정하십시오. 권한은 이 레벨 아래 모든 디렉토리와 파일에 적용됩니다. 결과적으로 JDBC와 SQLJ를 사용할 수 있습니다.

e. 예제 2에 표시되는 행을 추가하여, 읽기 권한을 부여하도록 Java 2 보안 정책을 편집하십시오. 읽기 권한을 추가하지 않은 경우 Java 프로그램을 실행하면 AccessControlExceptions 및 unpredictable 결과가 생성됩니다. Java 2 보안 정책과 JDBC 및 SQLJ를 사용할 수 있습니다.

예제 1:

```
grant codeBase "file:/usr/lpp/db2xxx/-" {
  permission java.security.AllPermission;
};
```

예제 2:

```
grant {

// allows anyone to read properties
permission java.util.PropertyPermission "*", "read";

};
```

결과

Java 애플리케이션이 호출되면 JVM이 클래스의 코드 소스를 판별하며 클래스에 적절한 권한을 부여하기 전에 보안 정책을 확인합니다.

제 8 장 Java 애플리케이션 문제점 해결

Java 애플리케이션 관련 문제점이 있는 경우 CICS가 제공하는 진단과 JVM을 사용하여 문제점의 원인을 판별할 수 있습니다.

이 태스크 정보

CICS는 Java 관련 문제점 진단에 도움을 줄 수 있는 통계, 메시지, 추적 정보를 제공합니다. Java에 제공되는 진단 도구와 인터페이스는 CICS보다 JVM에서 발생하는 상황에 대한 자세한 정보를 제공할 수 있습니다. CICS는 JVM에서 많은 활동을 인식할 수 없기 때문입니다.

JVM에 대한 실시간 및 오프라인 분석을 수행하며 무료로 제공되는 도구(예: JConsole 및 IBM Health Center)를 사용할 수 있습니다. 세부사항은 z/OS에서 Java 버전 7용 사용자 안내서의 진단 도구 사용의 내용을 참조하십시오.

Liberty JVM 서버에서 실행되는 웹 애플리케이션의 문제점을 해결하려면 213 페이지의 제 9 장 『Liberty JVM 서버 및 Java 웹 애플리케이션 문제점 해결』의 내용을 참조하십시오.

프로시저

1. JVM 서버를 시작할 수 없는 경우 Java 설치 설정이 올바른지 확인하십시오. 문제점을 일으킬 수 있는 원인을 판별하려면 JVM에 대한 `stderr` 파일의 오류와 CICS 메시지를 사용하십시오.
   a. Java SDK의 올바른 버전이 설치되고 CICS가 z/OS UNIX에서 해당 버전에 액세스할 수 있는지 확인하십시오. 지원되는 SDK 목록은 상위 레벨 언어 지원을 참조하십시오.
   b. **USSHOME** 시스템 초기화 매개변수가 CICS 영역에서 설정되었는지 확인하십시오. 이 매개변수는 z/OS UNIX에서 파일의 홈을 지정합니다.
   c. **JVMPROFILEDIR** 시스템 초기화 매개변수가 CICS 영역에서 올바르게 설정되었는지 확인하십시오. 이 매개변수는 z/OS UNIX에서 JVM 프로파일의 위치를 지정합니다.
   d. CICS 영역에 JVM 프로파일이 있는 z/OS UNIX 디렉토리에 대한 읽기 및 실행 액세스 권한이 있는지 확인하십시오.
   e. CICS 영역에 JVM의 작업 디렉토리에 대한 쓰기 액세스 권한이 있는지 확인하십시오. 이 디렉토리는 JVM 프로파일의 `WORK_DIR` 선택항목에서 지정됩니다.
   f. JVM 프로파일의 `JAVA_HOME` 선택항목이 Java SDK를 포함하는 디렉토리를 가리키는지 확인하십시오.

g. SDFJAUTH가 CICS 시동 JCL의 STEPLIB 연결에 있는지 확인하십시오.

h. WebSphere MQ 또는 DB2 DLL 파일을 사용하는 경우 CICS에서 이러한 파일의 64비트 버전을 사용할 수 있는지 확인하십시오.

i. Language Environment 인클레이브를 구성하기 위해 DFHAXRO를 수정하는 경우 런타임 선택항목이 200바이트를 초과하지 않고 선택항목이 올바른지 확인하십시오. CICS는 Language Environment로 선택항목을 전달하기 전에 지정된 선택항목의 유효성을 검사하지 않습니다. Language Environment의 오류 메시지에 대한 SYSOUT을 확인하십시오.

2. 설정이 올바른 경우 진단 정보를 수집하여 애플리케이션과 JVM에 대한 영향을 판별하십시오.

   a. JVM 프로파일에 `PRINT_JVM_OPTIONS=YES`를 추가하십시오. 이 선택항목을 지정하면 클래스 경로 내용을 포함하여 시동 시 JVM으로 전달되는 모든 선택항목이 SYSPRINT에 인쇄됩니다. 해당 프로파일의 이 선택항목으로 JVM이 시작될 때마다 정보가 생성됩니다.

   b. `dfhjvmout` 및 `dfhjvmerr` 파일에서 JVM의 오류 메시지와 정보를 확인하십시오. 이러한 파일은 JVM 프로파일에서 `WORK_DIR` 선택항목으로 지정된 디렉토리에 있습니다. `STDOUT` 및 `STDERR` 선택항목이 JVM 프로파일에서 변경된 경우 파일 이름이 다를 수 있습니다.

3. 애플리케이션에 장애가 발생하거나 성능이 저하되는 경우 애플리케이션을 디버그하십시오.

   - `java.lang.ClassNotFoundException` 오류를 수신하고 트랜잭션이 AJ05 코드로 이상 종료되는 경우 애플리케이션이 OSGi 프레임워크에서 IBM 또는 벤더 클래스에 액세스하지 못할 수 있습니다. 이 문제점 해결 방법에 대한 자세한 정보는 업그레이드의 JVM 서버의 Java 애플리케이션 업그레이드의 내용을 참조하십시오.
   - 애플리케이션 트랜잭션을 디버그하려면 CEDX 트랜잭션을 사용하십시오. Liberty JVM 서버의 경우 URI 맵을 사용하여 애플리케이션 트랜잭션에 대한 인바운드 애플리케이션 요청을 일치시키려면 해당 트랜잭션을 디버그하십시오. 기본 트랜잭션 CJSA를 사용하는 경우 DFHEDFTC 트랜잭션 클래스에서 MAXACTIVE 속성을 1로 설정해야 합니다. 이 설정은 여러 CJSA 태스크가 실행되어 잘못된 트랜잭션을 디버그할 수 있으므로 필요합니다. 프로덕션 환경에서는 CJSA 트랜잭션에서 CEDX를 사용하지 마십시오.
   - JVM 서버에서 디버거를 사용하려면 JVM 프로파일에서 일부 선택항목을 설정해야 합니다. 자세한 정보는 208 페이지의 『Java 애플리케이션 디버깅』의 내용을 참조하십시오.

4. 메모리 부족 오류를 수신하는 경우, 64비트 저장영역이 충분하지 않거나 애플리케이션에 메모리 누수가 발생했거나 힙 크기가 충분하지 않을 수 있습니다.

a. CICS 통계 또는 IBM Health Center와 같은 도구를 사용하여 JVM을 모니터하십시오. 애플리케이션에 메모리 누수가 발생하면 힙이 소진될 때까지 시간 경과에 따라 사용공간 정리 후 남아 있는 활성 데이터의 양이 점진적으로 증가합니다. JVM 서버 통계는 마지막 사용공간 정리 후 힙 *크기*와 힙의 최대 *크기*를 보고합니다. 자세한 정보는 165 페이지의 『IBM Health Center를 사용하여 Java 애플리케이션 분석』의 내용을 참조하십시오.

b. Language Environment에 대한 저장영역 보고서를 실행하여 저장영역 *크기*가 충분한지 여부를 확인하십시오. 자세한 정보는 174 페이지의 『JVM용 Language Environment 인클레이브 저장영역』의 내용을 참조하십시오.

5. Java 애플리케이션을 설치하거나 실행할 때 인코딩 오류를 수신하는 경우 충돌하거나 지원되지 않는 코드 페이지 조합을 설정했을 수 있습니다. z/OS의 JVM은 일반적으로 파일 인코딩을 위해 EBCDIC 코드 페이지를 사용합니다. 기본값은 IBM1047(또는 cp1047)이지만 JVM은 필요한 경우 파일 인코딩을 위해 다른 코드 페이지를 사용할 수 있습니다. CICS에서는 EBCDIC 코드 페이지가 문자 데이터를 처리해야 하며 모든 JCICS 호출이 EBCDIC 코드 페이지를 사용해야 합니다. 코드 페이지는 CICS 영역의 **LOCALCCSID** 시스템 초기화 매개변수에 설정됩니다.

a. JVM 서버 로그를 보고 **LOCALCCSID** 값과 관련된 경고 메시지가 실행되었는지 여부를 확인하십시오. 이 매개변수가 비EBCDIC 코드 페이지, JVM이 지원하지 않는 코드 페이지 또는 지원되지 않는 EBCDIC 코드 페이지(예: 930)로 설정되면 JVM 서버가 cp1047을 사용합니다.

b. JCICS 호출은 **LOCALCCSID** 시스템 초기화 매개변수에 지정된 코드 페이지를 사용합니다. 애플리케이션이 다른 코드 페이지를 예상하는 경우에는 인코딩 오류가 수신됩니다. JCICS에 다른 코드 페이지를 사용하려면 JVM 프로파일에서 **-Dcom.cics.jvmserver.override.ccsid=** 매개변수를 설정하십시오.

c. JVM 프로파일에서 **-Dcom.cics.jvmserver.override.ccsid=** 매개변수를 사용하는 경우 CCSID가 EBCDIC 코드 페이지인지 확인하십시오. JCICS 호출을 사용하는 경우 애플리케이션이 EBCDIC를 사용해야 합니다.

d. Axis2 JVM 서버에서 SOAP 처리를 실행하는 경우 **-Dfile.encoding** JVM 등록 정보가 EBCDIC 등록 정보를 지정하는지 확인하십시오. EBCDIC 이외 코드 페이지(예: UTF-8)를 지정하는 경우 웹 서비스 요청이 실패하고 응답에 손상된 데이터가 포함됩니다.

6. 메시지 DFHSJ0904가 표시되고 다음 예제와 유사한 문제점이 보고되면 JVM 서버의 스레드 한계에 도달했을 수 있습니다.

```
예외 'java.lang.Exception: CICSThreadExecutor: 태스크 45921에 대한
작업이 발견되지 않았습니다.
이 태스크가 시작된 작업의 제한시간이 이미 초과되었습니다.'
```

다음을 수행하여 문제점을 해결해 보십시오.

- JVMSERVER 자원의 **THREADLIMIT** 값을 늘리십시오.

- **THREADLIMIT**가 이미 허용되는 최대값으로 설정되어 있으면 한 대의 JVM 서버가 처리할 수 있는 것보다 더 많은 작업을 실행하려고 시도할 수 있습니다. 다중 JVM 서버 또는 다중 영역 간 워크로드의 밸런스를 조정하는 것을 고려하십시오.

  또는 다른 제한조건으로 CICS 시스템이 응답하지 않을 수 있습니다. 성능 문제점을 진단하려면 표준 프로시저를 따르십시오. 성능 개선에서 CICS 시스템의 성능 개선의 내용을 참조하십시오.

다음에 수행할 작업

문제점의 원인을 수정할 수 없으면 IBM 지원 센터에 문의하십시오. Java 문제점 보고를 위해서는 MustGather에 나열된 필수 정보를 제공해야 합니다.

199 페이지의『Java에 대한 진단』
CICS 진단 정보의 일반적인 소스에는 대부분 Java 애플리케이션에 적용되는 정보가 포함됩니다. CICS가 제공하는 정보뿐 아니라 여러 JVM 특정 인터페이스를 사용하여 문제점을 판별할 수 있습니다.

201 페이지의『JVM stdout, stderr, JVMTRACE 및 덤프 출력의 위치 제어』
JVM에서 실행되는 Java 애플리케이션의 출력은 일반적으로 JVM에 대한 JVM 프로파일에서 STDOUT 및 STDERR 선택항목으로 이름이 지정되는 z/OS UNIX 파일에 기록됩니다. z/OS UNIX에서 JVM의 작업 디렉토리에 JAVADUMP 파일이 기록되며 JAVA_DUMP_TDUMP_PATTERN 선택항목으로 이름이 지정된 파일에 자세한 Java TDUMP가 기록됩니다. 이러한 파일 이름은 대부분 런타임에 사용자 정의되어 해당 이름을 생성한 JVM을 고유하게 식별할 수 있습니다. JVMTRACE는 JVM 서버 시작 및 종료 시와 Java 애플리케이션 실행 시 CICS Java 추적이 작성되는 z/OS UNIX 파일의 이름을 지정합니다. 애플리케이션 개발 과정에서 Java 클래스를 사용하여 JVM의 출력 및 JVM 내부의 메시지 경로를 재지정할 수도 있습니다.

206 페이지의『JVM 서버의 OSGi 로그 파일 관리』
OSGi 프레임워크는 JVM 서버의 작업 디렉토리에 있는 로그 파일 세트에 오류를 기록합니다. 기본값이 환경에 적합하지 않은 경우 각 JVM 서버의 로그 파일 수와 크기를 관리할 수 있습니다.

207 페이지의『JVM 서버에 대한 CICS 구성요소 추적』
CICS는 Java로 생성된 로깅뿐 아니라 0, 1, 2 추적 레벨에 대한 SJ(JVM) 및 AP 도메인에 표준 추적점을 제공합니다. 이러한 추적점은 CICS가 JVM 서버를 설정 및 관리하기 위해 수행하는 조치를 추적합니다.

207 페이지의『JVM 서버에 대한 추적 활성화 및 관리』
SJ 및 AP 구성요소 추적을 켜고 JVM 서버 추적을 활성화할 수 있습니다. 내부 추

적 테이블에는 적은 양의 추적이 기록되지만 Java는 각 JVM 서버마다 zFS에서 고유한 파일에 로깅 정보를 기록합니다. 이 파일은 줄을 바꾸지 않으므로 zFS의 경우 해당 *크기*를 관리해야 합니다.

208 페이지의『Java 애플리케이션 디버깅』
CICS의 JVM은 Java 2 플랫폼에서 제공되는 표준 디버깅 메커니즘인 JPDA(Java Platform Debugger Architecture)를 지원합니다. 이 구조는 JVM에 원격 디버거 연결을 허용하는 API 세트를 제공합니다.

210 페이지의『CICS JVM 플러그인 메커니즘』
CICS는 JVM의 표준 JPDA 디버그 인터페이스 이외에 애플리케이션 디버깅에 유용한 인터셉션 지점(플러그인) 세트를 CICS Java 미들웨어에 제공합니다. 이러한 플러그인을 사용하면 애플리케이션 Java 코드가 실행되기 직전과 직후에 추가 Java 프로그램을 삽입할 수 있습니다.

Java에 대한 진단

CICS 진단 정보의 일반적인 소스에는 대부분 Java 애플리케이션에 적용되는 정보가 포함됩니다. CICS가 제공하는 정보뿐 아니라 여러 JVM 특정 인터페이스를 사용하여 문제점을 판별할 수 있습니다.

Java용 CICS 진단 도구

CICS에는 실행 Java 애플리케이션에서 수집할 수 있는 통계 및 모니터링 데이터가 있습니다. 오류가 발생하면 해당 로그에 트랜잭션 이상 종료 및 메시지가 기록됩니다. JVM(SJ) 도메인에 적용되는 이상 종료 및 메시지 목록은 참조 -> 진단의 CICS 메시지의 내용을 참조하십시오. Java와 관련된 메시지는 DFHSJxxxx 형식입니다.

추적을 활성화하여 추가 진단 정보를 생성할 수도 있습니다. JVM 도메인에 대한 추적점은 참조에서 JVM 도메인 추적점 -> 진단에 나열됩니다.

초기화 이후 첫 번째 JVM이 CICS 영역에서 시작되면 사용하는 Java 버전을 보여주는 DFHSJ0207 메시지를 CICS가 실행합니다.

Java SDK는 JVM에서 발생하는 상황에 대한 자세한 정보를 제공하는 진단 도구와 인터페이스를 제공합니다. JVM의 메시지와 진단 정보는 JVM의 `stderr` 로그 파일에 기록됩니다. Java 문제점이 발생하면 항상 이 파일을 참조하십시오. 예를 들어, CICS가 JVM이 이상 종료되었음을 나타내는 메시지를 발행하는 경우 `stderr` 로그 파일은 진단 정보의 기본 소스입니다. 201 페이지의『JVM stdout, stderr, JVMTRACE 및 덤프 출력의 위치 제어』에서는 JVM으로부터의 출력 위치를 제어하는 방법 및 JVM 내부 메시지와 JVM에서 실행되는 Java 애플리케이션의 출력 경로를 재지정하는 방법을 알려줍니다.

CICS용 Java 애플리케이션을 개발하는 경우 CICS에서의 스레드 안전성과 트랜잭션 분리를 위한 요구사항을 고려해야 합니다. Java 애플리케이션이 최초 사용 시 올바르게 작동하지만 이후 사용 시 올바르게 작동하지 않는 경우 분리 이슈에 따른 문제점일 수 있습니다.

OSGi 진단 파일

OSGi 프레임워크는 JVM 서버에서 OSGi 번들 및 서비스 관련 문제점 해결에 도움을 주기 위해 사용할 수 있는 진단 파일을 zFS에서 생성합니다.

OSGi 캐시

OSGi 캐시는 JVM 서버의 *$WORK_DIR*/*applid*/*jvmserver*/configuration/org.eclipse.osgi 디렉토리에 있습니다. *$WORK_DIR*은 JVM 서버의 작업 디렉토리이고 *applid*는 CICS APPLID이며 *jvmserver*는 JVMSERVER 자원의 이름입니다. OSGi 캐시에는 프레임워크를 실행하는 데 필요한 프레임워크 메타데이터와 기타 정보가 포함됩니다. 캐시는 JVM 서버가 시동될 때 대체됩니다.

OSGi 로그

OSGi 프레임워크에서 오류가 발생하는 경우 JVM 서버의 *$WORK_DIR*/*applid*/*jvmserver*/configuration/ 디렉토리에 OSGi 로그가 작성됩니다. 파일 확장자는 .log입니다.

JVM 진단 도구

CICS 문서는 일부 Java 진단 도구 및 인터페이스에 대한 정보를 제공합니다.

- 207 페이지의 『JVM 서버에 대한 추적 활성화 및 관리』에서는 JVM 서버의 라이프사이클과 서버 내부에서 실행되는 태스크를 추적하기 위해 CETR 트랜잭션이 제공하는 구성요소 추적을 사용할 수 있는 방법을 설명합니다. JVM 서버는 보조 또는 GTF 추적을 사용하지 않습니다. 대신 각 JVM 서버에 대해 고유하게 이름이 지정된 zFS의 파일에 추적이 기록됩니다.
- 208 페이지의 『Java 애플리케이션 디버깅』에서는 원격 디거버를 사용하여 JVM에서 실행되는 Java 애플리케이션에 대한 애플리케이션 코드를 진행하는 방법을 설명합니다. CICS는 또한 CICS Java 미들웨어에 인터셉션 지점(또는 "플러그인") 세트를 제공하므로 디버깅, 로깅 또는 기타 목적으로 애플리케이션 Java 코드가 실행되기 직전과 직후에 추가 Java 프로그램이 삽입될 수 있습니다. 자세한 정보는 210 페이지의 『CICS JVM 플러그인 메커니즘』의 내용을 참조하십시오.

JVM에는 보다 많은 진단 도구와 인터페이스를 사용할 수 있습니다. JVM에 대한 문제점 판별에 사용할 수 있는 추가 기능에 대한 정보는 IBM SDK for z/OS, Java Technology Edition, 버전 7(문제점 해결 절 참조)의 내용을 참조하십시오. 다음 기능은 유용한 진단 정보를 제공합니다.

- JVM의 내부 추적 기능은 CICS가 제공하는 인터페이스를 실행하지 않고 직접 사용할 수 있습니다. *진단 안내서*는 내부 추적 기능을 제어하고 JVM 추적 정보를 다양한 대상에 출력하기 위해 사용할 수 있는 시스템 등록 정보에 대한 정보를 제공합니다. 이러한 시스템 등록 정보를 사용하여 JVM 내 메소드 또는 클래스에서 추적을 출력할 수 있으며 매개변수 값을 찾아 메소드 호출에서 유형을 리턴할 수 있습니다.
- JVM에서 메모리 누수가 발생하는 경우 JVM에서 힙 덤프를 요청할 수 있습니다. 힙 덤프는 JVM의 힙에 있는 모든 활성 오브젝트(사용 중인 오브젝트)의 덤프를 생성합니다. IBM Health Center 및 메모리 분석기 도구(IBM Support Assistant에서 사용 가능)를 사용하여 메모리 누수를 분석할 수도 있습니다. Java 도구에 대한 자세한 정보는 Java용 IBM 모니터링 및 진단 도구 - Health Center의 내용을 참조하십시오.
- IBM 64-bit SDK for z/OS, Java Technology Edition와 함께 제공되는 HPROF 프로파일러는 JVM에서 실행되는 애플리케이션에 대한 성능 정보를 제공하므로 프로그램에서 가장 많은 메모리 또는 프로세서 시간을 사용하는 파트를 확인할 수 있습니다.
- JVM은 모니터링, 프로파일링, RAS(Reliability, Availability, and Serviceability)를 위한 인터페이스를 제공합니다.

CICS 환경에만 적용되지 않는 IBM JVM에 사용 가능한 모든 인터페이스, 선택항목 또는 시스템 등록 정보와 함께 IBM JVM 문서를 정보의 기본 소스로 사용하십시오.

JVM stdout, stderr, JVMTRACE 및 덤프 출력의 위치 제어

JVM에서 실행되는 Java 애플리케이션의 출력은 일반적으로 JVM에 대한 JVM 프로파일에서 STDOUT 및 STDERR 선택항목으로 이름이 지정되는 z/OS UNIX 파일에 기록됩니다. z/OS UNIX에서 JVM의 작업 디렉토리에 JAVADUMP 파일이 기록되며 JAVA_DUMP_TDUMP_PATTERN 선택항목으로 이름이 지정된 파일에 자세한 Java TDUMP가 기록됩니다. 이러한 파일 이름은 대부분 런타임에 사용자 정의되어 해당 이름을 생성한 JVM을 고유하게 식별할 수 있습니다. JVMTRACE는 JVM 서버 시작 및 종료 시와 Java 애플리케이션 실행 시 CICS Java 추적이 작성되는 z/OS UNIX 파일의 이름을 지정합니다. 애플리케이션 개발 과정에서 Java 클래스를 사용하여 JVM의 출력 및 JVM 내부의 메시지 경로를 재지정할 수도 있습니다.

CICS JVM에 대한 표준 설정의 경우, JVM 프로파일에서 STDOUT 선택항목으로 이름이 지정되는 파일이 System.out 요청에 사용되고 STDERR 선택항목으로 이름이 지정되는 파일이 System.err 요청에 사용됩니다. 출력 파일은 JVM 프로파일에서 WORK_DIR 선택항목으로 이름이 지정되는 작업 디렉토리에 있는 z/OS UNIX 파일입니다.

stdout 및 stderr 파일에 고정 파일 이름을 지정할 수 있습니다. 그러나 고정 파일 이름을 사용하는 경우, 해당 JVM 프로파일로 작성된 모든 JVM의 출력이 동일한 파일에 추가되고 다른 JVM으로부터의 출력은 레코드가 없는 헤더와 연결됩니다. 이는 문제점 판별을 위해 유용하지 않습니다.

보다 나은 선택은 stdout 및 stderr 파일에 변수 파일 이름을 지정하는 것입니다. CICS 영역 수명 중에 개별 JVM 각각에 대해 파일을 고유하게 지정할 수 있습니다. 추가 식별 정보를 포함할 수도 있습니다.

- &APPLID; 기호를 사용하여 파일 이름에 CICS 영역 applid를 포함할 수 있습니다.

파일 이름의 다른 식별 정보에는 &DATE; 및 &TIME; 기호가 포함됩니다.

&DATE;는 Dyymmdd 양식에서 현재 날짜로 대체됩니다.

&TIME;은 Thhmmss 형식에서 현재 시간으로 대체됩니다.

각 JVM 서버에 고유한 출력 또는 덤프 파일을 작성하려면 &JVMSERVER; 기호를 사용하십시오. 이 기호를 사용하는 경우 JVMSERVER 자원의 이름이 런타임에서 대체됩니다.

JVM에서 출력된 JAVADUMP 파일의 위치는 JVM 프로파일에서 WORK_DIR 선택항목으로 이름이 지정된 z/OS UNIX의 작업 디렉토리입니다. JAVADUMP 파일은 이름의 시간 소인으로 고유하게 식별되므로 해당 파일에 맞게 이름을 사용자 정의할 수 없습니다.

JVMTRACE 선택항목의 값을 설정하지 않으면 CICS가 각 JVM 서버에 대한 고유한 추적 파일을 자동으로 작성합니다. CICS는 &APPLID; 및 &JVMSERVER; 기호와 JVM 서버가 시작되는 날짜 및 시간을 사용합니다. 이 파일은 WORK_DIR 선택항목으로 지정된 디렉토리에 작성됩니다.

JVM 주소 공간을 포함하는 자세한 메모리 덤프 출력을 포함하는 JVM의 TDUMP 출력은 데이터 세트 대상에 기록됩니다. 대상의 이름은 JVM 프로파일에서 JAVA_DUMP_TDUMP_PATTERN 선택항목으로 지정됩니다. 이 값의 &APPLID;, &DATE;, &TIME; 기호를 사용하여 개별 JVM에 고유한 이름을 작성할 수 있습니다 (CICS와 함께 포함된 샘플 JVM 프로파일 참조).

JVM은 JAVADUMP 또는 TDUMP를 생성할 때 해당 stderr 파일에 정보를 기록합니다. JAVADUMP 및 TDUMP 파일 내용에 대한 자세한 정보는 IBM SDK for z/OS, Java Technology Edition, 버전 7(문제점 해결 절 참조)의 내용을 참조하십시오.

애플리케이션을 개발할 때 JVM 프로파일에서 USEROUTPUTCLASS 선택항목을 사용하여 JVM 및 JVM 내부의 메시지 출력을 인터셉트하여 경로를 재지정하는 Java 클래스의 이름을 지정할 수 있습니다. 출력 레코드에 시간 소인과 헤더를 추가할 수 있으며

JVM에서 실행되는 개별 트랜잭션의 출력을 식별할 수 있습니다. CICS는 이러한 태스크를 수행하는 샘플 클래스를 제공합니다. 이 선택항목을 지정하면 JVM 성능에 부정적인 영향을 미치므로 프로덕션 환경에서는 사용해서는 안됩니다.

『JVM stdout 및 stderr 출력 경로 재지정』
애플리케이션을 개발하는 과정에서 개발자가 USEROUTPUTCLASS 선택항목을 사용하여 CICS 영역에서 JVM stdout 및 stderr 출력을 구분하고 해당 경로를 선택한 식별 가능한 대상으로 지정할 수 있습니다. Java 클래스를 사용하여 출력 경로를 재지정할 수 있으며 출력 레코드에 시간 소인과 헤더를 추가할 수 있습니다. 덤프 출력은 이 방법으로 인터셉트할 수 없습니다.

204 페이지의 『샘플 클래스 com.ibm.cics.samples.SJMergedStream 및 com.ibm.cics.samples.SJTaskStream』
CICS 요청을 작성할 수 있는 Java 애플리케이션 스레드의 경우 JVM으로부터의 출력을 인터셉트하고 트랜지언트 데이터 큐에 출력을 작성하여 JVM 활동과 CICS 활동을 상관시키는 로그를 작성할 수 있습니다.

206 페이지의 『Java 덤프 선택항목 제어』
-Xdump 선택항목은 JVM에 덤프 선택항목을 지정하기 위해 JVM 프로파일에서 사용할 수 있습니다.

JVM stdout 및 stderr 출력 경로 재지정

애플리케이션을 개발하는 과정에서 개발자가 USEROUTPUTCLASS 선택항목을 사용하여 CICS 영역에서 JVM stdout 및 stderr 출력을 구분하고 해당 경로를 선택한 식별 가능한 대상으로 지정할 수 있습니다. Java 클래스를 사용하여 출력 경로를 재지정할 수 있으며 출력 레코드에 시간 소인과 헤더를 추가할 수 있습니다. 덤프 출력은 이 방법으로 인터셉트할 수 없습니다.

USEROUTPUTCLASS 선택항목을 지정하면 JVM 성능에 부정적인 영향을 미칩니다. 프로덕션 환경에서 최대 성과를 얻으려면 이 선택항목을 사용하지 마십시오.

애플리케이션 또는 시스템 코드에 의해 System.out() 또는 System.err()에 기록된 출력은 출력 방향 전환 클래스로 경로를 재지정할 수 있습니다. JVM 프로파일에서 STDOUT 및 STDERR 선택항목으로 이름이 지정된 z/OS UNIX 파일은 JVM에서 실행된 일부 메시지에 또는 USEROUTPUTCLASS 선택항목으로 이름이 지정된 클래스가 원하는 대상에 데이터를 쓸 수 없는 경우 계속 사용됩니다. 따라서 이러한 파일에 적합한 파일 이름을 계속 지정해야 합니다.

USEROUTPUTCLASS 선택항목을 사용하려면 원하는 Java 클래스 이름을 지정하여 USEROUTPUTCLASS=[java class]를 JVM 프로파일에서 지정하십시오. 클래스는 java.io.OutputStream을 확장합니다. 제공된 샘플 JVM 프로파일에는 제공된 샘플 클래스의 이름을 지정하는 주석 처리 선택항목
USEROUTPUTCLASS=com.ibm.cics.samples.SJMergedStream이 포함됩니다. 해당 프

로파일로 JVM의 출력을 처리하려면 이 선택항목의 주석을 취소하여 com.ibm.cics.samples.SJMergedStream 클래스를 사용하십시오. CICS는 또한 대체 샘플 Java 클래스, com.ibm.cics.samples.SJTaskStream을 제공합니다.

제공된 사용자 출력 클래스의 소스는 샘플로 제공되므로 원하는 대로 클래스를 수정하거나 샘플을 기반으로 직접 클래스를 작성할 수 있습니다.

JVM 서버의 경우 출력 방향 전환 클래스를 OSGi 프레임워크에서 클래스를 실행하기 위한 OSGi 번들로 패키지화합니다. 자세한 정보는 159 페이지의 『JVM stdout 및 stderr 출력 경로 재지정을 위한 Java 클래스 작성』의 내용을 참조하십시오.

샘플 클래스 com.ibm.cics.samples.SJMergedStream 및 com.ibm.cics.samples.SJTaskStream

CICS 요청을 작성할 수 있는 Java 애플리케이션 스레드의 경우 JVM으로부터의 출력을 인터셉트하고 트랜지언트 데이터 큐에 출력을 작성하여 JVM 활동과 CICS 활동을 상관시키는 로그를 작성할 수 있습니다.

출력이 인터셉트될 때 시간 소인, 태스크 및 트랜잭션 ID, 프로그램 이름을 추가할 수 있습니다. 따라서 여러 JVM의 출력을 포함하는 병합된 로그 파일을 작성할 수 있습니다. 이 로그 파일을 사용하여 JVM 활동과 CICS 활동을 상관시킬 수 있습니다. 샘플 클래스, com.ibm.cics.samples.SJMergedStream은 이와 같이 병합된 로그 파일을 작성하도록 설정됩니다.

com.ibm.cics.samples.SJMergedStream 클래스는 JVM 출력 방향을 트랜지언트 데이터 큐 CSJO(stdout 출력) 및 CSJE(stderr 출력 및 내부 메시지)로 지정합니다. 이러한 트랜지언트 데이터 큐는 DFHDCTG 그룹에 제공되어 CSSL로 경로가 지정되지만 필요한 경우 큐를 재정의할 수 있습니다.

이 클래스는 출력 경로를 재지정할 뿐 아니라 날짜, 시간, APPLID, TRANSID, 태스크 번호, 프로그램 이름을 포함하는 각 레코드에 헤더를 추가합니다. 결과적으로 JVM 출력 및 오류 메시지에 대한 두 개 병합 로그 파일이 생성되므로 출력 및 메시지의 소스를 쉽게 식별할 수 있습니다.

클래스는 `/usr/lpp/cicsts/cicsts52/lib` 디렉토리에 있는 `com.ibm.cics.samples.jar` 파일에 제공됩니다. 여기서 `/usr/lpp/cicsts/cicsts52`는 z/OS UNIX에서 CICS 파일의 설치 디렉토리입니다. 클래스 소스 또한 샘플로 제공되므로 클래스를 원하는 대로 수정하거나 샘플을 기반으로 고유한 클래스를 작성할 수 있습니다. 이러한 샘플을 OSGi 환경에서 사용하도록 특별한 조치를 수행해야 합니다. 자세한 정보는 159 페이지의 『JVM stdout 및 stderr 출력 경로 재지정을 위한 Java 클래스 작성』의 내용을 참조하십시오.

CICS 요청을 작성할 수 없고 CICS에서 첨부되지 않은 스레드에서 실행되는 Java 애플리케이션의 경우, CICS 기능을 사용하여 JVM으로부터의 출력 경로를 재지정할 수 없습니다. com.ibm.cics.samples.SJMergedStream 클래스는 계속 출력을 인터셉트하고 각 레코드에 헤더를 추가합니다. 출력은 z/OS UNIX 파일인 */work_dir/applid/*stdout/CSJO 및 */work_dir/applid/*stderr/CSJE에 기록되거나(위의 설명 참조) 이러한 파일을 사용할 수 없는 경우 JVM 프로파일에서 STDOUT 및 STDERR 선택항목으로 이름이 지정되는 z/OS UNIX 파일에 기록됩니다.

JVM 출력에 대한 병합 로그 파일 작성의 대안으로, 출력 방향을 단일 태스크에서 z/OS UNIX 파일로 지정하고 시간 소인과 헤더를 추가함으로써 단일 태스크에 고유한 출력 스트림을 제공할 수 있습니다. CICS 제공 샘플 클래스, com.ibm.cics.samples.SJTaskStream이 이 작업을 수행하도록 설정됩니다. 이 클래스는 각 태스크의 출력 경로를 두 개 z/OS UNIX 파일로 지정합니다. 두 파일은 각각 stdout 출력과 stderr 출력용으로, 태스크 번호를 사용하여 YYYYMMDD.task.*tasknumber* 형식으로 고유한 이름이 지정됩니다. z/OS UNIX 파일은 stdout 디렉토리(stdout 출력의 경우) 또는 stderr 디렉토리(stderr 출력의 경우)에 저장됩니다. CICS에서 첨부된 스레드에서 실행되는 Java 애플리케이션과 다른 스레드에서 실행되는 Java 애플리케이션 모두 프로세스가 동일합니다.

오류 처리

JVM으로 실행된 메시지의 길이는 다를 수 있으며 CSSL 큐의 최대 레코드 길이(133 바이트)가 일부 수신 메시지를 포함하는 데 충분하지 않을 수 있습니다. 이러한 경우 샘플 출력 방향 전환 클래스가 오류 메시지를 실행하며 메시지 텍스트가 영향을 받을 수 있습니다.

JVM에서 133바이트보다 긴 메시지를 수신하는 경우 CSJO 및 CSJE를 개별 트랜지언트 데이터 큐로 재정의하십시오. 해당 큐를 리전 외부 대상으로 설정하고 큐의 레코드 길이를 늘리십시오. 큐는 물리적 데이터 세트 또는 시스템 출력 데이터 세트에 할당할 수 있습니다. 이 경우 출력을 보기 위해 큐를 닫지 않아도 되므로 시스템 출력 데이터 세트가 보다 편리할 수 있습니다. 트랜지언트 데이터 큐 정의 방법에 대한 정보는 참조 -> 시스템 정의의 TDQUEUE 자원의 내용을 참조하십시오. CSJO 및 CSJE를 재정의하는 경우, DFHDCTG 그룹에 정의된 트랜지언트 데이터 큐의 경우와 동일한 방식으로 콜드 스타트에서 최대한 빨리 설치되는지 확인하십시오.

트랜지언트 데이터 큐 CSJO 및 CSJE에 액세스할 수 없으면 z/OS UNIX 파일 */work_dir/applid/*stdout/CSJO 및 */work_dir/applid/*stderr/CSJE에 출력이 기록됩니다. 여기서 *work_dir*은 JVM 프로파일에서 WORK_DIR 선택항목에 지정된 디렉토리이고 *applid*는 CICS 영역과 연관된 APPLID ID입니다. 이러한 파일을 사용할 수 없는 경우에는 JVM 프로파일에서 STDOUT 및 STDERR 선택항목으로 이름이 지정된 z/OS UNIX 파일에 출력이 작성됩니다.

샘플 출력 방향 전환 클래스가 오류를 발견하면 하나 이상의 오류 메시지가 실행됩니다. 출력 메시지를 처리하는 동안 오류가 발생하면 오류 메시지 방향이 System.err로 지정되며 방향 전환이 가능합니다. 그러나 오류 메시지를 처리하는 동안 오류가 발생한 경우에는 JVM 프로파일에서 STDERR 선택항목으로 이름이 지정된 파일로 새 오류 메시지가 전송됩니다. 이를 통해 Java 클래스에서 순환 루프를 방지합니다. 클래스는 호출 Java 프로그램으로 예외를 리턴하지 않습니다.

Java 덤프 선택항목 제어

-Xdump 선택항목은 JVM에 덤프 선택항목을 지정하기 위해 JVM 프로파일에서 사용할 수 있습니다.

Java 덤프 선택항목에 대한 정보는 IBM SDK for z/OS, Java Technology Edition, 버전 7(문제점 해결 절 참조)의 내용을 참조하십시오.

JVM 서버의 OSGi 로그 파일 관리

OSGi 프레임워크는 JVM 서버의 작업 디렉토리에 있는 로그 파일 세트에 오류를 기록합니다. 기본값이 환경에 적합하지 않은 경우 각 JVM 서버의 로그 파일 수와 크기를 관리할 수 있습니다.

이 태스크 정보

OSGi 프레임워크는 zFS의 *$WORK_DIR*/*applid*/*jvmserver*/configuration 디렉토리 내 로그 파일에 오류를 기록합니다. 여기서 *$WORK_DIR*은 JVM 서버의 작업 디렉토리이고 *applid*는 CICS APPLID이며 *jvmserver*는 JVMSERVER 자원의 이름입니다. *$WORK_DIR*의 값을 JVM 프로파일에서 설정하지 않으면 CICS 영역의 z/OS UNIX 홈 디렉토리에 로그가 출력됩니다.

OSGi 프레임워크는 크기가 1000KB에 도달할 때까지 계속 로그 파일에 기록합니다. 그 다음에는 OSGi 프레임워크가 추가 오류 메시지를 작성하기 위해 다른 로그 파일을 작성합니다. 디렉토리에는 최대 10개 로그 파일을 저장할 수 있습니다. 10번째 로그 파일이 가득 차면 OSGi 프레임워크가 가장 오래된 로그 파일을 덮어씁니다. 따라서 각 JVM 서버마다 최대 10,000KB의 저장영역을 zFS의 로그 파일에 할당할 수 있습니다.

OSGi 프레임워크가 사용하는 로그 파일의 수와 크기를 변경하는 선택항목을 JVM 프로파일에 추가하여 파일 수와 저장영역 사용량을 줄이거나 늘릴 수 있습니다.

프로시저

- 최대 로그 파일 수를 변경하려면 JVM 프로파일에 **eclipse.log.backup.max** 매개변수를 추가하십시오.
- 각 로그 파일의 최대 크기를 변경하려면 JVM 프로파일에 **eclipse.log.size.max** 매개변수를 추가하십시오.

예

다음 예제는 두 개 매개변수가 지정된 JVM 프로파일을 보여줍니다. 이 예제에서는 OSGi 프레임워크가 최대 다섯 개 로그 파일을 사용할 수 있으며 각 로그 파일의 최대 크기는 500KB입니다.

```
#Parameters to control the number and size of OSGi logs
#
-Declipse.log.backup.max=5
-Declipse.log.size.max=500
#
#
```

JVM 서버에 대한 CICS 구성요소 추적

CICS는 Java로 생성된 로깅뿐 아니라 0, 1, 2 추적 레벨에 대한 SJ(JVM) 및 AP 도메인에 표준 추적점을 제공합니다. 이러한 추적점은 CICS가 JVM 서버를 설정 및 관리하기 위해 수행하는 조치를 추적합니다.

CETR 트랜잭션을 사용하여 0, 1, 2 레벨에서 SJ 및 AP 도메인 추적점을 활성화할 수 있습니다. SJ 도메인의 모든 표준 추적점에 대한 세부사항은 참조에서 JVM 도메인 추적점 -> 진단의 내용을 참조하십시오.

SJ 및 AP 구성요소 추적

SJ 구성요소는 SJ 도메인의 예외와 처리를 내부 추적 테이블로 추적합니다. AP 구성요소는 OSGi 프레임워크에서 OSGi 번들 설치를 추적합니다. SJ 레벨 3, 4 추적은 zFS의 추적 파일에 기록되는 Java 로깅을 생성합니다. 추적 파일의 이름과 위치는 JVM 프로파일의 JVMTRACE 선택항목으로 결정됩니다.

SJ 레벨 4 추적은 추적 파일에서 상세 로깅 정보를 생성합니다. 이 추적 레벨을 사용하려면 파일에 대해 충분한 공간이 zFS에 있는지 확인해야 합니다. 추적 활성화 및 관리에 대한 자세한 정보는 『JVM 서버에 대한 추적 활성화 및 관리』의 내용을 참조하십시오.

JVM 서버에 대한 추적 활성화 및 관리

SJ 및 AP 구성요소 추적을 켜고 JVM 서버 추적을 활성화할 수 있습니다. 내부 추적 테이블에는 적은 양의 추적이 기록되지만 Java는 각 JVM 서버마다 zFS에서 고유한 파일에 로깅 정보를 기록합니다. 이 파일은 줄을 바꾸지 않으므로 zFS의 경우 해당 크기를 관리해야 합니다.

이 태스크 정보

JVM 서버 추적은 일반적으로 보조 또는 GTF 추적을 사용하지 않습니다. CICS는 내부 추적 테이블에 일부 정보를 기록합니다. 그러나 대부분의 진단 정보는 Java로 로그

하며 zFS에서 파일에 기록됩니다. 이 파일은 각 JVM 서버마다 고유한 이름이 지정됩니다. 기본 파일 이름의 형식은 *applid.jvmserver.*dfhjvmtrc이며 JVMSERVER 자원을 사용하도록 설정하는 경우 CICS에서 JVM의 작업 디렉토리에 작성됩니다. 추적 파일의 이름과 위치는 JVM 프로파일에서 변경할 수 있습니다. JVM 서버가 실행될 때 추적 파일을 삭제하거나 이름을 바꾸는 경우, CICS는 파일을 다시 작성하지 않고 로깅 정보가 다른 파일에 기록되지 않습니다.

프로시저

1. CETR 트랜잭션을 사용하여 JVM 서버에 대한 추적을 활성화하십시오. 두 개 구성요소를 사용하여 JVM 서버에 대한 추적 및 로깅 정보를 생성할 수 있습니다.
   - JVM 서버를 시작 및 중지하기 위해 CICS가 수행하는 조치를 추적하려면 SJ 구성요소를 선택하십시오. JVM은 zFS 파일에 진단 정보를 로그합니다.
   - OSGi 번들 설치를 추적하려면 AP 구성요소를 선택하십시오.
2. SJ 및 AP 구성요소에 대한 추적 레벨을 설정하십시오.
   - SJ 레벨 0은 JVM 서버 초기화 도중 오류 또는 OSGi 프레임워크 문제점과 같은 예외 추적만 생성합니다. SJ 레벨 1과 레벨 2는 SJ 도메인으로부터 추가 CICS 추적을 생성합니다. 이 추적은 내부 추적 테이블에 기록됩니다.
   - SJ 레벨 3은 JVM으로부터 추가 로깅을 생성합니다(예: OSGi 프레임워크의 경고 및 정보 메시지). 이 정보는 zFS에서 추적 파일에 작성됩니다.
   - SJ 레벨 4와 AP 레벨 2는 JVM 서버 처리에 대한 자세한 정보를 제공하는 디버그 정보를 CICS 및 JVM에서 생성합니다. 이 정보는 zFS에서 추적 파일에 작성됩니다.
3. *applid.jvmserver.*dfhjvmtrc 파일에서 추적 결과를 확인하십시오. 각 추적 항목은 날짜와 시간 소인을 갖습니다. JVMTRACE 프로파일 선택항목을 사용하여 이 추적 파일의 이름과 위치를 변경할 수 있습니다.
4. 파일 크기를 관리하려면 오래된 항목을 삭제하면 됩니다. JVMSERVER 자원을 사용하지 않는 경우, 파일을 삭제하거나 정보를 별도로 보관하려면 파일 이름을 바꿀 수 있습니다. JVMSERVER 자원을 사용하는 경우, 추적 파일이 이미 존재하면 CICS는 파일에 추적 항목을 추가하고 추적 파일이 없으면 zFS에서 파일을 작성합니다.

Java 애플리케이션 디버깅

CICS의 JVM은 Java 2 플랫폼에서 제공되는 표준 디버깅 메커니즘인 JPDA(Java Platform Debugger Architecture)를 지원합니다. 이 구조는 JVM에 원격 디버거 연결을 허용하는 API 세트를 제공합니다.

이 태스크 정보

JDPA를 지원하는 도구를 사용하여 CICS에서 실행되는 Java 애플리케이션을 디버그할 수 있습니다. 예를 들어, z/OS에서 Java SDK와 함께 포함되는 Java 디버거(JDB)를 사용할 수 있습니다. JPDA 원격 디버거를 연결하려면 JVM 프로파일에서 일부 선택항목을 설정해야 합니다.

IBM은 Health Center를 포함한 Java용 모니터링 및 진단 도구를 제공합니다. IBM Health Center는 IBM Support Assistant Workbench에서 사용할 수 있습니다. 이러한 도구는 IBM에서 무료로 다운로드할 수 있습니다(시작하기 안내서의 설명 참조).

프로시저

1. JVM 프로파일에 디버깅 선택항목을 추가하여 JVM를 디버그 모드에서 시작하십시오.

   ```
   -agentlib:jdwp=transport=dt_socket,server=y,address=port, suspend=n
   ```

   디버거에 원격으로 연결할 사용 가능 포트를 선택하십시오. JVM 프로파일을 여러 JVM 서버가 공유하는 경우 디버깅에 다른 JVM 프로파일을 사용할 수 있습니다.

   참고: suspend의 기본값은 y이지만 디버깅 에이전트가 연결하려고 시도할 때 이로 인해 JVM 서버가 잠길 수 있습니다. suspend 값을 n으로 지정하면 JVM 서버 잠김이 방지됩니다.
2. JVM에 디버거를 첨부하십시오. 연결 중에 오류가 발생하는 경우(예를 들어, 포트 값이 올바르지 않은 경우) JVM 표준 출력 및 표준 오류 스트림에 메시지가 작성됩니다.
3. 디버거를 사용하여 JVM의 초기 상태를 확인하십시오. 예를 들어, 시작되는 스레드의 ID와 로드되는 시스템 클래스를 확인하십시오. JVM은 실행을 일시중단합니다. Java 애플리케이션은 시작되지 않았습니다.
4. 전체 Java 클래스 이름과 소스 코드 행 번호를 지정하여 Java 애플리케이션 내 적합한 지점에서 중단점을 설정하십시오. 일반적으로 애플리케이션 클래스가 아직 로드되지 않았으므로 디버거는 클래스가 로드될 때까지 해당 중단점 활성화가 지연됨을 나타냅니다. CICS 미들웨어 코드에서 애플리케이션 실행이 다시 일시중단되는 지점인 애플리케이션 중단점까지 JVM을 실행하십시오.
5. 로드된 클래스와 변수를 검사하고 추가 중단점을 설정하여 필요에 따라 코드를 진행하십시오.
6. 디버그 세션을 종료하십시오. 애플리케이션을 실행하여 디버거와 CICS JVM 간의 연결이 닫히는 지점에서 완료되도록 설정할 수 있습니다. 일부 디버거는 JVM 강제 종료를 지원하므로 CICS 시스템 콘솔에서 이상 종료 및 오류 메시지가 나타날 수 있습니다.

CICS JVM 플러그인 메커니즘

CICS는 JVM의 표준 JPDA 디버그 인터페이스 이외에 애플리케이션 디버깅에 유용한 인터셉션 지점(플러그인) 세트를 CICS Java 미들웨어에 제공합니다. 이러한 플러그인을 사용하면 애플리케이션 Java 코드가 실행되기 직전과 직후에 추가 Java 프로그램을 삽입할 수 있습니다.

클래스 이름, 메소드 이름과 같은 애플리케이션에 대한 정보를 플러그인에 제공할 수 있습니다. 플러그인은 또한 JCICS API를 사용하여 애플리케이션에 대한 정보를 얻을 수 있으며 표준 JPDA 인터페이스와 함께 사용하여 특히 CICS에 대한 추가 디버그 기능을 제공할 수 있습니다. 플러그인은 CICS 사용자 종료와 유사한 방식으로 디버깅 이외 용도로도 사용할 수 있습니다.

Java 종료는 Java 프로그램이 호출되기 진전 직후에 호출되는 메소드를 제공하는 CICS Java 랩퍼 플러그인입니다.

플러그인을 전개하려면 플러그인을 OSGi 번들로 패키지화합니다. 자세한 정보는 140 페이지의 『JVM 서버에 OSGi 번들 전개』의 내용을 참조하십시오.

두 개 Java 프로그래밍 인터페이스가 제공됩니다.

두 인터페이스 모두 `com.ibm.cics.server.jar`에 제공되며 Javadoc에서 설명합니다. 자세한 정보는 41 페이지의 『CICS용 Java 클래스 라이브러리(JCICS)』의 내용을 참조하십시오.

Java 프로그래밍 인터페이스는 다음과 같습니다.

- DebugControl: `com.ibm.cics.server.debug.DebugControl`. 이 프로그래밍 인터페이스는 사용자가 제공한 구현에 대해 가능한 메소드 호출을 정의합니다.
- 플러그인: `com.ibm.cics.server.debug.Plugin`. 플러그인 구현을 등록하기 위해 사용하는 일반 용도의 프로그래밍 인터페이스입니다.

다음은 DebugControl 인터페이스 예제입니다.

```
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 plugin is registered
    public void init();
}
```

다음은 DebugControl 및 플러그인 인터페이스의 구현 예제입니다.

```
import com.ibm.cics.server.debug.*;

public class SampleCICSDebugPlugin
    implements Plugin, DebugControl
{
    // Implementation of the plugin initialiser
    public void init()
    {
        // This method is called when the CICS Java middleware loads and
        // registers the plugin. 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)
    {
    }
}
```

제 9 장 Liberty JVM 서버 및 Java 웹 애플리케이션 문제점 해결

Java 웹 애플리케이션 관련 문제점이 있는 경우 CICS가 제공하는 진단과 Liberty 프로파일을 사용하여 문제점의 원인을 판별할 수 있습니다.

CICS는 Liberty JVM 서버에서 Java 웹 애플리케이션 실행과 관련된 문제점을 진단하는 데 유용한 통계, 메시지, 추적을 제공합니다. 웹 애플리케이션을 실행하는 데 사용되는 Liberty 프로파일 기술은 또한 zFS에서 사용 가능한 진단을 생성합니다. 일반 설정 오류 및 애플리케이션 문제점은 195 페이지의 제 8 장 『Java 애플리케이션 문제점 해결』의 내용을 참조하십시오.

문제점 방지

CICS는 영역 APPLID의 값과 JVMSERVER 자원 이름을 사용하여 zFS 파일 시스템의 고유 파일 및 디렉토리 이름을 작성합니다. UNIX 시스템 서비스 쉘의 허용 가능한 일부 문자는 특수한 의미를 지닙니다. 예를 들면, 달러 기호($)는 환경 변수 이름의 시작을 의미합니다. 영역 APPLID 및 JVM 서버 이름에서 영숫자가 아닌 문자를 사용하지 마십시오. 이러한 문자를 사용할 경우 UNIX 시스템 서비스 쉘에서 백슬래시(\)를 이스케이프 문자로 사용해야 할 수 있습니다. 예를 들어 JVM 서버 MY$JVMS를 호출한 경우 다음을 수행하십시오.

```
cat CICSPRD.MY\$JVMS.D20140319.T124122.dfhjvmout
```

Liberty JVM 서버를 시작할 수 없음

1. Liberty JVM 서버를 시작할 수 없는 경우 설정이 올바른지 확인하십시오. CICS 메시지와 `WLP_OUTPUT_DIR` 아래에 있는 Liberty `messages.log` 파일의 오류를 사용하여 문제점의 원인을 판별하십시오.
2. JVM 프로파일의 **`-Dfile.encoding`** JVM 등록 정보가 ISO-8859-1 또는 UTF-8을 지정하는지 확인하십시오. 이는 Liberty 프로파일 서버가 지원하는 두 가지 코드 페이지입니다.다른 값을 설정하면 JVM 서버가 시작되지 않습니다.

dropins 디렉토리에 전개한 후 웹 애플리케이션을 사용할 수 없음

1. `dfhjvmerr`에서 CWWK0221E 오류 메시지를 수신하는 경우 JVM 프로파일과 `server.xml`에서 호스트 이름 및 포트 번호에 올바른 값을 설정했는지 확인하십시오. 이 오류 메시지는 지정된 포트 번호 또는 호스트 이름이 올바르지 않음을 나타냅니다. 호스트 이름이 올바르지 않거나 포트 번호가 사용 중일 수 있습니다.

Liberty JVM 서버를 사용하도록 설정한 후 CICS CPU가 증가함

1. Liberty JVM 서버가 `dropins` 디렉토리를 너무 자주 스캔하여 I/O 및 CPU 사용량이 지나치게 증가합니다. Liberty JVM 서버가 애플리케이션의 `dropins` 디렉토

리를 스캔하는 빈도는 구성할 수 있습니다. 구성 템플리트에서 제공되는 기본 간격은 5초이지만 이 값을 늘리거나 애플리케이션 스캔을 사용 안함으로 설정할 수 있습니다.

이 문제점을 수정하려면 server.xml 파일에서 다음 XML을 편집하십시오.

```
<config monitorInterval="5s" updateTrigger="polled"/>
<applicationMonitor updateTrigger="disabled" pollingRate="5s" dropins="dropins" dropinsEnabled="false"/>
```

웹 애플리케이션을 CICS 번들에 설치하는 경우에는 <config> 요소에서 풀링을 사용 안함으로 설정하지 마십시오. 그러나 애플리케이션 스캔은 사용 안함으로 설정하십시오. 이 설정 편집 방법에 대한 정보는 동적 갱신 제어의 내용을 참조하십시오.

애플리케이션을 사용할 수 없음

1. WAR 파일을 dropins 디렉토리에 복사하지만 애플리케이션은 사용할 수 없습니다. Liberty messages.log 파일에서 오류 메시지를 확인하십시오. CWWKZ0013E 오류 메시지를 수신하는 경우 Liberty JVM 서버에 이미 동일한 이름으로 실행되는 웹 애플리케이션이 있기 때문입니다. 문제점을 수정하려면 웹 애플리케이션의 이름을 변경하고 dropins 디렉토리에 전개하십시오.

웹 애플리케이션이 인증을 요청하지 않음

보안을 구성했지만 웹 애플리케이션이 인증을 요청하지 않습니다.

1. 웹 애플리케이션에 CICS 보안을 구성할 수는 있지만 웹 애플리케이션은 WAR 파일에 보안 제한조건이 포함되는 경우에만 보안을 사용합니다. 동적 웹 프로젝트의 web.xml 파일에서 애플리케이션 개발자가 보안 제한조건을 정의했는지 확인하십시오.
2. server.xml 파일에 올바른 보안 구성 정보가 포함되었는지 확인하십시오.모든 구성 오류는 dfhjvmerr에서 보고되며 유용한 정보를 제공할 수 있습니다. CICS 보안을 사용하는 경우 CICS 보안 기능이 지정되었는지 확인하십시오. CICS 보안이 꺼진 경우 애플리케이션 사용자를 인증하기 위해 기본 사용자 레지스트리를 지정했는지 확인하십시오.
3. <safAuthorization>에서 EJBRole을 이용하도록 server.xml이 구성되었는지 확인하고 <application-bnd> 요소에서 로컬 역할 맵핑을 확인하십시오.
   <application-bnd> 요소는 server.xml 또는 installedApps.xml에 <application> 요소와 함께 있습니다. CICS에서 로컬 역할 맵핑을 위해 추가한 기본 security-role은 'cicsAllAuthenticated'입니다.

웹 애플리케이션이 HTTP 403 오류 코드를 리턴

웹 애플리케이션은 웹 브라우저에서 HTTP 403 오류 코드를 리턴합니다. HTTP 403 인증 실패를 수신하면 사용자 ID가 철회되었거나 사용자에게 애플리케이션 트랜잭션을 실행할 수 있는 권한이 부여되지 않았기 때문입니다.

1. 오류 메시지 ICH408I의 CICS 메시지 로그를 보고 발생한 권한 부여 오류 유형을 확인하십시오.문제점을 수정하려면 사용자 ID가 올바른 암호를 갖고 트랜잭션을 실행할 수 있는 권한이 부여되었는지 확인하십시오.
2. 애플리케이션이 com.ibm.ws.webcontainer.util.Base64Decode 클래스에 대한 예외를 리턴하는 경우 오류 메시지의 `dfhjvmerr`을 확인하십시오.구성 오류 메시지(예: CWWKS4106E 또는 CWWKS4000E)가 있는 경우 서버가 다른 인코딩으로 작성된 구성 파일 액세스를 시도하기 때문입니다. 이 유형의 구성 오류는 **`file.encoding`** 값을 변경하고 JVM 서버를 다시 시작할 때 발생할 수 있습니다. 문제점을 수정하려면 이전 인코딩으로 되돌리고 JVM 서버를 다시 시작하거나 구성 파일을 삭제하면 됩니다. JVM 서버는 시작 시 올바른 파일 인코딩으로 파일을 다시 작성합니다.

웹 애플리케이션이 HTTP 500 오류 코드를 리턴

웹 애플리케이션은 웹 브라우저에서 HTTP 500 오류를 리턴합니다. HTTP 500 오류를 수신하면 구성 오류가 발생한 것입니다.

1. 오류의 특정 원인에 대한 자세한 정보를 제공할 수 있는 DFHSJ 메시지의 CICS 메시지 로그를 확인하십시오.
2. URIMAP을 사용하여 특정 트랜잭션에 대한 애플리케이션 요청을 실행하는 경우 URIMAP이 올바른 트랜잭션을 사용하는지 확인하십시오.
3. URIMAP이 올바른 트랜잭션을 사용하는 경우 SCHEME 및 USAGE 속성이 올바르게 설정되었는지 확인하십시오.SCHEME은 애플리케이션 요청(HTTP 또는 HTTPS)과 일치해야 합니다. USAGE 속성은 JVMSERVER로 설정해야 합니다.

웹 애플리케이션이 HTTP 503 오류 코드를 리턴

웹 애플리케이션은 웹 브라우저에서 HTTP 503 오류를 리턴합니다. HTTP 503 오류를 수신하는 경우 애플리케이션을 사용할 수 없습니다.

1. 추가 정보는 DFHSJ 메시지의 CICS 메시지 로그를 확인하십시오.
2. 애플리케이션에 대한 TRANSACTION 및 URIMAP 자원을 사용할 수 있는지 확인하십시오.이러한 자원이 애플리케이션의 일부로 CICS 번들에 패키지화되는 경우 BUNDLE 자원의 상태를 확인할 수 있습니다.
3. 요청은 완료되기 전에 영구 제거되었을 수 있습니다. 로그의 오류 메시지가 요청이 영구 제거된 이유를 설명합니다.

웹 애플리케이션이 예외를 리턴함

웹 애플리케이션이 웹 브라우저에서 예외를 리턴합니다. 예를 들어, 애플리케이션은 com.ibm.ws.webcontainer.util.Base64Decode 클래스에 대한 예외를 리턴합니다.

1. dfhjvmerr에서 오류 메시지를 확인하십시오.
2. 구성 오류 메시지(예: CWWKS4106E 또는 CWWKS4000E)가 있는 경우 서버가 다른 인코딩으로 작성된 구성 파일 액세스를 시도하기 때문입니다. 이 유형의 구성 오류는 **file.encoding** 값을 변경하고 JVM 서버를 다시 시작할 때 발생할 수 있습니다. 문제점을 수정하려면 이전 인코딩으로 되돌리고 JVM 서버를 다시 시작하거나 구성 파일을 삭제하면 됩니다. JVM 서버는 시작 시 올바른 파일 인코딩으로 파일을 다시 작성합니다.

오류 메시지 CWWKB0109E

Liberty 서버를 완전히 종료하는 데 실패하면 WLP_ZOS_PLATFORM=TRUE로 설정된 다음 Liberty JVM 서버가 messages.log 파일에 오류 메시지 CWWKB0109E를 기록합니다. 이 오류를 수정할 필요가 없으며 무시할 수 있습니다.

productInfo 스크립트를 사용하여 Liberty 프로파일의 무결성 검증

productInfo 스크립트를 사용하여 CICS 설치 또는 서비스 적용 이후 Liberty 프로파일 설치의 무결성을 검증할 수 있습니다.

1. 디렉토리를 CICS USSHOME 디렉토리로 변경하십시오.
2. productInfo는 Java를 사용하므로 Java가 경로에 포함되었는지 확인해야 합니다. 또는 **JAVA_HOME** 환경 변수를 JVM 프로파일의 **JAVA_HOME** 값으로 설정하십시오. 예:

   ```
   export JAVA_HOME=/usr/lpp/java/J7.0_64
   ```

3. 유효성 검증 선택항목 wlp/bin/productInfo validate를 제공하는 productInfo 스크립트를 실행하십시오. 오류가 보고되지 않아야 합니다. Liberty 프로파일 productInfo 스크립트에 대한 자세한 정보는 Liberty 프로파일 설치의 무결성 검증을 참조하십시오.

wlpenv 스크립트를 사용하여 Liberty 명령 실행

IBM 서비스가 하나 이상의 Liberty 프로파일 제공 명령(예: **productInfo** 또는 **server dump**)을 실행하도록 요청할 수 있습니다. 이러한 명령을 실행하려면 wlpenv 스크립트를 필요한 환경을 설정하기 위한 랩퍼로 사용할 수 있습니다. JVM 프로파일이 성공적으로 구문 분석된 후 Liberty JVM 서버를 사용으로 설정할 때마다 스크립트가 작성 및 갱신됩니다. 스크립트는 각 CICS 영역의 JVM 서버마다 고유하므로 JVM 프로파

일에 지정된 대로 *WORK_DIR*에서 작성되고 *APPLID*.*JVMSERVER*.wlpenv라고 지칭됩니다. 여기서 *APPLID*는 CICS 영역 APPLID의 값이고 *JVMSERVER*는 JVMSERVER 자원의 이름입니다.

UNIX 시스템 서비스 쉘에서 wlpenv 스크립트를 실행하려면 JVM 프로파일에 지정된 대로 디렉토리를 *WORK_DIR*로 변경하고 Liberty 프로파일 명령을 인수로 사용하여 스크립트를 실행하십시오. 예:

```
./CICSPRD.DFHWLP.wlpenv productInfo version
```

```
./CICSPRD.DFHWLP.wlpenv server dump --archive=package_file_name.dump.pax
--include=heap
```

server dump 명령의 경우 서버 이름을 제공하지 않습니다. wlpenv 스크립트가 마지막으로 JVM 서버를 사용했을 때 설정된 값으로 설정되기 때문입니다.

Liberty 명령에 대한 자세한 정보는 및 의 내용을 참조하십시오.

주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다. IBM은 다른 국가에서 이 책에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급했다고 해서 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산권을 침해하지 않는 한, 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 있습니다. 그러나 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이센스까지 부여하는 것은 아닙니다. 라이센스에 대한 의문사항은 다음으로 문의하십시오.

135-700
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터
전화번호: 080-023-8080

2바이트(DBCS) 정보에 관한 라이센스 문의는 한국 IBM 고객만족센터에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다.

IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여(단, 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증 없이 이 책을 "현상태대로" 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 책에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 변경된 사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통지 없이 언제든지 개선 및/또는 변경할 수 있습니다.

(i) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함) 간의 정보 교환 및 (ii) 교환된 정보의 상호 이용을 목적으로 정보를 원하는 프로그램 라이센스 사용자는 다음 주소로 문의하십시오.

135-700
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터

이러한 정보는 해당 조건(예를 들면, 사용료 지불 등)하에서 사용될 수 있습니다.

이 정보에 기술된 라이센스가 부여된 프로그램 및 프로그램에 대해 사용 가능한 모든 라이센스가 부여된 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이센스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

개인정보 보호정책 고려사항

SaaS(Software as a Service) 솔루션을 포함한 IBM 소프트웨어 제품(이하 "소프트웨어 오퍼링")은 제품 사용 정보를 수집하거나 일반 사용자의 사용 경험을 개선하거나 일반 사용자와의 상호 작용을 조정하거나 그 외의 용도로 쿠키나 기타 다른 기술을 사용할 수 있습니다. 대부분의 경우 소프트웨어 오퍼링은 개인 식별 정보를 수집하지 않습니다. IBM 소프트웨어 오퍼링 중 일부는 개인 식별 정보를 수집할 수 있도록 도와 줍니다. 이 소프트웨어 오퍼링이 쿠키를 사용하여 개인 식별 정보를 수집하는 경우, 이 오퍼링의 쿠키 사용에 대한 자세한 정보는 다음과 같습니다.

CICSPlex SM 웹 사용자 인터페이스:

WUI 기본 인터페이스: 전개된 구성에 따라 이 소프트웨어 오퍼링은 세션 관리, 인증, 사용자 용이성 향상 또는 기타 사용법 추적이나 기능을 위해 각 사용자의 사용자 이름 및 개인 식별 정보를 수집하는 세션 및 지속적 쿠키를 사용할 수 있습니다. 이러한 쿠키는 사용 안함으로 설정할 수 없습니다.

WUI 데이터 인터페이스: 전개된 구성에 따라 이 소프트웨어 오퍼링은 세션 관리, 인증 또는 기타 사용법 추적이나 기능을 위해 각 사용자의 사용자 이름 및 개인 식별 정보를 수집하는 세션 쿠키를 사용할 수 있습니다. 이러한 쿠키는 사용 안함으로 설정할 수 없습니다.

WUI Hello World 페이지: 전개된 구성에 따라 이 소프트웨어 오퍼링은 개인 식별 정보를 수집하지 않는 세션 쿠키를 사용할 수 있습니다. 이러한 쿠키는 사용 안함으로 설정할 수 없습니다.

CICS 탐색기: 전개된 구성에 따라 이 소프트웨어 오퍼링은 세션 관리, 인증 및 싱글 사인온 구성을 위해 사용자의 사용자 이름 및 비밀번호를 수집하는 세션 및 지속적 환경 설정을 사용할 수 있습니다. 사인온 중에 선택란을 선택하여 사용자 비밀번호를 암호화된 양식으로 저장하면 사용자의 명시적인 조치에 의해서만 사용으로 설정할 수 있지만 이러한 환경 설정은 사용 안함으로 설정할 수 없습니다.

이 소프트웨어 오퍼링을 위해 전개되는 구성이 귀하(고객)에게 쿠키 등의 기술로 일반 사용자의 개인 식별 정보를 수집하는 기능을 제공하는 경우, 통지 및 동의에 필요한 모든 요구사항을 포함하여 그러한 데이터 수집에 적용되는 모든 법률에 대해 직접 법적 자문을 구해야 합니다.

이러한 목적을 위해 쿠키를 비롯한 여러 기술을 사용하는 데 대한 자세한 정보는 IBM의 개인정보 보호정책(http://www.ibm.com/privacy/kr/ko/), IBM의 온라인 개인정보 보호정책(http://www.ibm.com/privacy/details/kr/ko/) 『쿠키, Web Beacons 및 기타 기술』 절과 『IBM 소프트웨어 제품 및 SaaS(Software-as-a-Service) 개인정보 보호정책』(http://www-01.ibm.com/software/info/product-privacy/)을 참조하십시오.

『상표』

상표

IBM, IBM 로고 및 ibm.com®은 전세계 여러 국가에 등록된 International Business Machines Corp.의 상표 또는 등록상표입니다. 기타 제품 및 서비스 이름은 IBM 또는 타사의 상표입니다. IBM 상표의 최신 목록은 웹 사이트의 저작권 및 상표 정보(www.ibm.com/legal/copytrade.shtml)에서 확인할 수 있습니다.

Java 및 모든 Java 기반 상표와 로고는 Oracle 및/또는 그 계열사의 상표 또는 등록상표입니다.

Linux는 미국 또는 기타 국가에서 사용되는 Linus Torvalds의 등록상표입니다.

Microsoft 및 Windows는 미국 또는 기타 국가에서 사용되는 Microsoft Corporation의 상표입니다.

UNIX는 미국 또는 기타 국가에서 사용되는 Open Group의 등록상표입니다.

참고 문헌

z/OS용 CICS Transaction Server의 CICS 서적

일반

z/OS용 CICS Transaction Server 프로그램 디렉토리, GI13-3326
z/OS용 CICS Transaction Server 새로운 기능, GC34-7302
*CICS TS 버전 3.1*에서 z/OS용 CICS Transaction Server 업그레이드, GC34-7296
*CICS TS 버전 3.2*에서 z/OS용 CICS Transaction Server 업그레이드, GC34-7297
*CICS TS 버전 4.1*에서 z/OS용 CICS Transaction Server 업그레이드, GC34-7298
*CICS TS 버전 4.2*에서 z/OS용 CICS Transaction Server 업그레이드, GC34-7299
*CICS TS 버전 5.1*에서 z/OS용 CICS Transaction Server 업그레이드, GC34-7300
z/OS용 CICS Transaction Server 설치 안내서, GC34-7279

CICS 액세스

CICS 인터넷 안내서, SC34-7281
CICS 웹 서비스 안내서, SC34-7301

관리

CICS 시스템 정의 안내서, SC34-7293
CICS 사용자 정의 안내서, SC34-7269
CICS 자원 정의 안내서, SC34-7290
CICS 운영 및 유틸리티 안내서, SC34-7285
CICS RACF 보안 안내서, SC34-7288
CICS 제공된 트랜잭션, SC34-7292

프로그래밍

CICS 애플리케이션 프로그래밍 안내서, SC34-7266
CICS 애플리케이션 프로그래밍 참조서, SC34-7267
CICS 시스템 프로그래밍 참조서, SC34-7294
CICS FEPI(Front End Programming Interface) 사용자 안내서, SC34-7277
CICS C++ OO 클래스 라이브러리, SC34-7270
CICS 분산 트랜잭션 프로그래밍 안내서, SC34-7275
CICS 비즈니스 트랜잭션 서비스, SC34-7268
CICS의 Java 애플리케이션, SC43-0655

진단

CICS 문제점 판별 안내서, GC34-7287
CICS 성능 안내서, SC34-7286

CICS 메시지 및 코드 Vol 1, GC34-7283
CICS 메시지 및 코드 Vol 2, GC34-7284
CICS 진단 참조서, GC34-7274
CICS 복구 및 다시 시작 안내서, SC34-7289
CICS 데이터 영역, GC34-7271
CICS 추적 항목, SC34-7295
CICS 디버깅 도구 인터페이스 참조서, GC34-7273

통신

CICS 상호 통신 안내서, SC34-7280
CICS 외부 인터페이스 안내서, SC34-7276

데이터베이스

CICS DB2 안내서, SC34-7272
CICS IMS 데이터베이스 제어 안내서, SC34-7278
CICS 공유 데이터 테이블 안내서, SC34-7291

---

z/OS용 CICS Transaction Server에 대한 CICSPlex SM 서적

일반

CICSPlex SM Concepts and Planning, SC34-7306
CICSPlex SM Web User Interface Guide, SC34-7316

관리(Administration and Management)

CICSPlex SM Administration, SC34-7303
CICSPlex SM Operations Views Reference, SC34-7312
CICSPlex SM Monitor Views Reference, SC34-7311
CICSPlex SM Managing Workloads, SC34-7309
CICSPlex SM Managing Resource Usage, SC34-7308
CICSPlex SM Managing Business Applications, SC34-7307

프로그래밍

CICSPlex SM Application Programming Guide, SC34-7304
CICSPlex SM Application Programming Reference, SC34-7305

진단

CICSPlex SM Resource Tables Reference Vol 1, SC34-7314
CICSPlex SM Resource Tables Reference Vol 2, SC34-7315
CICSPlex SM Messages and Codes, GC34-7310
CICSPlex SM Problem Determination, GC34-7313

기타 CICS 서적

다음 서적에는 CICS에 대한 자세한 정보가 포함되어 있으나 z/OS용 CICS Transaction Server, 버전 5 릴리스 2와 함께 제공되지는 않습니다.

Designing and Programming CICS Applications, SR23-9692

CICS Application Migration Aid Guide, SC33-0768

CICS Family: API Structure, SC33-1007

CICS Family: Client/Server Programming, SC33-1435

CICS Family: Interproduct Communication, SC34-6853

CICS Family: Communicating from CICS on System/390, SC34-6854

CICS Transaction Gateway for z/OS Administration, SC34-5528

CICS Family: General Information, GC33-0155

CICS 4.1 Sample Applications Guide, SC33-1173

CICS/ESA 3.3 XRF Guide , SC33-0661

기타 IBM 서적

다음 서적은 관련된 IBM 제품에 대한 정보를 포함합니다.

IBM Developer Kit and Runtime Environment, Java 2 Technology Edition Diagnostics Guide, SC34-6358

Persistent Reusable Java Virtual Machine User's Guide, SC34-6201

내게 필요한 옵션

내게 필요한 옵션 기능은 거동 장애나 시각 장애 같은 신체적 장애가 있는 사용자가 소프트웨어 제품을 사용할 수 있도록 도와줍니다.

다음 중 한 가지 방법으로 CICS 시스템을 설정, 실행, 유지보수하는 데 필요한 대부분의 태스크를 수행할 수 있습니다.

- CICS에 로그온된 3270 에뮬레이터 사용
- TSO에 로그온된 3270 에뮬레이터 사용
- 3270 에뮬레이터를 MVS 시스템 콘솔로 사용

IBM Personal Communications는 장애를 가진 사람들을 위한 내게 필요한 옵션 기능을 3270 에뮬레이션에 제공합니다. 이 제품을 사용하여 CICS 시스템에서 필요한 내게 필요한 옵션 기능을 제공할 수 있습니다.

색인

[가]

[나]

[다]

[라]

[마]

[바]

[사]

[아]

[자]

[차]

[카]

[타]

[파]

[하]

A

C

D

E

G

I

J

L

O

P

S

U

W

Z

[특수 문자]

SC43-0655-00