Red Hat Fuse 7.11

Apache Karaf에 배포

Apache Karaf 컨테이너에 애플리케이션 패키지 배포

Last Updated: 2023-07-25

Red Hat Fuse 7.11 Apache Karaf에 배포

Apache Karaf 컨테이너에 애플리케이션 패키지 배포

법적 공지

초록

이 가이드에서는 Apache Karaf 컨테이너에 애플리케이션을 배포하는 옵션에 대해 설명합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 CTO Chris Wright의 메시지 에서 참조하십시오.

I 부. 개발자 가이드

이 부분에는 개발자를 위한 정보가 포함되어 있습니다.

1장. OSGI 소개

초록

OSGi 사양은 복잡한 애플리케이션의 빌드, 배포 및 관리를 단순화하는 런타임 프레임워크를 정의하여 모듈식 애플리케이션 개발을 지원합니다.

1.1. 개요

Apache Karaf는 번들 배포 및 관리를 위한 OSGi 기반 런타임 컨테이너입니다. Apache Karaf는 기본 운영 체제 통합을 제공하며 라이프사이클이 운영 체제에 바인딩되도록 운영 체제로 통합할 수 있습니다.

Apache Karaf에는 다음과 같은 구조가 있습니다.

- Apache Karaf - OSGi 컨테이너 구현을 중심으로 하는 래퍼 계층으로, OSGi 컨테이너 배포를 런타임 서버로 지원합니다. Fuse에서 제공하는 런타임 기능에는 핫 배포, 관리 및 관리 기능이 포함됩니다.
- OSGi Framework - 종속성 및 번들 라이프사이클 관리를 포함하여 OSGi 기능을 구현합니다.

1.2. APACHE KARAF 아키텍처

Apache Karaf는 다음과 같은 기능을 통해 OSGi 계층을 확장합니다.

- **콘솔** - 콘솔은 서비스를 관리하고, 애플리케이션 및 라이브러리를 설치 및 관리하고, Fuse 런타임과 상호 작용합니다. Fuse 인스턴스를 관리하기 위한 콘솔 명령을 제공합니다. Apache Karaf 콘솔 참조를 참조하십시오.
- **로깅** - 로깅 하위 시스템은 로그 수준을 표시, 보기 및 변경할 수 있는 콘솔 명령을 제공합니다.
- **deployment** - **bundle:install** 및 **bundle:start** 명령을 사용하고 애플리케이션의 핫 배포를 사용하여 OSGi 번들의 수동 배포를 지원합니다. 6.1절. "핫 디플로이먼트"을 참조하십시오.
- **프로비저닝** - 애플리케이션 및 라이브러리를 설치하기 위한 여러 메커니즘을 제공합니다. 9장. *기능 배포*을 참조하십시오.
- **구성** - *InstallDir*/**etc** 폴더에 저장된 속성 파일이 지속적으로 모니터링되며 변경 사항은 구성 가능한 간격으로 관련 서비스로 자동 전파됩니다.
- **블루프린트** - OSGi 컨테이너와의 상호 작용을 간소화하는 종속성 주입 프레임워크입니다. 예를 들어 OSGi 서비스를 가져오고 내보낼 표준 XML 요소를 제공합니다. 블루프린트 구성 파일이 핫 배포 폴더에 복사되면 Red Hat Fuse는 OSGi 번들을 즉시 생성하고 블루프린트 컨텍스트를 인스턴스화합니다.

1.3. OSGI FRAMEWORK

1.3.1. 개요

OSGi Alliance 는 OSGi Service Platform 릴리스 4 의 기능 및 기능을 정의하는 독립 조직입니다. OSGi Service Platform은 복잡한 소프트웨어 애플리케이션의 구축, 배포 및 관리를 단순화하는 오픈 사양 세트입니다.

OSGi 기술은 종종 Java용 동적 모듈 시스템이라고 합니다. OSGi는 번들을 사용하여 Java 구성 요소를 모듈식으로 배포하고 종속 항목, 버전 관리, 클래스 경로 제어 및 클래스 로드를 처리하는 Java용 프레임워크

입니다. OSGi의 라이프사이클 관리를 사용하면 JVM을 종료하지 않고 번들을 로드, 시작 및 중지할 수 있습니다.

OSGi는 Java에 가장 적합한 런타임 플랫폼, 우수한 클래스 로드 아키텍처, 서비스 레지스트리를 제공합니다. 번들은 서비스를 내보내고, 프로세스를 실행하고, 종속성을 관리할 수 있습니다. 각 번들에는 OSGi 컨테이너에서 관리하는 요구 사항이 있을 수 있습니다.

Fuse는 Apache Felix 를 기본 OSGi 구현으로 사용합니다. 프레임워크 계층은 번들을 설치하는 컨테이너를 형성합니다. 프레임워크는 동적이고 확장 가능한 방식으로 번들 설치 및 업데이트를 관리하고 번들과 서비스 간의 종속성을 관리합니다.

1.3.2. OSGi 아키텍처

OSGi 프레임워크에는 다음이 포함됩니다.

- **bundles** - 애플리케이션을 구성하는 논리 모듈입니다. 1.5절. "OSGi 번들"을 참조하십시오.
- **서비스 계층** - 모듈과 포함된 구성 요소 간 통신을 제공합니다. 이 계층은 라이프사이클 계층과 긴밀하게 통합됩니다. 1.4절. "OSGi Services"을 참조하십시오.
- **라이프사이클 계층** - 기본 OSGi 프레임워크에 대한 액세스를 제공합니다. 이 계층은 개별 번들의 라이프사이클을 처리하므로 번들 시작 및 중지를 포함하여 애플리케이션을 동적으로 관리할 수 있습니다.
- **모듈 계층** - 번들 패키지, 종속성 확인 및 클래스 로드를 관리하는 API를 제공합니다.
- **실행 환경** - JVM 구성입니다. 이 환경에서는 번들이 작동할 수 있는 환경을 정의하는 프로필을 사용합니다.
- **보안 계층** - 추가 제약 조건 및 개선 사항이 포함된 Java 2 보안을 기반으로 하는 선택적 계층입니다.

프레임워크의 각 계층은 해당 계층에 따라 다릅니다. 예를 들어 라이프사이클 계층에는 모듈 계층이 필요합니다. 모듈 계층은 라이프사이클 및 서비스 계층 없이 사용할 수 있습니다.

1.4. OSGI SERVICES

1.4.1. 개요

OSGi 서비스는 이름/값 쌍으로 정의된 서비스 속성이 있는 Java 클래스 또는 서비스 인터페이스입니다. 서비스 속성은 동일한 서비스 인터페이스를 제공하는 서비스 공급자 간에 구별됩니다.

OSGi 서비스는 서비스 인터페이스에 의해 의미 체계적으로 정의되며 서비스 오브젝트로 구현됩니다. 서비스의 기능은 구현하는 인터페이스에 의해 정의됩니다. 따라서 다른 애플리케이션이 동일한 서비스를 구현할 수 있습니다.

서비스 인터페이스를 사용하면 번들이 구현이 아닌 바인딩 인터페이스로 상호 작용할 수 있습니다. 구현 세부 정보를 가능한 한 적은 수만큼 사용하여 서비스 인터페이스를 지정해야 합니다.

1.4.2. OSGi 서비스 레지스트리

OSGi 프레임워크에서 서비스 계층은 게시, 찾기, 서비스 모델을 사용하여 1.5절. "OSGi 번들" 와 포함된 구성 요소 간의 통신을 제공합니다. 서비스 계층에는 다음과 같은 서비스 레지스트리가 포함되어 있습니다.

- 서비스 공급자는 다른 번들에서 사용할 프레임워크에 서비스를 등록합니다.

- 서비스 요청자가 서비스를 찾고 서비스 공급자에 바인딩

서비스는 번들에 의해 소유되며, 내에서 실행됩니다. 번들은 하나 이상의 Java 인터페이스 아래에 프레임워크 서비스 레지스트리에 서비스 구현을 등록합니다. 따라서 프레임워크를 제어하는 다른 번들에서 서비스의 기능을 사용할 수 있으며 다른 번들은 서비스를 조회하고 사용할 수 있습니다. 조회는 Java 인터페이스 및 서비스 속성을 사용하여 수행됩니다.

각 번들은 인터페이스의 정규화된 이름과 해당 속성을 사용하여 서비스 레지스트리에 여러 서비스를 등록할 수 있습니다. 번들에서는 LDAP 구문과 함께 이름 및 속성을 사용하여 서비스의 서비스 레지스트리를 쿼리합니다.

번들은 게시, 검색 및 바인딩을 포함한 런타임 서비스 종속성 관리 활동을 담당합니다. 또한 번들은 번들에 바인딩된 서비스의 동적 가용성 (배타 또는 출발)으로 인한 변경 사항에도 대응할 수 있습니다.

1.4.1. 이벤트 알림

서비스 인터페이스는 번들에 의해 생성된 오브젝트로 구현됩니다. 번들은 다음을 수행할 수 있습니다.

- 서비스 등록
- 서비스 검색
- 등록 상태 변경 시 알림 수신

OSGi 프레임워크는 이벤트 알림 메커니즘을 제공하므로 서비스 요청자가 서비스 레지스트리의 변경이 발생할 때 알림 이벤트를 수신할 수 있습니다. 이러한 변경에는 특정 서비스의 게시 또는 검색 및 서비스가 등록, 수정 또는 등록 취소되는 시기가 포함됩니다.

1.4.2. 서비스 호출 모델

번들에서 서비스를 사용하려는 경우 서비스를 조회하고 Java 오브젝트를 일반 Java 호출로 호출합니다. 따라서 서비스에 대한 호출은 동기적으로 수행되며 동일한 스레드에서 발생합니다. 더 많은 비동기 처리를 위해 콜백을 사용할 수 있습니다. 매개 변수는 Java 개체 참조로 전달됩니다. XML과 마찬가지로 마샬링 또는 중간 표준 형식이 필요하지 않습니다. OSGi는 사용할 수 없는 서비스 문제에 대한 솔루션을 제공합니다.

1.4.3. OSGi 프레임워크 서비스

OSGi 프레임워크는 자체 서비스 외에도 다음과 같은 선택적 서비스를 제공하여 프레임워크의 작업을 관리합니다.

- **패키지 관리 서비스**- 관리 에이전트가 공유 패키지의 상태를 검사하여 Java 패키지 공유를 관리하는 정책을 정의할 수 있습니다. 또한 관리 에이전트가 패키지를 새로 고치고 필요에 따라 번들을 중지 및 다시 시작할 수 있습니다. 이 서비스를 사용하면 번들 내보내기를 제거하거나 업데이트할 때 관리 에이전트가 공유 패키지에 대해 결정을 내릴 수 있습니다.
  또한 서비스는 마지막 새로 고침 이후 제거되거나 업데이트된 내보낸 패키지를 새로 고치고 특정 번들을 명시적으로 확인하는 방법도 제공합니다. 이 서비스는 런타임 시 번들 간의 종속성을 추적하여 업그레이드로 인해 어떤 번들이 영향을 받을 수 있는지 확인할 수 있습니다.

- **시작 수준 서비스**- 관리 에이전트를 활성화하여 번들 시작 및 중지 순서를 제어합니다. 서비스는 각 번들에 시작 수준을 할당합니다. 관리 에이전트는 번들 시작 수준을 수정하고 적절한 번들을 시작하고 중지하는 프레임워크의 활성 시작 수준을 설정할 수 있습니다. 시작 수준이 작거나 같은 번들만 이 활성 시작 수준을 활성화할 수 있습니다.

- **URL 처리기 서비스**- 모든 구성 요소가 추가 URL 처리기를 제공할 수 있도록 URL 체계 및 콘텐츠 처리기를 사용하여 Java 런타임을 동적으로 확장합니다.

- **권한 관리 서비스**- OSGi 프레임워크 관리 에이전트가 특정 번들의 권한을 관리하고 모든 번들에 대한 기본값을 제공할 수 있습니다. 번들에는 권한 있는 코드를 실행할 권한이 있는지 확인하는 데 사용되는 단일 권한 세트가 있을 수 있습니다. 즉시 정책을 변경하고 새로 설치된 구성 요소에 대한 새 정책을 추가하여 권한을 동적으로 조작할 수 있습니다. 정책 파일은 번들을 제어하는 데 사용됩니다.

- **Conditional Permission Admin 서비스**- 권한을 확인할 때 특정 조건이 true 또는 false인 경우 적용할 수 있는 권한으로 Permission Admin 서비스를 확장합니다. 이러한 조건에 따라 권한이 적용되는 번들의 선택이 결정됩니다. 권한은 설정된 직후 활성화됩니다.

OSGi 프레임워크 서비스는 OSGi Alliance 웹 사이트의 릴리스 4 다운로드 페이지에서 제공되는 **OSGi Service Platform Release 4** 사양의 별도의 장에 자세히 설명되어 있습니다.

1.4.4. OSGi Compendium 서비스

OSGi 프레임워크 서비스 외에도 OSGi Alliance는 선택적 표준화된 호환성 서비스를 정의합니다. OSGi compendium 서비스는 로깅 및 기본 설정과 같은 작업에 대한 API를 제공합니다. 이러한 서비스는 OSGi Alliance 웹 사이트의 릴리스 4 다운로드 페이지에서 제공되는 **OSGi Service Platform, Service Compendium** 에 설명되어 있습니다.

Configuration Admin compendium 서비스는 구성 정보를 유지하고 관련 당사자에게 배포하는 중앙 허브와 같습니다. 구성 관리자 서비스는 배포된 번들에 대한 구성 정보를 지정하고 번들이 활성 상태일 때 해당 데이터를 수신하도록 합니다. 번들의 구성 데이터는 이름-값 쌍 목록입니다. 1.2절. "Apache Karaf 아키텍처"을 참조하십시오.

1.5. OSGI 번들

1.5.1. 개요

OSGi를 사용하면 애플리케이션을 번들로 모듈화할 수 있습니다. 각 번들은 외부 종속성을 명시적으로 선언하는 클래스, JAR 및 구성 파일의 밀접하게 결합된 동적으로 로드 가능한 컬렉션입니다. OSGi에서 번들은 기본 배포 형식입니다. 번들은 JAR에 패키지로 제공되는 애플리케이션이며 설치, 시작, 중지, 업데이트 및 제거할 수 있습니다.

OSGi는 번들 개발을 위한 동적이고 간결하며 일관된 프로그래밍 모델을 제공합니다. 개발 및 배포는 서비스의 사양(Java 인터페이스)을 구현에서 분리하여 간소화됩니다.

OSGi 번들 추상화를 사용하면 모듈에서 Java 클래스를 공유할 수 있습니다. 이는 정적 형태의 재사용입니다. 종속 번들이 시작될 때 공유 클래스를 사용할 수 있어야합니다.

번들은 OSGi 매니페스트 파일에 메타데이터가 있는 JAR 파일입니다. 번들에는 클래스 파일 및 선택적으로 다른 리소스 및 네이티브 라이브러리가 포함되어 있습니다. 번들의 어떤 패키지가 외부에 표시되는지(내보냄 패키지)와 번들에 필요한 외부 패키지(가져오기 패키지)를 명시적으로 선언할 수 있습니다.

모듈 계층은 번들과 다른 번들의 패키지 숨기기 간 Java 패키지 패키징 및 공유를 처리합니다. OSGi 프레임워크는 번들 간의 종속성을 동적으로 확인합니다. 프레임워크는 가져온 패키지 및 내보낸 패키지와 일치하도록 번들 확인을 수행합니다. 배포된 번들의 여러 버전을 관리할 수도 있습니다.

1.5.2. OSGi에서 클래스 로드

OSGi는 트리 모델이 아닌 클래스 로드에 그래프 모델을 사용합니다(JVM에서 사용됨). 번들은 런타임 클래스 로드 충돌 없이 클래스를 표준화된 방식으로 공유하고 다시 사용할 수 있습니다.

각 번들에는 고유한 내부 클래스 경로가 있으므로 필요한 경우 독립적인 단위로 사용될 수 있습니다.

OSGi에서 클래스 로드의 이점은 다음과 같습니다.

- 번들 간에 클래스를 직접 공유합니다. JAR을 상위 클래스 로더로 승격할 필요가 없습니다.
- 충돌 없이 동일한 클래스의 다른 버전을 동시에 배포할 수 있습니다.

2장. APACHE KARAF 시작 및 중지

초록

Apache Karaf는 서버를 시작하고 중지하기 위한 간단한 명령줄 도구를 제공합니다.

2.1. APACHE KARAF 시작

Apache Karaf 런타임을 배포하는 기본 방법은 활성 콘솔이 있는 독립 실행형 서버로 배포하는 것입니다. 콘솔 없이 런타임을 백그라운드 프로세스로 배포할 수도 있습니다.

2.1.1. 환경 설정

환경을 수정하지 않고도 설치의 **bin** 하위 디렉터리에서 Karaf 런타임을 직접 시작할 수 있습니다. 그러나 다른 폴더에서 시작하려면 다음과 같이 Karaf 설치의 **bin** 디렉토리를 **PATH** 환경 변수에 추가해야 합니다.

Windows

```
set PATH=%PATH%;InstallDir\bin
```

Linux/UNIX

```
export PATH=$PATH,InstallDir/bin`
```

2.1.2. 콘솔 모드에서 런타임 시작

설치 디렉토리에서 Karaf 런타임을 시작하는 경우 다음 명령을 사용합니다.

Windows

```
bin\fuse.bat
```

Linux/UNIX

```
./bin/fuse
```

Karaf가 올바르게 시작되면 콘솔에 다음이 표시됩니다.

```
Red Hat Fuse starting up. Press Enter to open the shell now...
100%
[========================================================================]

Karaf started in 8s. Bundle stats: 220 active, 220 total


 _____         _  _  _      _   ______
|  _ \ ___  __| | | | | | __ _| |_  |  ____|   _ ____  ____
| |_) / _ \/ _` | | |_| |/ _` | __| | |_ | | | / __|/ _ \
|  _ <  __/ (_| | |  _  | (_| | |_  |  _|| |_| \__ \  __/
|_| \_\___|\__,_| |_| |_|\__,_|\__| |_|   \__,_|___/___|

  Fuse (7.x.x.fuse-xxxxxx-redhat-xxxxx)
  http://www.redhat.com/products/jbossenterprisemiddleware/fuse/
```

```
Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

Open a browser to http://localhost:8181/hawtio to access the management console

Hit '<ctrl-d>' or 'shutdown' to shutdown Red Hat Fuse.

karaf@root()>
```

참고

버전 Fuse 6.2.1 이후 콘솔 모드에서 시작하면 두 개의 프로세스가 생성됩니다. 즉, Karaf 콘솔을 실행하는 상위 프로세스 **./bin/karaf** 와 **java** JVM에서 Karaf 서버를 실행하는 하위 프로세스가 생성됩니다. 그러나 종료 동작은 이전과 동일하게 유지됩니다. 즉, Ctrl-D 또는 **osgi:shutdown** 을 사용하여 콘솔에서 서버를 종료하여 두 프로세스를 모두 종료할 수 있습니다.

2.1.3. 서버 모드에서 런타임 시작

서버 모드에서 시작하면 로컬 콘솔 없이 백그라운드에서 Apache Karaf가 실행됩니다. 그런 다음 원격 콘솔을 사용하여 실행 중인 인스턴스에 연결합니다. 자세한 내용은 17.2절. "원격 연결 및 연결 해제" 을 참조하십시오.

서버 모드에서 Karaf를 시작하려면 다음을 실행합니다.

Windows

```
bin\start.bat
```

Linux/UNIX

```
./bin/start
```

2.1.4. 클라이언트 모드에서 런타임 시작

프로덕션 환경에서는 로컬 콘솔만 사용하여 런타임 인스턴스에 액세스할 수 있어야 합니다. 즉, SSH 콘솔 포트를 통해 원격으로 런타임에 연결할 수 없습니다. 다음 명령을 사용하여 클라이언트 모드에서 런타임을 시작하여 이 작업을 수행할 수 있습니다.

Windows

```
bin\fuse.bat client
```

Linux/UNIX

```
./bin/fuse client
```

참고

클라이언트 모드에서 시작하면 SSH 콘솔 포트만 비활성화됩니다(일반적으로 8101 포트). 기타 Karaf 서버 포트(예: Cryostat 관리 RMI 포트)가 정상적으로 열립니다.

2.1.5. 디버그 모드에서 Fuse 실행

디버그 모드에서 Fuse를 실행하면 오류를 보다 효율적으로 식별하고 해결할 수 있습니다. 이 옵션은 기본적으로 비활성화되어 있습니다. 활성화된 경우 Fuse는 포트 **5005** 에서 JDWP 소켓을 시작합니다.

디버그 모드에서 Fuse를 실행하는 방법은 세 가지가 있습니다.

- 2.1.5.1절. "Karaf 환경 변수 사용"
- 2.1.5.2절. "Fuse 디버그 실행"
- 2.1.5.3절. "Fuse 디버그 실행"

2.1.5.1. Karaf 환경 변수 사용

이 접근 방식은 **KARAF_DEBUG** 환경 변수(**=1**)를 활성화한 다음 컨테이너를 시작합니다.

```
$ export KARAF_DEBUG=1
$ bin/start
```

2.1.5.2. Fuse 디버그 실행

이 접근 방식은 **suspend** 옵션이 **n** (아니오)으로 설정된 **debug** 를 실행합니다.

```
$ bin/fuse debug
```

2.1.5.3. Fuse 디버그 실행

이 접근 방식은 **suspend** 옵션이 **y** (예)로 설정된 **debug** 를 실행합니다.

참고

suspend to yes로 설정하면 디버거가 연결될 때까지 JVM이 **main()** 를 실행하기 직전에 일시 중지되고 실행이 다시 시작됩니다.

```
$ bin/fuse debugs
```

2.2. APACHE KARAF 중지

콘솔 내에서 또는 stop 스크립트를 사용하여 Apache Karaf 인스턴스를 **중지** 할 수 있습니다.

2.2.1. 로컬 콘솔에서 인스턴스 중지

fuse 또는 **fuse 클라이언트를** 실행하여 Karaf 인스턴스를 시작한 경우 **karaf** > 프롬프트에서 다음 중 하나를 수행하여 중지할 수 있습니다.

- **shutdown**을 입력합니다.
- **Ctrl**+**D**누릅니다.

2.2.2. 서버 모드에서 실행 중인 인스턴스 중지

다음과 같이 **InstallDir/bin** 디렉토리에서 stop(**.CAST)을 호출하여 로컬로 실행 중인 Karaf 인스턴스(루트 컨테이너)를 중지** 할 수 있습니다.

Windows

```
bin\stop.bat
```

Linux/UNIX

```
./bin/stop
```

Karaf **stop** 스크립트에서 호출한 종료 메커니즘은 Apache Tomcat에서 구현된 종료 메커니즘과 유사합니다. Karaf 서버는 종료 알림을 수신하기 위해 전용 종료 포트(SSH 포트와**동일하지 않음**)를 엽니다. 기본적으로 shutdown 포트는 임의로 선택되지만 원하는 경우 특정 포트를 사용하도록 구성할 수 있습니다.

필요한 경우 **InstallDir/etc/config.properties** 파일에서 다음 속성을 설정하여 종료 포트를 사용자 지정할 수 있습니다.

karaf.shutdown.port

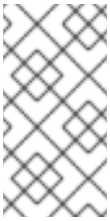
종료 포트로 사용할 TCP 포트를 지정합니다. 이 속성을 **-1** 로 설정하면 포트가 비활성화됩니다. 기본값은 **0** 입니다(랜덤 포트의 경우).

참고

bin/stop 스크립트를 사용하여 원격 호스트에서 실행 중인 Karaf 서버를 종료하려면 이 속성을 원격 호스트의 종료 포트와 동일하게 설정해야 합니다. 그러나 이 설정은 **etc/config.properties** 파일과 동일한 호스트에 있는 Karaf 서버에도 영향을 미칩니다.

karaf.shutdown.host

shutdown 포트가 바인딩된 호스트 이름을 지정합니다. 이 설정은 다중 홈 호스트에서 유용할 수 있습니다. 기본값은 **localhost** 입니다.



참고

bin/stop 스크립트를 사용하여 원격 호스트에서 실행 중인 Karaf 서버를 종료하려면 이 속성을 원격 호스트의 호스트 이름(또는 IP 주소)으로 설정해야 합니다. 그러나 이 설정은 **etc/config.properties** 파일과 동일한 호스트에 있는 Karaf 서버에도 영향을 미칩니다.

karaf.shutdown.port.file

Karaf 인스턴스가 시작되면 현재 종료 포트를 이 속성에서 지정한 파일에 씁니다. **stop** 스크립트는 이 속성에서 지정한 파일을 읽고 현재 종료 포트의 값을 검색합니다. 기본값은 **${karaf.data}/port** 입니다.

karaf.shutdown.command

종료를 트리거하려면 종료 포트로 전송해야 하는 UUID 값을 지정합니다. UUID 값이 시크릿을 유지하는 한 이는 기본 보안 수준을 제공합니다. 예를 들어 **etc/config.properties** 파일을 읽고 일반 사용자가 이 값을 읽지 못하도록 할 수 있습니다.
Apache Karaf가 처음으로 시작되면 임의의 UUID 값이 자동으로 생성되고 이 설정이 **etc/config.properties** 파일의 끝에 기록됩니다. 또는 **karaf.shutdown.command** 가 이미 설정된 경우 Karaf 서버는 기존 UUID 값을 사용합니다(필요한 경우 UUID 설정을 사용자 지정할 수 있음).

참고

bin/stop 스크립트를 사용하여 원격 호스트에서 실행 중인 Karaf 서버를 종료하려면 이 속성을 원격 호스트의 **karaf.shutdown.command** 값과 동일하게 설정해야 합니다. 그러나 이 설정은 **etc/config.properties** 파일과 동일한 호스트에 있는 Karaf 서버에도 영향을 미칩니다.

2.2.3. 원격 인스턴스 중지

17.3절. "원격 컨테이너 중지" 에 설명된 대로 원격 호스트에서 실행 중인 컨테이너 인스턴스를 중지할 수 있습니다.

3장. 기본 보안

이 장에서는 Karaf를 처음 시작하기 전에 보안을 구성하는 기본 단계에 대해 설명합니다. 기본적으로 Karaf는 안전하지만 서비스는 원격으로 액세스할 수 없습니다. 이 장에서는 Karaf가 노출하는 포트에 대한 보안 액세스를 활성화하는 방법을 설명합니다.

3.1. 기본 보안 구성

3.1.1. 개요

노출된 모든 포트에는 사용자 인증이 필요하고 사용자가 처음에 정의되어 있지 않기 때문에 Apache Karaf 런타임은 기본적으로 네트워크 공격에 대해 보호됩니다. 즉, Apache Karaf 런타임은 기본적으로 원격으로 액세스할 수 없습니다.

원격으로 런타임에 액세스하려면 여기에 설명된 대로 먼저 보안 구성을 사용자 지정해야 합니다.

3.1.2. 컨테이너를 시작하기 전에

Karaf 컨테이너에 대한 원격 액세스를 활성화하려면 컨테이너를 시작하기 전에 보안 JAAS 사용자를 만들어야 합니다.

3.1.3. 보안 JAAS 사용자 만들기

기본적으로 컨테이너에 대한 JAAS 사용자는 정의되어 있지 않으므로 원격 액세스를 효과적으로 비활성화합니다(로그인할 수 없음).

보안 JAAS 사용자를 생성하려면 **InstallDir/etc/users.properties** 파일을 편집하고 다음과 같이 새 user 필드를 추가합니다.

```
Username=Password,admin
```

여기서 **Username** 및 **Password** 는 새 사용자 자격 증명입니다. **admin** 역할은 이 사용자에게 컨테이너의 모든 관리 및 관리 기능에 액세스할 수 있는 권한을 제공합니다.

앞에 0인 숫자 사용자 이름을 정의하지 마십시오. 이러한 사용자 이름은 항상 로그인 시도가 실패합니다. 콘솔이 사용하는 Karaf 쉘은 입력이 숫자로 표시될 때 선행 0이 삭제되기 때문입니다. 예를 들면 다음과 같습니다.

```
karaf@root> echo 0123
123
karaf@root> echo 00.123
0.123
karaf@root>
```

주의

강력한 암호로 사용자 지정 사용자 자격 증명을 정의하는 것이 좋습니다.

3.1.4. 역할 기반 액세스 제어

Karaf 컨테이너는 Cryostat 프로토콜, Karaf 명령 콘솔 및 Fuse 관리 콘솔을 통해 액세스를 규제하는 역할 기반 액세스 제어를 지원합니다. 사용자에게 역할을 할당할 때 표 3.1. "액세스 제어를 위한 표준 역할" 에 설명된 액세스 수준을 제공하는 표준 역할 세트 중에서 선택할 수 있습니다.

표 3.1. 액세스 제어를 위한 표준 역할

역할	설명
뷰어	컨테이너에 대한 읽기 전용 액세스 권한을 부여합니다.
관리자	애플리케이션을 배포 및 실행하려는 일반 사용자에게 적절한 수준에서 읽기-쓰기 액세스 권한을 부여합니다. 그러나 중요한 컨테이너 구성 설정에 대한 액세스를 차단합니다.
admin	컨테이너에 대한 무제한 액세스 권한을 부여합니다.
ssh	SSH 포트를 통해 원격 콘솔 액세스 권한에 대한 권한을 부여합니다.

역할 기반 액세스 제어에 대한 자세한 내용은 역할 기반 액세스 제어를 참조하십시오.

3.1.5. Apache Karaf 컨테이너에서 노출된 포트

다음은 컨테이너에서 노출하는 포트는 다음과 같습니다.

- **콘솔 포트** Cryostat- Cryostat는 Apache Karaf 쉘 명령을 통해 컨테이너 인스턴스에 대한 원격 제어를 활성화합니다. 이 포트는 기본적으로 활성화되어 있으며 JAAS 인증 및 SSH에 의해 보호됩니다.
- **Cryostat 포트** Cryostat- Cryostatenables를 통해 컨테이너를 관리할 수 있습니다. 이 포트는 기본적으로 활성화되어 있으며 JAAS 인증에 의해 보호됩니다.
- **웹 콘솔 포트 Cryo**stat- Cryostat는 웹 콘솔 서블릿을 호스팅할 수 있는 포함된 Cryostat 컨테이너에 대한 액세스를 제공합니다. 기본적으로 Fuse Console은 Cryostat 컨테이너에 설치됩니다.

3.1.6. 원격 콘솔 포트 활성화

다음 조건이 모두 충족될 때마다 원격 콘솔 포트에 액세스할 수 있습니다.

- JAAS는 하나 이상의 로그인 인증 정보 세트로 구성됩니다.
- Karaf 런타임이 클라이언트 모드에서 **시작되지** 않았습니다(클라이언트 모드는 원격 콘솔 포트를 완전히 비활성화합니다).

예를 들어 컨테이너가 실행 중인 동일한 머신의 원격 콘솔 포트에 로그인하려면 다음 명령을 입력합니다.

```
./client -u Username -p Password
```

여기서 **Username** 및 **Password** 는 **ssh** 역할이 있는 JAAS 사용자의 자격 증명입니다. 원격 포트를 통해 Karaf 콘솔에 액세스하는 경우 권한은 **etc/users.properties** 파일의 사용자에게 할당된 역할에 따라 달라집니다. 전체 콘솔 명령 세트에 액세스하려면 사용자 계정에 **admin** 역할이 있어야합니다.

3.1.7. 원격 콘솔 포트의 보안 강화

다음 조치를 사용하여 원격 콘솔 포트에서 보안을 강화할 수 있습니다.

- JAAS 사용자 자격 증명에 강력한 암호가 있는지 확인합니다.
- X.509 인증서를 사용자 지정합니다(Java 키 저장소 파일인 **InstallDir/etc/host.key** 를 사용자 지정 키 쌍으로 교체).

3.1.8. Cryostat 포트 활성화

Cryostat 포트는 기본적으로 활성화되며 JAAS 인증에 의해 보호됩니다. Cryostat 포트에 액세스하려면 하나 이상의 로그인 인증 정보 세트를 사용하여 JAAS를 구성해야 합니다. Cryostat 포트에 연결하려면 Cryostat 클라이언트(예: **jconsole**)를 열고 다음 Cryostat URI에 연결합니다.

```
service:jmx:rmi:///jndi/rmi://localhost:1099/karaf-root
```

연결하려면 유효한 JAAS 자격 증명도 제공해야 합니다.

참고

일반적으로 tail of the Cryostat URI 형식은 **/karaf-*ContainerName*** 입니다. 컨테이너 이름을 **root** 에서 다른 이름으로 변경하는 경우 그에 따라 Cryostat URI를 수정해야 합니다.

3.1.9. Fuse Console 포트의 보안 강화

Fuse Console은 JAAS 인증에 의해 이미 보호됩니다. SSL 보안을 추가하려면 Cryostat HTTP 서버 보안을 참조하십시오.

4장. APACHE KARAF AS A SERVICE 설치

이 장에서는 제공된 템플릿을 사용하여 Apache Karaf 인스턴스를 시스템 서비스로 시작하는 방법에 대한 정보를 제공합니다.

4.1. 개요

서비스 스크립트 템플릿을 사용하여 운영 체제별 init 스크립트를 사용하여 Karaf 인스턴스를 실행할 수 있습니다. 이러한 템플릿은 **bin/contrib** 디렉토리에서 찾을 수 있습니다.

4.2. KARAF AS A SERVICE 실행

karaf-service.sh 유틸리티는 템플릿을 사용자 지정하는 데 도움이 됩니다. 이 유틸리티는 운영 체제와 기본 init 시스템을 자동으로 식별하고 즉시 사용 가능한 init 스크립트를 생성합니다. **JAVA_HOME** 및 몇 가지 다른 환경 변수를 설정하여 스크립트에 맞게 사용자 지정할 수도 있습니다.

생성된 스크립트는 다음 두 파일로 구성됩니다.

- init 스크립트
- init 구성 파일

4.3. SYSTEMD

karaf-service.sh 유틸리티가 **systemd** 를 식별하면 다음 세 개의 파일을 생성합니다.

- 루트 Apache Karaf 컨테이너를 관리하는 **systemd** 장치 파일입니다.
- 루트 Apache Karaf 컨테이너에서 사용하는 변수가 있는 **systemd** 환경 파일입니다.
- *(지원되지 않음)* Apache Karaf 하위 컨테이너를 관리하는 **systemd** 템플릿 단위 파일입니다.

예를 들어 **/opt/karaf-4** 에 설치된 Karaf 인스턴스에 대한 서비스를 설정하려면 서비스에 이름 **karaf-4**:

```
$ ./karaf-service.sh -k /opt/karaf-4 -n karaf-4
Writing service file "/opt/karaf-4/bin/contrib/karaf-4.service"
Writing service configuration file ""/opt/karaf-4/etc/karaf-4.conf"
Writing service file "/opt/karaf-4/bin/contrib/karaf-4@.service"
$ sudo cp /opt/karaf-4/bin/contrib/karaf-4.service /etc/systemd/system
$ sudo systemctl enable karaf-4.service
```

4.4. SYSV

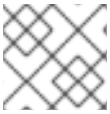
karaf-service.sh 유틸리티에서 SysV 시스템을 식별하면 다음 두 개의 파일을 생성합니다.

- 루트 Apache Karaf 컨테이너를 관리하는 init 스크립트입니다.
- 루트 Apache Karaf 컨테이너에서 사용하는 변수가 있는 환경 파일입니다.

예를 들어 **/opt/karaf-4** 에 설치된 Karaf 인스턴스에 대한 서비스를 설정하려면 서비스에 이름 **karaf-4**:

```
$ ./karaf-service.sh -k /opt/karaf-4 -n karaf-4
Writing service file "/opt/karaf-4/bin/contrib/karaf-4"
```

```
Writing service configuration file "/opt/karaf-4/etc/karaf-4.conf"
$ sudo ln -s /opt/karaf-4/bin/contrib/karaf-4 /etc/init.d/
$ sudo chkconfig karaf-4 on
```



참고

부팅 시 서비스를 시작하려면 운영 체제 init 가이드를 참조하십시오.

4.5. SOLARIS SMF

karaf-service.sh 유틸리티는 Solaris 운영 체제를 식별할 때 단일 파일을 생성합니다.

예를 들어 **/opt/karaf-4** 에 설치된 Karaf 인스턴스에 대한 서비스를 설정하려면 서비스에 이름 **karaf-4**:

```
$ ./karaf-service.sh -k /opt/karaf-4 -n karaf-4
Writing service file "/opt/karaf-4/bin/contrib/karaf-4.xml"
$ sudo svccfg validate /opt/karaf-4/bin/contrib/karaf-4.xml
$ sudo svccfg import /opt/karaf-4/bin/contrib/karaf-4.xml
```

참고

생성된 SMF 설명자는 일시적인 것으로 정의되므로 시작 방법을 한 번만 실행할 수 있습니다.

4.6. WINDOWS

Windows 서비스로 Apache Karaf 설치는 **winsw** 를 통해 지원됩니다.

Apache Karaf를 Windows 서비스로 설치하려면 다음 단계를 수행합니다.

1. **karaf-service-win.exe** 파일의 이름을 **karaf-4.exe** 로 변경합니다.
2. **karaf-service-win.xml** 파일의 이름을 **karaf-4.xml** 로 변경합니다.
3. 필요에 따라 서비스 설명자를 사용자 지정합니다.
4. 서비스 실행 파일을 사용하여 서비스를 설치, 시작 및 중지합니다.

예를 들면 다음과 같습니다.

```
C:\opt\apache-karaf-4\bin\contrib> karaf-4.exe install
C:\opt\apache-karaf-4\bin\contrib> karaf-4.exe start
```

4.7. KARAF-SERVICE.SH 옵션

다음과 같이 **karaf-service.sh** 유틸리티에 대한 옵션을 명령줄 옵션으로 지정하거나 환경 변수를 설정하여 지정할 수 있습니다.

명령줄 옵션	환경 변수	설명
-k	**KARAF_SERVICE_PATH**	Karaf 설치 경로

명령줄 옵션	환경 변수	설명
-d	**KARAF_SERVICE_DATA**	Karaf 데이터 경로(기본값: **${KARAF_SERVICE_PATH}/data**)
-c	**KARAF_SERVICE_CONF**	Karaf 구성 파일(기본값: **${KARAF_SERVICE_PATH}/etc/$KARAF_SERVICE_NAME}.conf**)
-t	**KARAF_SERVICE_ETC**	Karaf **etc** 경로(기본값: **${KARAF_SERVICE_PATH}/etc}**)
-p	**KARAF_SERVICE_PIDFILE**	Karaf PID 경로(기본값: **${KARAF_SERVICE_DATA}/$KARAF_SERVICE_NAME}.pid**)
-n	**KARAF_SERVICE_NAME**	Karaf 서비스 이름(기본값: **karaf**)
-e	**KARAF_ENV**	서비스에 대한 환경 변수 설정 **NAME=VALUE** 를 지정합니다 (한 번 이상 지정할 수 있음)
-u	**KARAF_SERVICE_USER**	Karaf 사용자
-g	**KARAF_SERVICE_GROUP**	Karaf 그룹(기본값: **${KARAF_SERVICE_USER}**)
-l	**KARAF_SERVICE_LOG**	Karaf 콘솔 로그(기본값: **${KARAF_SERVICE_DATA}/log/${KARAF_SERVICE_NAME}-console.log**)
-f	**KARAF_SERVICE_TEMPLATE**	사용할 템플릿 파일
-x	**KARAF_SERVICE_EXECUTABLE**	Karaf 실행 이름(기본값: **karaf** Cryostat- Cryostat **데몬** 및 **중지** 명령을 지원해야 함)
-h		도움말 메시지

5장. OSGI 번들 빌드

초록

이 장에서는 Maven을 사용하여 OSGi 번들을 빌드하는 방법을 설명합니다. 번들 빌드의 경우 Maven 번들 플러그인은 OSGi 번들 헤더 생성을 자동화할 수 있기 때문에 핵심 역할을 수행합니다. 전체 샘플 프로젝트를 생성하는 Maven archetypes는 번들 프로젝트의 시작점도 제공할 수 있습니다.

5.1. 번들 프로젝트 생성

5.1.1. Maven archetypes를 사용하여 번들 프로젝트 생성

빠른 시작을 돕기 위해 Maven archetype을 호출하여 Maven 프로젝트의 초기 개요를 생성할 수 있습니다 (MM archetype은 프로젝트 마법사와 유사합니다). 다음 Maven archetype은 OSGi 번들을 빌드하는 프로젝트를 생성합니다.

5.1.2. Apache Camel archetype

Apache Camel OSGi archetype은 OSGi 컨테이너에 배포할 수 있는 경로를 빌드하기 위한 프로젝트를 생성합니다.

다음 예제에서는 Maven archetype 명령을 좌표와 함께 *GroupId*:*ArtifactId*:*Version* , 을 사용하여 **camel-blueprint** 프로젝트를 생성하는 방법을 보여줍니다.

```
mvn archetype:generate \
 -DarchetypeGroupId=org.apache.camel.archetypes \
 -DarchetypeArtifactId=camel-archetype-blueprint \
 -DarchetypeVersion=2.23.2.fuse-7_11_1-00015-redhat-00002
```

이 명령을 실행한 후 Maven에서 **GroupId**,**ArtifactId**, **Version** 을 지정하라는 메시지를 표시합니다.

5.1.3. 번들 빌드

기본적으로 이전 archetypes는 지정된 아티팩트 ID인 *ArtifactId* 와 이름이 동일한 새 디렉터리에 프로젝트를 생성합니다. 새 프로젝트에서 정의한 번들을 빌드하려면 명령 프롬프트를 열고 프로젝트 디렉터리(즉, **pom.xml** 파일이 포함된 디렉터리)로 이동하여 다음 Maven 명령을 입력합니다.

```
mvn install
```

이 명령의 영향은 모든 Java 소스 파일을 컴파일하여 *ArtifactId*/**target** 디렉터리에 번들 JAR을 생성한 다음 로컬 Maven 리포지토리에 생성된 JAR을 설치하는 것입니다.

5.2. 기존 MAVEN 프로젝트 수정

5.2.1. 개요

이미 Maven 프로젝트가 있고 OSGi 번들을 생성하도록 수정하려면 다음 단계를 수행합니다.

1. 5.2.2절. “패키지 유형을 번들로 변경”.
2. 5.2.3절. “POM에 번들 플러그인을 추가합니다.” .

3. 5.2.4절. "번들 플러그인 사용자 정의".
4. 5.2.5절. "JDK 컴파일러 버전 사용자 정의".

5.2.2. 패키지 유형을 번들로 변경

프로젝트의 **pom.xml** 파일에서 패키지 유형을 **번들로 변경하여 OSGi 번들을** 생성하도록 Maven을 구성합니다. 다음 예와 같이 **패키징** 요소의 내용을 **번들** 로 변경합니다.

```
<project ... >
  ...
  <packaging>bundle</packaging>
  ...
</project>
```

이 설정의 영향은 Maven 번들 플러그인인 **maven-bundle-plugin** 을 선택하여 이 프로젝트에 대한 패키징을 수행하는 것입니다. 그러나 이 설정은 POM에 번들 플러그인을 명시적으로 추가할 때까지는 적용되지 않습니다.

5.2.3. POM에 번들 플러그인을 추가합니다.

Maven 번들 플러그인을 추가하려면 다음 샘플 **플러그인** 요소를 복사하여 프로젝트 **pom.xml** 파일의 **project/build/plugins** 섹션에 붙여넣습니다.

```
<project ... >
  ...
  <build>
    <defaultGoal>install</defaultGoal>
    <plugins>
      ...
    <plugin>
       <groupId>org.apache.felix</groupId>
       <artifactId>maven-bundle-plugin</artifactId>
       <version>3.3.0</version>
       <extensions>true</extensions>
       <configuration>
         <instructions>
           <Bundle-SymbolicName>${project.groupId}.${project.artifactId}
           </Bundle-SymbolicName>
           <Import-Package>*</Import-Package>
         </instructions>
       </configuration>
     </plugin>
    </plugins>
  </build>
  ...
</project>
```

여기서 bundle 플러그인은 **instructions** 요소의 설정으로 구성됩니다.

5.2.4. 번들 플러그인 사용자 정의

Apache CXF용 번들 플러그인 구성에 대한 몇 가지 구체적인 권장 사항은 5.3절. "번들로 웹 서비스 패키징" 에서 참조하십시오.

5.2.5. JDK 컴파일러 버전 사용자 정의

POM 파일에서 JDK 버전을 지정하는 것은 거의 항상 필요합니다. 코드에서 Java 언어의 최신 기능을 사용하는 경우(예: 일반, 정적 가져오기 등) POM에서 JDK 버전을 사용자 지정하지 않은 경우 Maven은 소스 코드를 컴파일하지 못합니다. **JAVA_HOME** 및 **PATH** 환경 변수를 JDK의 올바른 값으로 설정하는 것만으로는 충분하지 **않으며** POM 파일도 수정해야 합니다.

POM 파일을 구성하려면 JDK 1.8에 도입된 Java 언어 기능을 허용하도록 다음 **maven-compiler-plugin** 플러그인을 POM에 추가합니다(아직 없는 경우).

```
<project ... >
  ...
  <build>
    <defaultGoal>install</defaultGoal>
    <plugins>
      ...
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

5.3. 번들로 웹 서비스 패키징

5.3.1. 개요

이 섹션에서는 Apache CXF 애플리케이션의 기존 Maven 프로젝트를 수정하여 프로젝트에서 Red Hat Fuse OSGi 컨테이너에 배포에 적합한 OSGi 번들을 생성하는 방법을 설명합니다. Maven 프로젝트를 변환하려면 프로젝트의 POM 파일과 프로젝트의 블루프린트 파일(**META-INF/spring**에 배치됨)을 수정해야 합니다.

5.3.2. POM 파일을 수정하여 번들을 생성

번들을 생성하도록 Maven POM 파일을 구성하려면 기본적으로 POM의 패키지 유형을 번들로 변경하고 POM **번들** 플러그인을 POM에 추가하는 두 가지 변경 사항이 있습니다. 자세한 내용은 5.1절. "번들 프로젝트 생성"의 내용을 참조하십시오.

5.3.3. 필수 가져오기 패키지

애플리케이션이 Apache CXF 구성 요소를 사용하려면 해당 패키지를 애플리케이션의 번들로 가져와야 합니다. Apache CXF의 종속성이 복잡하기 때문에 Maven 번들 플러그인 또는 **bnd** 툴을 사용하여 필요한 가져오기를 자동으로 결정할 수 없습니다. 명시적으로 선언해야 합니다.

다음 패키지를 번들로 가져와야 합니다.

```
javax.jws
javax.wsdl
```

```
javax.xml.bind
javax.xml.bind.annotation
javax.xml.namespace
javax.xml.ws
org.apache.cxf.bus
org.apache.cxf.bus.spring
org.apache.cxf.bus.resource
org.apache.cxf.configuration.spring
org.apache.cxf.resource
org.apache.cxf.jaxws
org.springframework.beans.factory.config
```

5.3.4. 샘플 Maven 번들 플러그인 지침

예 5.1. "필수 가져오기 패키지 구성" 필수 패키지를 가져오기 위해 POM에서 Maven 번들 플러그인을 구성하는 방법을 보여줍니다. 필수 가져오기 패키지는 **Import-Package** 요소 내에 쉼표로 구분된 목록으로 표시됩니다. 와일드카드인 * 의 모양을 목록의 마지막 요소로 기록해 둡니다. 와일드카드를 사용하면 현재 번들의 Java 소스 파일을 스캔하여 가져올 추가 패키지를 검색할 수 있습니다.

예 5.1. 필수 가져오기 패키지 구성

```
<project ... >
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            ...
            <Import-Package>
              javax.jws,
              javax.wsdl,
              javax.xml.bind,
              javax.xml.bind.annotation,
              javax.xml.namespace,
              javax.xml.ws,
              org.apache.cxf.bus,
              org.apache.cxf.bus.spring,
              org.apache.cxf.bus.resource,
              org.apache.cxf.configuration.spring,
              org.apache.cxf.resource,
              org.apache.cxf.jaxws,
              org.springframework.beans.factory.config,
              *
            </Import-Package>
            ...
          </instructions>
        </configuration>
      </plugin>
    </plugins>
```

```
    </build>
    ...
</project>
```

5.3.5. 코드 생성 플러그인 추가

웹 서비스 프로젝트에서는 일반적으로 코드를 생성해야 합니다. Apache CXF는 Cryostat-WS 프런트 엔드에 대한 두 개의 Maven 플러그인을 제공하므로 코드 생성 단계를 빌드에 통합할 수 있습니다. 플러그인의 선택은 다음과 같이 Java 우선 접근 방식 또는 WSDL 우선 접근 방식을 사용하여 서비스를 개발하는지 여부에 따라 달라집니다.

- **Java 우선 접근 방식- cxf-java2ws-plugin** 플러그인을 사용합니다.
- **WSDL 우선 접근 방식- cxf-codegen-plugin** 플러그인을 사용합니다.

5.3.6. OSGi 구성 속성

OSGi 구성 관리 서비스는 구성 설정을 OSGi 번들에 전달하는 메커니즘을 정의합니다. 구성에 이 서비스를 사용할 필요는 없지만 일반적으로 번들 애플리케이션을 구성하는 가장 편리한 방법입니다. 블루프린트는 OSGi 구성을 지원하여 OSGi 구성 관리 서비스에서 가져온 값을 사용하여 블루프린트 파일의 변수를 대체할 수 있습니다.

OSGi 구성 속성을 사용하는 방법에 대한 자세한 내용은 5.3.7절. “번들 플러그인 구성” 및 9.6절. “기능에 OSGi 구성 추가” 을 참조하십시오.

5.3.7. 번들 플러그인 구성

5.3.7.1. 개요

번들 플러그인에는 작동하는 데 정보가 거의 필요하지 않습니다. 모든 필수 속성은 기본 설정을 사용하여 유효한 OSGi 번들을 생성합니다.

기본값만 사용하여 유효한 번들을 생성할 수 있지만 일부 값을 수정해야 합니다. 플러그인의 **instructions** 요소 내에 대부분의 속성을 지정할 수 있습니다.

5.3.7.2. 구성 속성

일반적으로 사용되는 일부 구성 속성은 다음과 같습니다.

- Bundle-SymbolicName
- Bundle-Name
- Bundle-Version
- export-Package
- private-Package
- Import-Package

5.3.7.3. 번들의 심볼릭 이름 설정

기본적으로 번들 플러그인은 **Bundle-SymbolicName** 속성 값을 *groupId* **+ "." +** *artifactId* 로 설정합니다.

- *groupId* 에 섹션이 하나만 있는 경우 클래스가 포함된 첫 번째 패키지 이름이 반환됩니다.
  예를 들어, Id 그룹이 **commons-logging:commons-logging** 인 경우 번들의 심볼릭 이름은 **org.apache.commons.logging** 입니다.

- *artifactId* 가 *groupId* 의 마지막 섹션과 동일한 경우 *groupId* 가 사용됩니다.
  예를 들어 POM에서 그룹 ID와 아티팩트 ID를 **org.apache.maven:maven** 로 지정하는 경우 번들의 심볼릭 이름은 **org.apache.maven** 입니다.

- *artifactId* 가 *groupId* 의 마지막 섹션으로 시작되면 해당 부분이 제거됩니다.
  예를 들어 POM에서 그룹 ID 및 아티팩트 ID를 **org.apache.maven:maven-core** 로 지정하는 경우 번들의 심볼릭 이름은 **org.apache.maven.core** 입니다.

번들의 심볼릭 이름에 고유한 값을 지정하려면 예 5.2. "번들의 심볼릭 이름 설정" 과 같이 플러그인의 **instructions** 요소에 **Bundle-SymbolicName** 자식을 추가합니다.

예 5.2. 번들의 심볼릭 이름 설정

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
     ...
    </instructions>
  </configuration>
</plugin>
```

5.3.7.4. 번들 이름 설정

기본적으로 번들의 이름은 **${project.name}** 로 설정됩니다.

번들 이름에 대한 자체 값을 지정하려면 예 5.3. "번들 이름 설정" 과 같이 플러그인의 **instructions** 요소에 **Bundle-Name** 자식을 추가합니다.

예 5.3. 번들 이름 설정

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Bundle-Name>JoeFred</Bundle-Name>
     ...
    </instructions>
  </configuration>
</plugin>
```

5.3.7.5. 번들 버전 설정

기본적으로 번들의 버전은 **${project.version}** 로 설정됩니다. 모든 대시(**-**)는 점(**.**)으로 교체되고 숫자는 최대 4자리로 채워집니다. 예를 들어 **4.2-SNAPSHOT** 은 **4.2.0.SNAPSHOT** 이 됩니다.

번들 버전에 대한 자체 값을 지정하려면 예 5.4. "번들 버전 설정" 과 같이 플러그인의 **instructions** 요소에 **Bundle-Version** 자식을 추가합니다.

예 5.4. 번들 버전 설정

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
    <Bundle-Version>1.0.3.1</Bundle-Version>
    ...
   </instructions>
  </configuration>
</plugin>
```

5.3.7.6. 내보낸 패키지 지정

기본적으로 OSGi 매니페스트의 **Export-Package** 목록은 기본 패키지 **. impl** 또는 **.internal** 을 포함하는 패키지를 **제외하고** 로컬 Java 소스 코드의 모든 패키지(**src/main/java**아래)로 채워집니다.

중요

플러그인 구성에서 **Private-Package** 요소를 사용하고 내보낼 패키지 목록을 지정하지 않는 경우 기본 동작에는 번들의 **Private-Package** 요소에 나열된 패키지만 포함됩니다. 패키지를 내보내지 않습니다.

기본 동작으로 인해 패키지가 매우 크고 비공개로 유지해야 하는 패키지를 내보낼 수 있습니다. 내보낸 패키지 목록을 변경하려면 플러그인의 **instructions** 요소에 **Export-Package** 자식을 추가할 수 있습니다.

Export-Package 요소는 번들에 포함할 패키지 목록을 지정하고 내보낼 패키지 목록을 지정합니다. 패키지 이름은 ***** 와일드카드 기호를 사용하여 지정할 수 있습니다. 예를 들어 **com.fuse.demo.*** 항목에는 **com.fuse.demo** 로 시작하는 프로젝트의 클래스 경로에 있는 모든 패키지가 포함되어 있습니다.

제외할 패키지를 지정할 수 있습니다! **!** 항목을 접두사로 지정합니다. 예를 들어 **!com.fuse.demo.private** 항목은 **com.fuse.demo.private** 패키지를 제외합니다.

패키지를 제외할 때 목록의 항목 순서가 중요합니다. 목록은 처음부터 순서대로 처리되며 후속 일치하지 않는 항목은 무시됩니다.

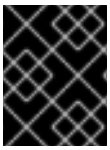
예를 들어 com.fuse.demo 패키지를 제외하고 **com.fuse.demo .demo 로 시작하는 모든 패키지를 포함하려면 다음을 사용하여 패키지를 나열합니다.**

```
!com.fuse.demo.private,com.fuse.demo.*
```

그러나 **com.fuse.demo.*,!com.fuse.demo.private** 을 사용하여 패키지를 나열하면 **com.fuse.demo.private** 이 첫 번째 패턴과 일치하므로 번들에 포함됩니다.

5.3.7.7. 개인 패키지 지정

패키지를 내보내지 **않고** 번들에 포함할 패키지 목록을 지정하려면 번들 플러그인 구성에 **Private-Package** 명령을 추가할 수 있습니다. 기본적으로 **Private-Package** 명령을 지정하지 않으면 로컬 Java 소스의 모든 패키지가 번들에 포함됩니다.



중요

패키지가 **Private-Package** 요소 및 **Export-Package** 요소의 항목과 일치하는 경우 **Export-Package** 요소가 우선합니다. 패키지가 번들에 추가되어 내보내집니다.

Private-Package 요소는 번들에 포함할 패키지 목록을 지정하는 **Export-Package** 요소와 유사하게 작동합니다. 번들 플러그인은 목록을 사용하여 번들에 포함할 프로젝트의 클래스 경로의 모든 클래스를 찾습니다. 이러한 패키지는 번들에 패키지되지만 내보내지 않습니다(**Export-Package** 명령으로도 선택하지 않는 한).

예 5.5. "번들에 개인 패키지 포함" 번들에 개인 패키지를 포함하는 구성 표시

예 5.5. 번들에 개인 패키지 포함

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Private-Package>org.apache.cxf.wsdlFirst.impl</Private-Package>
     ...
    </instructions>
  </configuration>
</plugin>
```

5.3.7.8. 가져온 패키지 지정

기본적으로 번들 플러그인은 OSGi 매니페스트의 **Import-Package** 속성을 번들 콘텐츠에서 참조하는 모든 패키지 목록으로 채웁니다.

기본 동작은 일반적으로 대부분의 프로젝트에 충분하지만 목록에 자동으로 추가되지 않는 패키지를 가져오려는 인스턴스를 찾을 수 있습니다. 기본 동작으로 인해 원하지 않는 패키지를 가져올 수도 있습니다.

번들에서 가져올 패키지 목록을 지정하려면 플러그인의 **instructions** 요소에 **Import-Package** 자식을 추가합니다. 패키지 목록의 구문은 **Export-Package** 요소 및 **Private-Package** 요소와 동일합니다.

중요

Import-Package 요소를 사용하면 플러그인에서 번들의 콘텐츠를 자동으로 검사하여 필요한 가져오기가 있는지 확인하지 않습니다. 번들 내용이 스캔되었는지 확인하려면 패키지 목록의 마지막 항목으로 * 를 배치해야 합니다.

예 5.6. "번들로 가져온 패키지 지정" 번들로 가져온 패키지를 지정하기 위한 구성 표시

예 5.6. 번들로 가져온 패키지 지정

```
<plugin>
  <groupId>org.apache.felix</groupId>
```

```
  <artifactId>maven-bundle-plugin</artifactId>
  <configuration>
   <instructions>
     <Import-Package>javax.jws, javax.wsdl, org.apache.cxf.bus, org.apache.cxf.bus.spring,
org.apache.cxf.bus.resource, org.apache.cxf.configuration.spring, org.apache.cxf.resource,
org.springframework.beans.factory.config, * </Import-Package>
      ...
   </instructions>
  </configuration>
</plugin>
```

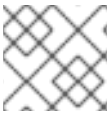
5.3.7.9. 더 많은 정보

번들 플러그인 구성에 대한 자세한 내용은 다음을 참조하십시오.

- olink:OsgiDependencies/OsgiDependencies
- Apache Felix 문서
- Peter Kriens' aQute Software Consultancy 웹 사이트

5.3.8. OSGI configAdmin 파일 이름 지정 규칙

PID 문자열(symbolic-name 구문)은 OSGI 사양의 하이픈을 허용합니다. 그러나 하이픈은 Apache **Felix.fileinstall** 및 **config:edit** 쉘 명령으로 해석되어 "관리 서비스" 및 "관리 서비스 팩토리"를 구분합니다. 따라서 PID 문자열에서 하이픈을 사용하지 않는 것이 좋습니다.



참고

구성 파일 이름은 PID 및 팩토리 PID와 관련이 있습니다.

6장. 핫 디플로이먼트 VS 수동 배포

초록

Fuse는 핫 배포 또는 수동 배포라는 두 가지 파일 배포 방법을 제공합니다. 관련 번들 컬렉션을 배포해야 하는 경우 단일 대신 *기능*으로 함께 배포하는 것이 좋습니다(9장. *기능 배포*참조).

6.1. 핫 디플로이먼트

6.1.1. 핫 배포 디렉토리

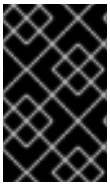
Fuse는 **FUSE_HOME/deploy** 디렉터리의 파일을 모니터링하고 hot은 이 디렉터리의 모든 항목을 배포합니다. 파일이 이 디렉터리에 복사될 때마다 런타임 내에 설치되고 시작됩니다. 나중에 **FUSE_HOME/deploy** 디렉터리에서 파일을 업데이트하거나 삭제할 수 있으며 변경 사항은 자동으로 처리됩니다.

예를 들어, 번들을 빌드한 경우 *ProjectDir*/**target/foo-1.0-SNAPSHOT.jar** 에서는 다음과 같이 FUSE_HOME/**deploy** 디렉터리에 복사하여 이 번들을 배포할 수 있습니다(UNIX 플랫폼에서 작업한다고 가정).

```
% cp ProjectDir/target/foo-1.0-SNAPSHOT.jar FUSE_HOME/deploy
```

6.2. 번들 배포 취소 시 핫

hot deploy 디렉토리에서 번들 배포를 취소하려면 **Apache Karaf 컨테이너가 실행되는 동안 FUSE_HOME/deploy 디렉터리에서 번들 파일을 삭제하면 됩니다**.



중요

컨테이너가 종료되는 동안 핫 배포 취소 메커니즘이 작동하지 **않습니다**. Karaf 컨테이너를 종료하고 **FUSE_HOME/deploy** 디렉터리에서 번들 파일을 삭제한 다음 Karaf 컨테이너를 다시 시작하면 컨테이너를 다시 시작한 후 번들이 배포되지 **않습니다**.

bundle:uninstall console 명령을 사용하여 번들 배포를 취소할 수도 있습니다.

6.3. 수동 배포

6.3.1. 개요

Fuse 콘솔에서 명령을 실행하여 수동으로 번들 배포 및 배포를 취소할 수 있습니다.

6.3.2. 번들 설치

bundle:install 명령을 사용하여 OSGi 컨테이너에 하나 이상의 번들을 설치합니다. 이 명령에는 다음 구문이 있습니다.

```
bundle:install [-s] [--start] [--help] UrlList
```

여기서 *UrlList* 는 배포할 각 번들의 위치를 지정하는 공백으로 구분된 URL 목록입니다. 다음 명령 인수가 지원됩니다.

-s

설치 후 번들을 시작합니다.

--start

마찬가지로 **-s**.

--help

명령 구문을 표시하고 설명합니다.

예를 들어, 번들 설치 및 시작, *ProjectDir*/**target/foo-1.0-SNAPSHOT.jar** 에서는 Karaf 콘솔 프롬프트에서 다음 명령을 입력합니다.

```
bundle:install -s file:ProjectDir/target/foo-1.0-SNAPSHOT.jar
```

참고

Windows 플랫폼에서는 이 명령에서 **파일** URL에 올바른 구문을 사용해야 합니다. 자세한 내용은 15.1절. “파일 URL 핸들러” 을 참조하십시오.

6.3.3. 번들 설치 제거

번들을 설치 제거하려면 먼저 **bundle:list** 명령을 사용하여 번들 ID를 가져와야 합니다. 그러면 bundle ID를 인수로 사용하는 **bundle:uninstall** 명령을 사용하여 번들을 제거할 수 있습니다.

예를 들어 **A Camel OSGi Service Unit** 이라는 번들을 이미 설치한 경우 콘솔 프롬프트에서 **bundle:list** 를 입력하면 다음과 같은 출력이 생성될 수 있습니다.

```
...
[ 181] [Resolved   ] [            ] [       ] [   60] A Camel OSGi Service Unit (1.0.0.SNAPSHOT)
```

다음 콘솔 명령을 입력하여 ID **181** 을 사용하여 번들을 제거할 수 있습니다.

```
bundle:uninstall 181
```

6.3.4. 번들을 찾기 위한 URL 체계

bundle:install 명령에 대한 위치 URL을 지정할 때 다음 스키마 유형을 포함하는 Fuse에서 지원하는 URL 체계를 사용할 수 있습니다.

- 15.1절. “파일 URL 핸들러”.
- 15.2절. “HTTP URL 처리기”.
- 15.3절. “mvn URL Handler”.

6.4. BUNDLE:WATCH를 사용하여 자동으로 번들 재배포

개발 환경에서는 개발자가 지속적으로 변경 및 다시 빌드되는 경우 일반적으로 번들을 여러 번 다시 설치해야 합니다. **bundle:watch** 명령을 사용하면 로컬 Maven 리포지토리가 변경될 때 Karaf에 로컬 Maven 리포지토리를 모니터링하고 특정 번들을 자동으로 다시 설치하도록 지시할 수 있습니다.

예를 들어 번들 ID가 포함된 특정 번들이 있는 경우 **751**- 사용자는 명령을 입력하여 자동 재배포를 활성화할 수 있습니다.

```
bundle:watch 751
```

이제 Maven 아티팩트를 다시 빌드하고 로컬 Maven 리포지토리에 설치할 때마다(예: Maven 프로젝트에서 **mvn install** 을 실행하는 경우) Karaf 컨테이너는 변경된 Maven 아티팩트를 자동으로 다시 설치합니다. 자세한 내용은 Apache Karaf Console Reference 를 참조하십시오.

중요

bundle:watch 명령을 사용하는 것은 개발 환경 전용입니다. 프로덕션 환경에서 사용하지 **않는** 것이 좋습니다.

7장. 라이프사이클 관리

7.1. 번들 라이프사이클 상태

OSGi 환경의 애플리케이션에는 번들의 라이프사이클이 적용됩니다. 번들의 라이프사이클 상태는 다음과 같습니다.

1. **Installed** - 모든 번들이 설치된 상태로 시작됩니다. 설치된 상태의 번들은 모든 종속 항목을 해결하기를 기다리고 있으며, 해결되면 번들이 해결된 상태로 이동합니다.

2. **해결됨** - 다음 조건이 충족되면 번들이 해결된 상태로 이동합니다.

   - 런타임 환경은 번들에서 지정한 환경을 충족하거나 초과합니다.
   - 번들에서 가져온 모든 패키지는 확인된 상태에 있거나 현재 번들과 동시에 확인된 상태로 이동할 수 있는 번들에 의해 노출됩니다.
   - 모든 필수 번들은 해결된 상태에 있거나 현재 번들과 동시에 확인될 수 있습니다.

   중요

   애플리케이션을 시작하기 전에 애플리케이션의 모든 번들이 해결된 상태에 있어야 합니다.

   위의 조건 중 하나라도 충족되지 않으면 번들이 다시 설치된 상태로 이동합니다. 예를 들어 가져온 패키지가 포함된 번들을 컨테이너에서 제거할 때 이러한 상황이 발생할 수 있습니다.

3. starting - **시작** 상태는 해결된 상태와 활성 상태 간의 Transmissionory 상태입니다. 번들이 시작되면 컨테이너는 번들에 대한 리소스를 생성해야 합니다. 컨테이너가 제공되는 경우 번들의 bundle activator의 **start()** 메서드도 호출합니다.

4. **active** - 활성 상태의 번들을 수행할 수 있습니다. 활성 상태의 번들은 번들의 콘텐츠에 따라 다릅니다. 예를 들어 Cryostat-WS 서비스 공급자를 포함하는 번들은 서비스를 통해 요청을 수락할 수 있음을 나타냅니다.

5. **stop** - 중지 상태는 활성 상태와 확인된 상태 간의 Transmissionory 상태입니다. 번들이 중지되면 컨테이너가 번들의 리소스를 정리해야 합니다. 컨테이너가 제공되는 경우 번들의 bundle activator의 **stop()** 메서드도 호출합니다.

6. uninstalled - 번들을 **제거하면** 해결된 상태에서 uninstalled 상태로 이동합니다. 이 상태의 번들은 해결된 상태 또는 다른 상태로 다시 전환할 수 없습니다. 명시적으로 다시 설치해야 합니다.

애플리케이션 개발자에게 가장 중요한 라이프사이클 상태는 시작 상태 및 중지 상태입니다. 애플리케이션에서 노출하는 끝점은 시작 상태 중에 게시됩니다. 게시된 끝점은 중지 상태 중에 중지됩니다.

7.2. 번들 설치 및 해결

bundle:install 명령을 사용하여(**-s** 플래그 제외) 번들을 설치할 때 커널은 지정된 번들을 설치하고 해결된 상태로 전환하려고 합니다. 어떤 이유로든 번들 확인이 실패하는 경우(예: 종속 항목 중 하나가 만족스럽지 않은 경우) 커널은 번들을 설치된 상태로 유지합니다.

나중에(예: 누락된 종속성을 설치한 후) **bundle:resolve** 명령을 호출하여 다음과 같이 번들을 확인된 상태로 이동할 수 있습니다.

```
bundle:resolve 181
```

여기서 인수**181** 은 확인하려는 번들의 ID입니다.

7.3. 번들 시작 및 중지

bundle:start 명령을 사용하여 설치된 또는 확인된 상태에서 하나 이상의 번들을 시작할 수 있습니다. 예를 들어 ID, 181, 185 및 186으로 번들을 시작하려면 다음 console 명령을 입력합니다.

```
bundle:start 181 185 186
```

bundle:stop 명령을 사용하여 하나 이상의 번들을 중지할 수 있습니다. 예를 들어 ID가 181, 185 및 186인 번들을 중지하려면 다음 console 명령을 입력합니다.

```
bundle:stop 181 185 186
```

bundle:restart 명령을 사용하여 하나 이상의 번들(즉, started 상태에서 확인된 상태로 전환)을 다시 시작한 다음 다시 시작할 수 있습니다. 예를 들어 ID가 181, 185 및 186인 번들을 다시 시작하려면 다음 console 명령을 입력합니다.

```
bundle:restart 181 185 186
```

7.4. 번들 시작 수준

*시작 수준*은 모든 번들과 연결되어 있습니다. 시작 수준은 번이 활성화/시작되는 순서를 제어하는 양의 정수 값입니다. 시작 수준이 낮은 번들은 시작 수준이 높은 번들보다 먼저 시작됩니다. 따라서 시작 수준 **1** 이 있는 번들이 먼저 시작되고 커널에 속하는 번들은 대부분의 다른 번들을 실행하는 데 필요한 사전 요구 사항을 제공하므로 시작 수준이 낮은 경향이 있습니다.

일반적으로 사용자 번들의 시작 수준은 60 이상입니다.

7.5. 번들의 시작 수준 지정

bundle:start-level 명령을 사용하여 특정 번들의 시작 수준을 설정합니다. 예를 들어 ID **181** 로 번들을 구성하려면 시작 수준이 **70** 인 경우 다음 console 명령을 입력합니다.

```
bundle:start-level 181 70
```

7.6. 시스템 시작 수준

OSGi 컨테이너 자체에는 시작 수준이 연결되어 있으며 이 *시스템 시작 수준*은 활성화할 수 있는 번들을 결정합니다. 시작 수준이 시스템 시작 수준보다 **작거나 같은** 번들만 활성화될 수 있습니다.

현재 시스템 시작 수준을 검색하려면 다음과 같이 콘솔에서 **system:start-level** 을 입력합니다.

```
karaf@root()> system:start-level
Level 100
```

시스템 시작 수준을 변경하려면 다음과 같이 새 시작 수준을 **system:start-level** 명령에 인수로 제공합니다.

```
system:start-level 200
```

8장. 종속성 문제 해결

8.1. 종속 항목 누락

OSGi 번들을 Red Hat Fuse 컨테이너에 배포할 때 발생할 수 있는 가장 일반적인 문제는 하나 이상의 종속 항목이 누락되었다는 것입니다. 이 문제는 일반적으로 번들 시작의 부작용으로 OSGi 컨테이너의 번들을 해결하려고 할 때 나타납니다. 번들이 해결되거나 시작되지 않으며 **ClassNotFound** 오류가 기록됩니다 (로그를 보려면 **log:display** console 명령을 사용하거나 **FUSE_HOME/data/log** 디렉터리에서 로그 파일을 확인하십시오).

종속성이 누락된 두 가지 기본 원인: 필수 기능 또는 번들이 컨테이너에 설치되지 않았거나 번들의 **Import-Package** 헤더가 불완전합니다.

8.2. 필수 기능 또는 번들이 설치되지 않았습니다.

번들 확인을 시도하기 전에 번들에 필요한 모든 기능과 번들이 **이미** OSGi 컨테이너에 설치되어 있어야 합니다. 특히 Apache Camel에는 각 구성 요소가 별도의 기능으로 설치된 모듈식 아키텍처가 있으므로 필요한 구성 요소 중 하나를 설치하는 것을 잊어버리기 쉽습니다.

참고

번들 패키징을 기능으로 사용하는 것이 좋습니다. 기능을 사용하면 번들을 모든 종속 항목과 함께 패키징하여 모두 동시에 설치되었는지 확인할 수 있습니다. 자세한 내용은 9장. *기능 배포*의 내용을 참조하십시오.

8.3. IMPORT-PACKAGE 헤더가 불완전합니다.

필요한 모든 기능과 번들이 이미 설치되어 있고 아직 **ClassNotFound** 오류가 있는 경우 번들의 **MANIFEST.MF** 파일에서 **Import-Package** 헤더가 불완전함을 의미합니다. **maven-bundle-plugin** (5.2절. "기존 Maven 프로젝트 수정"참조)은 번들의 **Import-Package** 헤더를 생성할 때 큰 도움이 되지만 다음 사항에 유의하십시오.

- Maven 번들 플러그인 구성의 **Import-Package** 요소에 와일드카드 ***** 를 포함해야 합니다. 와일드카드는 플러그인을 전달하여 Java 소스 코드를 스캔하고 패키지 종속 항목 목록을 자동으로 생성합니다.
- Maven 번들 플러그인은 동적 종속성을 파악할 수 **없습니다**. 예를 들어 Java 코드가 클래스 로더를 명시적으로 호출하여 클래스 로더를 동적으로 로드하는 경우 번들 플러그인은 이를 고려하지 않고 필요한 Java 패키지가 생성된 **Import-Package** 헤더에 나열되지 않습니다.
- 블루프린트 XML 파일(예: **OSGI-INF/blueprint** 디렉토리)을 정의하는 경우 블루프린트 XML 파일에서 발생하는 모든 종속성은 **런타임 시 자동으로 해결됩니다**.

8.4. 누락된 종속성을 추적하는 방법

누락된 종속 항목을 추적하려면 다음 단계를 수행합니다.

1. **bundle:diag** console 명령을 사용합니다. 이렇게 하면 번들이 비활성화된 이유에 대한 정보가 제공됩니다. 사용 정보는 Apache Karaf 콘솔 참조 에서 참조하십시오.
2. 빠른 검사를 수행하여 모든 필수 번들 및 기능이 실제로 OSGi 컨테이너에 설치되어 있는지 확인합니다. **bundle:list** 를 사용하여 설치된 번들을 확인하고 **features:list** 를 사용하여 설치된 기능을 확인할 수 있습니다.

3. **bundle:install** console 명령을 사용하여 번들을 설치(시작하지 않음)합니다. 예를 들면 다음과 같습니다.

```
karaf@root()> bundle:install MyBundleURL
```

4. **bundle:dynamic-import** console 명령을 사용하여 방금 설치한 번들에서 동적 가져오기를 활성화합니다. 예를 들어 번들의 번들 ID가 218인 경우 다음 명령을 입력하여 이 번들에서 동적 가져오기를 활성화합니다.

```
karaf@root()> bundle:dynamic-import 218
```

이 설정을 사용하면 OSGi에서 컨테이너에 이미 설치된 번들을 사용하여 종속성을 확인할 수 있으므로 일반적인 종속성 확인 메커니즘(**Import-Package** 헤더 기반)을 효과적으로 우회할 수 있습니다. **이는 버전 검사를 우회하기 때문에 정상적인 배포에 대해 기억되지 않습니다. 잘못된 버전의 패키지를 쉽게 선택하여 애플리케이션이 오작동할 수 있습니다.**

5. **이제 번들을 확인할 수 있습니다. 예를 들어 번들 ID가 218인 경우 followng console 명령을 입력합니다.**

```
karaf@root()> bundle:resolve 218
```

6. **이제 번들이 해결되었다고 가정하면 package :imports명령을 사용하여 번들에 연결된 모든 패키지 목록을 가져올 수 있습니다. 예를 들어 번들 ID가 218인 경우 다음 console 명령을 입력합니다.**

```
karaf@root()> package:imports -b 218
```

콘솔 창에 종속 패키지 목록이 표시됩니다.

```
Package                          | Version       | Optional  | ID  | Bundle Name
-------------------------------------------------------------------------------------
org.apache.jasper.servlet        | [2.2.0,3.0.0) | resolved  | 217 | org.ops4j.pax.web.pax-web-runtime
org.jasypt.encryption.pbe        |               | resolved  | 217 | org.ops4j.pax.web.pax-web-runtime
org.ops4j.pax.web.jsp            | [7.0.0,)      | resolved  | 217 | org.ops4j.pax.web.pax-web-runtime
org.ops4j.pax.web.service.spi.model | [7.0.0,)   |           | 217 | org.ops4j.pax.web.pax-web-runtime
org.ops4j.pax.web.service.spi.util | [7.0.0,)    |           | 217 | org.ops4j.pax.web.pax-web-runtime
...
```

7. **번들 JAR 파일의 압축을 풀고 META-INF/MANIFEST.MF 파일의 Import-Package 헤더 아래에 나열된 패키지를 확인합니다. 이 목록을 이전 단계에서 발견된 패키지 목록과 비교합니다. 이제 매니페스트의 Import-Package 헤더에서 누락된 패키지 목록을 컴파일하고 이러한 패키지 이름을 프로젝트의 POM 파일에 있는 Maven 번들 플러그인 구성의 Import-Package 요소에 추가합니다.**

8. **동적 가져오기 옵션을 취소하려면 OSGi 컨테이너에서 이전 번들을 제거해야 합니다. 예를 들어 번들 ID가 218인 경우 다음 명령을 입력합니다.**

```
karaf@root()> bundle:uninstall 218
```

9. 이제 가져온 패키지 목록을 사용하여 번들을 다시 빌드하고 OSGi 컨테이너에서 테스트할 수 있습니다.

addurl :experimental: :toclevels: 4 :numbered:

9장. 기능 배포

초록

애플리케이션 및 기타 툴은 일반적으로 여러 OSGi 번들로 구성되므로 서로 종속되거나 관련 번들을 더 큰 배포 단위로 집계하는 것이 편리합니다. 따라서 Red Hat Fuse는 확장 가능한 배포 단위인 *기능*을 제공하므로 단일 단계에서 여러 번들(및 선택적으로 다른 기능에 종속 항목)을 배포할 수 있습니다.

9.1. 기능 생성

9.1.1. 개요

기본적으로 기능은 기능 저장소라는 특수한 종류의 XML 파일에 새 기능요소를 추가하여 생성됩니다. 기능을 생성하려면 다음 단계를 수행합니다.

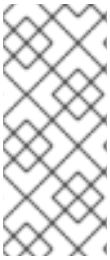
1. 9.2절. “사용자 정의 기능 리포지토리 생성”.
2. 9.3절. “사용자 정의 기능 리포지토리에 기능 추가”.
3. 9.4절. “기능 서비스에 로컬 리포지토리 URL을 추가합니다.”.
4. 9.5절. “기능에 종속 기능 추가”.
5. 9.6절. “기능에 OSGi 구성 추가”.

9.2. 사용자 정의 기능 리포지토리 생성

사용자 지정 기능 리포지토리를 아직 정의하지 않은 경우 다음과 같이 생성할 수 있습니다. 파일 시스템에서 기능 리포지토리에 대한 편리한 위치(예: C:\Projects\features.xml-)를 선택하고 즐겨 찾는 텍스트 편집기를 사용하여 다음 행을 추가합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="CustomRepository">
</features>
```

여기서 **name** 속성을 설정하여 리포지토리의 이름을 지정해야 합니다.



참고

Maven 리포지토리 또는 **OBR**과 달리 기능 리포지토리는 번들의 스토리지 위치를 제공하지 않습니다. 기능 리포지토리는 번들에 대한 참조 집계를 저장합니다. 번들 자체는 다른 위치에 저장됩니다(예: 파일 시스템 또는 **Maven** 리포지토리에).

9.3. 사용자 정의 기능 리포지토리에 기능 추가

사용자 지정 기능 리포지토리에 기능을 추가하려면 새 기능 요소를 루트 기능 요소의 자식으로 삽입합니다. 번들 하위 요소를 삽입하여 기능에 속하는 여러 번들을 나열할 수 있습니다. 예를 들어 단일 번들이 포함된 **example-camel-bundle** 이라는 기능을 추가하려면 **C:\Projects\camel-bundle\target\camel-bundle-1.0-SNAPSHOT.jar** 에서는 다음과 같이 **feature** 요소를 추가합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="MyFeaturesRepo">
  <feature name="example-camel-bundle">
    <bundle>file:C:/Projects/camel-bundle/target/camel-bundle-1.0-SNAPSHOT.jar</bundle>
  </feature>
</features>
```

번들 요소의 내용은 유효한 **URL**일 수 있으며 번들 위치를 지정합니다(**15장. URL 처리기** 참조). 선택적으로 **feature** 요소에서 **version** 속성을 지정하여 **0**이 아닌 버전을 기능에 할당할 수 있습니다(예: **features:install** 명령의 선택적 인수로 버전을 지정할 수 있습니다).

기능 서비스가 새 기능 항목을 성공적으로 구문 분석하는지 확인하려면 다음 콘솔 명령 쌍을 입력합니다.

```
JBossFuse:karaf@root> features:refreshurl
JBossFuse:karaf@root> features:list
...
[uninstalled] [0.0.0             ] example-camel-bundle             MyFeaturesRepo
...
```

features:list 명령은 일반적으로 다소 긴 기능 목록을 생성하지만, 목록을 다시 스크롤하여 새 기능(이 경우 **example-camel-bundle)**에 대한 항목을 찾을 수 있어야 합니다. **features:refreshurl** 명령은 커널이 모든 기능 리포지토리를 다시 읽도록 합니다. 이 명령을 실행하지 않은 경우 커널은 리포지토리에 대한 최근 변경 사항을 인식하지 못합니다(특히 새 기능이 목록에 표시되지 않음).

긴 기능 목록을 스크롤하지 않으려면 다음과 같이 **example-camel-bundle** 기능에 대해 **grep** 을 실행할 수 있습니다.

```
JBossFuse:karaf@root> features:list | grep example-camel-bundle
[uninstalled] [0.0.0                ] example-camel-bundle                MyFeaturesRepo
```

grep 명령(표준 **UNIX** 패턴 일치 유틸리티)이 쉘에 빌드되므로 이 명령은 **Windows** 플랫폼에서도 작동합니다.

9.4. 기능 서비스에 로컬 리포지토리 URL을 추가합니다.

Apache Karaf에서 새로운 기능 리포지토리를 사용하려면 **features:addurl console** 명령을 사용하여 기능 리포지토리를 추가해야 합니다. 예를 들어, 커널에 사용할 수 있는 **C:\Projects\features.xml** 리포지토리의 내용을 만들려면 다음 **console** 명령을 입력합니다.

```
features:addurl file:C:/Projects/features.xml
```

지원되는 **URL** 형식을 사용하여 **features:addurl** 에 대한 인수를 지정할 수 있는 위치(*15장. URL 처리기*참조).

다음과 같이 **features:listUrl console** 명령을 입력하여 리포지토리의 **URL**이 올바르게 등록되어 있는지 확인할 수 있습니다.

```
JBossFuse:karaf@root> features:listUrl
file:C:/Projects/features.xml
mvn:org.apache.ode/ode-jbi-karaf/1.3.3-fuse-01-00/xml/features
mvn:org.apache.felix.karaf/apache-felix-karaf/1.2.0-fuse-01-00/xml/features
```

9.5. 기능에 종속 기능 추가

기능이 다른 기능에 종속되는 경우 기능 요소를 원래 기능 요소의 하위 항목으로 추가하여 이러한 종속성을 지정할 수 있습니다. 각 자식 기능 요소에는 현재 기능이 종속된 기능의 이름이 포함됩니다. 종속 기능을 사용하여 기능을 배포할 때 종속성 메커니즘은 종속 기능이 컨테이너에 설치되어 있는지 여부를 확인합니다. 그렇지 않은 경우 종속성 메커니즘은 누락된 종속 항목(및 재귀 종속성)을 자동으로 설치합니다.

예를 들어 사용자 지정 **Apache Camel** 기능인 **example-camel-bundle** 의 경우 어떤 표준 **Apache Camel** 기능이 종속되어 있는지 명시적으로 지정할 수 있습니다. 이제 **OSGi** 컨테이너에 필요한 기능이 사전 배포되지 않은 경우에도 애플리케이션을 성공적으로 배포하고 실행할 수 있는 이점이 있습니다. 예를 들어 다음과 같이 **Apache Camel** 종속 항목을 사용하여 **example-camel-bundle** 기능을 정의할 수 있습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="MyFeaturesRepo">
```

```
  <feature name="example-camel-bundle">
    <bundle>file:C:/Projects/camel-bundle/target/camel-bundle-1.0-SNAPSHOT.jar</bundle>
    <feature version="7.11.1.fuse-7_11_1-00013-redhat-00003">camel-core</feature>
    <feature version="7.11.1.fuse-7_11_1-00013-redhat-00003">camel-spring-osgi</feature>
  </feature>
</features>
```

version 속성을 지정하는 것은 선택 사항입니다. 있는 경우 지정된 버전의 기능을 선택할 수 있습니다.

9.6. 기능에 OSGI 구성 추가

애플리케이션에서 **OSGi** 구성 관리 서비스를 사용하는 경우 기능 정의의 **config** 하위 요소를 사용하여 이 서비스에 대한 구성 설정을 지정할 수 있습니다. 예를 들어 **prefix** 속성 값이 **MyTransform** 임을 지정하려면 기능 구성에 다음 **config** 하위 요소를 추가합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="MyFeaturesRepo">
  <feature name="example-camel-bundle">
    <config name="org.fusesource.fuseesb.example">
      prefix=MyTransform
    </config>
  </feature>
</features>
```

여기서 **config** 요소의 **name** 속성이 속성 설정의 영구 **ID** 를 지정합니다. 여기서 영구 **ID**는 속성 이름의 이름 범위 역할을 합니다. **config** 요소의 내용은 **Java** 속성 파일과 동일한 방식으로 구문 분석됩니다.

구성 요소의 설정은 다음과 같이 **InstallDir/etc** 디렉터리에 있는 **Java** 속성 파일의 설정으로 재정의할 수 있습니다.

```
InstallDir/etc/org.fusesource.fuseesb.example.cfg
```

실제로 이전 구성 속성을 사용하는 방법의 예로 **OSGi** 구성 속성에 액세스하는 다음 블루프린트 **XML** 파일을 고려하십시오.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:cm="http://aries.apache.org/blueprint/xmlns/blueprint-cm/v1.1.0">

    <!-- osgi blueprint property placeholder -->
    <cm:property-placeholder id="placeholder"
                             persistent-id="org.fusesource.fuseesb.example">
        <cm:default-properties>
```

```
        <cm:property name="prefix" value="DefaultValue"/>
      </cm:default-properties>
    </cm:property-placeholder>

    <bean id="myTransform" class="org.fusesource.fuseesb.example.MyTransform">
      <property name="prefix" value="${prefix}"/>
    </bean>

  </blueprint>
```

*이 블루프린트 **XML** 파일이 **example-camel-bundle** 번들에 배포되면 속성 참조 **${prefix}** 가 **feature** 리포지토리의 **config** 요소에 의해 지정되는 **MyTransform** 값으로 교체됩니다.*

9.7. OSGI 구성 자동 배포

*기능에 **configfile** 요소를 추가하면 기능이 설치된 동시에 **OSGi** 구성 파일이 **InstallDir/etc** 디렉터리에 추가되도록 할 수 있습니다. 즉, 기능 및 관련 구성을 동시에 편리하게 설치할 수 있습니다.*

*예를 들어 **org.fusesource.fuseesb.example.cfg** 구성 파일이 **mvn:org.fusesource.fuseesb.example/configadmin/1.0/cfg** 의 **Maven** 리포지토리에 보관되면 다음 요소를 기능에 추가하여 구성 파일을 배포할 수 있습니다.*

```
<configfile finalname="etc/org.fusesource.fuseesb.example.cfg">
  mvn:org.fusesource.fuseesb.example/configadmin/1.0/cfg
</configfile>
```

10장. 기능 배포

10.1. 개요

다음 방법 중 하나로 기능을 배포할 수 있습니다.

- **features:install** 을 사용하여 콘솔에 설치합니다.
- 핫 배포를 사용합니다.
- 부팅 구성을 수정합니다(**first boot only!**).

10.2. 콘솔에 설치

기능 리포지토리에 기능을 추가하고 기능 리포지토리를 등록하여 기능을 생성한 후에는 **features:install console** 명령을 사용하여 기능을 비교적 쉽게 배포할 수 있습니다. 예를 들어 **example-camel-bundle** 기능을 배포하려면 다음 콘솔 명령 쌍을 입력합니다.

```
JBossFuse:karaf@root> features:refreshurl
JBossFuse:karaf@root> features:install example-camel-bundle
```

features:install 을 호출하기 전에 **features:refreshurl** 명령을 호출하는 것이 좋습니다. 커널이 아직 선택되지 않은 기능 리포지토리의 기능에 최근 변경이 이루어진 경우입니다. **features:install** 명령은 기능 이름을 인수로 사용합니다(선택 사항, 기능 버전을 두 번째 인수로 사용).

참고

기능은 플랫 네임스페이스를 사용합니다. 따라서 기능 이름을 지정할 때는 기존 기능과 이름이 충돌하지 않도록 주의해야 합니다.

10.3. 콘솔에서 설치 제거

기능을 설치 제거하려면 다음과 같이 **features:uninstall** 명령을 호출합니다.

```
JBossFuse:karaf@root> features:uninstall example-camel-bundle
```

참고

설치 제거 후에도 **features:list** 를 호출할 때 기능이 계속 표시되지만 이제 상태는 **[설치되지 않음]** 으로 플래그가 지정됩니다.

10.4. 핫 디플로이먼트

기능 리포지토리 파일을 **InstallDir/deploy** 디렉터리에 복사하여 기능 리포지토리의 모든 기능을 핫 디플로이먼트할 수 있습니다.

한 번에 전체 기능 리포지토리를 핫 디플로이먼트할 가능성은 낮기 때문에 배포하려는 기능 리포지토리 또는 기능 설명자 를 줄이는 것이 더 편리합니다. 기능 설명자에는 기능 리포지토리와 정확히 동일한 구문이 있지만 다른 스타일로 작성됩니다. 차이점은 기능 설명자가 기능 리포지토리의 기존 기능에 대한 참조로만 구성된다는 것입니다.

예를 들어 다음과 같이 **example-camel-bundle** 기능을 로드할 기능 설명자를 정의할 수 있습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="CustomDescriptor">
  <repository>RepositoryURL</repository>
  <feature name="hot-example-camel-bundle">
    <feature>example-camel-bundle</feature>
  </feature>
</features>
```

리포지토리 요소는 사용자 정의 기능 리포지토리, **RepositoryURL** (*15장. URL 처리기* 에 설명된 **URL** 형식 중 하나를 사용할 수 있음)을 지정합니다. **hot-example-camel-bundle** 기능은 기존 기능 **example-camel-bundle** 에 대한 참조일 뿐입니다.

10.5. 기능 파일 배포 취소

hot deploy 디렉토리에서 기능 파일 배포를 취소하려면 **Apache Karaf** 컨테이너가 실행되는 동안 **InstallDir/deploy** 디렉토리에서 기능 파일을 삭제하면 됩니다.

중요

컨테이너가 종료되는 동안 핫 배포 취소 메커니즘이 작동하지 않습니다. **Karaf** 컨테이너를 종료하고 **deploy/** 에서 기능 파일을 삭제한 다음 **Karaf** 컨테이너를 다시 시작하면 컨테이너를 다시 시작한 후 기능이 배포되지 않습니다 (하지만 **features:uninstall console** 명령을 사용하여 수동으로 기능 배포 취소할 수 있음).

10.6. 부팅 구성에 기능 추가

여러 호스트에 배포하기 위해 **Apache Karaf**의 사본을 프로비저닝하려면 **Apache Karaf**가 처음으로 부팅될 때 설치된 기능 컬렉션을 결정하는 부팅 구성에 기능을 추가하는 데 관심이 있을 수 있습니다.

설정 파일인 **/etc/org.apache.karaf.features.cfg** 에서는 다음 설정을 포함합니다.

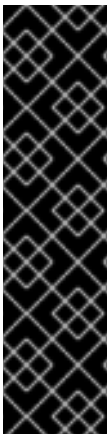
```
...
#
# Comma separated list of features repositories to register by default
#
featuresRepositories = \
    mvn:org.apache-extras.camel-extra.karaf/camel-extra/2.21.0.fuse-000032-redhat-2/xml/features, \
    mvn:org.apache.karaf.features/spring-legacy/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.apache.activemq/artemis-features/2.4.0.amq-710008-redhat-1/xml/features, \
    mvn:org.jboss.fuse.modules.patch/patch-features/7.0.0.fuse-000163-redhat-2/xml/features, \
    mvn:org.apache.karaf.features/framework/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.jboss.fuse/fuse-karaf-framework/7.0.0.fuse-000163-redhat-2/xml/features, \
    mvn:org.apache.karaf.features/standard/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.apache.karaf.features/enterprise/4.2.0.fuse-000191-redhat-1/xml/features, \
    mvn:org.apache.camel.karaf/apache-camel/2.21.0.fuse-000055-redhat-2/xml/features, \
    mvn:org.apache.cxf.karaf/apache-cxf/3.1.11.fuse-000199-redhat-1/xml/features, \
    mvn:io.hawt/hawtio-karaf/2.0.0.fuse-000145-redhat-1/xml/features

#
# Comma separated list of features to install at startup
#
featuresBoot = \
    instance/4.2.0.fuse-000191-redhat-1, \
    cxf-commands/3.1.11.fuse-000199-redhat-1, \
    log/4.2.0.fuse-000191-redhat-1, \
    pax-cdi-weld/1.0.0, \
    camel-jms/2.21.0.fuse-000055-redhat-2, \
    ssh/4.2.0.fuse-000191-redhat-1, \
    camel-cxf/2.21.0.fuse-000055-redhat-2, \
    aries-blueprint/4.2.0.fuse-000191-redhat-1, \
    cxf/3.1.11.fuse-000199-redhat-1, \
    cxf-http-undertow/3.1.11.fuse-000199-redhat-1, \
    pax-jdbc-pool-narayana/1.2.0, \
    patch/7.0.0.fuse-000163-redhat-2, \
    cxf-rs-description-swagger2/3.1.11.fuse-000199-redhat-1, \
    feature/4.2.0.fuse-000191-redhat-1, \
    camel/2.21.0.fuse-000055-redhat-2, \
    jaas/4.2.0.fuse-000191-redhat-1, \
    camel-jaxb/2.21.0.fuse-000055-redhat-2, \
    camel-paxlogging/2.21.0.fuse-000055-redhat-2, \
    deployer/4.2.0.fuse-000191-redhat-1, \
    diagnostic/4.2.0.fuse-000191-redhat-1, \
    patch-management/7.0.0.fuse-000163-redhat-2, \
    bundle/4.2.0.fuse-000191-redhat-1, \
    kar/4.2.0.fuse-000191-redhat-1, \
    camel-csv/2.21.0.fuse-000055-redhat-2, \
```

```
package/4.2.0.fuse-000191-redhat-1, \
scr/4.2.0.fuse-000191-redhat-1, \
maven/4.2.0.fuse-000191-redhat-1, \
war/4.2.0.fuse-000191-redhat-1, \
camel-mail/2.21.0.fuse-000055-redhat-2, \
fuse-credential-store/7.0.0.fuse-000163-redhat-2, \
framework/4.2.0.fuse-000191-redhat-1, \
system/4.2.0.fuse-000191-redhat-1, \
pax-http-undertow/6.1.2, \
camel-jdbc/2.21.0.fuse-000055-redhat-2, \
shell/4.2.0.fuse-000191-redhat-1, \
management/4.2.0.fuse-000191-redhat-1, \
service/4.2.0.fuse-000191-redhat-1, \
camel-undertow/2.21.0.fuse-000055-redhat-2, \
camel-blueprint/2.21.0.fuse-000055-redhat-2, \
camel-spring/2.21.0.fuse-000055-redhat-2, \
hawtio/2.0.0.fuse-000145-redhat-1, \
camel-ftp/2.21.0.fuse-000055-redhat-2, \
wrap/2.5.4, \
config/4.2.0.fuse-000191-redhat-1, \
transaction-manager-narayana/5.7.2.Final
```

이 구성 파일에는 다음 두 가지 속성이 있습니다.

- ***featuresRepositories-comma**로 구분된 시작 시 로드할 기능 리포지토리 목록입니다.*
- ***featuresBoot-comma**로 구분된 시작 시 설치할 기능 목록입니다.*

*구성을 수정하여 **Fuse**가 시작될 때 설치된 기능을 사용자 지정할 수 있습니다. 사전 설치된 기능으로 **Fuse**를 배포하려는 경우 이 구성 파일을 수정할 수도 있습니다.*



중요

*기능을 추가하는 이 방법은 특정 **Apache Karaf** 인스턴스가 처음 부팅되는 경우에만 효과가 있습니다. **featuresRepositories** 설정 및 **featuresBoot** 설정이 이후에 수행된 모든 변경 사항은 컨테이너를 다시 시작하더라도 무시됩니다.*

*그러나 **InstallDir/data/cache** 의 전체 콘텐츠를 삭제하여 컨테이너가 초기 상태로 되돌리도록 할 수 있습니다(컨테이너의 모든 사용자 지정 설정이 손실됨).*

11장. PLAIN JAR 배포

초록

Apache Karaf에 애플리케이션을 배포하는 다른 방법은 일반 JAR 파일을 사용하는 것입니다. 일반적으로 배포 메타데이터가 포함되지 않은 라이브러리입니다. 일반 JAR은 WAR 또는 OSGi 번들도 아닙니다.

일반 JAR이 번들의 종속성으로 발생하는 경우 번들 헤더를 JAR에 추가해야 합니다. JAR이 공용 API를 노출하는 경우 일반적으로 가장 좋은 솔루션은 기존 JAR을 번들로 변환하여 다른 번들과 JAR을 공유할 수 있도록 하는 것입니다. 이 장의 지침을 사용하여 오픈 소스 Bnd 툴을 사용하여 변환 프로세스를 자동으로 수행합니다.

Bnd 툴에 대한 자세한 내용은 Bnd 툴 웹 사이트를 참조하십시오.

11.1. 랩 스키마를 사용하여 JAR 변환

11.1.1. 개요

기존 URL 형식과 함께 사용할 수 있는 wrap: protocol을 사용하여 JAR을 번들로 변환하는 옵션이 있습니다. wrap: 프로토콜은 Bnd 유틸리티를 기반으로 합니다.

11.1.2. 구문

wrap: protocol에는 다음과 같은 기본 구문이 있습니다.

```
wrap:LocationURL
```

wrap: 프로토콜은 JAR을 찾는 모든 URL의 접두사를 지정할 수 있습니다. URL 위치 URL의 위치 부분인 LocationURL 은 일반 JAR을 가져오고 wrap: 프로토콜의 URL 처리기를 가져온 다음 JAR을 번들로 자동으로 변환합니다.

참고

또한 wrap: 프로토콜은 더 정교한 구문을 지원하므로 Bnd 속성 파일을 지정하거나 URL에 개별 Bnd 속성을 지정하여 변환을 사용자 지정할 수 있습니다. 그러나 일반적으로 wrap: 프로토콜은 기본 설정에서만 사용됩니다.

11.1.3. 기본 속성

wrap: 프로토콜은 **Bnd** 유틸리티를 기반으로 하므로 **Bnd**와 동일한 기본 속성을 사용하여 번들을 생성합니다.

11.1. 랩 및 설치

다음 예제에서는 단일 **console** 명령을 사용하여 원격 **Maven** 리포지토리에서 일반 **commons-logging JAR**을 다운로드하고 동적으로 **OSGi** 번들로 변환한 다음 이를 설치한 후 **OSGi** 컨테이너에서 시작하는 방법을 보여줍니다.

```
karaf@root> bundle:install -s wrap:mvn:commons-logging/commons-logging/1.1.1
```

1. reference

랩: 프로토콜은 다양한 오픈 소스 **OSGi** 유틸리티에 대한 프로젝트인 **Pax** 프로젝트에서 제공합니다. **wrap: protocol**에 대한 전체 문서는 **Wrap Protocol** 참조 페이지를 참조하십시오.

12장. OSGI SERVICES

초록

OSGi 코어 프레임워크는 OSGi 서비스 레지스트리에서 Java 오브젝트를 서비스로 등록하여 번들이 상호 작용할 수 있는 간단한 메커니즘을 제공하는 OSGi 서비스 계층 을 정의합니다. OSGi 서비스 모델의 강점 중 하나는 Java 오브젝트를 서비스로 제공할 수 있다는 것입니다. 서비스 클래스에 적용해야 하는 특정 제약 조건, 상속 규칙 또는 주석이 없습니다. 이 장에서는 OSGi 블루프린트 컨테이너 를 사용하여 OSGi 서비스를 배포하는 방법을 설명합니다.

12.1. 블루프린트 컨테이너

초록

블루프린트 컨테이너는 OSGi 컨테이너와의 상호 작용을 간소화하는 종속성 주입 프레임워크입니다. 블루프린트 컨테이너는 OSGi 서비스 registry-예: OSGi 서비스를 가져오고 내보낼 표준 XML 요소를 제공하는 구성 기반 접근 방식을 지원합니다.

12.1.1. 블루프린트 구성

JAR 파일의 블루프린트 파일의 위치

번들 JAR 파일의 루트와 관련하여 블루프린트 구성 파일의 표준 위치는 다음과 같습니다.

```
OSGI-INF/blueprint
```

접미사 .xml .xml 이 있는 모든 파일은 블루프린트 구성 파일로 해석됩니다. 즉, 패턴과 일치하는 모든 파일, OSGI-INF/blueprint/*.xml.xml .xml입니다.

Maven 프로젝트의 블루프린트 파일 위치

Maven 프로젝트 ProjectDir 의 컨텍스트에서 블루프린트 구성 파일의 표준 위치는 다음과 같습니다.

```
ProjectDir/src/main/resources/OSGI-INF/blueprint
```

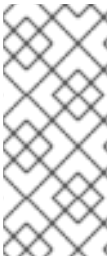
블루프린트 네임스페이스 및 루트 요소

블루프린트 구성 요소는 다음 XML 네임스페이스와 연결됩니다.

```
http://www.osgi.org/xmlns/blueprint/v1.0.0
```

블루프린트 구성의 루트 요소는 블루프린트 이므로 블루프린트 **XML** 구성 파일에는 일반적으로 다음과 같은 개요 형식이 있습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  ...
</blueprint>
```



참고

블루프린트 루트 요소에서는 스키마 위치가 블루프린트 프레임워크에 이미 알려져 있기 때문에 **xsi:schemaLocation** 특성을 사용하여 블루프린트 스키마의 위치를 지정할 필요가 없습니다.

블루프린트 매니페스트 구성

블루프린트 구성의 일부 측면은 **JAR**의 매니페스트 파일인 **META-INF/MANIFEST.MF** 의 헤더에 의해 다음과 같이 제어됩니다.

- 사용자 정의 블루프린트 파일 위치.
- 필수 종속 항목.

사용자 정의 블루프린트 파일 위치

블루프린트 구성 파일을 비표준 위치(즉, **OSGI-INF/blueprint/*.xml**이외의 위치)에 배치해야 하는 경우 매니페스트 파일의 **Bundle-Blueprint** 헤더에 쉼표로 구분된 대체 위치 목록을 지정할 수 있습니다.

```
Bundle-Blueprint: lib/account.xml, security.bp, cnf/*.xml
```

필수 종속 항목

OSGi 서비스의 종속 항목은 기본적으로 필수입니다(참조 요소 또는 참조 목록 요소에서 가용성 속성을 선택 사항으로 설정하여 이 속성을 변경할 수 있음). 종속성을 필수로 선언하면 해당 종속성 및 종속성을 항상 사용할 수 없으면 번들이 제대로 작동할 수 없습니다.

일반적으로 블루프린트 컨테이너는 초기화하는 동안 유예 기간을 통과하며 이 기간 동안 모든 필수 종속성을 해결하려고 시도합니다. 이 시점에서 필수 종속 항목을 확인할 수 없는 경우(기본 제한 시간은 **5**

분), 컨테이너 초기화가 중단되고 번들이 시작되지 않습니다. **Bundle-SymbolicName** 매니페스트 헤더에 다음 설정을 추가하여 유예 기간을 구성할 수 있습니다.

blueprint.graceperiod

true (기본값)가 활성화되고 블루프린트 컨테이너는 초기화 중에 필수 종속성이 해결될 때까지 기다립니다. **false** 인 경우 유예 기간을 건너뛰고 컨테이너는 필수 종속성이 해결되었는지 여부를 확인하지 않습니다.

blueprint.timeout

유예 기간(밀리초)을 지정합니다. 기본값은 **300000(5분)**입니다.

예를 들어 유예 기간이 **10**초인 경우 매니페스트 파일에 다음 **Bundle-SymbolicName** 헤더를 정의할 수 있습니다.

```
Bundle-SymbolicName: org.fusesource.example.osgi-client;
 blueprint.graceperiod:=true;
 blueprint.timeout:= 10000
```

Bundle-SymbolicName 헤더의 값은 **Semi-colon** 구분 목록입니다. 여기서 첫 번째 항목은 실제 번들 심볼릭 이름, 두 번째 항목인 블루프린트**.graceperiod:=true** 에서는 유예 기간을 활성화하고 세 번째 항목인 블루프린트**.timeout:= 10000** 을 지정합니다.

12.1.2. 서비스 Cryostat 정의

개요

블루프린트 컨테이너를 사용하면 **8080** 요소를 사용하여 **Java** 클래스를 인스턴스화할 수 있습니다. 이러한 방식으로 모든 주요 애플리케이션 오브젝트를 생성할 수 있습니다. 특히 **8080** 요소를 사용하여 **OSGi** 서비스 인스턴스를 나타내는 **Java** 오브젝트를 생성할 수 있습니다.

블루프린트 **8080** 요소

블루프린트 빈 요소는 블루프린트 스키마 네임스페이스인 **http://www.osgi.org/xmlns/blueprint/v1.0.0** 에 정의되어 있습니다.

샘플 빈

다음 예제에서는 블루프린트의 **8080** 요소를 사용하여 몇 가지 다른 유형의 빈을 생성하는 방법을 보여줍니다.

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
```

```
  <bean id="label" class="java.lang.String">
    <argument value="LABEL_VALUE"/>
  </bean>

  <bean id="myList" class="java.util.ArrayList">
    <argument type="int" value="10"/>
  </bean>

  <bean id="account" class="org.fusesource.example.Account">
    <property name="accountName" value="john.doe"/>
    <property name="balance" value="10000"/>
  </bean>

</blueprint>
```

*마지막 **8080** 예제에서 참조하는 **Account** 클래스를 다음과 같이 정의할 수 있습니다.*

```
package org.fusesource.example;

public class Account
{
    private String accountName;
    private int balance;

    public Account () { }

    public void setAccountName(String name) {
        this.accountName = name;
    }

    public void setBalance(int bal) {
        this.balance = bal;
    }
    ...
}
```

참고 자료

블루프린트 빈 정의에 대한 자세한 내용은 다음 참조를 참조하십시오.

- *Spring 동적 모듈 참조 가이드 v2.0, 블루프린트 장.*
- *OSGi Compendium Services R4.2 사양의 섹션 121 블루프린트 컨테이너 사양.*

12.1.3. 속성을 사용하여 블루프린트 구성

개요

이 섹션에서는 **Camel** 컨텍스트 외부에 있는 파일에 보관된 속성을 사용하여 블루프린트를 구성하는 방법을 설명합니다.

블루프린트 빈 구성

블루프린트 빈은 외부 파일의 속성과 함께 대체할 수 있는 변수를 사용하여 구성할 수 있습니다. **ext** 네임스페이스를 선언하고 블루프린트 **xml**에 속성 자리 표시자 **pin**을 추가해야 합니다. **Property-Placeholder Cryostat**를 사용하여 속성 파일의 위치를 블루프린트로 선언합니다.

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:ext="http://aries.apache.org/blueprint/xmlns/blueprint-ext/v1.2.0">

  <ext:property-placeholder>
    <ext:location>file:etc/ldap.properties</ext:location>
  </ext:property-placeholder>
  ...
  <bean ...>
      <property name="myProperty" value="${myProperty}" />
  </bean>
</blueprint>
```

속성 소유자 구성 옵션의 사양은 http://aries.apache.org/schemas/blueprint-ext/blueprint-ext.xsd 에서 확인할 수 있습니다.

12.2. 서비스 내보내기

개요

이 섹션에서는 **Java** 오브젝트를 **OSGi** 서비스 레지스트리로 내보내는 방법을 설명하므로 **OSGi** 컨테이너의 다른 번들에 서비스로 액세스할 수 있습니다.

단일 인터페이스로 내보내기

단일 인터페이스 이름으로 **OSGi** 서비스 레지스트리로 서비스를 내보내려면 **ref** 특성을 사용하여 관련 서비스 빈을 참조하는 **service** 요소를 정의하고 **interface** 특성을 사용하여 게시된 인터페이스를 지정합니다.

예를 들어 예 12.1. “단일 인터페이스를 사용한 샘플 서비스 내보내기” 에 표시된 블루프린트 구성 코드를 사용하여 **org.fusesource.example.Account** 인터페이스 이름 아래에 **SavingsAccountImpl** 클래스의 인스턴스를 내보낼 수 있습니다.

예 12.1. 단일 인터페이스를 사용한 샘플 서비스 내보내기

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="savings" class="org.fusesource.example.SavingsAccountImpl"/>

  <service ref="savings" interface="org.fusesource.example.Account"/>

</blueprint>
```

*여기서 **ref** 속성은 해당 빈 인스턴스의 **ID**를 지정하고 **interface** 속성은 서비스가 **OSGi** 서비스 레지스트리에 등록된 공용 **Java** 인터페이스의 이름을 지정합니다. 이 예제에서 사용되는 클래스 및 인터페이스는 다음과 같습니다. 예 12.2. "샘플 계정 클래스 및 인터페이스"*

예 12.2. 샘플 계정 클래스 및 인터페이스

```
package org.fusesource.example

public interface Account { ... }

public interface SavingsAccount { ... }

public interface CheckingAccount { ... }

public class SavingsAccountImpl implements SavingsAccount
{
   ...
}

public class CheckingAccountImpl implements CheckingAccount
{
   ...
}
```

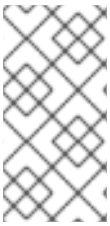
여러 인터페이스로 내보내기

*여러 인터페이스 이름으로 서비스를 **OSGi** 서비스 레지스트리로 내보내려면 **ref** 특성을 사용하여 관련 서비스 빈을 참조하는 **service** 요소를 정의하고 **interfaces** 하위 요소를 사용하여 게시된 인터페이스를 지정합니다.*

*예를 들어 다음 블루프린트 구성 코드를 사용하여 공용 **Java** 인터페이스 목록 **org.fusesource.example. Account** 및 **org.fusesource.example.SavingsAccount** 목록 아래에 **SavingsAccountImpl** 클래스의 인스턴스를 내보낼 수 있습니다.*

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <bean id="savings" class="org.fusesource.example.SavingsAccountImpl"/>
  <service ref="savings">
    <interfaces>
```

```
    <value>org.fusesource.example.Account</value>
    <value>org.fusesource.example.SavingsAccount</value>
   </interfaces>
  </service>
  ...
</blueprint>
```



참고

interface 특성 및 **interfaces** 요소는 동일한 **service** 요소에서 동시에 사용할 수 없습니다. 둘 중 하나를 사용해야 합니다.

자동 내보내기로 내보내기

구현된 모든 공용 **Java** 인터페이스에서 **OSGi** 서비스 레지스트리에 서비스를 내보내려면 **auto-export** 특성을 사용하여 쉽게 수행할 수 있습니다.

예를 들어 구현된 모든 공용 인터페이스 아래에 **SavingsAccountImpl** 클래스의 인스턴스를 내보내려면 다음 블루프린트 구성 코드를 사용합니다.

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <bean id="savings" class="org.fusesource.example.SavingsAccountImpl"/>
  <service ref="savings" auto-export="interfaces"/>
  ...
</blueprint>
```

여기서 **auto-export** 속성의 **interface** 값은 블루프린트가 **SavingsAccountImpl** 에서 구현한 모든 공용 인터페이스를 등록해야 함을 나타냅니다. **auto-export** 속성에 다음과 같은 유효한 값이 있을 수 있습니다.

비활성화됨

자동 내보내기를 비활성화합니다. 이는 기본값입니다.

interfaces

구현된 모든 공용 **Java** 인터페이스 아래에 서비스를 등록합니다.

class-hierarchy

Object 클래스를 제외한 자체 유형(클래스) 및 모든 슈퍼 유형(**super-classes**) 아래에 서비스를 등록합니다.

all-classes

class-hierarchy 옵션과 유사하지만 구현된 모든 공용 Java 인터페이스도 포함됩니다.

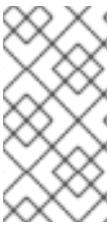
서비스 속성 설정

또한 OSGi 서비스 레지스트리를 사용하면 서비스 속성을 등록된 서비스와 연결할 수 있습니다. 그러면 서비스 클라이언트가 서비스 속성을 사용하여 서비스를 검색하거나 필터링할 수 있습니다. 서비스 속성을 내보낸 서비스와 연결하려면 하나 이상의 빈 요소(각 서비스 속성에 대한 하나의 빈:entry 요소)가 포함된 service-properties 하위 요소를 추가합니다.

예를 들어 bank.name 문자열 속성을 비용 절감 계정 서비스와 연결하려면 다음 블루프린트 구성을 사용할 수 있습니다.

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
        xmlns:beans="http://www.springframework.org/schema/beans"
        ...>
 ...
 <service ref="savings" auto-export="interfaces">
  <service-properties>
    <beans:entry key="bank.name" value="HighStreetBank"/>
  </service-properties>
 </service>
 ...
</blueprint>
```

여기서 bank.name 문자열 속성에는 HighStreetBank 값이 있습니다. 문자열 이외의 유형의 서비스 속성을 정의할 수 있습니다. 즉, 기본 유형, 배열 및 컬렉션도 지원됩니다. 이러한 유형을 정의하는 방법에 대한 자세한 내용은 Spring Reference Guide 에서 Advertised 속성 설정 제어를 참조하십시오.



참고

블루프린트 네임스페이스에 속하는 항목 요소입니다. Spring의 블루프린트 구현에서 beans:entry 요소의 사용은 비표준입니다.

기본 서비스 속성

다음과 같이 서비스 요소를 사용하여 서비스를 내보낼 때 자동으로 설정할 수 있는 두 가지 서비스 속성이 있습니다.

- *osGi.service.blueprint.compname-is always set to the id of the service's speaker element, unless the Cryostat is inlined (that is, the mutual is defined as a child element of the service element). 인라인 빈은 항상 익명입니다.*

- ***ranking** 속성이 **0**이 아닌 경우 **service.ranking-is** 자동으로 설정됩니다.*

순위 속성 지정

*번들이 서비스 레지스트리에서 서비스를 조회하고 일치하는 서비스를 두 개 이상 찾으면 순위를 사용하여 반환된 서비스를 확인할 수 있습니다. 규칙은 조회가 여러 서비스와 일치할 때마다 순위가 가장 높은 서비스가 반환됩니다. 서비스 순위는 음수가 아닌 정수일 수 있으며 **0** 은 기본값입니다. 서비스 요소에서 순위 속성을 설정하여 서비스 순위를 지정할 수 있습니다. 예를 들면 다음과 같습니다.*

```
<service ref="savings" interface="org.fusesource.example.Account" ranking="10"/>
```

등록 리스너 지정

*서비스 등록 및 등록 취소 이벤트를 추적하려면 등록 및 등록 취소 이벤트 알림을 수신하는 등록 리스너 콜백 빈을 정의할 수 있습니다. 등록 리스너를 정의하려면 **registration-listener** 하위 요소를 **service** 요소에 추가합니다.*

*예를 들어 다음 블루프린트 구성은 **registration-listener** 요소에서 참조하는 리스너 **8080** **listenerBean** 을 정의하여 계정 서비스가 등록되거나 등록되지 않을 때마다 리스너 **28**이 콜백을 수신하도록 합니다.*

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" ...>
  ...
  <bean id="listenerBean" class="org.fusesource.example.Listener"/>

  <service ref="savings" auto-export="interfaces">
    <registration-listener
        ref="listenerBean"
        registration-method="register"
        unregistration-method="unregister"/>
  </service>
  ...
</blueprint>
```

***registration-listener** 요소의 **ref** 속성이 리스너 **8080**의 **ID**를 참조하는 경우 **registration-method** 속성은 등록 콜백을 수신하는 리스너 메서드의 이름을 지정하고, **unregistration-method** 속성은 등록 콜백을 수신하는 리스너 메서드의 이름을 지정합니다.*

*다음 **Java** 코드는 등록 및 등록 취소 이벤트 알림을 수신하는 **Listener** 클래스의 샘플 정의를 보여줍니다.*

```
package org.fusesource.example;

public class Listener
```

```
{
    public void register(Account service, java.util.Map serviceProperties) {
        ...
    }

    public void unregister(Account service, java.util.Map serviceProperties) {
        ...
    }
}
```

메서드 이름, 등록 및 등록 취소 는 각각 **registration-method** 및 **unregistration-method** 속성으로 지정됩니다. 이러한 메서드의 서명은 다음 구문을 준수해야 합니다.

- 첫 번째 메서드 인수- 서비스 오브젝트 유형에서 할당할 수 있는 모든 유형 **T**입니다. 즉, 서비스 클래스의 모든 슈퍼타입 클래스 또는 서비스 클래스에 의해 구현된 모든 인터페이스입니다. 이 인수에는 서비스 빈에서 프로토타입 으로 범위를 선언하지 않는 한 서비스 인스턴스가 포함됩니다. 이 경우 이 인수가 **null** 입니다(범위가 프로토타입 인 경우 등록 시 사용 가능한 서비스 인스턴스가 없습니다).

- 두 번째 메서드 인수- **java.util.Map** 유형 또는 **java.util.Dictionary** 유형이어야 합니다. 이 맵에는 이 서비스 등록과 연결된 서비스 속성이 포함되어 있습니다.

12.3. 서비스 가져오기

개요

이 섹션에서는 **OSGi** 서비스 레지스트리로 내보낸 **OSGi** 서비스에 대한 참조를 가져오고 사용하는 방법을 설명합니다. 참조 요소 또는 **reference -list** 요소를 사용하여 **OSGi** 서비스를 가져올 수 있습니다. **reference** 요소는 상태 비저장 서비스에 액세스하는 데 적합하지만 **reference-list** 요소는 상태 저장 서비스에 액세스하는 데 적합합니다.

서비스 참조 관리

OSGi 서비스 참조를 가져오기 위한 다음 모델이 지원됩니다.

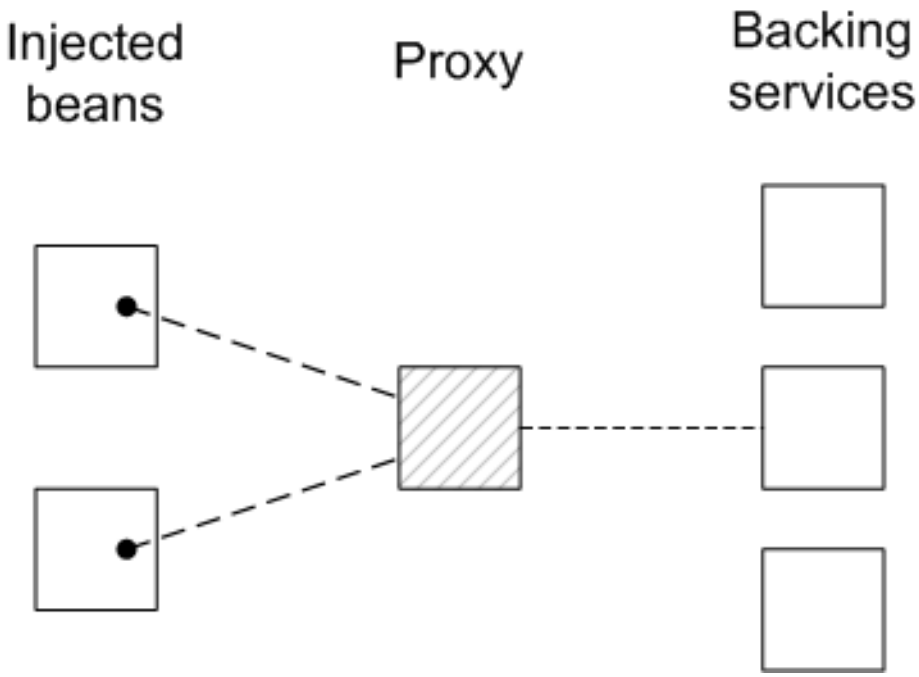
- *참조 관리자*.

- *참조 목록 관리자*.

참조 관리자

참조 관리자 인스턴스는 블루프린트 참조 요소에 의해 생성됩니다. 이 요소는 단일 서비스 참조를 반환하며 상태 비저장 서비스에 액세스하는 데 권장되는 접근 방식입니다. 그림 12.1. "상태 비저장 서비스 참조" 참조 관리자를 사용하여 상태 비저장 서비스에 액세스하기 위한 모델의 개요를 보여줍니다.

그림 12.1. 상태 비저장 서비스 참조



클라이언트 블루프린트 컨테이너의 빈은 **OSGi** 서비스 레지스트리의 서비스 오브젝트(지원 서비스) 에서 지원하는 프록시 오브젝트(제공오브젝트)와 함께 삽입됩니다. 이 모델은 다음과 같은 방법으로 상태 비저장 서비스를 서로 바꿔 사용할 수 있다는 사실을 명시적으로 활용합니다.

- 참조 요소의 기준과 일치하는 여러 서비스 인스턴스가 있는 경우 참조 관리자는 해당 인스턴스 중 하나를 백업 인스턴스로 임의로 선택할 수 있습니다.

- 백업 서비스가 사라지면 참조 관리자는 동일한 유형의 다른 사용 가능한 서비스 중 하나를 사용하여 즉시 전환할 수 있습니다. 따라서 하나의 메서드 호출에서 다음 방법까지 프록시가 동일한 백업 서비스에 연결된 상태로 유지된다는 보장은 없습니다.

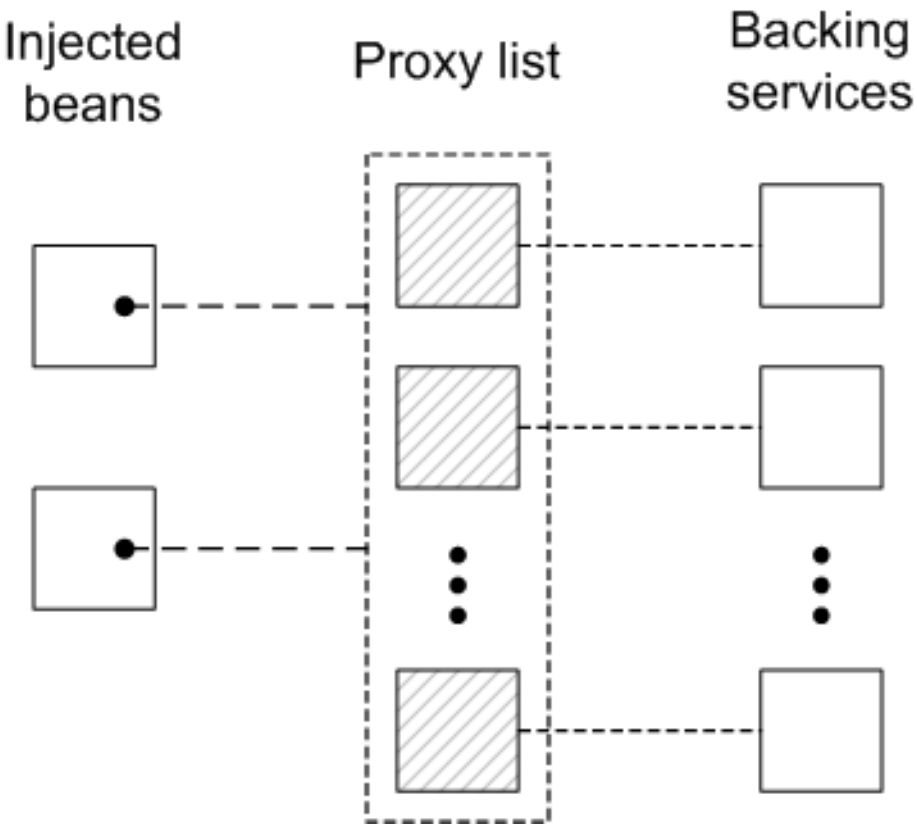
따라서 클라이언트와 백업 서비스 간의 계약은 상태 비저장이므로 클라이언트는 항상 동일한 서비스 인스턴스에 대해 이야기하고 있다고 가정 해서는 안 됩니다. 일치하는 서비스 인스턴스가 없는 경우 프록시는 **ServiceUnavailable** 예외를 **throw**하기 전에 특정 시간 동안 기다립니다. 시간 초과는 참조 요소에서 **timeout** 특성을 설정하여 구성할 수 있습니다.

참조 목록 관리자

참조 목록 관리자 인스턴스는 블루프린트 참조 목록 요소에 의해 생성됩니다. 이 요소는 서비스 참조 목록을 반환하고 상태 저장 서비스에 액세스하는 데 권장되는 접근 방식입니다. 그림 12.2. "상태 저장 서

비스에 대한 참조 목록" 참조 목록 관리자를 사용하여 상태 저장 서비스에 액세스하기 위한 모델의 개요를 보여줍니다.

그림 12.2. 상태 저장 서비스에 대한 참조 목록



*클라이언트 블루프린트 컨테이너의 빈은 프록시 오브젝트 목록을 포함하는 **java.util.List** 오브젝트(제공된 오브젝트)와 함께 삽입됩니다. 각 프록시는 **OSGi** 서비스 레지스트리의 고유한 서비스 인스턴스에서 지원합니다. 상태 비저장 모델과 달리 백업 서비스는 여기에서 서로 바꿔 사용할 수 있는 것으로 간주되지 않습니다. 사실, 목록에 있는 각 프록시의 라이프사이클은 해당 백업 서비스의 라이프사이클과 밀접하게 연결되어 있습니다. 서비스가 **OSGi** 레지스트리에 등록되면 해당 프록시가 동기적으로 생성되고 프록시 목록에 추가됩니다. 서비스가 **OSGi** 레지스트리에서 등록 해제되는 경우 해당 프록시는 프록시 목록에서 동시에 제거됩니다.*

*따라서 프록시와 해당 백업 서비스 간의 계약은 상태 저장 이므로 클라이언트는 특정 프록시에서 메서드를 호출할 때 항상 동일한 백업 서비스와 통신할 때 이를 가정할 수 있습니다. 그러나 백업 서비스를 사용할 수 없게 되어 프록시가 오래될 수 있습니다. 오래된 프록시에서 메서드를 호출하려고 하면 **ServiceUnavailable** 예외가 생성됩니다.*

*인터페이스(**stateless**)로 일치*

***stateles** 서비스 참조를 얻는 가장 간단한 방법은 참조 요소의 **interface** 특성을 사용하여 일치시킬 인터페이스를 지정하는 것입니다. 인터페이스 특성 값이 서비스의 상위 유형인 경우 또는 특성 값이 서비스에서 구현된 **Java** 인터페이스인 경우 서비스는 일치하는 것으로 간주됩니다(**interface** 속성은 **Java** 클래스 또는 **Java** 인터페이스를 지정할 수 있음).*

예를 들어 상태 비저장 **SavingsAccount** 서비스(예 12.1. “단일 인터페이스를 사용한 샘플 서비스 내보내기”참조)를 참조하려면 다음과 같이 참조 요소를 정의합니다.

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference id="savingsRef"
             interface="org.fusesource.example.SavingsAccount"/>

  <bean id="client" class="org.fusesource.example.client.Client">
    <property name="savingsAccount" ref="savingsRef"/>
  </bean>

</blueprint>
```

참조 요소가 **ID**를 사용하여 참조 관리자 **8080**을 생성하는 위치, **savingsRef.** 참조된 서비스를 사용하려면 다음과 같이 **savingsRef** 빈을 클라이언트 클래스 중 하나에 삽입합니다.

클라이언트 클래스에 삽입된 **useful** 속성은 **SavingsAccount** 에서 할당할 수 있는 모든 유형일 수 있습니다. 예를 들어 다음과 같이 **Client** 클래스를 정의할 수 있습니다.

```
package org.fusesource.example.client;

import org.fusesource.example.SavingsAccount;

public class Client {
    SavingsAccount savingsAccount;

    // Bean properties
    public SavingsAccount getSavingsAccount() {
        return savingsAccount;
    }

    public void setSavingsAccount(SavingsAccount savingsAccount) {
        this.savingsAccount = savingsAccount;
    }
    ...
}
```

인터페이스(상태)로 일치

상태 저장 서비스 참조를 얻는 가장 간단한 방법은 **reference-list** 요소에서 **interface** 특성을 사용하여 일치하는 인터페이스를 지정하는 것입니다. 그런 다음 참조 목록 관리자는 인터페이스 속성 값이 서비스의 상위 유형이거나 서비스에서 구현하는 **Java** 인터페이스인 모든 서비스 목록을 가져옵니다(**interface** 속성은 **Java** 클래스 또는 **Java** 인터페이스를 지정할 수 있음).

예를 들어 상태 저장 **SavingsAccount** 서비스(예 12.1. “단일 인터페이스를 사용한 샘플 서비스 내보내기”참조)를 참조하려면 다음과 같이 **reference-list** 요소를 정의합니다.

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference-list id="savingsListRef"
                  interface="org.fusesource.example.SavingsAccount"/>

  <bean id="client" class="org.fusesource.example.client.Client">
    <property name="savingsAccountList" ref="savingsListRef"/>
  </bean>

</blueprint>
```

***reference-list** 요소가 **ID**를 사용하여 참조 목록 관리자 **8080**을 생성하는 경우 **savingsListRef.** 참조된 서비스 목록을 사용하려면 다음과 같이 **savingsListRef** 빈 참조를 클라이언트 클래스 중 하나에 삽입합니다.*

*기본적으로 **savingsAccountList 8080** 속성은 서비스 오브젝트 목록(예: **java.util.List<SavingsAccount>**)입니다. 클라이언트 클래스를 다음과 같이 정의할 수 있습니다.*

```
package org.fusesource.example.client;

import org.fusesource.example.SavingsAccount;

public class Client {
    java.util.List<SavingsAccount> accountList;

    // Bean properties
    public java.util.List<SavingsAccount> getSavingsAccountList() {
        return accountList;
    }

    public void setSavingsAccountList(
                    java.util.List<SavingsAccount> accountList
    ) {
        this.accountList = accountList;
    }
    ...
}
```

인터페이스 및 구성 요소 이름으로 일치

*상태 비저장 서비스의 인터페이스 및 구성 요소 이름(예: **ID**)을 모두 일치시키려면 다음과 같이 참조 요소의 **interface** 속성과 **component-name** 속성을 모두 지정합니다.*

```
<reference id="savingsRef"
           interface="org.fusesource.example.SavingsAccount"
           component-name="savings"/>
```

*상태 저장 서비스의 인터페이스 및 구성 요소 이름(**bean ID**)을 모두 일치시키려면 다음과 같이*

reference-list 요소에서 **interface** 속성과 **component-name** 속성을 모두 지정합니다.

```
<reference-list id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        component-name="savings"/>
```

필터를 사용하여 서비스 속성 일치

필터에 대해 서비스 속성을 일치시켜 서비스를 선택할 수 있습니다. 필터는 참조 요소 또는 **reference-list** 요소에 대한 **filter** 특성을 사용하여 지정됩니다. **filter** 속성의 값은 **LDAP** 필터 표현식 이어야 합니다. 예를 들어 **bank.name** 서비스 속성이 **HighStreetBank** 와 같은 경우 일치하는 필터를 정의하려면 다음 **LDAP** 필터 표현식을 사용할 수 있습니다.

```
(bank.name=HighStreetBank)
```

두 서비스 속성 값을 일치시키려면 식을 논리 와 결합한 **& amp;** 함께 사용할 수 있습니다. 예를 들어 **foo** 속성이 **FooValue** 와 같고 **bar** 속성이 **BarValue** 와 같도록 하려면 다음 **LDAP** 필터 표현식을 사용할 수 있습니다.

```
(&(foo=FooValue)(bar=BarValue))
```

LDAP 필터 표현식의 전체 구문은 **OSGi** 코어 사양 의 **3.2.7** 섹션을 참조하십시오.

필터는 **interface** 및 **component-name** 설정과 결합할 수도 있습니다. 이 경우 지정된 조건이 모두 일치해야 합니다.

예를 들어 **SavingsAccount** 유형의 상태 비저장 서비스와 **high StreetBank** 와 같은 **bank.name** 서비스 속성을 일치시키려면 다음과 같이 참조 요소를 정의할 수 있습니다.

```
<reference id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        filter="(bank.name=HighStreetBank)"/>
```

SavingsAccount 유형의 상태 저장 서비스와 **HighStreetBank** 와 같은 **bank.name** 서비스 속성을 일치시키려면 다음과 같이 참조 목록 요소를 정의할 수 있습니다.

```
<reference-list id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        filter="(bank.name=HighStreetBank)"/>
```

필수 또는 선택 사항 지정

기본적으로 **OSGi** 서비스에 대한 참조는 필수라고 가정합니다(필수 종속 항목참조). 요소에 가용성 특성을 설정하여 참조 요소 또는 참조 목록 요소의 종속성 동작을 사용자 지정할 수 있습니다.

availability 속성의 두 가지 가능한 값이 있습니다.

- 필수 (기본값)는 일반 블루프린트 컨테이너 초기화 중에 종속성을 해석 해야 함을 의미합니다.
- 선택 사항 인 에서는 초기화 중에 종속성을 해결할 필요가 없습니다.

다음 참조 요소의 예제에서는 참조 가 필수 종속성임을 명시적으로 선언하는 방법을 보여줍니다.

```
<reference id="savingsRef"
        interface="org.fusesource.example.SavingsAccount"
        availability="mandatory"/>
```

참조 리스너 지정

예를 들어 일부 서비스 참조를 선택적 가용성으로 선언한 경우와 같이 **OSGi** 환경의 동적 특성을 해결하기 위해 백업 서비스가 레지스트리에 바인딩되는 시기와 레지스트리에서 바인딩되지 않는 시기를 추적하는 것이 유용한 경우가 많습니다. 서비스 바인딩 및 바인딩 해제 이벤트의 알림을 수신하려면 **reference-listener** 요소를 참조 요소 또는 **reference -list** 요소의 자식으로 정의할 수 있습니다.

예를 들어 다음 블루프린트 구성은 참조 리스너를 **ID**가 있는 참조 관리자의 자식으로 정의하는 방법을 보여줍니다. **savingsRef:**

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference id="savingsRef"
          interface="org.fusesource.example.SavingsAccount"
          >
    <reference-listener bind-method="onBind" unbind-method="onUnbind">
      <bean class="org.fusesource.example.client.Listener"/>
    </reference-listener>
  </reference>

  <bean id="client" class="org.fusesource.example.client.Client">
    <property name="savingsAcc" ref="savingsRef"/>
  </bean>

</blueprint>
```

이전 구성에서는 **org.fusesource.example.client.Listener** 유형의 인스턴스를 **bind** 및 **unbind** 이벤트를 수신 대기하는 콜백으로 등록합니다. **savingsRef** 참조 관리자의 백업 서비스가 바인딩되거나 바인딩 해제될 때마다 이벤트가 생성됩니다.

다음 예제에서는 **Listener** 클래스의 예제 구현을 보여줍니다.

```
package org.fusesource.example.client;

import org.osgi.framework.ServiceReference;

public class Listener {

    public void onBind(ServiceReference ref) {
        System.out.println("Bound service: " + ref);
    }

    public void onUnbind(ServiceReference ref) {
        System.out.println("Unbound service: " + ref);
    }

}
```

메서드 이름, **onBind** 및 **onUnbind** 는 각각 **bind-method** 및 **unbind-method** 속성으로 지정됩니다. 이러한 콜백 방법은 모두 **org.osgi.framework.ServiceReference** 인수를 사용합니다.

12.4. OSGI 서비스 게시

12.4.1. 개요

이 섹션에서는 **OSGi** 컨테이너에서 간단한 **OSGi** 서비스를 생성, 빌드 및 배포하는 방법을 설명합니다. 서비스는 간단한 **Hello World Java** 클래스이며 **OSGi** 구성은 블루프린트 구성 파일을 사용하여 정의됩니다.

12.4.2. 사전 요구 사항

Maven 빠른 시작 **archetype**을 사용하여 프로젝트를 생성하려면 다음과 같은 사전 요구 사항이 있어야 합니다.

- **Maven** 설치-**Maven**은 **Apache**의 무료 오픈 소스 빌드 툴입니다. http://maven.apache.org/download.html 에서 최신 버전을 다운로드할 수 있습니다(최소 버전 **2.0.9**).

- *인터넷 연결- 빌드를 수행하는whilst, Maven은 외부 리포지토리를 동적으로 검색하고 즉시 필요한 아티팩트를 다운로드합니다. 이 기능이 작동하려면 빌드 머신이 인터넷에 연결되어 있어야 합니다.*

12.4.3. Maven 프로젝트 생성

maven-archetype-quickstart archetype은 일반 Maven 프로젝트를 생성한 다음 원하는 목적에 맞게 사용자 지정할 수 있습니다. 좌표를 사용하여 Maven 프로젝트를 생성하려면 org.fusesource.example:osgi-service 를 입력합니다.

```
mvn archetype:create
-DarchetypeArtifactId=maven-archetype-quickstart
-DgroupId=org.fusesource.example
-DartifactId=osgi-service
```

이 명령의 결과는 생성된 프로젝트의 파일이 포함된 ProjectDir/osgi-service 디렉터리입니다.

참고

기존 제품의 그룹 ID와 충돌하는 아티팩트의 그룹 ID를 선택하지 않도록 주의하십시오! 이로 인해 프로젝트의 패키지와 기존 제품의 패키지가 충돌할 수 있습니다(일반적으로 프로젝트 Java 패키지 이름의 루트로 그룹 ID가 사용되므로).

12.4.4. POM 파일 사용자 정의

다음과 같이 OSGi 번들을 생성하려면 POM 파일을 사용자 지정해야 합니다.

1. *5.1절. "번들 프로젝트 생성" 에 설명된 POM 사용자 지정 단계를 따르십시오.*

2. *Maven 번들 플러그인의 구성에서 bundle 지침을 수정하여 다음과 같이 org.fusesource.example.service 패키지를 내보냅니다.*

```
<project ... >
 ...
 <build>
  ...
  <plugins>
   ...
   <plugin>
    <groupId>org.apache.felix</groupId>
```

```
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <Bundle-SymbolicName>${pom.groupId}.${pom.artifactId}</Bundle-SymbolicName>
        <Export-Package>org.fusesource.example.service</Export-Package>
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

12.4.5. 서비스 인터페이스 작성

ProjectDir/osgi-service/src/main/java/org/fusesource/example/service 하위 디렉토리를 생성합니다. 이 디렉터리에서 원하는 텍스트 편집기를 사용하여 파일 **HelloWorldSvc.java** 를 생성하고 예 12.3. "The HelloWorldSvc Interface" 의 코드를 추가합니다.

예 12.3. The HelloWorldSvc Interface

```
package org.fusesource.example.service;

public interface HelloWorldSvc
{
    public void sayHello();
}
```

12.4.6. 서비스 클래스 작성

ProjectDir/osgi-service/src/main/java/org/fusesource/example/service/impl 하위 디렉토리를 생성합니다. 이 디렉터리에서 원하는 텍스트 편집기를 사용하여 파일 **HelloWorldSvcImpl.java** 를 생성하고 예 12.4. "HelloWorldSvcImpl 클래스" 의 코드를 추가합니다.

예 12.4. HelloWorldSvcImpl 클래스

```
package org.fusesource.example.service.impl;

import org.fusesource.example.service.HelloWorldSvc;

public class HelloWorldSvcImpl implements HelloWorldSvc {

    public void sayHello()
    {
        System.out.println( "Hello World!" );
```

```
    }
}
```

12.4.7. 블루프린트 파일 작성

블루프린트 구성 파일은 클래스 경로의 **OSGI-INF/blueprint** 디렉터리에 저장된 **XML** 파일입니다. 프로젝트에 블루프린트 파일을 추가하려면 먼저 다음 하위 디렉터리를 생성합니다.

```
ProjectDir/osgi-service/src/main/resources
ProjectDir/osgi-service/src/main/resources/OSGI-INF
ProjectDir/osgi-service/src/main/resources/OSGI-INF/blueprint
```

여기서 **src/main/resources** 는 모든 **JAR** 리소스의 표준 **Maven** 위치입니다. 이 디렉터리에 있는 리소스 파일은 생성된 번들 **JAR**의 루트 범위에 자동으로 패키지됩니다.

예 12.5. “서비스 내보내기를 위한 블루프린트 파일” 빈 요소를 사용하여 **HelloWorldSvc 8080**을 만든 다음 **service** 요소를 사용하여**ans**를 **OSGi** 서비스로 내보내는 샘플 블루프린트 파일을 표시합니다.

ProjectDir/osgi-service/src/main/resources/OSGI-INF/blueprint 디렉터리 아래에서 원하는 텍스트 편집기를 사용하여 **config.xml** 을 생성하고 예 12.5. “서비스 내보내기를 위한 블루프린트 파일” 에서 **XML** 코드를 추가합니다.

예 12.5. 서비스 내보내기를 위한 블루프린트 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="hello" class="org.fusesource.example.service.impl.HelloWorldSvcImpl"/>

  <service ref="hello" interface="org.fusesource.example.service.HelloWorldSvc"/>

</blueprint>
```

12.4.8. 서비스 번들 실행

osgi-service 프로젝트를 설치하고 실행하려면 다음 단계를 수행합니다.

1. 프로젝트 빌드- 명령 프롬프트를 열고 디렉터리를 **ProjectDir/osgi-service** 로 변경합니다. **Maven**을 사용하여 다음 명령을 입력하여 데모를 빌드합니다.

```
mvn install
```

이 명령이 성공적으로 실행되면 **ProjectDir/osgi-service/target** 디렉터리에 번들 파일 **osgi-service-1.0-SNAPSHOT.jar.**

2. **osgi-service** 번들을 설치하고 시작합니다. **Red Hat Fuse** 콘솔에서 다음 명령을 입력합니다.

```
Jkaraf@root()> bundle:install -s file:ProjectDir/osgi-service/target/osgi-service-1.0-SNAPSHOT.jar
```

여기서 **ProjectDir** 은 **Maven** 프로젝트를 포함하는 디렉터리이며 **-s** 플래그는 즉시 번들을 시작하도록 컨테이너를 지시합니다. 예를 들어 프로젝트 디렉터리가 **Windows** 머신의 **C:\Projects** 인 경우 다음 명령을 입력합니다.

```
karaf@root()> bundle:install -s file:C:/Projects/osgi-service/target/osgi-service-1.0-SNAPSHOT.jar
```

참고

Windows 머신에서 파일 **URL**을 포맷하는 방법을 주의하십시오. 파일 **URL** 처리기에서 이해하는 구문에 대한 자세한 내용은 **15.1절. "파일 URL 핸들러"** 을 참조하십시오.

3. 서비스가 생성되었는지 확인하려면번들이 성공적으로 시작되었는지 확인하려면 다음 **Red Hat Fuse console** 명령을 입력합니다.

```
karaf@root()> bundle:list
```

이 목록에는 **osgi-service** 번들에 대한 행이 표시되어야 합니다. 예를 들면 다음과 같습니다.

```
[ 236] [Active     ] [Created     ] [       ] [   60] osgi-service (1.0.0.SNAPSHOT)
```

12.5. OSGI 서비스 액세스

12.5.1. 개요

이 섹션에서는 OSGi 컨테이너에 간단한 OSGi 클라이언트를 생성, 빌드 및 배포하는 방법을 설명합니다. 클라이언트는 OSGi 레지스트리에서 간단한 Hello World 서비스를 찾고 sayHello() 메서드를 호출합니다.

12.5.2. 사전 요구 사항

Maven 빠른 시작 archetype을 사용하여 프로젝트를 생성하려면 다음과 같은 사전 요구 사항이 있어야 합니다.

- Maven 설치-Maven은 Apache의 무료 오픈 소스 빌드 툴입니다. http://maven.apache.org/download.html 에서 최신 버전을 다운로드할 수 있습니다(최소 버전 2.0.9).
- 인터넷 연결- 빌드를 수행하는whilst, Maven은 외부 리포지토리를 동적으로 검색하고 즉시 필요한 아티팩트를 다운로드합니다. 이 기능이 작동하려면 빌드 머신이 인터넷에 연결되어 있어야 합니다.

12.5.3. Maven 프로젝트 생성

maven-archetype-quickstart archetype은 일반 Maven 프로젝트를 생성한 다음 원하는 목적에 맞게 사용자 지정할 수 있습니다. 좌표를 사용하여 Maven 프로젝트를 생성하려면 org.fusesource.example:osgi-client 를 입력합니다.

```
mvn archetype:create
-DarchetypeArtifactId=maven-archetype-quickstart
-DgroupId=org.fusesource.example
-DartifactId=osgi-client
```

이 명령의 결과는 생성된 프로젝트의 파일이 포함된 ProjectDir/osgi-client 디렉터리입니다.

참고

기존 제품의 그룹 ID와 충돌하는 아티팩트의 그룹 ID를 선택하지 않도록 주의하십시오! 이로 인해 프로젝트의 패키지와 기존 제품의 패키지가 충돌할 수 있습니다(일반적으로 프로젝트 Java 패키지 이름의 루트로 그룹 ID가 사용되므로).

12.5.4. POM 파일 사용자 정의

다음과 같이 OSGi 번들을 생성하려면 POM 파일을 사용자 지정해야 합니다.

1. 5.1절. “번들 프로젝트 생성” 에 설명된 POM 사용자 지정 단계를 따르십시오.

2. 클라이언트는 osgi-service 번들에 정의된 HelloWorldSvc Java 인터페이스를 사용하므로 osgi-service 번들에 Maven 종속성을 추가해야 합니다. osgi-service 번들의 Maven 조정이 org.fusesource.example:osgi-service:1.0-SNAPSHOT 이라고 가정하면 클라이언트의 POM 파일에 다음 종속성을 추가해야 합니다.

```
<project ... >
 ...
 <dependencies>
  ...
  <dependency>
     <groupId>org.fusesource.example</groupId>
     <artifactId>osgi-service</artifactId>
     <version>1.0-SNAPSHOT</version>
  </dependency>
 </dependencies>
 ...
</project>
```

12.5.5. 블루프린트 파일 작성

클라이언트 프로젝트에 블루프린트 파일을 추가하려면 먼저 다음 하위 디렉터리를 생성합니다.

```
ProjectDir/osgi-client/src/main/resources
ProjectDir/osgi-client/src/main/resources/OSGI-INF
ProjectDir/osgi-client/src/main/resources/OSGI-INF/blueprint
```

ProjectDir/osgi-client/src/main/resources/OSGI-INF/blueprint 디렉터리 아래에서 원하는 텍스트 편집기를 사용하여 config.xml 을 생성하고 예 12.6. “서비스 가져오기를 위한 블루프린트 파일” 에서 XML 코드를 추가합니다.

예 12.6. 서비스 가져오기를 위한 블루프린트 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <reference id="helloWorld"
      interface="org.fusesource.example.service.HelloWorldSvc"/>

  <bean id="client"
      class="org.fusesource.example.client.Client"
      init-method="init">
    <property name="helloWorldSvc" ref="helloWorld"/>
```

```
  </bean>

</blueprint>
```

여기서 **reference** 요소는 **OSGi** 레지스트리에서 **HelloWorldSvc** 유형의 서비스를 찾는 참조 관리자를 생성합니다. **Cryo stat** 요소는 **Client** 클래스의 인스턴스를 생성하고 서비스 참조를 **8080** 속성 **helloWorldSvc** 로 삽입합니다. 또한 **init-method** 속성은 빈 초기화 단계에서 **Client.init()** 메서드가 호출되도록 지정합니다(즉, 서비스 참조가 클라이언트사이에 삽입된 후).

12.5.6. 클라이언트 클래스 작성

ProjectDir/osgi-client/src/main/java/org/fusesource/example/client 디렉터리 아래의 텍스트 편집기를 사용하여 파일을 생성하고, **Client.java** 에서 **Java** 코드를 추가합니다. 예 12.7. “클라이언트 클래스”

예 12.7. 클라이언트 클래스

```
package org.fusesource.example.client;

import org.fusesource.example.service.HelloWorldSvc;

public class Client {
    HelloWorldSvc helloWorldSvc;

    // Bean properties
    public HelloWorldSvc getHelloWorldSvc() {
        return helloWorldSvc;
    }

    public void setHelloWorldSvc(HelloWorldSvc helloWorldSvc) {
        this.helloWorldSvc = helloWorldSvc;
    }

    public void init() {
        System.out.println("OSGi client started.");
        if (helloWorldSvc != null) {
            System.out.println("Calling sayHello()");
            helloWorldSvc.sayHello();  // Invoke the OSGi service!
        }
    }

}
```

클라이언트 클래스는 삽입을 통해 **Hello World** 서비스에 대한 참조를 수신할 수 있는 **helloWorldSvc 8080** 속성에 대한 **getter** 및 **setter** 메서드를 정의합니다. **init()** 메서드는 속성 삽입 후 **8080** 초기화 단계에서 호출되므로 일반적으로 이 메서드 범위 내에서 **Hello World** 서비스를 호출할 수 있습니다.

12.5.7. 클라이언트 번들 실행

osgi-client 프로젝트를 설치하고 실행하려면 다음 단계를 수행합니다.

1. 프로젝트 빌드- 명령 프롬프트를 열고 디렉터리를 **ProjectDir/osgi-client** 로 변경합니다. **Maven**을 사용하여 다음 명령을 입력하여 데모를 빌드합니다.

```
mvn install
```

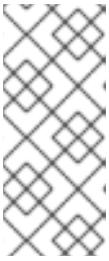
이 명령이 성공적으로 실행되면 **ProjectDir/osgi-client/target** 디렉터리에 번들 파일 **osgi-client-1.0-SNAPSHOT.jar.**

2. **osgi-service** 번들을 설치하고 시작합니다. **Red Hat Fuse** 콘솔에서 다음 명령을 입력합니다.

```
karaf@root()> bundle:install -s file:ProjectDir/osgi-client/target/osgi-client-1.0-SNAPSHOT.jar
```

여기서 **ProjectDir** 은 **Maven** 프로젝트를 포함하는 디렉터리이며 **-s** 플래그는 즉시 번들을 시작하도록 컨테이너를 지시합니다. 예를 들어 프로젝트 디렉터리가 **Windows** 머신의 **C:\Projects** 인 경우 다음 명령을 입력합니다.

```
karaf@root()> bundle:install -s file:C:/Projects/osgi-client/target/osgi-client-1.0-SNAPSHOT.jar
```



참고

Windows 머신에서 파일 **URL**을 포맷하는 방법을 주의하십시오. 파일 **URL** 처리기에서 이해하는 구문에 대한 자세한 내용은 **15.1절. “파일 URL 핸들러”** 을 참조하십시오.

3. 클라이언트 출력- 클라이언트 번들이 성공적으로 시작되면 콘솔에 다음과 같은 출력이 즉시 표시됩니다.

```
Bundle ID: 239
OSGi client started.
Calling sayHello()
Hello World!
```

12.6. APACHE CAMEL과의 통합

12.6.1. 개요

Apache Camel은 **Cryostat** 언어를 사용하여 **OSGi** 서비스를 쉽게 호출할 수 있는 방법을 제공합니다. 이 기능은 **Apache Camel** 애플리케이션을 **OSGi** 컨테이너에 배포할 때마다 자동으로 사용할 수 있으며 특별한 구성이 필요하지 않습니다.

12.6.2. 레지스트리 연결

Apache Camel 경로가 **OSGi** 컨테이너에 배포되면 **CamelContext** 에서 빈 인스턴스를 해결하기 위해 레지스트리 체인을 자동으로 설정합니다. 레지스트리 체인은 **OSGi** 레지스트리와 블루프린트 레지스트리로 구성됩니다. 이제 특정 **8080** 클래스 또는 빈 인스턴스를 참조하려고 하면 레지스트리에서 다음과 같이 빈을 확인합니다.

1. 먼저 **OSGi** 레지스트리에서 **8080**을 조회합니다. 클래스 이름이 지정된 경우 **OSGi** 서비스의 인터페이스 또는 클래스와 일치하십시오.

2. **OSGi** 레지스트리에 일치하는 항목이 없는 경우 블루프린트 레지스트리에 대체하십시오.

12.6.3. OSGi 서비스 인터페이스 샘플

단일 메서드인 **getGreeting()** 을 정의하는 다음 **Java** 인터페이스로 정의된 **OSGi** 서비스를 고려하십시오.

```
package org.fusesource.example.hello.boston;

public interface HelloBoston {
    public String getGreeting();
}
```

12.6.4. 샘플 서비스 내보내기

HelloBoston OSGi 서비스를 구현하는 번들을 정의할 때 다음 블루프린트 구성을 사용하여 서비스를 내보낼 수 있습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">

  <bean id="hello" class="org.fusesource.example.hello.boston.HelloBostonImpl"/>

  <service ref="hello" interface="org.fusesource.example.hello.boston.HelloBoston"/>

</blueprint>
```

-

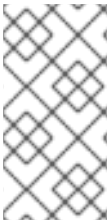
여기서 **HelloBoston** 인터페이스는 **HelloBostonImpl** 클래스에서 구현되어 있지 않다고 가정합니다.

12.6.5. Java DSL에서 OSGi 서비스 호출

HelloBoston OSGi 서비스가 포함된 번들을 배포한 후 **Java DSL**을 사용하여 **Apache Camel** 애플리케이션에서 서비스를 호출할 수 있습니다. **Java DSL**에서는 다음과 같이 **Cryostat** 언어를 통해 **OSGi** 서비스를 호출합니다.

```
from("timer:foo?period=5000")
  .bean(org.fusesource.example.hello.boston.HelloBoston.class, "getGreeting")
  .log("The message contains: ${body}")
```

8080 명령에서 첫 번째 인수는 **OSGi** 인터페이스 또는 클래스이며 **OSGi** 서비스 번들에서 내보낸 인터페이스와 일치해야 합니다. 두 번째 인수는 호출하려는 **8080** 메서드의 이름입니다. **Quarkus** 명령 구문에 대한 자세한 내용은 **Apache Camel Development Guide Cryostat Integration** 을 참조하십시오.



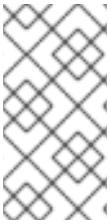
참고

이 방법을 사용하면 **OSGi** 서비스를 암시적으로 가져옵니다. 이 경우 **OSGi** 서비스를 명시적으로 가져올 필요가 없습니다.

12.6.6. XML DSL에서 OSGi 서비스 호출

XML DSL에서는 **Cryostat** 언어를 사용하여 **HelloBoston OSGi** 서비스를 호출할 수도 있지만 구문은 약간 다릅니다. **XML DSL**에서는 다음과 같이 **method** 요소를 사용하여 **Cryostat** 언어를 통해 **OSGi** 서비스를 호출합니다.

```
<beans ...>
  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
      <from uri="timer:foo?period=5000"/>
      <setBody>
         <method ref="org.fusesource.example.hello.boston.HelloBoston" method="getGreeting"/>
      </setBody>
      <log message="The message contains: ${body}"/>
    </route>
  </camelContext>
</beans>
```



참고

*이 방법을 사용하면 **OSGi** 서비스를 암시적으로 가져옵니다. 이 경우 **OSGi** 서비스를 명시적으로 가져올 필요가 없습니다.*

13장. JMS 브로커를 사용하여 배포

초록

Fuse 7.11은 기본 내부 브로커와 함께 제공되지 않지만 4개의 외부 JMS 브로커와 상호 작용하도록 설계되었습니다.

Fuse 7.11 컨테이너에는 지원되는 외부 브로커를 위한 브로커 클라이언트 라이브러리가 포함되어 있습니다.

Fuse 7.11 메시징에 사용할 수 있는 외부 브로커, 클라이언트 및 Camel 구성 요소 조합에 대한 자세한 내용은 지원 구성을 참조하십시오.

13.1. AMQ 7 빠른 시작

AMQ 7 브로커를 사용하여 앱의 설정 및 배포를 설명하기 위해 빠른 시작이 제공됩니다.

빠른 시작 다운로드

Fuse 소프트웨어 다운로드 페이지에서 모든 빠른 시작을 설치할 수 있습니다.

다운로드한 zip 파일의 내용을 로컬 폴더(예: quickstarts 라는 폴더)로 추출합니다.

퀵스타트 설정

1. quickstarts/camel/camel-jms 폴더로 이동합니다.
2. mvn clean install 을 입력하여 퀵스타트를 빌드합니다.
3. org.ops4j.connectionfactory-amq7.cfg 파일을 /camel/camel-jms/src/main 디렉터리에서 Fuse 설치의 FUSE_HOME/etc 디렉토리로 복사합니다. 올바른 브로커 URL 및 인증 정보에 대한 콘텐츠를 확인합니다. 기본적으로 브로커 URL은 AMQ 7의 CORE 프로토콜에 따라 tcp://localhost:61616으로 설정됩니다. 인증 정보는 admin/admin으로 설정됩니다. 이러한 세부 정보를 외부 브로커에 맞게 변경하십시오.
4. Windows에서 ./bin/fuse 를 실행하거나 Linux에서 ./bin/fuse 를 실행하여 Fuse를 시작합니

다.

5. **Fuse** 콘솔에서 다음 명령을 입력합니다.

```
feature:install pax-jms-pool artemis-jms-client camel-blueprint camel-jms

install -s mvn:org.jboss.fuse.quickstarts/camel-jms/${project.version}
```

번들을 배포할 때 **Fuse**에서 번들 **ID**를 제공합니다.

6. **log:display** 를 입력하여 시작 로그 정보를 확인합니다. 번들이 성공적으로 배포되었는지 확인합니다.

```
12:13:50.445 INFO [Blueprint Event Dispatcher: 1] Attempting to start Camel Context jms-example-context
12:13:50.446 INFO [Blueprint Event Dispatcher: 1] Apache Camel 2.21.0.fuse-000030 (CamelContext: jms-example-context) is starting
12:13:50.446 INFO [Blueprint Event Dispatcher: 1] JMX is enabled
12:13:50.528 INFO [Blueprint Event Dispatcher: 1] StreamCaching is not in use. If using streams then its recommended to enable stream caching. See more details at http://camel.apache.org/stream-caching.html
12:13:50.553 INFO [Blueprint Event Dispatcher: 1] Route: file-to-jms-route started and consuming from: file://work/jms/input
12:13:50.555 INFO [Blueprint Event Dispatcher: 1] Route: jms-cbr-route started and consuming from: jms://queue:incomingOrders?transacted=true
12:13:50.556 INFO [Blueprint Event Dispatcher: 1] Total 2 routes, of which 2 are started
```

퀵스타트 실행

1. **Camel** 경로가 실행되면 **/camel/camel-jms/work/jms/input** 디렉터리가 생성됩니다. **/camel/camel-jms/src/main/data** 디렉토리에서 파일을 / **camel/camel-jms/work/jms/input** 디렉토리로 복사합니다.

2. **.../src/main/data** 파일에 복사된 파일은 주문 파일입니다. **1**분 정도 기다린 다음 **/camel/camel-jms/work/jms/output** 디렉터리를 확인합니다. 파일은 대상 국가에 따라 별도의 디렉터리로 정렬됩니다.

- **order1.xml,order2.xml** 및 **order4.xml** 의 **/camel/camel-jms/work/jms/output/others/**

- */camel/camel-jms/work/jms/output/us의 order3.xml 및 order5.xml*

- */camel/camel-jms/work/jms/output/fr의 order6.xml*

3. ***log:display** 를 사용하여 로그 메시지를 확인합니다.*

```
Receiving order order1.xml
Sending order order1.xml to another country
Done processing order1.xml
```

1. ***Camel** 명령은 컨텍스트에 대한 세부 정보를 표시합니다.*

*컨텍스트 세부 정보를 표시하려면 **camel:context-list** 를 사용합니다.*

```
Context             Status              Total #       Failed #      Inflight #    Uptime
-------             ------              -------       --------      ----------    ------
jms-example-context   Started               12             0             0   3 minutes
```

***context에 Camel** 경로를 표시하려면 **camel:route-list** 를 사용합니다.*

```
Context             Route               Status              Total #       Failed #      Inflight #    Uptime
-------             -----               ------              -------       --------      ----------    ------
jms-example-context   file-to-jms-route   Started               6             0             0   3 minutes
jms-example-context   jms-cbr-route       Started               6             0             0   3 minutes
```

*교환 통계를 표시하려면 **camel:route-info** 를 사용합니다.*

```
karaf@root()> camel:route-info jms-cbr-route jms-example-context
Camel Route jms-cbr-route
    Camel Context: jms-example-context
    State: Started
    State: Started

Statistics
    Exchanges Total: 6
    Exchanges Completed: 6
    Exchanges Failed: 0
    Exchanges Inflight: 0
    Min Processing Time: 2 ms
    Max Processing Time: 12 ms
    Mean Processing Time: 4 ms
```

```
Total Processing Time: 29 ms
Last Processing Time: 4 ms
Delta Processing Time: 1 ms
Start Statistics Date: 2018-01-30 12:13:50
Reset Statistics Date: 2018-01-30 12:13:50
First Exchange Date: 2018-01-30 12:19:47
Last Exchange Date: 2018-01-30 12:19:47
```

13.2. ARTEMIS 핵심 클라이언트 사용

Artemis 코어 클라이언트는 qpid-jms-client 대신 외부 브로커에 연결하는 데 사용할 수 있습니다.

Artemis 코어 클라이언트를 사용하여 연결

1. Artemis 코어 클라이언트를 활성화하려면 Fuse를 시작합니다. FUSE_HOME 디렉토리로 이동하여 Linux에서 ./bin/fuse 를 입력하거나 Windows에서 bin\fuse.skip 를 입력합니다.

2. 다음 명령을 사용하여 Artemis 클라이언트를 기능으로 추가합니다. feature:install artemis-core-client

3. 코드를 작성할 때 Camel 구성 요소를 연결 팩토리와 연결해야 합니다.

연결 팩토리를 가져옵니다.

```
import org.apache.qpid.jms.JmsConnectionFactory;
```

연결을 설정합니다.

```
ConnectionFactory connectionFactory = new
JmsConnectionFactory("amqp://localhost:5672");
    try (Connection connection = connectionFactory.createConnection()) {
```

14장. 장애 조치 배포

초록

Red Hat Fuse는 간단한 잠금 파일 시스템 또는 **JDBC** 잠금 메커니즘을 사용하여 페일오버 기능을 제공합니다. 두 경우 모두 컨테이너 수준 잠금 시스템을 사용하면 빠른 페일오버 성능을 제공하기 위해 번들을 보조 커널 인스턴스로 사전 로드할 수 있습니다.

14.1. 간단한 잠금 파일 시스템 사용

14.1.1. 개요

Red Hat Fuse를 처음 시작하면 설치 디렉터리의 루트에 잠금 파일이 생성됩니다. 기본 인스턴스가 실패하면 동일한 호스트 시스템에 있는 보조 인스턴스에 잠금이 전달됩니다.

14.1.2. 잠금 파일 시스템 구성

잠금 파일 장애 조치 배포를 구성하려면 예 14.1. "파일 잠금 Cryostat 구성" 에 속성을 포함하도록 기본 설치와 보조 설치 모두에서 **etc/system.properties** 파일을 편집합니다.

예 14.1. 파일 잠금 Cryostat 구성

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.SimpleFileLock
karaf.lock.dir=PathToLockFileDirectory
karaf.lock.delay=10000
```

- **Karaf.lock -** 잠금파일이 기록되는지 여부를 지정합니다.
- **Karaf.lock.class-** 잠금을 구현하는 **Java** 클래스를 지정합니다. 간단한 파일 잠금의 경우 항상 **org.apache.karaf.main.SimpleFileLock** 이어야 합니다.
- **Karaf.lock.dir-** 잠금 파일이 작성된 디렉토리를 지정합니다. 기본 설치와 보조 설치 둘 다 동

일 해야 합니다.

- **Karaf.lock.delay-** 밀리초 단위로, 잠금 재조정 시도 사이의 지연을 지정합니다.

14.2. JDBC 잠금 시스템 사용

14.2.1. 개요

JDBC 잠금 메커니즘은 **Red Hat Fuse** 인스턴스가 별도의 시스템에 존재하는 페일오버 배포용입니다.

이 시나리오에서는 기본 인스턴스에는 데이터베이스에 호스팅되는 잠금 테이블에 대한 잠금이 있습니다. 기본 인스턴스가 잠금을 끊으면 대기 중인 보조 프로세스에서 잠금에 대한 액세스 권한을 얻고 해당 컨테이너를 완전히 시작합니다.

14.2.2. classpath에 JDBC 드라이버 추가

JDBC 잠금 시스템에서는 **primary/secondary** 설정의 각 인스턴스의 **classpath**에 **JDBC** 드라이버가 있어야 합니다. 다음과 같이 **JDBC** 드라이버를 **classpath**에 추가합니다.

1. **JDBC** 드라이버 **JAR** 파일을 각 **Red Hat Fuse** 인스턴스의 **ESBInstallDir/lib/ext** 디렉터리에 복사합니다.

2. **CLASSPATH** 변수에 **JDBC** 드라이버 **JAR**을 포함하도록 **bin/karaf** 시작 스크립트를 수정합니다.

   예를 들어 **JDBC JAR** 파일 **JDBCJarFile.jar** 가 제공되는 경우 다음과 같이 시작 스크립트를 수정할 수 있습니다 **(*NIX** 운영 체제**).**

```
...
# Add the jars in the lib dir
for file in "$KARAF_HOME"/lib/karaf*.jar
do
    if [ -z "$CLASSPATH" ]; then
        CLASSPATH="$file"
    else
        CLASSPATH="$CLASSPATH:$file"
```

```
    fi
done
CLASSPATH="$CLASSPATH:$KARAF_HOME/lib/JDBCJarFile.jar"
```

참고

MySQL 드라이버 **JAR** 또는 **PostgreSQL** 드라이버 **JAR**을 추가하는 경우 **karaf-** 접두사를 추가하여 드라이버 **JAR**의 이름을 변경해야합니다. 그렇지 않으면 **Apache Karaf**가 중단되고 로그에 **Apache Karaf**가 드라이버를 찾을 수 없음을 알려줍니다.

14.2.3. JDBC 잠금 시스템 구성

JDBC 잠금 시스템을 구성하려면 다음과 같이 기본/초 배포 시 각 인스턴스에 대해 **etc/system.properties** 파일을 업데이트하십시오.

예 14.2. JDBC 잠금 파일 구성

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.DefaultJDBCLock
karaf.lock.level=50
karaf.lock.delay=10000
karaf.lock.jdbc.url=jdbc:derby://dbserver:1527/sample
karaf.lock.jdbc.driver=org.apache.derby.jdbc.ClientDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

이 예제에서는 **sample**이라는 데이터베이스가 아직 없는 경우 생성됩니다. 잠금 테이블을 얻는 첫 번째 **Red Hat Fuse** 인스턴스는 기본 인스턴스입니다. 데이터베이스에 대한 연결이 손실되면 기본 인스턴스가 정상적으로 종료되어 데이터베이스 서비스가 복원될 때 보조 인스턴스가 기본 인스턴스가 될 수 있습니다. 이전 기본 인스턴스에는 수동 재시작이 필요합니다.

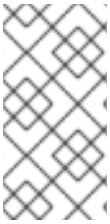
14.2.4. Oracle에서 JDBC 잠금 구성

JDBC 잠금 시나리오에서 **Oracle**을 데이터베이스로 사용하는 경우 **etc/system.properties** 파일의 **karaf.lock.class** 속성은 **org.apache.karaf.main.lock.OracleJDBCLock** 을 가리켜야 합니다.

또는 다음과 같이 **system.properties** 파일을 설정에 대해 정상적으로 구성합니다.

예 14.3. Oracle JDBC 잠금 파일 구성

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.OracleJDBCLock
karaf.lock.jdbc.url=jdbc:oracle:thin:@hostname:1521:XE
karaf.lock.jdbc.driver=oracle.jdbc.OracleDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```



참고

***karaf.lock.jdbc.url** 에는 활성 **Oracle** 시스템 **ID(SID)**가 필요합니다. 즉, 이 특정 잠금을 사용하기 전에 데이터베이스 인스턴스를 수동으로 생성해야 합니다.*

14.2.5. Derby에서 JDBC 잠금 구성

***JDBC** 잠금 시나리오에서 **Derby**를 데이터베이스로 사용하는 경우 **etc/system.properties** 파일의 **karaf.lock.class** 속성은 **org.apache.karaf.main.lock.DerbyJDBCLock** 를 가리켜야 합니다. 예를 들어 다음과 같이 **system.properties** 파일을 구성할 수 있습니다.*

예 14.4. Derby의 JDBC 잠금 파일 구성

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.DerbyJDBCLock
karaf.lock.jdbc.url=jdbc:derby://127.0.0.1:1527/dbname
karaf.lock.jdbc.driver=org.apache.derby.jdbc.ClientDriver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

14.2.6. MySQL에서 JDBC 잠금 구성

***JDBC** 잠금 시나리오에서 **MySQL**을 데이터베이스로 사용하는 경우 **etc/system.properties** 파일의 **karaf.lock.class** 속성은 **org.apache.karaf.main.lock.MySQLJDBCLock** 을 가리켜야 합니다. 예를 들어 다음과 같이 **system.properties** 파일을 구성할 수 있습니다.*

예 14.5. MySQL의 JDBC 잠금 파일 구성

```
karaf.lock=true
```

```
karaf.lock.class=org.apache.karaf.main.lock.MySQLJDBCLock
karaf.lock.jdbc.url=jdbc:mysql://127.0.0.1:3306/dbname
karaf.lock.jdbc.driver=com.mysql.jdbc.Driver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=30
```

14.2.7. PostgreSQL에서 JDBC 잠금 구성

JDBC 잠금 시나리오에서 **PostgreSQL**을 데이터베이스로 사용하는 경우 **etc/system.properties** 파일의 **karaf.lock.class** 속성은 **org.apache.karaf.main.lock.PostgreSQLJDBCLock** 을 가리켜야 합니다. 예를 들어 다음과 같이 **system.properties** 파일을 구성할 수 있습니다.

예 14.6. PostgreSQL에 대한 JDBC 잠금 파일 구성

```
karaf.lock=true
karaf.lock.class=org.apache.karaf.main.lock.PostgreSQLJDBCLock
karaf.lock.jdbc.url=jdbc:postgresql://127.0.0.1:5432/dbname
karaf.lock.jdbc.driver=org.postgresql.Driver
karaf.lock.jdbc.user=user
karaf.lock.jdbc.password=password
karaf.lock.jdbc.table=KARAF_LOCK
karaf.lock.jdbc.clustername=karaf
karaf.lock.jdbc.timeout=0
```

14.2.8. JDBC 잠금 클래스

다음 **JDBC** 잠금 클래스는 현재 **Apache Karaf**에서 제공합니다.

```
org.apache.karaf.main.lock.DefaultJDBCLock
org.apache.karaf.main.lock.DerbyJDBCLock
org.apache.karaf.main.lock.MySQLJDBCLock
org.apache.karaf.main.lock.OracleJDBCLock
org.apache.karaf.main.lock.PostgreSQLJDBCLock
```

14.3. 컨테이너 수준 잠금

14.3.1. 개요

컨테이너 수준 잠금을 사용하면 더 빠른 페일오버 성능을 제공하기 위해 번들을 보조 커널 인스턴스에 사전 로드할 수 있습니다. 컨테이너 수준 잠금은 간단한 파일 및 **JDBC** 잠금 메커니즘 모두에서 지원됩니

다.

14.3.2. 컨테이너 수준 잠금 구성

컨테이너 수준 잠금을 구현하려면 기본/초별 설정의 각 시스템의 **etc/system.properties** 파일에 다음을 추가합니다.

> 예 14.7. 컨테이너 수준 잠금 구성
>
> ```
> karaf.lock=true
> karaf.lock.level=50
> karaf.lock.delay=10000
> ```

karaf.lock.level 속성은 **Red Hat Fuse** 인스턴스에 **OSGi** 컨테이너를 가져오는 부팅 프로세스의 범위를 알려줍니다. 그러면 동일한 시작 수준 또는 낮음이 할당된 번들이 해당 **Fuse** 인스턴스에서도 시작됩니다.

번들 시작 수준은 **BundleName.jar=level** 형식으로 **etc/startup. properties** 형식으로 지정됩니다. 코어 시스템 번들에는 **50** 미만의 수준이 있으며, 여기서 사용자 번들의 수준이 **50**보다 큽니다.

표 14.1. 번들 시작 수준

시작 수준	동작
1	'cold' 대기 인스턴스입니다. 코어 번들은 컨테이너에 로드되지 않습니다. 보조 인스턴스는 서버를 시작하기 위해 획득된 잠금이 완료될 때까지 기다립니다.
<50	'hot' 대기 인스턴스입니다. 코어 번들은 컨테이너에 로드됩니다. 보조 인스턴스는 사용자 수준 번들을 시작하기 위해 획득된 잠금이 완료될 때까지 기다립니다. 이 수준에서 각 보조 인스턴스에 대해 콘솔에 액세스할 수 있습니다.
>50	사용자 번들이 시작되므로 이 설정은 권장되지 않습니다.

14.3.3. 포트 충돌 방지

동일한 호스트에 **'hot'** 예비를 사용하는 경우 바인딩 충돌을 방지하기 위해 **Cryostat** 원격 포트를 고유한 값으로 설정해야 합니다. 다음을 포함하도록 **fuse** 시작 스크립트(또는 하위 인스턴스에서 **karaf** 스크립트)를 편집할 수 있습니다.

```
DEFAULT_JAVA_OPTS="-server $DEFAULT_JAVA_OPTS -
Dcom.sun.management.jmxremote.port=1100 -
Dcom.sun.management.jmxremote.authenticate=false"
```

15장. URL 처리기

Red Hat Fuse에는 리소스의 위치(예: 콘솔 명령에 대한 인수로)를 지정하는 URL을 제공해야 하는 많은 컨텍스트가 있습니다. 일반적으로 URL을 지정할 때 Fuse의 기본 제공 URL 처리기에서 지원하는 모든 체계를 사용할 수 있습니다. 이 부록은 사용 가능한 모든 URL 처리기의 구문을 설명합니다.

15.1. 파일 URL 핸들러

15.1.1. 구문

파일 URL의 구문은 pathName입니다. 여기서 PathName 은 Classpath에서 사용할 수 있는 파일의 상대 또는 절대 경로 이름입니다. 제공된 PathName 은 Java의 기본 제공 파일 URL 처리기 에서 구문 분석합니다. 따라서 PathName 구문에는 Java 경로 이름의 일반적인 규칙이 적용됩니다. 특히 Windows에서는 각 백슬래시를 다른 백슬래시로 이스케이프하거나 슬래시로 교체해야 합니다.

15.1.2. 예

예를 들어 Windows에서 경로 이름 C:\Projects\camel-bundle\target\foo-1.0-SNAPSHOT.jar 를 고려하십시오. 다음 예제에서는 Windows의 파일 URL에 대한 올바른 대안을 보여줍니다.

```
file:C:/Projects/camel-bundle/target/foo-1.0-SNAPSHOT.jar
file:C:\\Projects\\camel-bundle\\target\\foo-1.0-SNAPSHOT.jar
```

다음 예제에서는 Windows의 파일 URL에 대한 몇 가지 잘못된 대안을 보여줍니다.

```
file:C:\Projects\camel-bundle\target\foo-1.0-SNAPSHOT.jar       // WRONG!
file://C:/Projects/camel-bundle/target/foo-1.0-SNAPSHOT.jar     // WRONG!
file://C:\\Projects\\camel-bundle\\target\\foo-1.0-SNAPSHOT.jar // WRONG!
```

15.2. HTTP URL 처리기

15.2.1. 구문

HTTP URL에는 표준 구문, http:Host[:Port]/[Path][#AnchorName][?Query]. https 스키마를 사용하여 보안 HTTP URL을 지정할 수도 있습니다. 제공된 HTTP URL은 Java의 기본 제공 HTTP URL 처리기에서 구문 분석되므로 HTTP URL은 Java 애플리케이션의 일반적인 방식으로 작동합니다.

15.3. MVN URL HANDLER

15.3.1. 개요

Maven을 사용하여 번들을 빌드하거나 Maven 리포지토리에서 특정 번들을 사용할 수 있는 경우 Mvn 처리기 스키마를 사용하여 번들을 찾을 수 있습니다.

참고

Mvn URL 처리기에서 로컬 및 원격 Maven 아티팩트를 찾을 수 있도록 Mvn URL 처리기 구성을 사용자 지정해야 할 수 있습니다. 자세한 내용은 15.3.5절. "Mvn URL 처리기 구성"의 내용을 참조하십시오.

15.3.2. 구문

Mvn URL 구문은 다음과 같습니다.

```
mvn:[repositoryUrl!]groupId/artifactId[/[version][/[packaging][/[classifier]]]]
```

여기서 repositoryUrl 은 선택적으로 Maven 리포지토리의 URL을 지정합니다. groupId,artifactId,version,packaging, classifier 는 Maven 아티팩트를 찾기 위한 표준 Maven 좌표입니다.

15.3.3. 좌표 생략

Mvn URL을 지정하는 경우 groupId 및 artifactId 좌표만 필요합니다. 다음 예제에서는 groupId,org.fusesource.example 및 artifactId,bundle-demo 를 사용하여 Maven 번들을 참조합니다.

```
mvn:org.fusesource.example/bundle-demo
mvn:org.fusesource.example/bundle-demo/1.1
```

첫 번째 예와 같이 버전이 생략되면 기본값은 LATEST 이며 사용 가능한 Maven 메타데이터를 기반으로 최신 버전으로 해석됩니다.

패키징 또는 버전 값을 지정하지 않고 분류자 값을 지정하려면 Mvn URL에 간격을 둘 수 있습니다. 마찬가지로 버전 값 없이 패키징 값을 지정하려면 다음을 수행합니다. 예를 들면 다음과 같습니다.

```
mvn:groupId/artifactId///classifier
mvn:groupId/artifactId/version//classifier
mvn:groupId/artifactId//packaging/classifier
mvn:groupId/artifactId//packaging
```

15.3.4. 버전 범위 지정

Mvn URL에 version 값을 지정하는 경우 간단한 버전 번호 대신 버전 범위(표준 Maven 버전 범위 구문을 사용하여)를 지정할 수 있습니다. 대괄호[및]-를 사용하여 포함 범위를 나타냅니다(및)- 독점 범위를 나타냅니다. 예를 들어 범위 [1.0.4,2.0) 는 1.0.4 Cryostat v < 2.0을 충족하는 모든 버전 v 와 일치합니다. 다음과 같이 Mvn URL에서 이 버전 범위를 사용할 수 있습니다.

```
mvn:org.fusesource.example/bundle-demo/[1.0.4,2.0)
```

15.3.5. Mvn URL 처리기 구성

Mvn URL을 처음 사용하기 전에 다음과 같이 Mvn URL 처리기 설정을 사용자 지정해야 할 수 있습니다.

1. 15.3.6절. “Mvn URL 설정 확인”.
2. 15.3.7절. “구성 파일 편집”.
3. 15.3.8절. “로컬 리포지토리의 위치 사용자 지정”.

15.3.6. Mvn URL 설정 확인

Mvn URL 핸들러는 로컬 Maven 리포지토리에 대한 참조를 확인하고 원격 Maven 리포지토리 목록을 유지 관리합니다. Mvn URL을 확인할 때 처리기는 먼저 로컬 리포지토리를 검색한 다음 지정된 Maven Artifiact를 찾기 위해 원격 리포지토리를 검색합니다. Mvn URL을 해결하는 데 문제가 있는 경우 가장 먼저 수행해야 하는 작업은 처리기 설정을 확인하여 URL을 해결하기 위해 사용 중인 로컬 리포지토리와 원격 리포지토리를 확인하는 것입니다.

Mvn URL 설정을 확인하려면 콘솔에 다음 명령을 입력합니다.

```
JBossFuse:karaf@root> config:edit org.ops4j.pax.url.mvn
JBossFuse:karaf@root> config:proplist
```

config:edit 명령은 config 유틸리티의 초점을 org.ops4j.pax.url.mvn 영구 ID에 속하는 속성으로 전환합니다. config:proplist 명령은 현재 영구 ID에 대한 모든 속성 설정을 출력합니다. org.ops4j.pax.url.mvn 에 중점을 두고 다음과 유사한 목록이 표시됩니다.

```
org.ops4j.pax.url.mvn.defaultRepositories = file:/path/to/JBossFuse/jboss-fuse-7.11.1.fuse-7_11_1-00013-redhat-00003/system@snapshots@id=karaf.system,file:/home/userid/.m2/repository@snapshots@id=local,file:
```

```
path/to/JBossFuse/jboss-fuse-7.11.1.fuse-7_11_1-00013-redhat-00003/local-
repo@snapshots@id=karaf.local-repo,file:/path/to/JBossFuse/jboss-fuse-7.11.1.fuse-7_11_1-00013-
redhat-00003/system@snapshots@id=child.karaf.system
  org.ops4j.pax.url.mvn.globalChecksumPolicy = warn
  org.ops4j.pax.url.mvn.globalUpdatePolicy = daily
  org.ops4j.pax.url.mvn.localRepository = /path/to/JBossFuse/jboss-fuse-7.11.1.fuse-7_11_1-00013-
redhat-00003/data/repository
  org.ops4j.pax.url.mvn.repositories = http://repo1.maven.org/maven2@id=maven.central.repo,
https://maven.repository.redhat.com/ga@id=redhat.ga.repo,
https://maven.repository.redhat.com/earlyaccess/all@id=redhat.ea.repo,
https://repository.jboss.org/nexus/content/groups/ea@id=fuseearlyaccess
  org.ops4j.pax.url.mvn.settings = /path/to/jboss-fuse-7.11.1.fuse-7_11_1-00013-redhat-
00003/etc/maven-settings.xml
  org.ops4j.pax.url.mvn.useFallbackRepositories = false
  service.pid = org.ops4j.pax.url.mvn
```

where the localRepository setting shows the local repository location currently used by the handler and the repositories setting shows the remote repository list currently used by the handler.

15.3.7. 구성 파일 편집

***Mvn URL** 처리기의 속성 설정을 사용자 지정하려면 다음 구성 파일을 편집합니다.*

```
InstallDir/etc/org.ops4j.pax.url.mvn.cfg
```

*이 파일의 설정을 사용하면 로컬 **Maven** 리포지토리의 위치를 명시적으로 지정하고 **Maven** 리포지토리, **Maven** 프록시 서버 설정 등을 제거할 수 있습니다. 이러한 설정에 대한 자세한 내용은 설정 파일의 주석을 참조하십시오.*

15.3.8. 로컬 리포지토리의 위치 사용자 지정

*특히 로컬 **Maven** 리포지토리가 기본이 아닌 위치에 있는 경우 로컬로 빌드하는 **Maven** 아티팩트에 액세스하려면 명시적으로 구성해야 할 수 있습니다. **org.ops4j.pax.url.mvn.cfg** 구성 파일에서 **org.ops4j.pax.url.mvn.localRepository** 속성의 주석을 제거하고 로컬 **Maven** 리포지토리의 위치로 설정합니다. 예를 들면 다음과 같습니다.*

```
# Path to the local maven repository which is used to avoid downloading
# artifacts when they already exist locally.
# The value of this property will be extracted from the settings.xml file
# above, or defaulted to:
#     System.getProperty( "user.home" ) + "/.m2/repository"
#
org.ops4j.pax.url.mvn.localRepository=file:E:/Data/.m2/repository
```

15.3.9. reference

mvn URL 구문에 대한 자세한 내용은 원래 Pax URL Mvn Protocol 설명서를 참조하십시오.

15.4. 래핑 URL 핸들러

15.4.1. 개요

번들로 패키지되지 않은 JAR 파일을 참조해야 하는 경우 Wrap URL 처리기를 사용하여 동적으로 변환할 수 있습니다. Wrap URL 핸들러의 구현은 Peter Krien의 오픈 소스 Bnd 유틸리티를 기반으로 합니다.

15.4.2. 구문

Wrap URL 구문에는 다음과 같은 구문이 있습니다.

```
wrap:locationURL[,instructionsURL][$instructions]
```

locationURL 은 JAR을 찾는 모든 URL일 수 있습니다(참조된 JAR이 번들로 포맷 되지 않음). 선택적 instructionsURL 은 번들 변환 수행 방법을 지정하는 Bnd 속성 파일을 참조합니다. 선택적 명령 은 번들 변환 수행 방법을 지정하는 Bnd 속성의 앰퍼샌드 및 delimited 목록입니다.

15.4.3. 기본 지침

대부분의 경우 기본 Bnd 명령은 API JAR 파일을 래핑하는 데 적합합니다. 기본적으로 Wrap은 표 15.1. “JAR 트래핑을 위한 기본 지침” 와 같이 JAR의 META-INF/Manifest.mf 파일에 매니페스트 헤더를 추가합니다.

표 15.1. JAR 트래핑을 위한 기본 지침

매니페스트 헤더	기본값
Import-Package	***;resolution:=optional**
export-Package	래핑된 JAR의 모든 패키지.
Bundle-SymbolicName	설정된 **[a-zA-Z0-9_-]** 에 없는 JAR 파일의 이름은 밑줄 _ 로 대체됩니다.

15.4.4. 예

다음 **Wrap URL**은 **Maven** 리포지토리에서 **commons-logging JAR**의 버전 **1.1**을 찾고 기본 **Bnd** 속성을 사용하여 **OSGi** 번들로 변환합니다.

```
wrap:mvn:commons-logging/commons-logging/1.1
```

다음 **Wrap URL**은 파일의 **Bnd** 속성을 사용합니다. **E:\Data\Examples\commons-logging-1.1.bnd:**

```
wrap:mvn:commons-logging/commons-logging/1.1,file:E:/Data/Examples/commons-logging-1.1.bnd
```

다음 **Wrap URL**은 **Bundle-SymbolicName** 속성 및 **Bundle-Version** 속성을 명시적으로 지정합니다.

```
wrap:mvn:commons-logging/commons-logging/1.1$Bundle-SymbolicName=apache-comm-
log&Bundle-Version=1.1
```

이전 **URL**을 명령줄 인수로 사용하는 경우 다음과 같이 명령줄에서 처리하지 못하도록 달러 기호 **\$**를 이스케이프해야 할 수 있습니다.

```
wrap:mvn:commons-logging/commons-logging/1.1\$Bundle-SymbolicName=apache-comm-
log&Bundle-Version=1.1
```

15.4.5. reference

랩 **URL** 처리기에 대한 자세한 내용은 다음 참조를 참조하십시오.

- **Bnd** 툴 문서**: Bnd** 속성 및 **Bnd** 명령 파일에 대한 자세한 내용은 .
- 원래 **Pax URL Wrap Protocol** 설명서

15.5. 충돌 URL 핸들러

15.5.1. 개요

OSGi 컨테이너에 **WAR** 파일을 배포해야 하는 경우 여기에 설명된 대로 **WAR URL** 앞에 **war:** 를 추가하여 **requisite** 매니페스트 헤더를 **WAR** 파일에 자동으로 추가할 수 있습니다.

15.5.2. 구문

WAR URL은 다음 구문 중 하나를 사용하여 지정됩니다.

```
war:warURL
warref:instructionsURL
```

선전 체계를 사용하는 첫 번째 구문은 기본 지침을 사용하여 번들로 변환되는 WAR 파일을 지정합니다. warURL 은 WAR 파일을 찾는 모든 URL일 수 있습니다.

warref 스키마를 사용하는 두 번째 구문은 변환 명령(이 처리기에 고유한 일부 명령 포함)을 포함하는 Bnd 속성 파일인 instructionsURL 을 지정합니다. 이 구문에서 참조된 WAR 파일의 위치는 URL에 명시적으로 표시되지 않습니다. WAR 파일은 속성 파일의 (필수) WAR-URL 속성으로 대신 지정됩니다.

15.5.3. WAR별 속성/구성

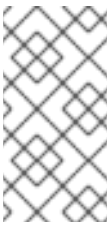
.bnd 명령 파일의 일부 속성은 다음과 같이 war URL 처리기에 고유합니다.

WAR-URL

(필수) 번들로 변환될 war 파일의 위치를 지정합니다.

Web-ContextPath

웹 컨테이너 내부에 배포된 후 이 웹 애플리케이션에 액세스하는 데 사용되는 URL 경로 조각을 지정합니다.



참고

이전 버전의 PAX Web은 이제 더 이상 사용되지 않는 Webapp-Context 속성을 사용했습니다.

15.5.4. 기본 지침

기본적으로 war URL 처리기는 표 15.2. “WAR 파일 래퍼에 대한 기본 지침” 에 표시된 대로 WAR의 META-INF/Manifest.mf 파일에 매니페스트 헤더를 추가합니다.

표 15.2. WAR 파일 래퍼에 대한 기본 지침

매니페스트 헤더	기본값
Import-Package	**javax.,org.xml.,org.w3c.***
export-Package	패키지를 내보내지 않습니다.
Bundle-SymbolicName	**[a-zA-Z0-9_-\.]** 세트에 없는 모든 문자가 마침표로 교체되는 WAR 파일의 이름입니다 **.**
Web-ContextPath	기본값이 없습니다. 그러나 *WAR Extender* 는 기본적으로 **Bundle-SymbolicName** 값을 사용합니다.
Bundle-ClassPath	명시적으로 지정된 클래스 경로 항목 외에도 다음 항목이 자동으로 추가됩니다. • **.** • **webpage-INF/classes** • **WEB-INF/lib** 디렉토리의 모든 JAR.

15.5.5. 예

다음 war URL은 Maven 리포지토리에서 wicket-examples WAR의 버전 1.4.7을 찾고 기본 지침을 사용하여 OSGi 번들로 변환합니다.

```
war:mvn:org.apache.wicket/wicket-examples/1.4.7/war
```

다음 Wrap URL은 Web-ContextPath 를 명시적으로 지정합니다.

```
war:mvn:org.apache.wicket/wicket-examples/1.4.7/war?Web-ContextPath=wicket
```

다음 war URL은 wicket-examples-1.4.7.bnd 파일의 WAR-URL 속성에서 참조하는 WAR 파일을 변환한 다음 .bnd 파일의 다른 지침을 사용하여 WAR를 OSGi 번들로 변환합니다.

```
warref:file:E:/Data/Examples/wicket-examples-1.4.7.bnd
```

15.5.6. reference

war URL 구문에 대한 자세한 내용은 원래 Pax URL war Protocol 설명서 를 참조하십시오.

II 부. 사용자 가이드

*이 부분에는 **Red Hat Fuse**의 **Apache Karaf**에 대한 구성 및 준비 정보가 포함되어 있습니다.*

16장. APACHE KARAF 사용자 배포 가이드 소개

초록

Apache Karaf 가이드에 배포 가이드의 이 사용자 가이드 섹션을 사용하기 전에 **Apache Karaf** 에 설치되는 지침에 따라 최신 버전의 **Red Hat Fuse**를 설치해야 합니다.

16.1. FUSE 구성 소개

OSGi 구성 관리 서비스는 배포된 서비스에 대한 구성 정보를 지정하고 서비스가 활성 상태일 때 해당 데이터를 수신하도록 합니다.

16.2. OSGI 구성

구성은 **FUSE_HOME/etc** 디렉터리의 **.cfg** 파일에서 읽은 이름-값 쌍 목록입니다. 파일은 **Java** 속성 파일 형식을 사용하여 해석됩니다. 파일 이름은 구성할 서비스의 영구 식별자(**PID**)에 매핑됩니다. **OSGi**에서 **PID**는 컨테이너를 다시 시작할 때마다 서비스를 식별하는 데 사용됩니다.

16.3. 구성 파일

다음 파일을 사용하여 **Red Hat Fuse** 런타임을 구성할 수 있습니다.

표 **16.1. Fuse** 구성 파일

파일 이름	설명
config.properties	컨테이너의 기본 구성 파일입니다.
custom.properties	컨테이너의 사용자 지정 속성에 대한 기본 구성 파일입니다.
keys.properties	SSH 키 기반 프로토콜을 사용하여 Fuse 런타임에 액세스할 수 있는 사용자를 나열합니다. 파일의 내용은 ***username=publicKey,role***형식을 사용합니다.
org.apache.karaf.features.repos.cfg	기능 리포지토리 URL은 다음과 같습니다.
org.apache.karaf.features.cfg	등록할 기능 리포지토리 목록과 Fuse가 처음 시작될 때 설치할 기능 리포지토리 목록을 구성합니다.

파일 이름	설명
org.apache.karaf.jaas.cfg	Karaf JAAS 로그인 모듈에 대한 옵션을 구성합니다. 주로 암호화된 암호를 구성하는 데 사용됩니다(기본적으로 비활성화됨).
org.apache.karaf.log.cfg	**로그** 콘솔 명령의 출력을 구성합니다.
org.apache.karaf.management.cfg	Cryostat 시스템을 구성합니다.
org.apache.karaf.shell.cfg	원격 콘솔의 속성을 구성합니다.
org.ops4j.pax.logging.cfg	로깅 시스템을 구성합니다.
org.ops4j.pax.transx.tm.narayana.cfg	Narayana 트랜잭션 관리자 구성
org.ops4j.pax.url.mvn.cfg	추가 URL 확인자를 구성합니다.
org.ops4j.pax.web.cfg	기본 Cryostat 컨테이너(웹 서버)를 구성합니다. Red Hat Fuse Apache CXF 보안 가이드에서 Cryostat HTTP 서버 보안을 참조하십시오.
startup.properties	컨테이너에서 시작되는 번들과 해당 시작 수준을 지정합니다. 항목은 ***bundle=start-level*** 형식을 사용합니다.
system.properties	Java 시스템 속성을 지정합니다. 이 파일에 설정된 모든 속성은 런타임에 **System.getProperties()** 를 사용하여 사용할 수 있습니다.
users.properties	Fuse 런타임에 원격으로 또는 웹 콘솔을 통해 액세스할 수 있는 사용자를 나열합니다. 파일의 내용은 ***username=password*** 형식인***role*을 사용합니다**.
SetEnv 또는 **setenv.rate**	이 파일은 **/bin** 디렉터리에 있습니다. JVM 옵션을 설정하는 데 사용됩니다. 파일의 내용은 **JAVA_MIN_MEM=*512M*** 형식을 사용합니다. 여기서 *512M* 은 Java 메모리의 최소 크기입니다. 자세한 내용은 16.6절. "Java 옵션 설정"를 참조하십시오.

16.4. 고급 CRYOSTAT 구성

16.4.1. IO 구성

PAXWEB-1255부터 리스너가 사용하는 XNIO 작업자 및 버퍼 풀의 구성을 변경할 수 있습니다. undertow.xml 템플릿에는 일부 IO 관련 매개변수의 기본값을 지정하는 섹션이 있습니다.

```
<!-- Only "default" worker and buffer-pool are supported and can be used to override the default
values used by all listeners
    buffer-pool:
     - buffer-size defaults to:
        - when < 64MB of Xmx: 512
        - when < 128MB of Xmx: 1024
        - when >= 128MB of Xmx: 16K - 20
     - direct-buffers defaults to:
        - when < 64MB of Xmx: false
        - when >= 64MB of Xmx: true
    worker:
     - io-threads defaults to Math.max(Runtime.getRuntime().availableProcessors(), 2);
     - task-core-threads and task-max-threads default to io-threads * 8
-->

<!--
<subsystem xmlns="urn:jboss:domain:io:3.0">
    <buffer-pool name="default" buffer-size="16364" direct-buffers="true" />
    <worker name="default" io-threads="8" task-core-threads="64" task-max-threads="64" task-
keepalive="60000" />
</subsystem>
-->
```

*다음 **buffer-pool** 매개변수를 지정할 수 있습니다.*

buffer-size

IO 작업에 사용되는 버퍼의 크기를 지정합니다. 지정하지 않으면 사용 가능한 메모리에 따라 크기가 계산됩니다.

direct-buffers

java.nio.ByteBuffer#allocateDirect 또는 java.nio.ByteBuffer#allocate를 사용해야 하는지 여부를 결정합니다.

다음 작업자 매개변수를 지정할 수 있습니다.

io-threads

작업자에 대해 생성할 I/O 스레드 수입니다. 지정하지 않으면 스레드 수가 CPU 수 × 2로 설정됩니다.

task-core-threads

코어 작업 스레드 풀의 스레드 수입니다.

task-max-threads

작업자 작업 스레드 풀의 최대 스레드 수입니다. 지정하지 않으면 최대 스레드 수가 **CPU** 수 **× 16** 으로 설정됩니다.

16.4.2. 작업자 IO 구성

Cryostat 스레드 풀과 해당 이름은 서비스별 또는 번들 기준으로 구성할 수 있으므로 **Hawtio** 콘솔에서 모니터링하고 디버깅을 보다 효율적으로 수행할 수 있습니다.

번들 블루프린트 구성 파일(일반적으로 **Maven** 프로젝트의 **src/main/resources/OSGI-INF/blueprint** 디렉터리에 저장됨)에서 다음 예와 같이 **workerIOName** 및 **ThreadPool**을 구성할 수 있습니다.

예 16.1. httpu:engine-factory 요소가 workerIOName 및 ThreadPool 구성

```
<httpu:engine-factory>
   <httpu:engine port="9001">
      <httpu:threadingParameters minThreads="99" maxThreads="777" workerIOThreads="8"
workerIOName="WorkerIOTest"/>
   </httpu:engine>
</httpu:engine-factory>
```

다음의 **threadingParameters** 를 지정할 수 있습니다.

minThreads

작업자 작업 스레드 풀에 대한 **"코어"** 스레드 수를 지정합니다. 일반적으로 이는 **CPU** 코어당 최소 **10**개 이상 높은 수준이어야 합니다.

maxThreads

작업자 작업 스레드 풀에 대한 최대 스레드 수를 지정합니다.

다음 작업자 매개변수를 지정할 수 있습니다.

workerIOThreads

작업자에 대해 생성할 **I/O** 스레드 수를 지정합니다. 지정하지 않으면 기본값이 선택됩니다. **CPU** 코어당 하나의 **IO** 스레드가 적절한 기본값입니다.

workerIOName

작업자의 이름을 지정합니다. 지정하지 않으면 기본 **"XNIO-1"**이 선택됩니다.

16.5. 구성 파일 이름 지정 규칙

구성 파일에 대한 파일 이름 지정 규칙은 구성이 **OSGi Managed Service** 또는 **OSGi Managed Service** 팩토리용인지에 따라 다릅니다.

OSGi Managed 서비스의 구성 파일은 다음 명명 규칙을 따릅니다.

```
<PID>.cfg
```

여기서 **<PID >**는 **OSGi** 관리 서비스의 영구 **ID** 입니다(**OSGi** 구성 관리자 사양에 정의됨). 영구 **ID**는 일반적으로 점으로 구분되어 있습니다(예: **org.ops4j.pax.web**).

OSGi Managed Service Cryostat의 구성 파일은 다음과 같은 명명 규칙을 따릅니다.

```
<PID>-<InstanceID>.cfg
```

여기서 **& lt;PID >**는 **OSGi Managed Service Cryostat**의 영구 **ID** 입니다. 관리형 서비스 팩토리의 **< PID >**의 경우 하이픈 뒤에 임의의 인스턴스 **ID**인 **< InstanceID>**를 추가할 수 있습니다. 그러면 관리형 서비스 팩토리에서 찾은 각 **< InstanceID>**에 대한 고유한 서비스 인스턴스를 생성합니다.

16.6. JAVA 옵션 설정

Java 옵션은 **Linux**의 **/bin/setenv** 파일을 사용하거나 **Windows**용 **bin/setenv.rate** 파일을 사용하여 설정할 수 있습니다. 이 파일을 사용하여 **Java** 옵션 그룹 **JAVA_MIN_MEM, JAVA_MAX_MEM, JAVA_PERM_MEM, JAVA_MAX_PERM_MEM**을 직접 설정합니다. 다른 **Java** 옵션은 **EXTRA_JAVA_OPTS** 변수를 사용하여 설정할 수 있습니다.

예를 들어 **JVM**에 최소 메모리를 할당하려면 다음을 사용합니다.

```
JAVA_MIN_MEM=512M # Minimum memory for the JVM
```

```
To set a Java option other than the direct options, use
```

```
EXTRA_JAVA_OPTS="Java option"
```

```
For example,
```

```
EXTRA_JAVA_OPTS="-XX:+UseG1GC"
```

16.7. 구성 콘솔 명령

Fuse 7.11의 구성을 변경하거나 확인하는 데 사용할 수 있는 여러 콘솔 명령이 있습니다.

config: 명령에 대한 자세한 내용은 Apache Karaf 콘솔 참조 의 구성 섹션을 참조하십시오.

16.8. JMX CONFIGMBEAN

Cryostat 계층에서는 구성 관리 전용입니다.

ConfigMBean 오브젝트 이름은 org.apache.karaf:type=config,name=*' 입니다.

14.1.2.1. 속성

구성에는 모든 구성 PID 목록이 포함되어 있습니다.

14.1.2.2. 작업

표 16.2. Cryostat Cryostat 작업

작업 이름	설명
listProperties(pid)	구성 pid의 속성 목록(property=value formatted)을 반환합니다.
deleteProperty(pid, property)	구성 pid에서 속성을 삭제합니다.Deletes the property from the configuration pid.
appendProperty(pid, property, value)	구성 pid의 속성 값 끝에 값을 추가합니다.
setProperty(pid, property, value)	구성 pid의 속성 값에 대한 값을 설정합니다.
delete(pid)	pid로 식별된 구성을 삭제합니다.
create(pid)	pid를 사용하여 빈(속성 없이) 구성을 생성합니다.
update(pid, properties)	제공된 속성 맵으로 pid로 식별되는 구성을 업데이트합니다.

16.9. 콘솔 사용

16.9.1. 사용 가능한 명령

콘솔에서 사용 가능한 명령 목록을 보려면 도움말 을 사용할 수 있습니다.

```
karaf@root()> help
bundle                       Enter the subshell
bundle:capabilities              Displays OSGi capabilities of a given bundles.
bundle:classes                   Displays a list of classes/resources contained in the bundle
bundle:diag                    Displays diagnostic information why a bundle is not Active
bundle:dynamic-import            Enables/disables dynamic-import for a given bundle.
bundle:find-class              Locates a specified class in any deployed bundle
bundle:headers                   Displays OSGi headers of a given bundles.
bundle:id                      Gets the bundle ID.
...
```

간단한 설명과 함께 모든 명령 목록이 있습니다.

tab 키를 사용하여 모든 명령의 빠른 목록을 가져올 수 있습니다.

```
karaf@root()> Display all 294 possibilities? (y or n)
...
```

16.9.2. 하위 쉘 및 완료 모드

명령에는 범위와 이름이 있습니다. 예를 들어, 명령 **feature:list** 에는 범위가 있으며 **list as name**이 있습니다.

카프는 범위별로 명령을 "그룹"으로 합니다. 각 범위는 하위 쉘을 형성합니다.

정규화된 이름**(scope:name)**을 사용하여 명령을 직접 실행할 수 있습니다.

```
karaf@root()> feature:list
...
```

또는 하위 쉘을 입력하고 명령 컨텍스트를 하위 쉘로 입력합니다.

```
karaf@root()> feature
karaf@root(feature)> list
```

하위 쉘 이름(여기 기능)을 입력하여 직접 하위 쉘을 입력할 수 있습니다. 하위 쉘에서 다른 쉘로 직접 **"switch"**할 수 있습니다.

```
karaf@root()> feature
karaf@root(feature)> bundle
karaf@root(bundle)>
```

프롬프트에 () 사이의 현재 하위 쉘이 표시됩니다.

exit 명령은 상위 하위 쉘로 이동합니다.

```
karaf@root()> feature
karaf@root(feature)> exit
karaf@root()>
```

완료 모드는 **tab** 키의 동작과 **help** 명령을 정의합니다.

다음 세 가지 모드를 사용할 수 있습니다.

- **GLOBAL**
- **FIRST**
- **SUBSHELL**

etc/org.apache.karaf.shell.cfg 파일의 **completionMode** 속성을 사용하여 기본 완료 모드를 정의할 수 있습니다. 기본적으로 다음을 수행할 수 있습니다.

```
completionMode = GLOBAL
```

shell:completion 명령을 사용하여 "오전기 중에"(카운트 쉘 콘솔을 사용하는 동안) 완료 모드를 변경할 수도 있습니다.

```
karaf@root()> shell:completion
GLOBAL
karaf@root()> shell:completion FIRST
karaf@root()> shell:completion
FIRST
```

shell:completion 은 사용된 현재 완료 모드에 대해 알려줍니다. 원하는 새 완료 모드를 제공할 수도 있습니다.

GLOBAL 완료 모드는 **Karaf 4.0.0** (대부분의 전환 목적으로)의 기본 완료 모드입니다.

GLOBAL 모드는 실제로 하위 쉘을 사용하지 않습니다. 이전 **Karaf** 버전과 동일한 동작입니다.

tab 키를 입력하면 하위 쉘이 무엇이든 완료 시 모든 명령 및 모든 별칭이 표시됩니다.

```
karaf@root()> <TAB>
karaf@root()> Display all 273 possibilities? (y or n)
...
karaf@root()> feature
karaf@root(feature)> <TAB>
karaf@root(feature)> Display all 273 possibilities? (y or n)
```

첫 번째 완료 모드는 **GLOBAL** 완료 모드의 대안입니다.

루트 수준 하위 쉘에 **tab** 키를 입력하면 완료 시 모든 하위 쉘의 명령과 별칭(**GLOBAL** 모드)이 표시됩니다. 그러나 하위 쉘에 있는 경우 **tab** 키를 입력하면 완료에 현재 하위 쉘의 명령만 표시됩니다.

```
karaf@root()> shell:completion FIRST
karaf@root()> <TAB>
karaf@root()> Display all 273 possibilities? (y or n)
...
karaf@root()> feature
karaf@root(feature)> <TAB>
karaf@root(feature)>
info install list repo-add repo-list repo-remove uninstall version-list
karaf@root(feature)> exit
karaf@root()> log
karaf@root(log)> <TAB>
karaf@root(log)>
clear display exception-display get log set tail
```

SUBSHELL 완료 모드는 실제 하위 쉘 모드입니다.

루트 수준에서 **Tab** 키를 입력하면 완료에 하위 쉘 명령(하위 쉘로 이동)과 글로벌 별칭이 표시됩니다. 하위 쉘에 있으면 **TAB** 키를 입력하면 완료에 현재 하위 쉘의 명령이 표시됩니다.

```
karaf@root()> shell:completion SUBSHELL
karaf@root()> <TAB>
karaf@root()>
* bundle cl config dev feature help instance jaas kar la ld lde log log:list man package region service
shell ssh system
karaf@root()> bundle
karaf@root(bundle)> <TAB>
karaf@root(bundle)>
capabilities classes diag dynamic-import find-class headers info install list refresh requirements
resolve restart services start start-level stop
uninstall update watch
karaf@root(bundle)> exit
karaf@root()> camel
karaf@root(camel)> <TAB>
karaf@root(camel)>
backlog-tracer-dump backlog-tracer-info backlog-tracer-start backlog-tracer-stop context-info
context-list context-start context-stop endpoint-list route-info route-list route-profile route-reset-stats
route-resume route-show route-start route-stop route-suspend
```

16.9.3. UNIX와 같은 환경

Karaf 콘솔은 환경과 같은 전체 **Unix**를 제공합니다.

16.9.3.1. 도움말 또는 사람

이미 사용 가능한 모든 명령을 표시하는 **help** 명령의 사용법을 살펴보았습니다.

그러나 **help** 명령을 사용하여 명령 또는 **help** 명령의 별칭인 **man** 명령에 대한 세부 정보를 가져올 수도 있습니다. 명령에 **--help** 옵션을 사용하여 다른 양식을 사용하여 명령 도움말을 가져올 수도 있습니다.

따라서 이러한 명령

```
karaf@root()> help feature:list
karaf@root()> man feature:list
karaf@root()> feature:list --help
```

모두 동일한 도움말 출력을 생성합니다.

```
DESCRIPTION
        feature:list

        Lists all existing features available from the defined repositories.

SYNTAX
        feature:list [options]

OPTIONS
        --help
                Display this help message
        -o, --ordered
                Display a list using alphabetical order
        -i, --installed
                Display a list of all installed features only
        --no-format
                Disable table rendered output
```

16.9.3.2. 완료

***Tab** 키를 입력하면 **Karaf**가 완료하려고 합니다.*

- *하위 쉘*
- *명령*
- *별칭*
- *명령 인수*
- *명령 옵션*

16.9.3.3. 별칭

별칭은 지정된 명령과 연결된 다른 이름입니다.

shell:alias 명령은 새 별칭을 생성합니다. 예를 들어 실제 **feature:list -i** 명령에 대한 **list- installed-features** 별칭을 생성하려면 다음을 수행할 수 있습니다.

```
karaf@root()> alias "list-features-installed = { feature:list -i }"
karaf@root()> list-features-installed
Name       | Version | Required | State   | Repository     | Description
--------------------------------------------------------------------------------------------------------------------------------
---
feature    | 4.0.0   | x        | Started | standard-4.0.0 | Features Support
shell      | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Shell
deployer   | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Deployer
bundle     | 4.0.0   | x        | Started | standard-4.0.0 | Provide Bundle support
config     | 4.0.0   | x        | Started | standard-4.0.0 | Provide OSGi ConfigAdmin support
diagnostic | 4.0.0   | x        | Started | standard-4.0.0 | Provide Diagnostic support
instance   | 4.0.0   | x        | Started | standard-4.0.0 | Provide Instance support
jaas       | 4.0.0   | x        | Started | standard-4.0.0 | Provide JAAS support
log        | 4.0.0   | x        | Started | standard-4.0.0 | Provide Log support
package    | 4.0.0   | x        | Started | standard-4.0.0 | Package commands and mbeans
service    | 4.0.0   | x        | Started | standard-4.0.0 | Provide Service support
system     | 4.0.0   | x        | Started | standard-4.0.0 | Provide System support
kar        | 4.0.0   | x        | Started | standard-4.0.0 | Provide KAR (KARaf archive) support
ssh        | 4.0.0   | x        | Started | standard-4.0.0 | Provide a SSHd server on Karaf
management | 4.0.0   | x        | Started | standard-4.0.0 | Provide a JMX MBeanServer and a set of
MBeans in
```

로그인 시 **Apache Karaf** 콘솔은 별칭을 생성할 수 있는 **etc/shell.init.script** 파일을 읽습니다. **Unix** 의 **bashrc** 또는 프로필 파일과 유사합니다.

```
ld = { log:display $args } ;
lde = { log:exception-display $args } ;
la = { bundle:list -t 0 $args } ;
ls = { service:list $args } ;
cl = { config:list "(service.pid=$args)" } ;
halt = { system:shutdown -h -f $args } ;
help = { *:help $args | more } ;
man = { help $args } ;
log:list = { log:get ALL } ;
```

여기에서는 기본적으로 사용 가능한 별칭을 확인할 수 있습니다.

- **LD**는 로그를 표시하는 짧은 양식입니다(**log:display** 명령).
- **LDE** 는 예외를 표시하는 짧은 양식입니다(**log:exception-display** 명령).

- *La 는 모든 번들을 나열하는 짧은 양식입니다(bundle:list -t 0 명령)*
- *LS 는 모든 서비스를 나열하는 짧은 양식입니다(service:list 명령).*
- *CL 은 모든 구성을 나열하는 짧은 양식입니다(config:list 명령).*
- *halt 는 Apache Karaf를 종료하는 간단한 양식입니다(system:shutdown -h -f 명령).*
- *도움말은 도움말을 표시하는 간단한 양식입니다(*: help 명령까지)*
- *man 은 도움말과 동일합니다(help command)*
- *log:list 는 모든 로거 및 수준 (log:get ALL 명령)을 표시합니다.*

etc/shell.init.script 파일에서 고유한 별칭을 생성할 수 있습니다.

16.9.3.4. 키 바인딩

대부분의 Unix 환경에서와 마찬가지로 Karaf 콘솔은 몇 가지 주요 바인딩을 지원합니다.

- *명령 기록에서 탐색할 화살표 키*
- *CTRL-D to logout/shutdown Karaf*
- *CTRL-R: 이전에 실행된 명령을 검색*
- *CTRL-U를 사용하여 현재 행을 제거합니다.*

16.9.3.5. Pipe

한 명령의 출력을 입력으로 다른 명령에 파이프할 수 있습니다. | 문자를 사용하는 파이프입니다.

```
karaf@root()> feature:list |grep -i war
pax-war                  | 4.1.4                    |          | Uninstalled | org.ops4j.pax.web-4.1.4  | Provide
support of a full WebContainer
pax-war-tomcat           | 4.1.4                    |          | Uninstalled | org.ops4j.pax.web-4.1.4  |
war                      | 4.0.0                    |          | Uninstalled | standard-4.0.0           | Turn Karaf as
a full WebContainer
blueprint-web            | 4.0.0                    |          | Uninstalled | standard-4.0.0           | Provides
an OSGI-aware Servlet ContextListener fo
```

16.9.3.6. grep, more, find, …

Karaf 콘솔은 Unix 환경과 유사한 몇 가지 핵심 명령을 제공합니다.

- *shell:alias 는 기존 명령에 대한 별칭을 생성*
- *shell:cat 은 파일 또는 URL의 내용을 표시합니다.*
- *shell:clear 현재 콘솔 디스플레이를 지웁니다.*
- *shell:completion 이 표시되거나 현재 완료 모드를 변경합니다.*
- *shell:date 는 현재 날짜를 표시합니다 (선택적으로 형식을 사용)*
- *shell:each 는 인수 목록에서 호출을 실행합니다.*
- *shell:echo echoes 및 print arguments to stdout*
- *shell:edit 는 현재 파일 또는 URL에서 텍스트 편집기를 호출합니다.*
- *shell:env 가 쉘 세션 변수 값을 표시하거나 설정합니다.*

- *shell:exec 가 시스템 명령을 실행합니다.*
- *shell:grep 은 지정된 패턴과 일치하는 행을 출력합니다.*
- *shell:head 는 입력의 첫 번째 행을 표시합니다.*
- *shell:history 는 명령 기록을 출력합니다.*
- *shell:if 를 사용하여 스크립트에서 조건(if, then, else blocks)을 사용할 수 있습니다.*
- *shell:info 는 현재 Karaf 인스턴스에 대한 다양한 정보를 출력합니다.*
- *shell:java 는 Java 애플리케이션을 실행합니다.*
- *shell:less file pager*
- *shell:logout 의 현재 세션과 쉘의 연결을 끊습니다.*
- *shell:more 는 파일 페이지입니다.*
- *shell:new 에서 새 Java 오브젝트 생성*
- *shell: arguments 형식 및 출력*
- *shell:sleep s for a bit then wakes up*
- *shell:sort 쓰기는 모든 파일의 연결을 stdout에 정렬*

- **shell:source** 는 스크립트에 포함된 명령을 실행합니다.
- **shell:stack-traces-print** 는 명령 실행 시 예외가 발생할 때 콘솔의 전체 스택 추적을 출력합니다.
- **shell:tac** 은 **STDIN**을 캡처하고 문자열로 반환합니다.
- **shell:tail** 이 입력의 마지막 행을 표시합니다.
- **shell:threads** 현재 스레드를 출력합니다.
- **shell:watch** 는 주기적으로 명령을 실행하고 출력을 새로 고칩니다.
- **shell:wc** 는 각 파일에 대한 줄 바꿈, 단어 및 바이트 수를 출력합니다.
- **shell: condition**이 **True**인 동안 **loop**

정규화된 명령 이름을 사용할 필요는 없으며 고유 한 명령 이름을 직접 사용할 수 있습니다. 따라서 **'shell:head'** 대신 **'head'**를 사용할 수 있습니다.

또한 **help** 명령 또는 **-- help** 옵션을 사용하여 이러한 명령의 세부 정보와 모든 옵션을 찾을 수 있습니다.

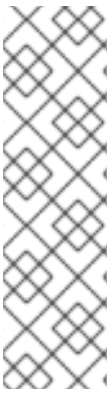
16.9.3.7. 스크립팅

Apache Karaf Console은 **Unix**의 **bash** 또는 **csh**와 유사한 전체 스크립팅 언어를 지원합니다.

각 **(shell:each)** 명령은 목록에서 반복할 수 있습니다.

```
karaf@root()> list = [1 2 3]; each ($list) { echo $it }
1
2
```

```
3
```



참고

*셸을 사용하여 동일한 루프를 작성할 수 있습니다. **while** 명령:*

```
karaf@root()> a = 0 ; while { %((a+=1) <= 3) } { echo $a }
1
2
3
```

이전 예에서와 같이 목록을 직접 만들거나 일부 명령도 목록을 반환할 수 있습니다.

*콘솔에서 **$list** 를 사용하여 액세스할 수 있는 이름 목록을 사용하여 "세션" 변수를 생성했습니다.*

***$it** 변수는 현재 오브젝트(**list**에서 현재 반복된 값)에 해당하는 암시적 변수입니다.*

***[]** 로 목록을 생성하면 **Apache Karaf** 콘솔에서 **Java Cryostat**를 생성합니다. 즉, **Cryostat** 개체(예: **get** 또는 **size for instance**)에서 사용할 수 있는 방법을 사용할 수 있습니다.*

```
karaf@root()> list = ["Hello" world]; echo ($list get 0) ($list get 1)
Hello world
```

*여기에서는 개체의 메서드를 직접 호출하면 (오브젝트 메서드 인수) 를 사용할 수 있습니다. 여기에서 **($list get 0)** 는 **$list.get(0)** 을 의미합니다. 여기서 **$list** 는 **Cryostat**입니다.*

클래스 표기법은 오브젝트에 대한 세부 정보를 표시합니다.

```
karaf@root()> $list class
...
ProtectionDomain     ProtectionDomain  null
 null
 <no principals>
 java.security.Permissions@6521c24e (
 ("java.security.AllPermission" "<all permissions>" "<all actions>")
)
```

```
Signers             null
SimpleName          ArrayList
TypeParameters      [E]
```

변수를 지정된 유형으로 "캐스팅"할 수 있습니다.

```
karaf@root()> ("hello world" toCharArray)
[h, e, l, l, o,  , w, o, r, l, d]
```

실패하면 캐스팅 예외가 표시됩니다.

```
karaf@root()> ("hello world" toCharArray)[0]
Error executing command: [C cannot be cast to [Ljava.lang.Object;
```

shell:source 명령을 사용하여 스크립트를 "콜"할 수 있습니다.

```
karaf@root> shell:source script.txt
True!
```

여기서 **script.txt** 에는 다음이 포함됩니다.

```
foo = "foo"
if { $foo equals "foo" } {
  echo "True!"
}
```

참고

스크립트를 작성할 때 공백이 중요합니다. 예를 들어 다음 스크립트는 올바르지 않습니다.

```
if{ $foo equals "foo" } ...
```

다음과 같이 실패합니다.

```
karaf@root> shell:source script.txt
Error executing command: Cannot coerce echo "true!"() to any of []
```

if 문 뒤에 공백이 누락되어 있기 때문입니다.

별칭에 대해 etc/shell.init.script 파일에서 init 스크립트를 생성할 수 있습니다. 별칭을 사용하여 스크립트 이름을 지정할 수도 있습니다. 사실 별칭은 스크립트일 뿐입니다.

자세한 내용은 개발자 가이드의 스크립팅 섹션을 참조하십시오.

16.9.4. 보안

Apache Karaf 콘솔은 RBAC(역할 기반 액세스 제어) 보안 메커니즘을 지원합니다. 즉, 콘솔에 연결된 사용자에 따라 사용자의 그룹 및 역할에 따라 일부 명령을 실행하거나 인수에 허용되는 값을 제한할 수 있습니다.

콘솔 보안은 이 사용자 가이드의 보안 섹션에 자세히 설명되어 있습니다.

16.10. 프로비저닝

Apache Karaf는 Karaf 기능의 개념을 사용하여 애플리케이션 및 모듈 프로비저닝을 지원합니다.

16.10.1. 애플리케이션

애플리케이션을 프로비저닝하면 모든 모듈, 구성 및 전송 애플리케이션을 설치합니다.

16.10.2. OSGi

기본적으로 OSGi 애플리케이션 배포를 지원합니다.

OSGi 애플리케이션은 OSGi 번들 세트입니다. OSGi 번들은 일반 Cryostat 파일로, 추가 메타데이터가 MANIFEST에 있습니다.

OSGi에서 번들은 다른 번들에 따라 달라질 수 있습니다. 따라서 OSGi 애플리케이션을 배포하려면 대부분의 경우 애플리케이션에 필요한 많은 다른 번들을 먼저 배포해야 합니다.

따라서 먼저 이러한 번들을 찾아서 번들을 설치해야 합니다. 다시 말하지만 이러한 "종속성" 번들은 자체 종속성을 충족하기 위해 다른 번들이 필요할 수 있습니다.

일반적으로 애플리케이션에는 구성이 필요합니다(사용자 가이드의 [구성 섹션|구성] 참조). 따라서 애플리케이션을 시작하기 전에 종속성 번들을 추가하거나 구성을 생성해야 합니다.

우리가 볼 수 있듯이 애플리케이션 프로비저닝은 매우 길고 단단할 수 있습니다.

16.10.3. 기능 및 해결 방법

***Apache Karaf**는 애플리케이션을 프로비저닝할 수 있는 간단하고 유연한 방법을 제공합니다.*

***Apache Karaf**에서 애플리케이션 프로비저닝은 **Apache Karaf** "기능"입니다.*

기능은 다음과 같이 애플리케이션을 설명합니다.

- *이름*
- *버전*
- *선택적 설명(연장 설명 포함)*
- *번들 세트*
- *선택 사항으로 설정된 구성 또는 구성 파일*
- *필요한 경우 종속성 기능 세트*

*기능을 설치하면 **Apache Karaf**가 기능에 설명된 모든 리소스를 설치합니다. 즉, 기능에 설명된 모든 번들, 구성 및 종속성 기능을 자동으로 확인하고 설치합니다.*

*기능 확인자는 서비스 요구 사항을 확인하고 요구 사항과 일치하는 서비스를 제공하는 번들을 설치합니다. 기본 모드는 "새 스타일"에 대해서만 이 동작을 활성화합니다(기본적으로 스키마가 **1.3.0**) 이상인*

기능 **XML**이 있습니다. "기존 스타일" 기능 리포지토리(카르프 **2** 또는 **3**)에는 적용되지 않습니다.

serviceRequirements 속성을 사용하여 **etc/org.apache.karaf.features.cfg** 파일에서 서비스 요구 사항 적용 모드를 변경할 수 있습니다.

```
serviceRequirements=default
```

가능한 값은 다음과 같습니다.

- 비활성화: "오래된 스타일" 및 "새 스타일" 기능 리포지토리 모두에 대해 서비스 요구 사항이 완전히 무시됩니다.
- 기본값: **"old style"** 기능 리포지토리는 서비스 요구 사항이 무시되고 "새 스타일" 기능 리포지토리에 대해 활성화됩니다.
- 적용: "기존 스타일" 및 "새 스타일" 기능 리포지토리에 대해 서비스 요구 사항을 항상 확인합니다.

또한 기능은 요구 사항을 정의할 수도 있습니다. 이 경우 **Karaf**는 요구 사항을 충족하기 위해 기능을 제공하는 번들 또는 기능을 자동으로 추가할 수 있습니다.

기능에는 설치, 시작, 중지, 업데이트, 제거 등 전체 라이프사이클이 있습니다.

16.10.4. 리포지토리 기능

기능은 기능 **XML** 설명자에 설명되어 있습니다. 이 **XML** 파일에는 기능 집합에 대한 설명이 포함되어 있습니다.

기능 **XML** 설명자의 이름은 **"features repository"**입니다. 기능을 설치하려면 기능을 제공하는 기능 리포지토리를 등록해야 합니다(후에 설명된 대로 **feature:repo-add** 명령 또는 **FeatureMBean** 사용).

예를 들어 다음 **XML** 파일(또는 **"features** 리포지토리")은 **feature1** 및 **feature2** 기능을 설명합니다.

```
<features xmlns="http://karaf.apache.org/xmlns/features/v1.3.0">
```

```
 <feature name="feature1" version="1.0.0">
  <bundle>...</bundle>
  <bundle>...</bundle>
 </feature>
 <feature name="feature2" version="1.1.0">
  <feature>feature1</feature>
  <bundle>...</bundle>
 </feature>
</features>
```

XML 기능에는 스키마가 있습니다. 자세한 내용은 [Features XML Schema 섹션|provisioning-schema]을 참조하십시오. feature1 기능은 버전 1.0.0 에서 사용할 수 있으며 두 개의 번들을 포함합니다. < bundle/ > 요소에는 번들 아티팩트에 대한 URL이 포함되어 있습니다(자세한 내용은 [Artifacts 리포지토리 및 URL 섹션|urls] 참조). feature1 기능(feature:install 또는 later에 설명된 대로 FeatureMBean 사용)을 설치하는 경우 Apache Karaf는 설명된 두 개의 번들을 자동으로 설치합니다. feature2 기능은 버전 1.1.0 에서 사용할 수 있으며 feature1 기능과 번들에 대한 참조가 포함되어 있습니다. & lt;feature/& gt; 요소에는 기능 이름이 포함되어 있습니다. 특정 기능 버전은 <feature / > 요소의 version 속성을 사용하여 정의할 수 있습니다(<feature version="1.0.0">feature1</feature>). version 속성을 지정하지 않으면 Apache Karaf가 사용 가능한 최신 버전을 설치합니다. feature2 기능(후에 설명된 대로 feature:install 또는 FeatureMBean 사용)을 설치하는 경우 Apache Karaf는 feature1 (아직 설치되지 않은 경우) 및 번들을 자동으로 설치합니다.

기능 리포지토리는 기능 XML 파일에 URL을 사용하여 등록됩니다.

기능 상태는 Apache Karaf 캐시(KARAF_DATA 폴더)에 저장됩니다. Apache Karaf를 다시 시작할 수 있으며 이전에 설치한 기능은 다시 시작한 후 계속 설치되고 사용할 수 있습니다. 새로 다시 시작하거나 Apache Karaf 캐시를 삭제하면(KARAF_DATA 폴더 삭제) 이전에 등록된 리포지토리와 설치된 모든 기능이 손실됩니다. 기능 리포지토리를 등록하고 기능을 다시 설치해야 합니다. 이 동작을 방지하려면 기능을 부팅 기능으로 지정할 수 있습니다.

16.10.5. 부팅 기능

일부 기능을 부팅 기능으로 설명할 수 있습니다. feature:install 또는 FeatureMBean을 사용하여 이전에 설치하지 않은 경우에도 Apache Karaf에 의해 부팅 기능이 자동으로 설치됩니다.

Apache Karaf 기능 구성은 etc/org.apache.karaf.features.cfg 구성 파일에 있습니다.

이 구성 파일에는 부팅 기능을 정의하는 데 사용할 두 가지 속성이 포함되어 있습니다.

- *featuresRepositories 에는 기능 리포지토리(기능 XML) URL의 목록(콤마로 구분)이 포함되어 있습니다.*

- ***featuresBoot*** *에는 부팅 시 설치할 기능의 목록(마마로 구분)이 포함되어 있습니다.*

16.10.6. 기능 업그레이드

동일한 기능을 설치하여 릴리스를 업데이트할 수 있습니다(동일한 ***SNAPSHOT*** *버전 또는 다른 버전).*

기능 라이프사이클 덕분에 기능의 상태(시작, 중지 등)를 제어할 수 있습니다.

시뮬레이션을 사용하여 업데이트가 수행할 작업을 확인할 수도 있습니다.

16.10.7. 덮어쓰기

기능에 정의된 번들은 ***etc/overrides.properties*** *파일을 사용하여 재정의할 수 있습니다. 파일의 각 행은 하나의 덮어쓰기를 정의합니다. 구문은* ***<bundle-uri>[;range="[min,max"]*** *지정된 번들이 재정의된 번들의 버전 및 범위가 일치하는 경우, 동일한 심볼릭 이름으로 기능 정의의 모든 번들을 재정의합니다. 범위를 지정하지 않으면 마이크로 버전 수준에서의 호환성이 가정됩니다.*

예를 들어 재정의 ***mvn:org.ops4j.pax.logging/pax-logging-service/1.8.5****는* ***pax-logging-service 1.8.3****을 덮어쓰지만* ***1.8.6*** *또는* ***1.7.0****은 덮어씁니다.*

16.10.8. 기능 번들

16.10.8.1. 시작 수준

기본적으로 기능에 배포된 번들의 ***start-level****은* ***etc/config.properties*** *구성 파일에 정의된 값과 동일합니다.* ***karaf.startlevel.bundle*** *속성입니다.*

이 값은 기능 ***XML****에서* ***< bundle/ >*** *요소의* ***start-level*** *속성에 의해* ***"overrided"****일 수 있습니다.*

```
<feature name="my-project" version="1.0.0">
  <bundle start-level="80">mvn:com.mycompany.myproject/myproject-dao</bundle>
  <bundle start-level="85">mvn:com.mycompany.myproject/myproject-service</bundle>
</feature>
```

start-level *속성은* ***myproject-dao*** *번들을 사용하는 번들보다 먼저 시작하는 것을 보장합니다.*

*시작 수준을 사용하는 대신 더 나은 솔루션은 **OSGi** 프레임워크에 필요한 패키지 또는 서비스를 정의하여 종속 항목을 알리는 것입니다. 시작 수준을 설정하는 것보다 더 강력합니다.*

16.10.8.2. 시뮬레이션, 시작 및 중지

***feature:install** 명령에 **-t** 옵션을 사용하여 기능 설치를 시뮬레이션할 수 있습니다.*

번들을 시작하지 않고 설치할 수 있습니다. 기본적으로 기능의 번들이 자동으로 시작됩니다.

*기능은 번들을 자동으로 시작하지 않도록 지정할 수 있습니다(번들은 해결된 상태로 유지됨). 이를 위해 기능은 **< bundle/ >** 요소에서 **false**로 **start** 속성을 지정할 수 있습니다.*

```
<feature name="my-project" version="1.0.0">
  <bundle start-level="80" start="false">mvn:com.mycompany.myproject/myproject-dao</bundle>
  <bundle start-level="85" start="false">mvn:com.mycompany.myproject/myproject-service</bundle>
</feature>
```

16.10.8.3. 종속성

***<bundle />** 요소에서 번들에 **true**로 설정된 종속성 특성을 사용하여 종속성으로 플래그를 지정할 수 있습니다.*

이 정보는 확인자가 설치할 전체 번들 목록을 계산하는 데 사용할 수 있습니다.

16.10.9. 종속 기능

기능은 다른 기능 집합에 따라 달라질 수 있습니다.

```
<feature name="my-project" version="1.0.0">
  <feature>other</feature>
  <bundle start-level="80" start="false">mvn:com.mycompany.myproject/myproject-dao</bundle>
  <bundle start-level="85" start="false">mvn:com.mycompany.myproject/myproject-service</bundle>
</feature>
```

***my-project** 기능이 설치되면 다른 기능도 자동으로 설치됩니다.*

종속 기능에 대한 버전 범위를 정의할 수 있습니다.

```
<feature name="spring-dm">
  <feature version="[2.5.6,4)">spring</feature>
  ...
</feature>
```

범위에서 사용할 수 있는 가장 높은 버전의 기능이 설치됩니다.

단일 버전이 지정되면 범위가 열린 것으로 간주됩니다.

아무것도 지정하지 않으면 사용 가능한 최고가 설치됩니다.

*정확한 버전을 지정하려면 **[3.1,3.1]** 과 같은 폐쇄 범위를 사용합니다.*

16.10.9.1. 기능 사전 요구 사항

*사전 요구 사항 기능은 특별한 종류의 종속성입니다. 종속 기능 태그에 사전 요구 사항 속성을 추가하는 경우 실제 기능을 설치하기 전에 종속된 기능의 설치 및 번들도 활성화합니다. 지정된 기능에 포함된 번들이 사전 설치된 **URL**을 사용하는 경우(예: 랩 또는전) 이 유용할 수 있습니다.*

16.10.10. 기능 구성

*기능 **XML**의 **<config/ >** 요소는 기능을 통해 구성을 생성 및/또는 채울 수 있습니다(구성 **PID**로 식별됨).*

```
<config name="com.foo.bar">
  myProperty = myValue
</config>
```

***< config/ >** 요소의 **name** 속성은 구성 **PID**에 해당합니다(자세한 내용은 **[Configuration section|configuration]** 참조).*

*기능 설치는 **etc** 폴더에 **com.foo.bar.cfg** 라는 파일을 삭제하는 것과 동일한 효과가 있습니다.*

< config/> 요소의 내용은 key=value 표준에 따라 속성 집합입니다.

16.10.11. 기능 구성 파일

< config/ > 요소를 사용하는 대신 기능은 < configfile/ > 요소를 지정할 수 있습니다.

```
<configfile finalname="/etc/myfile.cfg" override="false">URL</configfile>
```

Apache Karaf 구성 계층을 직접 조작하는 대신(< config/ > 요소를 사용할 때) < configfile/ > 요소는 URL로 지정된 파일을 직접 가져와서 최종 이름 특성에 지정된 위치에 파일을 복사합니다.

지정하지 않으면 위치는 KARAF_BASE 변수에서 상대적입니다. ${karaf.home}, ${karaf.base}, ${karaf.etc} 또는 시스템 속성과 같은 변수를 사용할 수도 있습니다.

예를 들면 다음과 같습니다.

```
<configfile finalname="${karaf.etc}/myfile.cfg" override="false">URL</configfile>
```

파일이 이미 원하는 위치에 이미 있고 기존 파일에 사용자 지정이 포함될 수 있으므로 구성 파일의 배포를 건너뜁니다. 이 동작은 재정의를 통해 true로 설정할 수 있습니다.

파일 URL은 Apache Karaf에서 지원하는 모든 URL입니다(자세한 내용은 사용자 가이드의 [Artifacts 리포지토리 및 URL|urls] 참조).

16.10.11.1. 요구 사항

기능은 예상되는 요구 사항을 지정할 수도 있습니다. 기능 해결자는 요구 사항을 충족하기 위해 노력합니다. 이를 위해 기능 및 번들 기능을 확인하고 요구 사항을 충족하기 위해 번들을 자동으로 설치합니다.

예를 들어 기능에는 다음이 포함될 수 있습니다.

```
<requirement>osgi.ee;filter:=&quot;(&amp;(osgi.ee=JavaSE)(!(version&gt;=1.8)))&quot;
</requirement>
```

*요구 사항은 **JDK** 버전이 **1.8**이 아닌 경우에만 기능이 작동하도록 지정합니다(기본적으로 **1.7**).*

기능 확인자는 선택적 종속성을 충족할 때 번들을 새로 고칠 수 있으며 선택적 가져오기를 다시 조정할 수 있습니다.

16.10.12. 명령

16.10.12.1. feature:repo-list

***feature:repo-list** 명령은 등록된 모든 기능 리포지토리를 나열합니다.*

```
karaf@root()> feature:repo-list
Repository              | URL
------------------------------------------------------------------------------
org.ops4j.pax.cdi-0.12.0 | mvn:org.ops4j.pax.cdi/pax-cdi-features/0.12.0/xml/features
org.ops4j.pax.web-4.1.4  | mvn:org.ops4j.pax.web/pax-web-features/4.1.4/xml/features
standard-4.0.0           | mvn:org.apache.karaf.features/standard/4.0.0/xml/features
enterprise-4.0.0         | mvn:org.apache.karaf.features/enterprise/4.0.0/xml/features
spring-4.0.0             | mvn:org.apache.karaf.features/spring/4.0.0/xml/features
```

*각 리포지토리에는 **XML** 기능에 대한 이름과 **URL**이 있습니다.*

***Apache Karaf**는 기능 리포지토리 **URL**을 등록할 때 기능 **XML**을 구문 분석합니다(후에 설명된 대로 **feature:repo-add** 명령 또는 **FeatureMBean** 사용). **Apache Karaf**가 **features** 리포지토리 **URL**을 다시 로드하도록 강제 적용하려면 (및 기능 정의를 업데이트) **-r** 옵션을 사용할 수 있습니다.*

```
karaf@root()> feature:repo-list -r
Reloading all repositories from their urls

Repository              | URL
------------------------------------------------------------------------------
org.ops4j.pax.cdi-0.12.0 | mvn:org.ops4j.pax.cdi/pax-cdi-features/0.12.0/xml/features
org.ops4j.pax.web-4.1.4  | mvn:org.ops4j.pax.web/pax-web-features/4.1.4/xml/features
standard-4.0.0           | mvn:org.apache.karaf.features/standard/4.0.0/xml/features
enterprise-4.0.0         | mvn:org.apache.karaf.features/enterprise/4.0.0/xml/features
spring-4.0.0             | mvn:org.apache.karaf.features/spring/4.0.0/xml/features
```

16.10.12.2. feature:repo-add

*기능 리포지토리를 등록하려면 **Apache Karaf**에서 사용할 수 있는 새로운 기능을 사용하려면 **feature:repo-add** 명령을 사용해야 합니다.*

feature:repo-add 명령에는 name/url 인수가 필요합니다. 이 인수는 다음을 허용합니다.

- *기능 리포지토리 URL입니다. 기능 XML 파일에 대한 URL입니다. 사용자 가이드의 [Artifacts 리포지토리 및 URL 섹션|urls]에 설명된 모든 URL이 지원됩니다.*

- *etc/org.apache.karaf.features.repos.cfg 구성 파일에 정의된 기능 리포지토리 이름입니다.*

etc/org.apache.karaf.features.repos.cfg 는 "사전 설치/사용 가능한" 기능 리포지토리 목록을 정의합니다.

```
################################################################################
#
#   Licensed to the Apache Software Foundation (ASF) under one or more
#   contributor license agreements.  See the NOTICE file distributed with
#   this work for additional information regarding copyright ownership.
#   The ASF licenses this file to You under the Apache License, Version 2.0
#   (the "License"); you may not use this file except in compliance with
#   the License.  You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
#   Unless required by applicable law or agreed to in writing, software
#   distributed under the License is distributed on an "AS IS" BASIS,
#   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#   See the License for the specific language governing permissions and
#   limitations under the License.
#
################################################################################

#
# This file describes the features repository URL
# It could be directly installed using feature:repo-add command
#
enterprise=mvn:org.apache.karaf.features/enterprise/LATEST/xml/features
spring=mvn:org.apache.karaf.features/spring/LATEST/xml/features
cellar=mvn:org.apache.karaf.cellar/apache-karaf-cellar/LATEST/xml/features
cave=mvn:org.apache.karaf.cave/apache-karaf-cave/LATEST/xml/features
camel=mvn:org.apache.camel.karaf/apache-camel/LATEST/xml/features
camel-extras=mvn:org.apache-extras.camel-extra.karaf/camel-extra/LATEST/xml/features
cxf=mvn:org.apache.cxf.karaf/apache-cxf/LATEST/xml/features
cxf-dosgi=mvn:org.apache.cxf.dosgi/cxf-dosgi/LATEST/xml/features
cxf-xkms=mvn:org.apache.cxf.services.xkms/cxf-services-xkms-features/LATEST/xml
activemq=mvn:org.apache.activemq/activemq-karaf/LATEST/xml/features
jclouds=mvn:org.apache.jclouds.karaf/jclouds-karaf/LATEST/xml/features
openejb=mvn:org.apache.openejb/openejb-feature/LATEST/xml/features
wicket=mvn:org.ops4j.pax.wicket/features/LATEST/xml/features
hawtio=mvn:io.hawt/hawtio-karaf/LATEST/xml/features
```

```
pax-cdi=mvn:org.ops4j.pax.cdi/pax-cdi-features/LATEST/xml/features
pax-jdbc=mvn:org.ops4j.pax.jdbc/pax-jdbc-features/LATEST/xml/features
pax-jpa=mvn:org.ops4j.pax.jpa/pax-jpa-features/LATEST/xml/features
pax-web=mvn:org.ops4j.pax.web/pax-web-features/LATEST/xml/features
pax-wicket=mvn:org.ops4j.pax.wicket/pax-wicket-features/LATEST/xml/features
ecf=http://download.eclipse.org/rt/ecf/latest/site.p2/karaf-features.xml
decanter=mvn:org.apache.karaf.decanter/apache-karaf-decanter/LATEST/xml/features
```

feature:repo-add 명령에 기능 리포지토리 이름을 직접 제공할 수 있습니다. **PAX JDBC**를 설치하려면 다음을 수행할 수 있습니다.

```
karaf@root()> feature:repo-add pax-jdbc
Adding feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/LATEST/xml/features
```

선택적 버전 인수를 제공하지 않으면 **Apache Karaf**는 사용 가능한 기능 리포지토리의 최신 버전을 설치합니다. **version** 인수를 사용하여 대상 버전을 지정할 수 있습니다.

```
karaf@root()> feature:repo-add pax-jdbc 1.3.0
Adding feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
```

etc/org.apache.karaf.features.repos.cfg 구성 파일에 정의된 기능 리포지토리 이름을 제공하는 대신 **feature:repo-add** 명령에 기능 리포지토리 **URL**을 직접 제공할 수 있습니다.

```
karaf@root()> feature:repo-add mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
Adding feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
```

기본적으로 **feature:repo-add** 명령은 기능 리포지토리만 등록하고 기능은 설치되지 않습니다. **-i** 옵션을 지정하면 **feature:repo-add** 명령은 기능 리포지토리를 등록하고 이 기능 리포지토리에 설명된 모든 기능을 설치합니다.

```
karaf@root()> feature:repo-add -i pax-jdbc
```

16.10.12.3. feature:repo-refresh

Apache Karaf는 등록할 때 **features repository XML**을 구문 분석합니다(**feature:repo-add** 명령 또는 **FeatureMBean** 사용). 기능 리포지토리 **XML**이 변경되면 변경 사항을 로드하기 위해 기능 리포지토리를 새로 고치려면 **Apache Karaf**에 표시되어야 합니다.

feature:repo-refresh 명령은 **features** 리포지토리를 새로 고칩니다.

인수가 없으면 명령은 모든 기능 리포지토리를 새로 고칩니다.

```
karaf@root()> feature:repo-refresh
Refreshing feature url mvn:org.ops4j.pax.cdi/pax-cdi-features/0.12.0/xml/features
Refreshing feature url mvn:org.ops4j.pax.web/pax-web-features/4.1.4/xml/features
Refreshing feature url mvn:org.apache.karaf.features/standard/4.0.0/xml/features
Refreshing feature url mvn:org.apache.karaf.features/enterprise/4.0.0/xml/features
Refreshing feature url mvn:org.apache.karaf.features/spring/4.0.0/xml/features
```

모든 기능 리포지토리를 새로 고치는 대신 **URL** 또는 기능 리포지토리 이름을 제공하여 새로 고칠 **features** 리포지토리를 지정할 수 있습니다(선택 사항).

```
karaf@root()> feature:repo-refresh mvn:org.apache.karaf.features/standard/4.0.0/xml/features
Refreshing feature url mvn:org.apache.karaf.features/standard/4.0.0/xml/features
```

```
karaf@root()> feature:repo-refresh pax-jdbc
Refreshing feature url mvn:org.ops4j.pax.jdbc/pax-jdbc-features/LATEST/xml/features
```

16.10.12.4. feature:repo-remove

feature:repo-remove 명령은 등록된 시스템에서 **features** 리포지토리를 제거합니다.

feature:repo-remove 명령에는 인수가 필요합니다.

- **features** 리포지토리 이름(**feature:repo-list** 명령 출력의 리포지토리 열에 표시됨)
- **features repository URL (feature:repo-list** 명령 출력의 **URL** 열에 표시됨)

```
karaf@root()> feature:repo-remove org.ops4j.pax.jdbc-1.3.0
```

```
karaf@root()> feature:repo-remove mvn:org.ops4j.pax.jdbc/pax-jdbc-features/1.3.0/xml/features
```

기본적으로 **feature:repo-remove** 명령은 등록된 항목에서 **features** 리포지토리만 제거합니다. **features** 리포지토리에서 제공하는 기능은 제거되지 않습니다.

-u 옵션을 사용하는 경우 **feature:repo-remove** 명령은 **features** 리포지토리에서 설명하는 모든 기능을 제거합니다.

```
karaf@root()> feature:repo-remove -u org.ops4j.pax.jdbc-1.3.0
```

16.10.12.5. feature:list

feature:list 명령은 사용 가능한 모든 기능(다른 등록된 기능 리포지토리에서 제공)을 나열합니다.

```
Name                    | Version                  | Required | State       | Repository             |
Description
-------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------
pax-cdi                 | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Provide
CDI support
pax-cdi-1.1             | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.1 support
pax-cdi-1.2             | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.2 support
pax-cdi-weld            | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Weld
CDI support
pax-cdi-1.1-weld        | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld CDI 1.1 support
pax-cdi-1.2-weld        | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld CDI 1.2 support
pax-cdi-openwebbeans    | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
OpenWebBeans CDI support
pax-cdi-web             | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Web
CDI support
pax-cdi-1.1-web         | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Web
CDI 1.1 support
...
```

기능을 알파벳 이름으로 정렬하려면 **-o** 옵션을 사용할 수 있습니다.

```
karaf@root()> feature:list -o
Name                    | Version                  | Required | State       | Repository             |
Description
-------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------
deltaspike-core         | 1.2.1                    |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike core support
deltaspike-data         | 1.2.1                    |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike data support
deltaspike-jpa          | 1.2.1                    |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike jpa support
deltaspike-partial-bean | 1.2.1                    |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Apache Deltaspike partial bean support
pax-cdi                 | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Provide
CDI support
pax-cdi-1.1             | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.1 support
pax-cdi-1.1-web         | 0.12.0                   |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 | Web
CDI 1.1 support
```

```
pax-cdi-1.1-web-weld        | 0.12.0                    |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld Web CDI 1.1 support
pax-cdi-1.1-weld            | 0.12.0                    |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Weld CDI 1.1 support
pax-cdi-1.2                 | 0.12.0                    |          | Uninstalled | org.ops4j.pax.cdi-0.12.0 |
Provide CDI 1.2 support
...
```

*기본적으로 **feature:list** 명령은 현재 상태(설치됨 또는 설치되지 않음)를 모두 표시합니다.*

***-i** 옵션을 사용하면 설치된 기능만 표시됩니다.*

```
karaf@root()> feature:list -i
Name            | Version | Required | State   | Repository     | Description
----------------------------------------------------------------------------------------------------------------
aries-proxy     | 4.0.0   |          | Started | standard-4.0.0 | Aries Proxy
aries-blueprint | 4.0.0   | x        | Started | standard-4.0.0 | Aries Blueprint
feature         | 4.0.0   | x        | Started | standard-4.0.0 | Features Support
shell           | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Shell
shell-compat    | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Shell Compatibility
deployer        | 4.0.0   | x        | Started | standard-4.0.0 | Karaf Deployer
bundle          | 4.0.0   | x        | Started | standard-4.0.0 | Provide Bundle support
config          | 4.0.0   | x        | Started | standard-4.0.0 | Provide OSGi ConfigAdmin support
diagnostic      | 4.0.0   | x        | Started | standard-4.0.0 | Provide Diagnostic support
instance        | 4.0.0   | x        | Started | standard-4.0.0 | Provide Instance support
jaas            | 4.0.0   | x        | Started | standard-4.0.0 | Provide JAAS support
log             | 4.0.0   | x        | Started | standard-4.0.0 | Provide Log support
package         | 4.0.0   | x        | Started | standard-4.0.0 | Package commands and mbeans
service         | 4.0.0   | x        | Started | standard-4.0.0 | Provide Service support
system          | 4.0.0   | x        | Started | standard-4.0.0 | Provide System support
kar             | 4.0.0   | x        | Started | standard-4.0.0 | Provide KAR (KARaf archive) support
ssh             | 4.0.0   | x        | Started | standard-4.0.0 | Provide a SSHd server on Karaf
management      | 4.0.0   | x        | Started | standard-4.0.0 | Provide a JMX MBeanServer and a set of
MBeans in
wrap            | 0.0.0   | x        | Started | standard-4.0.0 | Wrap URL handler
```

16.10.12.6. feature:install

***feature:install** 명령은 기능을 설치합니다.*

***feature** 인수가 필요합니다. **feature** 인수는 기능의 이름 또는 기능의 이름/버전입니다. 기능 이름(버전이 아님)만 제공되면 사용 가능한 최신 버전이 설치됩니다.*

```
karaf@root()> feature:install eventadmin
```

***t** 또는 **--simulate** 옵션을 사용하여 설치를 시뮬레이션할 수 있습니다. 이 옵션은 수행하는 작업만 표*

시하지만 작동하지 않습니다.

```
karaf@root()> feature:install -t -v eventadmin
Adding features: eventadmin/[4.0.0,4.0.0]
No deployment change.
  Managing bundle:
    org.apache.felix.metatype / 1.0.12
```

설치할 기능 버전을 지정할 수 있습니다.

```
karaf@root()> feature:install eventadmin/4.0.0
```

*기본적으로 **feature:install** 명령은 상세하지 않습니다. **feature:install** 명령으로 수행한 작업에 대한 몇 가지 세부 정보를 사용하려면 **-v** 옵션을 사용할 수 있습니다.*

```
karaf@root()> feature:install -v eventadmin
Adding features: eventadmin/[4.0.0,4.0.0]
No deployment change.
Done.
```

*기능에 이미 설치된 번들이 포함된 경우 기본적으로 **Apache Karaf**는 이 번들을 새로 고칩니다. 경우에 따라 이 새로 고침으로 인해 다른 실행 중인 애플리케이션에 문제가 발생할 수 있습니다. 설치된 번들의 자동 새로 고침을 비활성화하려면 **-r** 옵션을 사용할 수 있습니다.*

```
karaf@root()> feature:install -v -r eventadmin
Adding features: eventadmin/[4.0.0,4.0.0]
No deployment change.
Done.
```

***-s** 또는 **--no-auto-start** 옵션을 사용하여 기능을 통해 설치한 번들을 시작하지 않도록 결정할 수 있습니다.*

```
karaf@root()> feature:install -s eventadmin
```

16.10.12.7. feature:start

*기본적으로 기능을 설치하면 자동으로 설치됩니다. 그러나 **feature:install** 명령에 **-s** 옵션을 지정할 수 있습니다.*

기능(시작 또는 아님)을 설치하는 즉시 기능에 정의된 번들에서 제공하는 모든 패키지를 사용할 수 있으며 다른 번들의 배선에 사용할 수 있습니다.

기능을 시작할 때 모든 번들이 시작되므로 이 기능에서도 서비스를 노출합니다.

16.10.12.8. 기능: 중지

또한 기능을 중지할 수 있습니다. 즉, 기능이 제공하는 모든 서비스가 서비스 레지스트리에서 중지 및 제거됩니다. 그러나 이 패키지는 계속 사용할 수 있습니다(번들은 해결된 상태입니다.).

16.10.12.9. feature:uninstall

feature:uninstall 명령은 기능을 제거합니다. **feature:install** 명령으로 **feature:uninstall** 명령에는 **feature** 인수가 필요합니다. **feature** 인수는 기능의 이름 또는 기능의 이름/버전입니다. 기능 이름(버전이 아님)만 제공되면 사용 가능한 최신 버전이 설치됩니다.

```
karaf@root()> feature:uninstall eventadmin
```

기능 확인자는 기능 제거 중에 관련됩니다. 다른 기능에서 사용하지 않는 경우 제거된 기능에 의해 설치된 전송적 기능을 자체적으로 제거할 수 있습니다.

16.10.13. deployer

배포 폴더에서 직접 파일을 삭제하여 기능 **XML**을 **"hot deploy "**할 수 있습니다.

Apache Karaf는 기능 배포자를 제공합니다.

배포 폴더에 기능 **XML**을 삭제하면 기능 배포자가 수행합니다. * **features deployer**는 **features XML**을 **features repository**로 등록합니다. * **install** 속성이 **"auto"**로 설정된 기능은 기능 배포자에 의해 자동으로 설치됩니다.

예를 들어 배포 폴더에 다음 **XML**을 삭제하면 **feature1** 및 **feature2**가 자동으로 설치되지만 **feature3**은 설치되지 않습니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<features name="my-features" xmlns="http://karaf.apache.org/xmlns/features/v1.3.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://karaf.apache.org/xmlns/features/v1.3.0
http://karaf.apache.org/xmlns/features/v1.3.0">
```

```
    <feature name="feature1" version="1.0" install="auto">
        ...
    </feature>

    <feature name="feature2" version="1.0" install="auto">
        ...
    </feature>

    <feature name="feature3" version="1.0">
        ...
    </feature>

</features>
```

16.10.14. JMX FeatureMBean

Cryostat 계층에서는 기능 및 기능 리포지토리인 **FeatureMBean**을 관리하는 전용 권한이 있습니다.

FeatureMBean 오브젝트 이름은 **org.apache.karaf:type=feature,name=*** 입니다.

16.10.14.1. 속성

FeatureMBean은 다음 두 가지 속성을 제공합니다.

- 기능은 사용 가능한 모든 기능의 표형 데이터 집합입니다.
- 리포지토리 는 등록된 모든 기능 리포지토리의 테이블 형식 데이터 집합입니다.

Repositories 속성은 다음 정보를 제공합니다.

- **name** 은 **features** 리포지토리의 이름입니다.
- **URI** 는 이 리포지토리의 기능 **XML**에 대한 **URI**입니다.
- 기능은 이 기능 리포지토리에서 제공하는 모든 기능(이름 및 버전)의 테이블 형식 데이터 집합입니다.

- *리포지토리 는 이 기능 리포지토리에서 "가져오기"하는 기능 리포지토리의 테이블 형식 데이터 집합입니다.*

***Feature** 속성은 다음 정보를 제공합니다.*

- ***name** 은 기능의 이름입니다.*
- ***version** 은 기능의 버전입니다.*
- ***installed** 는 부울입니다. **true**인 경우 해당 기능이 현재 설치되어 있음을 나타냅니다.*
- *번 들은 기능에 설명된 모든 번들(**bundles URL**)의 테이블 형식 데이터 세트입니다.*
- *구성은 기능에 설명된 모든 구성의 표형 데이터 집합입니다.*
- *구성 파일은 기능에 설명된 모든 구성 파일의 테이블 형식 데이터 집합입니다.*
- *종속성은 기능에 설명된 모든 종속 기능의 테이블 형식 데이터 집합입니다.**Dependency is a tabular data set of all dependent features described in the feature.***

16.10.14.2. 작업

- ***addRepository(url)** 는 **URL**이 있는 **features** 리포지토리를 추가합니다. **url** 은 **feature:repo-add** 명령과 마찬가지로 이름이 될 수 있습니다.*
- ***addRepository(url, install)** 는 **url** 을 사용하여 **features** 리포지토리를 추가하고 **install** 이 **true**인 경우 모든 번들을 자동으로 설치합니다. **url** 은 **feature:repo-add** 명령과 같은 이름 일 수 있습니다.*
- ***removeRepository(url)** 는 **URL**이 있는 **features** 리포지토리를 제거합니다. **url** 은 **feature:repo-remove** 명령과 마찬가지로 이름이 될 수 있습니다.*

- *installFeature(name) 는 이름으로 기능을 설치합니다.*
- *installFeature(이름, 버전) 는 이름과 버전으로 기능을 설치합니다.*
- *installFeature(name, noClean, noRefresh) 는 실패한 경우 번들을 정리하지 않고 이미 설치된 번들을 새로 고침하지 않고 이름으로 기능을 설치합니다.*
- *installFeature(name, version, noClean, noRefresh) '는 오류가 발생할 경우 번들을 정리하지 않고 '이름 및 버전으로 기능을 설치합니다.*
- *uninstallFeature(name) 는 이름으로 기능을 제거합니다.*
- *uninstallFeature(name, version) 는 이름과 버전이 있는 기능을 제거합니다.*

16.10.14.3. 알림

FeatureMBean은 두 가지 유형의 알림을 보냅니다(서브스크립션 및 대응 가능).

- *기능 리포지토리가 변경되는 경우(추가 또는 제거됨).*
- *기능이 변경되면(설치 또는 제거됨).*

17장. 원격 연결을 사용하여 컨테이너 관리

컨테이너를 관리하기 위해 로컬 콘솔을 사용하는 것이 항상 바람직하지는 않습니다. **Red Hat Fuse**에는 여러 가지 방법으로 컨테이너를 원격으로 관리할 수 있습니다. 원격 컨테이너의 명령 콘솔을 사용하거나 원격 클라이언트를 시작할 수 있습니다.

17.1. 원격 액세스를 위한 컨테이너 구성

17.1.1. 개요

기본 모드 또는 2.1.3절. “서버 모드에서 런타임 시작” 에서 **Red Hat Fuse** 런타임을 시작하면 다른 **Fuse** 콘솔에서 **SSH**를 통해 액세스할 수 있는 원격 콘솔을 활성화합니다. 원격 콘솔은 로컬 콘솔의 모든 기능을 제공하며 원격 사용자가 컨테이너 및 내부에서 실행되는 서비스를 완전히 제어할 수 있습니다.

참고

2.1.4절. “클라이언트 모드에서 런타임 시작” 에서 실행하면 **Fuse** 런타임에서 원격 콘솔을 비활성화합니다.

17.1.2. 원격 액세스를 위한 독립 실행형 컨테이너 구성

SSH 호스트 이름과 포트 번호는 **INSTALL_DIR/etc/org.apache.karaf.shell.cfg** 구성 파일에 구성됩니다. 원격 액세스용 포트 변경 는 **8102**로 사용되는 포트를 변경하는 샘플 구성을 보여줍니다.

원격 액세스용 포트 변경

```
sshPort=8102
sshHost=0.0.0.0
```

17.2. 원격 연결 및 연결 해제

원격 컨테이너에 연결하는 방법은 다음 두 가지가 있습니다. **Red Hat Fuse** 명령 쉘을 이미 실행 중인 경우 원격 컨테이너에 연결하려면 **console** 명령을 호출해야 합니다. 또는 명령줄에서 직접 유틸리티를 실행하여 원격 컨테이너에 연결할 수도 있습니다.

17.2.1. 원격 컨테이너에서 독립 실행형 컨테이너에 연결

17.2.1.1. 개요

컨테이너의 모든 명령 콘솔을 사용하여 원격 컨테이너에 액세스할 수 있습니다. **SSH**를 사용하여 로컬 컨테이너의 콘솔은 원격 컨테이너에 연결되고 원격 컨테이너의 명령 콘솔로 작동합니다.

17.2.1.2. ssh:ssh console 명령 사용

ssh:ssh console 명령을 사용하여 원격 컨테이너의 콘솔에 연결합니다.

ssh:ssh 명령 구문

```
ssh:ssh -l username -P password -p port hostname
```

-l

원격 컨테이너 연결에 사용되는 사용자 이름입니다. 관리자 권한이 있는 유효한 **JAAS** 로그인 자격 증명을 사용합니다.

-P

원격 컨테이너 연결에 사용되는 암호입니다.

-p

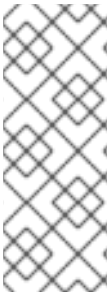
원하는 컨테이너의 원격 콘솔에 액세스하는 데 사용되는 **SSH** 포트입니다. 기본적으로 이 값은 **8101** 입니다. 포트 번호 변경에 대한 자세한 내용은 17.1.2절. “원격 액세스를 위한 독립 실행형 컨테이너 구성” 을 참조하십시오.

hostname

원격 컨테이너가 실행 중인 시스템의 호스트 이름입니다. 호스트 이름 변경에 대한 자세한 내용은 17.1.2절. “원격 액세스를 위한 독립 실행형 컨테이너 구성” 을 참조하십시오.

주의

etc/users.properties 파일에서 사용자 이름과 암호를 사용자 지정하는 것이 좋습니다.



참고

원격 컨테이너가 **Oracle VM Server for Cryostat** 인스턴스에 배포된 경우 기본 **SSH** 포트 값 **8101** 이 이미 **Logical Domains Manager** 데몬에 의해 사용되고 있을 가능성이 높습니다. 이 경우 **17.1.2절. "원격 액세스를 위한 독립 실행형 컨테이너 구성"** 에 설명된 대로 컨테이너의 **SSH** 포트를 재구성해야 합니다.

올바른 컨테이너에 연결되어 있는지 확인하려면 **Karaf** 콘솔 프롬프트에 **shell:info** 를 입력하여 현재 연결된 인스턴스에 대한 정보를 반환합니다.

17.2.1.3. 원격 콘솔에서 연결 해제

원격 콘솔에서 연결을 끊으려면 **logout** 을 입력하거나 프롬프트에서 **Ctrl+D** 누릅니다.

원격 컨테이너의 연결이 끊어지고 콘솔은 다시 로컬 컨테이너를 관리합니다.

17.2.2. 클라이언트 명령줄 유틸리티를 사용하여 컨테이너에 연결

17.2.2.1. 원격 클라이언트 사용

원격 클라이언트를 사용하면 전체 **Fuse** 컨테이너를 로컬로 시작하지 않고도 원격 **Red Hat Fuse** 컨테이너에 안전하게 연결할 수 있습니다.

예를 들어 동일한 머신의 서버 모드에서 실행되는 **Fuse** 인스턴스에 빠르게 연결하려면 명령 프롬프트를 열고 다음과 같이 클라이언트**[.CAST]** 스크립트(**InstallDir/bin** 디렉터리에 있는) 스크립트를 실행합니다.

```
client
```

일반적으로 원격 인스턴스에 연결할 호스트 이름, 포트, 사용자 이름 및 암호를 제공합니다. 더 큰 스크립트 내에서 클라이언트를 사용하는 경우(예: 테스트 모음) 다음과 같이 **console** 명령을 추가할 수 있습니다.

```
client -a 8101 -h hostname -u username -p password shell:info
```

또는 **-p** 옵션을 생략하면 암호를 입력하라는 메시지가 표시됩니다.

독립 실행형 컨테이너의 경우 관리자 권한이 있는 유효한 **JAAS** 사용자 자격 증명을 사용합니다.

클라이언트에 사용 가능한 옵션을 표시하려면 다음을 입력합니다.

```
client --help
```

Karaf 클라이언트 도움말

```
Apache Felix Karaf client
  -a [port]     specify the port to connect to
  -h [host]     specify the host to connect to
  -u [user]     specify the user name
  -p [password] specify the password
  --help        shows this help message
  -v            raise verbosity
  -r [attempts] retry connection establishment (up to attempts times)
  -d [delay]    intra-retry delay (defaults to 2 seconds)
  [commands]    commands to run
If no commands are specified, the client will be put in an interactive mode
```

17.2.2.2. 원격 클라이언트 기본 인증 정보

자격 증명을 제공하지 않고 **bin/client** 를 사용하여 **Karaf** 컨테이너에 로그인할 수 있다는 것을 알게 되어 반가운 경우가 있을 수 있습니다. 이는 원격 클라이언트 프로그램이 기본 자격 증명을 사용하도록 미리 구성되어 있기 때문입니다. 인증 정보를 지정하지 않으면 원격 클라이언트는 다음과 같은 기본 인증 정보(순위)를 자동으로 사용하려고 합니다.

- 기본 **SSH** 키 **Cryostat- Cryostattries**를 사용하여 기본 **Apache Karaf SSH** 키를 사용하여 로그인합니다. 이 로그인이 성공할 수 있도록 허용하는 해당 구성 항목은 **etc/keys.properties** 파

일에서 기본적으로 주석 처리됩니다.

- 기본 사용자 이름/암호 자격 증명 모음 - 사용자 이름 및 암호의 **admin/admin** 조합을 사용하여 로그인할 수 있습니다. 이 로그인이 성공할 수 있도록 허용하는 해당 구성 항목은 **etc/users.properties** 파일에서 기본적으로 주석 처리됩니다.

따라서 **users.properties** 의 기본 **admin/admin** 자격 증명의 주석을 제거하여 **Karaf** 컨테이너에 새 사용자를 생성하면 **bin/client** 유틸리티에서 인증 정보를 제공하지 않고 로그인할 수 있습니다.

중요

보안을 위해 **Karaf** 컨테이너를 처음 설치할 때 **Fuse**에서 기본 인증 정보를 사용하지 않도록 설정했습니다. 그러나 기본 암호 또는 **SSH** 공개 키를 변경 하지 않고 이러한 기본 인증 정보의 주석을 간단히 제거하는 경우 **Karaf** 컨테이너에 보안 허점이 열립니다. 프로덕션 환경에서는 절대 이 작업을 수행하지 않아야 합니다. 인증 정보를 제공하지 않고 **bin/client** 를 사용하여 컨테이너에 로그인할 수 있는 경우 컨테이너가 안전하지 않으며 프로덕션 환경에서 이를 해결하기 위한 단계를 수행해야 합니다.

17.2.2.3. 원격 클라이언트 콘솔에서 연결 해제

원격 클라이언트를 사용하여 원격 콘솔을 열 때 명령을 전달하는 대신 명령을 전달하는 데 사용하는 경우 연결을 해제해야 합니다. 원격 클라이언트의 콘솔에서 연결을 끊으려면 **logout** 을 입력하거나 프롬프트에서 **Ctrl-D** 를 누릅니다.

클라이언트가 연결을 해제하고 종료합니다.

17.2.3. SSH 명령줄 유틸리티를 사용하여 컨테이너에 연결

17.2.3.1. 개요

ssh 명령줄 유틸리티(**UNIX**와 같은 운영 체제의 표준 유틸리티)를 사용하여 인증 메커니즘이 공개 키 암호화를 기반으로 하는 **Red Hat Fuse** 컨테이너에 로그인할 수도 있습니다(공개 키를 먼저 컨테이너에 설치해야 함). 예를 들어 컨테이너가 **TCP** 포트 **8101**에서 수신 대기하도록 구성된 경우 다음과 같이 로그인할 수 있습니다.

```
ssh -p 8101 jdoe@localhost
```

중요

키 기반 로그인은 현재 **Fabric** 컨테이너가 아닌 독립 실행형 컨테이너에서만 지원됩니다.

17.2.3.2. 사전 요구 사항

키 기반 **SSH** 로그인을 사용하려면 다음 사전 요구 사항을 충족해야 합니다.

- 컨테이너는 **PublickeyLoginModule** 이 설치된 독립 실행형(**Fabric**은 지원되지 않음)이어야 합니다.
- **SSH** 키 쌍이 생성되어 있어야 합니다(17.2.3.4절. “새 SSH 키 쌍 생성”참조).
- **SSH** 키 쌍의 공개 키를 컨테이너에 설치해야 합니다(17.2.3.5절. “컨테이너에 SSH 공개 키 설치”참조).

17.2.3.3. 기본 키 위치

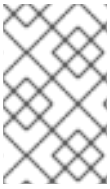
ssh 명령은 기본 키 위치에서 개인 키를 자동으로 찾습니다. 위치를 명시적으로 지정하는 데 문제가 저장되므로 기본 위치에 키를 설치하는 것이 좋습니다.

***NIX** 운영 체제에서는 **RSA** 키 쌍의 기본 위치는 다음과 같습니다.

```
~/.ssh/id_rsa
~/.ssh/id_rsa.pub
```

Windows 운영 체제에서 **RSA** 키 쌍의 기본 위치는 다음과 같습니다.

```
C:\Documents and Settings\Username\.ssh\id_rsa
C:\Documents and Settings\Username\.ssh\id_rsa.pub
```



참고

Red Hat Fuse는 **RSA** 키만 지원합니다. **DSA** 키가 작동하지 않습니다.

17.2.3.4. 새 SSH 키 쌍 생성

ssh-keygen 유틸리티를 사용하여 **RSA** 키 쌍을 생성합니다. 새 명령 프롬프트를 열고 다음 명령을 입력합니다.

```
ssh-keygen -t rsa -b 2048
```

앞의 명령은 키 길이가 **2048**비트인 **RSA** 키를 생성합니다. 그러면 키 쌍의 파일 이름을 지정하라는 메시지가 표시됩니다.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/Username/.ssh/id_rsa):
```

return을 입력하여 기본 위치에 키 쌍을 저장합니다. 그런 다음 전달 문구를 입력하라는 메시지가 표시됩니다.

```
Enter passphrase (empty for no passphrase):
```

선택적으로 여기에 통과 문구를 입력하거나 반환을 두 번 입력하여 패스 문구를 선택하지 않을 수 있습니다.

참고

Fabric console 명령을 실행하는 데 동일한 키 쌍을 사용하려면 **Fabric**에서 암호화된 개인 키 사용을 지원하지 않으므로 전달 문구 를 선택하는 것이 좋습니다.

17.2.3.5. 컨테이너에 SSH 공개 키 설치

SSH 키 쌍을 사용하여 **Red Hat JBoss Fuse** 컨테이너에 로그인하려면 **INSTALL_DIR/etc/keys.properties** 파일에 새 사용자 항목을 생성하여 컨테이너에 **SSH** 공개 키를 설치해야 합니다. 이 파일의 각 사용자 항목은 다음 형식으로 한 줄에 표시됩니다.

```
Username=PublicKey,Role1,Role2,...
```

예를 들어 공개 키 파일 **~/.ssh/id_rsa.pub** 에게 다음과 같은 내용이 있습니다.

```
ssh-rsa
AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH7WT2
NWPq/xfW6MPbLm1Vs14E7
```

```
gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnfqimFQ8E+4P208UewwI1VBNaFpEy9nXzrith1y
rv8iIDGZ3RSAHHAAAAFQCX
YFCPFSMLzLKSuYKi64QL8Fgc9QAAAnEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0Hgmd
RWVeOutRZT+ZxBxCBgLRJFnEj6Ewo
FhO3zwkyjMim4TwWeotifI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKL
Zl6Ae1UlZAFMO/7PSSoAAACB
AKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXfskI8wO63OzP/qLmcJM0+JbcRU/53Jj7uyk
31drV2qxhIOsLDC9dGCWj4
7Y7TyhPdXh/0dthTRBy6bqGtRPxGa7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx
jdoe@doemachine.local
```

***InstallDir/etc/keys.properties** 파일에 다음 항목을 추가하여 **admin** 역할로 **jdoe** 사용자를 생성할 수 있습니다(한 줄에 있음).*

```
jdoe=AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH
7WT2NWPq/xfW6MPbLm1Vs14E7
gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnfqimFQ8E+4P208UewwI1VBNaFpEy9nXzrith1y
rv8iIDGZ3RSAHHAAAAFQCX
YFCPFSMLzLKSuYKi64QL8Fgc9QAAAnEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0Hgmd
RWVeOutRZT+ZxBxCBgLRJFnEj6Ewo
FhO3zwkyjMim4TwWeotifI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWfBpKL
Zl6Ae1UlZAFMO/7PSSoAAACB
AKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXfskI8wO63OzP/qLmcJM0+JbcRU/53Jj7uyk
31drV2qxhIOsLDC9dGCWj4
7Y7TyhPdXh/0dthTRBy6bqGtRPxGa7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx,g:admingroup
```

중요

***id_rsa.pub** 파일의 전체 내용을 여기에 삽입하지 마십시오. 공개 키 자체를 나타내는 기호 블록을 삽입합니다.*

17.2.3.6. 공개 키 인증이 지원되는지 확인

*컨테이너를 시작한 후 다음과 같이 **jaas:realms console** 명령을 실행하여 공개 키 인증이 지원되는지 확인할 수 있습니다.*

```
karaf@root()> jaas:realms
Index │ Realm Name │ Login Module Class Name
──────┼────────────┼──────────────────────────────────────────────────────────────
──────-
1 │ karaf │ org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
2 │ karaf │ org.apache.karaf.jaas.modules.publickey.PublickeyLoginModule
3 │ karaf │ org.apache.karaf.jaas.modules.audit.FileAuditLoginModule
4 │ karaf │ org.apache.karaf.jaas.modules.audit.LogAuditLoginModule
5 │ karaf │ org.apache.karaf.jaas.modules.audit.EventAdminAuditLoginModule
karaf@root()>
```

PublickeyLoginModule 이 설치되어 있는지 확인해야 합니다. 이 구성을 사용하면 사용자 이름/암호 자격 증명 또는 공개 키 자격 증명을 사용하여 컨테이너에 로그인할 수 있습니다.

17.2.3.7. etc/keys.properties에 ssh 역할 추가

etc/keys.properties 에 정의된 관리자 그룹에 는 다음 예와 같이 **ssh** 역할이 포함되어야 합니다.

```
#
# For security reason, the default auto-signed key is disabled.
# The user guide describes how to generate/update the key.
#
#karaf=AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIl
H7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E
+4P208UewwI1VBNaFpEy9nXzrith1yrv8iIDGZ3RSAHHAAAAFQCXYFCPFSMLzLKSuYKi64QL8Fgc9
QAAAIEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6
EwoFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuW
fBpKLZl6Ae1UlZAFMO/7PSSoAAACBAKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXfskl8
wO63OzP/qLmcJM0+JbcRU/53JjTuyk31drV2qxhIOsLDC9dGCWj47Y7TyhPdXh/0dthTRBy6bqGtRPxG
a7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx,_g_:admingroup
_g_\:admingroup = group,admin,manager,viewer,systembundles,ssh
```

ssh 역할이 **admingroup** 정의에 포함되지 않은 경우 **etc/keys.properties** 를 편집하고 **ssh** 역할을 추가해야 합니다.

17.2.3.8. 키 기반 SSH를 사용하여 로그인

이제 키 기반 **SSH** 유틸리티를 사용하여 컨테이너에 로그인할 준비가 되었습니다. 예를 들면 다음과 같습니다.

```
$ ssh -p 8101 jdoe@localhost

_____         _   _   _       _    ______
|  _ \ ___  __| | | | | |__ _| |_  |  ____|   _ ___  ___
| |_) / _ \/ _` | |_| |/ _` | __| | |__ | | | / __|/ _ \
|  _ <  __/ (_| |  _  | (_| | |_  |  __|| |_| \__ \  __/
|_| \_\___|\__,_|_| |_|\__,_|\__| |_|    \__,_|___/\___|

  Fuse (7.x.x.fuse-xxxxxx-redhat-xxxxx)
  http://www.redhat.com/products/jbossenterprisemiddleware/fuse/

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.

Open a browser to http://localhost:8181/hawtio to access the management console

Hit '<ctrl-d>' or 'shutdown' to shutdown Red Hat Fuse.

karaf@root()>
```

참고

*암호화된 개인 키를 사용하는 경우 **ssh** 유틸리티에서 패스 문구를 입력하라는 메시지를 표시합니다.*

17.3. 원격 컨테이너 중지

***ssh:ssh** 명령 또는 원격 클라이언트를 사용하여 원격 콘솔에 연결한 경우 **osgi:shutdown** 명령을 사용하여 원격 인스턴스를 중지할 수 있습니다.*

참고

*원격 콘솔에서 **Ctrl+D** 누르면 원격 연결을 닫고 로컬 셸로 돌아갑니다.*

18장. MAVEN으로 빌드

초록

Maven은 Apache Maven 프로젝트에서 사용할 수 있는 오픈 소스 빌드 시스템입니다. 이 장에서는 몇 가지 기본 Maven 개념에 대해 설명하고 Red Hat Fuse에서 작동하도록 Maven을 설정하는 방법을 설명합니다. 기본적으로 모든 빌드 시스템을 사용하여 OSGi 번들을 빌드할 수 있습니다. 그러나 Maven은 Red Hat Fuse에서 효과적으로 지원하는 것이 좋습니다.

18.1. MAVEN 디렉터리 STRUCTURE

18.1.1. 개요

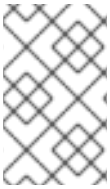
Maven 빌드 시스템의 가장 중요한 원칙 중 하나는 Maven 프로젝트의 모든 파일에 대한 표준 위치가 있다는 것입니다. 이 원칙에는 몇 가지 장점이 있습니다. 한 가지 장점은 Maven 프로젝트에는 일반적으로 동일한 디렉터리 레이아웃이 있어 프로젝트에서 파일을 쉽게 찾을 수 있다는 것입니다. 또 다른 장점은 Maven과 통합된 다양한 툴이 초기 구성이 거의 필요하지 않다는 것입니다. 예를 들어 Java 컴파일러는 src/main/java 아래에 있는 모든 소스 파일을 컴파일하고 결과를 대상/클래스 에 배치해야 함을 알고 있습니다.

18.1.2. 표준 디렉터리 레이아웃

예 18.1. “표준 Maven 디렉터리 레이아웃” OSGi 번들 프로젝트 빌드와 관련된 표준 Maven 디렉터리 레이아웃의 요소를 보여줍니다. 또한 블루프린트 구성 파일(MM에 의해 정의되지 않음)의 표준 위치도 표시됩니다.

예 18.1. 표준 Maven 디렉터리 레이아웃

```
ProjectDir/
  pom.xml
  src/
    main/
      java/
        ...
      resources/
        META-INF/

        OSGI-INF/
          blueprint/
            *.xml
    test/
      java/
      resources/
  target/
    ...
```



참고

표준 디렉터리 레이아웃을 재정의할 수 있지만 **Maven**에서는 권장되지 않습니다.

18.1.3. POM.xml 파일

pom.xml 파일은 현재 프로젝트를 빌드하는 방법에 대한 완전한 설명이 포함된 현재 프로젝트의 **POM(Project Object Model)**입니다. **pom.xml** 파일은 완전히 자체 포함 될 수 있지만 상위 **POM** 파일에서 설정을 가져올 수 있습니다.

프로젝트를 빌드한 후 생성된 **JAR** 파일의 다음 위치에 **pom.xml** 파일의 사본이 자동으로 포함됩니다.

```
META-INF/maven/groupId/artifactId/pom.xml
```

18.1.4. src 및 대상 디렉터리

src/ 디렉터리에는 프로젝트를 개발하는 동안 작업할 모든 코드와 리소스 파일이 포함되어 있습니다.

target/ 디렉터리에는 빌드 결과(일반적으로 **JAR** 파일)와 빌드 중에 생성된 모든 중간 파일이 포함됩니다. 예를 들어 빌드를 수행한 후 **target/classes/** 디렉터리에는 리소스 파일의 복사본과 컴파일된 **Java** 클래스가 포함됩니다.

18.1.5. 기본 및 테스트 디렉토리

src/main/ 디렉터리에는 아티팩트를 빌드하는 데 필요한 모든 코드와 리소스가 포함되어 있습니다.

src/test/ 디렉터리에는 컴파일된 아티팩트에 대해 단위 테스트를 실행하기 위한 모든 코드와 리소스가 포함되어 있습니다.

18.1.6. Java 디렉터리

각 **java /** 하위 디렉터리에는 표준 **Java** 디렉터리 레이아웃이 있는 **Java** 소스 코드(***.java** 파일)가 포함되어 있습니다(즉, 디렉터리 경로 이름이 **.** 문자 대신 **Java** 패키지 이름을 미러링하는 경우). **src/main/java/** 디렉터리에는 번들 소스 코드가 포함되어 있고 **src/test/java/** 디렉터리에는 단위 테스트 소스 코드가 포함되어 있습니다.

18.1.7. 리소스 디렉터리

번들에 포함할 구성 파일, 데이터 파일 또는 Java 속성이 있는 경우 src/main/resources/ 디렉터리에 배치해야 합니다. src/main/resources/ 아래의 파일 및 디렉터리는 Maven 빌드 프로세스에서 생성되는 JAR 파일의 루트로 복사됩니다.

src/test/resources/ 아래의 파일은 테스트 단계에서만 사용되며 생성된 JAR 파일에 복사 되지 않습니다.

18.1.8. 블루프린트 컨테이너

OSGi R4.2는 블루프린트 컨테이너를 정의합니다. Red Hat Fuse는 블루프린트 컨테이너를 기본적으로 지원하므로 블루프린트 구성 파일인 OSGI-INF/blueprint/*.xml 을 프로젝트에 포함할 수 있습니다. 블루프린트 컨테이너에 대한 자세한 내용은 12장. OSGi Services 에서 참조하십시오.

18.2. APACHE KARAF용 BOM 파일

BOM(Maven bill of materials) 파일의 목적은 잘 작동하는 Maven 종속성 버전 집합을 제공하여 모든 Maven 아티팩트에 대해 버전을 개별적으로 정의할 필요가 없도록 하는 것입니다.

Apache Karaf용 Fuse BOM은 다음과 같은 이점을 제공합니다.

- POM에 종속성을 추가할 때 버전을 지정할 필요가 없도록 Maven 종속 항목에 대한 버전을 정의합니다.
- 특정 버전의 Fuse에서 완전히 테스트 및 지원되는 선별된 종속성 세트를 정의합니다.
- Fuse 업그레이드 간소화.

중요

Fuse BOM에서 정의한 종속성 세트만 Red Hat에서 지원합니다.

Maven BOM 파일을 Maven 프로젝트에 통합하려면 다음 예와 같이 프로젝트의 pom.xml 파일(또는 상위 POM 파일)에 dependencyManagement 요소를 지정합니다.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project ...>
  ...
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <!-- configure the versions you want to use here -->
    <fuse.version>7.11.1.fuse-sb2-7_11_1-00022-redhat-00002</fuse.version>

  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.jboss.redhat-fuse</groupId>
        <artifactId>fuse-karaf-bom</artifactId>
        <version>${fuse.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```

참고

org.jboss.redhat-fuse BOM은 **Fuse 7**의 새로운 기능이며 **BOM** 버전 관리를 단순화하도록 설계되었습니다. **Fuse** 빠른 시작 및 **Maven archetypes**는 여전히 이전 **BOM** 스타일을 사용하지만 새 버전을 사용하도록 리팩토링되지 않았기 때문입니다. **BOM**은 모두 올바르며 **Maven** 프로젝트에서 사용할 수 있습니다. 예정된 **Fuse** 릴리스에서 빠른 시작 및 **Maven archetypes**가 새로운 **BOM**을 사용하도록 리팩터링됩니다.

종속성 관리 메커니즘을 사용하여 **BOM**을 지정하면 아티팩트 버전을 지정하지 않고 **Maven** 종속성을 **POM**에 추가할 수 있습니다. 예를 들어 **camel-velocity** 구성 요소에 대한 종속성을 추가하려면 **POM**의 종속성 요소에 다음 **XML** 조각을 추가합니다.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-velocity</artifactId>
</dependency>
```

이 종속성 정의에서 **version** 요소를 생략하는 방법을 참조하십시오.

19장. MAVEN INDEXER 플러그인

Maven 플러그인에서 Maven Central이 아티팩트를 빠르게 검색할 수 있도록 하려면 Maven 플러그인에 필요합니다.

Maven Indexer 플러그인을 배포하려면 다음 명령을 사용합니다.

사전 요구 사항

Maven Indexer 플러그인을 배포하기 전에 Apache Karaf preparing to Use Maven 섹션의 지침을 따라야 합니다.

Maven Indexer 플러그인 배포

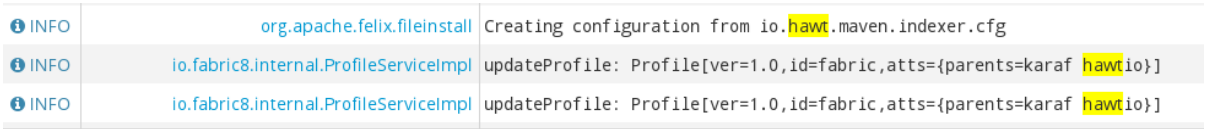
1. Karaf 콘솔로 이동하여 다음 명령을 입력하여 Maven Indexer 플러그인을 설치합니다.

```
features:install hawtio-maven-indexer
```

2. 다음 명령을 입력하여 Maven Indexer 플러그인을 구성합니다.

```
config:edit io.hawt.maven.indexer
config:proplist
config:propset repositories 'https://maven.oracle.com'
config:proplist
config:update
```

3. Maven Indexer 플러그인이 배포될 때까지 기다립니다. 이 작업은 몇 분 정도 걸릴 수 있습니다. 로그 탭에 표시하려면 아래에 표시된 것과 같은 메시지를 확인합니다.



Maven Indexer 플러그인이 배포되면 다음 명령을 사용하여 Maven Indexer 플러그인 구성에 외부 Maven 리포지토리를 추가합니다.

```
config:edit io.hawt.maven.indexer
config:proplist
config:propset repositories external repository
config:proplist
config:update
```

20장. LOG

Apache Karaf는 동적이고 강력한 로깅 시스템을 제공합니다.

지원 대상:

- OSGi 로그 서비스
- Apache Log4j v1 및 v2 프레임워크
- Apache Commons Logging 프레임워크
- 로그백 프레임워크
- SLF4J 프레임워크
- 네이티브 Java Util Logging 프레임워크

즉, 애플리케이션에서 모든 로깅 프레임워크를 사용할 수 있으며 Apache Karaf는 중앙 로그 시스템을 사용하여 로거, appenders 등을 관리합니다.

20.1. 구성 파일

초기 로그 구성이 etc/org.ops4j.pax.logging.cfg 에서 로드됩니다.

이 파일은 표준 Log4j2 구성 파일입니다.

다양한 Log4j2 요소가 있습니다.

- 로거

- ***appender***
- *레이아웃*

파일에 직접 고유한 초기 구성을 추가할 수 있습니다.

기본 구성은 다음과 같습니다.

```
#
#  Copyright 2005-2018 Red Hat, Inc.
#
#  Red Hat licenses this file to you under the Apache License, version
#  2.0 (the "License"); you may not use this file except in compliance
#  with the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
#  Unless required by applicable law or agreed to in writing, software
#  distributed under the License is distributed on an "AS IS" BASIS,
#  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
#  implied.  See the License for the specific language governing
#  permissions and limitations under the License.
#

#
# Internal Log4j2 configuration
#
log4j2.status = WARN
log4j2.verbose = false
log4j2.dest = out

#
# Common pattern layouts for appenders defined as reusable properties
# See https://logging.apache.org/log4j/2.x/manual/layouts.html#PatternLayout
# references will be replaced by felix.fileinstall
#
log4j2.pattern = %d{DEFAULT} | %-5.5p | %-20.20t | %-32.32c{1.} | %X{bundle.id} -
%X{bundle.name} - %X{bundle.version} | %m%n
#log4j2.pattern = %d{DEFAULT} %-5.5p {%t} [%C.%M()] (%F:%L) : %m%n

#
# Appenders configuration
#

# JDBC Appender
log4j2.appender.jdbc.type = JDBC
log4j2.appender.jdbc.name = JdbcAppender
log4j2.appender.jdbc.tableName = EVENTS
```

```
log4j2.appender.jdbc.cs.type = DataSource
log4j2.appender.jdbc.cs.lazy = true
log4j2.appender.jdbc.cs.jndiName = osgi:service/jdbc/logdb
log4j2.appender.jdbc.c1.type = Column
log4j2.appender.jdbc.c1.name = DATE
log4j2.appender.jdbc.c1.isEventTimestamp = true
log4j2.appender.jdbc.c2.type = Column
log4j2.appender.jdbc.c2.name = LEVEL
log4j2.appender.jdbc.c2.pattern = %level
log4j2.appender.jdbc.c2.isUnicode = false
log4j2.appender.jdbc.c3.type = Column
log4j2.appender.jdbc.c3.name = SOURCE
log4j2.appender.jdbc.c3.pattern = %logger
log4j2.appender.jdbc.c3.isUnicode = false
log4j2.appender.jdbc.c4.type = Column
log4j2.appender.jdbc.c4.name = THREAD_ID
log4j2.appender.jdbc.c4.pattern = %thread
log4j2.appender.jdbc.c4.isUnicode = false
log4j2.appender.jdbc.c5.type = Column
log4j2.appender.jdbc.c5.name = MESSAGE
log4j2.appender.jdbc.c5.pattern = %message
log4j2.appender.jdbc.c5.isUnicode = false


# Console appender not used by default (see log4j2.rootLogger.appenderRefs)
log4j2.appender.console.type = Console
log4j2.appender.console.name = Console
log4j2.appender.console.layout.type = PatternLayout
log4j2.appender.console.layout.pattern = ${log4j2.pattern}

# Rolling file appender
log4j2.appender.rolling.type = RollingRandomAccessFile
log4j2.appender.rolling.name = RollingFile
log4j2.appender.rolling.fileName = ${karaf.data}/log/fuse.log
log4j2.appender.rolling.filePattern = ${karaf.data}/log/fuse-%i.log.gz
# uncomment to not force a disk flush
#log4j2.appender.rolling.immediateFlush = false
log4j2.appender.rolling.append = true
log4j2.appender.rolling.layout.type = PatternLayout
log4j2.appender.rolling.layout.pattern = ${log4j2.pattern}
log4j2.appender.rolling.policies.type = Policies
log4j2.appender.rolling.policies.size.type = SizeBasedTriggeringPolicy
log4j2.appender.rolling.policies.size.size = 16MB
log4j2.appender.rolling.strategy.type = DefaultRolloverStrategy
log4j2.appender.rolling.strategy.max = 20

# Audit file appender
log4j2.appender.audit.type = RollingRandomAccessFile
log4j2.appender.audit.name = AuditRollingFile
log4j2.appender.audit.fileName = ${karaf.data}/security/audit.log
log4j2.appender.audit.filePattern = ${karaf.data}/security/audit.log.%i
log4j2.appender.audit.append = true
log4j2.appender.audit.layout.type = PatternLayout
log4j2.appender.audit.layout.pattern = ${log4j2.pattern}
log4j2.appender.audit.policies.type = Policies
log4j2.appender.audit.policies.size.type = SizeBasedTriggeringPolicy
```

```
log4j2.appender.audit.policies.size.size = 8MB

# OSGi appender
log4j2.appender.osgi.type = PaxOsgi
log4j2.appender.osgi.name = PaxOsgi
log4j2.appender.osgi.filter = *

#
# Loggers configuration
#

# Root logger
log4j2.rootLogger.level = INFO
log4j2.rootLogger.appenderRef.RollingFile.ref = RollingFile
log4j2.rootLogger.appenderRef.PaxOsgi.ref = PaxOsgi
log4j2.rootLogger.appenderRef.Console.ref = Console
log4j2.rootLogger.appenderRef.Console.filter.threshold.type = ThresholdFilter
log4j2.rootLogger.appenderRef.Console.filter.threshold.level = ${karaf.log.console:-OFF}
#log4j2.rootLogger.appenderRef.Sift.ref = Routing

# Spifly logger
log4j2.logger.spifly.name = org.apache.aries.spifly
log4j2.logger.spifly.level = WARN

# Security audit logger
log4j2.logger.audit.name = org.apache.karaf.jaas.modules.audit
log4j2.logger.audit.level = INFO
log4j2.logger.audit.additivity = false
log4j2.logger.audit.appenderRef.AuditRollingFile.ref = AuditRollingFile

# help with identification of maven-related problems with pax-url-aether
#log4j2.logger.aether.name = shaded.org.eclipse.aether
#log4j2.logger.aether.level = TRACE
#log4j2.logger.http-headers.name = shaded.org.apache.http.headers
#log4j2.logger.http-headers.level = DEBUG
#log4j2.logger.maven.name = org.ops4j.pax.url.mvn
#log4j2.logger.maven.level = TRACE
```

기본 구성은 파일 ***appender****를 사용하여* ***INFO*** *로그 수준으로* ***ROOT*** *로거를 정의합니다. 로그 수준을* ***Log4j2*** *유효한 값으로 변경할 수 있습니다. 가장 상세 정보에서 최소 상세 정보까지* ***TRACE, DEBUG, INFO, ERROR*** *또는* ***FATAL****을 지정할 수 있습니다.*

OSGi appender

osgi:* appender*는 로그 메시지를* ***OSGi*** *로그 서비스로 보내는 특수* ***appender****입니다.*

stdout appender

stdout *콘솔* ***appender****는 사전 구성되었지만 기본적으로 활성화되어 있지 않습니다. 이* ***appender****를 사용하면 로그 메시지를 표준 출력에 직접 표시할 수 있습니다.* ***Apache Karaf****를 서버 모드에서 실행하려는 경우 콘솔 없이 유용할 수 있습니다.*

이를 활성화하려면 **stdout appender**를 **rootLogger** 에 추가해야 합니다.

```
log4j2.rootLogger=INFO, out, stdout, osgi:*
```

out appender

out appender는 기본 항목입니다. **10 1MB** 로그 파일을 유지 관리하고 순환하는 롤링 파일 **appender**입니다. 로그 파일은 기본적으로 **data/log/fuse.log** 에 있습니다.

Sift appender

sift appender 는 기본적으로 활성화되어 있지 않습니다. 이 **appender**를 사용하면 배포된 번들 당 하나의 로그 파일을 사용할 수 있습니다. 기본적으로 로그 파일 이름 형식은 번들 심볼릭 이름(데이터/로그 폴더)을 사용합니다. 런타임 시 이 파일을 편집할 수 있습니다. **Apache Karaf**는 파일을 다시 로드하고 변경 사항이 적용됩니다. **Apache Karaf**를 다시 시작할 필요가 없습니다. 다른 설정 파일은 **Apache Karaf: etc/org.apache.karaf.log.cfg** 에서 사용합니다. 이 파일은 로그 명령에서 사용하는 로그 서비스를 구성합니다(나중에 참조).

JDBC appender

jdbc appender에는 **lazy** 플래그가 있으며, **true** (활성화)인 경우 데이터 소스를 사용할 수 없는 경우 데이터베이스에 로깅이 추가되지 않습니다. 그러나 **jndi**, 데이터 소스 또는 연결이 다시되면 로깅이 다시 시작됩니다.

```
log4j2.appender.jdbc.cs.lazy = true
```

중요

로깅 메시지가 손실되지 않도록 하려면 긴급 추가 기능도 다시 구성하는 것이 좋습니다.

20.2. 명령

etc/org.ops4j.pax.logging.cfg 파일을 변경하는 대신 **Apache Karaf**는 로그 구성을 동적으로 변경하고 로그 콘텐츠를 볼 수 있는 명령 세트를 제공합니다.

20.2.1. log:clear

log:clear 명령은 로그 항목을 지웁니다.

20.2.2. log:display

log:display 명령은 로그 항목을 표시합니다.

기본적으로 **rootLogger** 의 로그 항목을 표시합니다.

```
karaf@root()> log:display
2015-07-01 19:12:46,208 | INFO  | FelixStartLevel  | SecurityUtils                    | 16 -
org.apache.sshd.core - 0.12.0 | BouncyCastle not registered, using the default JCE provider
2015-07-01 19:12:47,368 | INFO  | FelixStartLevel  | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Starting JMX OSGi agent
```

로거 인수를 사용하여 특정 로거의 로그 항목을 표시할 수도 있습니다.

```
karaf@root()> log:display ssh
2015-07-01 19:12:46,208 | INFO  | FelixStartLevel  | SecurityUtils                    | 16 -
org.apache.sshd.core - 0.12.0 | BouncyCastle not registered, using the default JCE provider
```

기본적으로 모든 로그 항목이 표시됩니다. **Apache Karaf** 컨테이너가 장기 실행 중인 경우 매우 길 수 있습니다. **n** 옵션을 사용하여 표시할 항목 수를 제한할 수 있습니다.

```
karaf@root()> log:display -n 5
2015-07-01 06:53:24,143 | INFO  | JMX OSGi Agent   | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.framework.BundleStateMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=bundleState,version=1.7,framework=org.apache.felix.framework,uuid=5335370f-
9dee-449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,150 | INFO  | JMX OSGi Agent   | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.framework.PackageStateMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=packageState,version=1.5,framework=org.apache.felix.framework,uuid=5335370f-
9dee-449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,150 | INFO  | JMX OSGi Agent   | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.framework.ServiceStateMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=serviceState,version=1.7,framework=org.apache.felix.framework,uuid=5335370f-
9dee-449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,152 | INFO  | JMX OSGi Agent   | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering
org.osgi.jmx.framework.wiring.BundleWiringStateMBean to MBeanServer
com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.core:type=wiringState,version=1.1,framework=org.apache.felix.framework,uuid=5335370f-9dee-
449f-9b1c-cabe74432ed1
2015-07-01 06:53:24,501 | INFO  | FelixStartLevel  | RegionsPersistenceImpl           | 78 -
org.apache.karaf.region.persist - 4.0.0 | Loading region digraph persistence
```

etc/org.apache.karaf.log.cfg 파일의 **size** 속성을 사용하여 저장 및 유지되는 항목 수를 제한할 수도 있습니다.

```
#
# The number of log statements to be displayed using log:display. It also defines the number
# of lines searched for exceptions using log:display exception. You can override this value
# at runtime using -n in log:display.
#
size = 500
```

기본적으로 각 로그 수준은 다른 색상으로 표시됩니다. **ERROR/FATAL**은 빨간색, **DEBUG in purple, INFO in cyan** 등입니다. **no-color** 옵션을 사용하여 색상을 비활성화할 수 있습니다.

로그 항목 형식 패턴은 **etc/org.ops4j.pax.logging.cfg** 파일에 정의된 변환 패턴을 사용하지 않습니다. 기본적으로 **etc/org.apache.karaf.log.cfg** 에 정의된 **pattern** 속성을 사용합니다.

```
#
# The pattern used to format the log statement when using log:display. This pattern is according
# to the log4j2 layout. You can override this parameter at runtime using log:display with -p.
#
pattern = %d{ISO8601} | %-5.5p | %-16.16t | %-32.32c{1} | %X{bundle.id} - %X{bundle.name} -
%X{bundle.version} | %m%n
```

또한 **-p** 옵션을 사용하여 동적으로(한 번 실행) 패턴을 변경할 수도 있습니다.

```
karaf@root()> log:display -p "\%d - \%c - \%m\%n"
2015-07-01 07:01:58,007 - org.apache.sshd.common.util.SecurityUtils - BouncyCastle not registered,
using the default JCE provider
2015-07-01 07:01:58,725 - org.apache.aries.jmx.core - Starting JMX OSGi agent
2015-07-01 07:01:58,744 - org.apache.aries.jmx.core - Registering MBean with ObjectName
[osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=6361fc65-
8df4-4886-b0a6-479df2d61c83] for service with service.id [13]
2015-07-01 07:01:58,747 - org.apache.aries.jmx.core - Registering
org.osgi.jmx.service.cm.ConfigurationAdminMBean to MBeanServer
com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=6361fc65-
8df4-4886-b0a6-479df2d61c83
```

패턴은 날짜에 **%d**, 클래스의 **%c**, 로그 메시지의 경우 **%m**과 같은 키워드를 사용할 수 있는 일반 **Log4j2** 패턴입니다.

20.2.3. log:exception-display

log:exception-display 명령은 마지막으로 발생한 예외를 표시합니다.

log:display 명령과 마찬가지로 **log:exception-display** 명령은 기본적으로 **rootLogger** 를 사용하지만 **logger** 인수를 사용하여 로거를 지정할 수 있습니다.

20.2.4. log:get

log:get 명령은 로거의 현재 로그 수준을 표시합니다.

기본적으로 표시된 로그 수준은 루트 로거에서 가져온 것입니다.

```
karaf@root()> log:get
Logger                         | Level
-------------------------------------------------------------------------------
ROOT                           | INFO
org.apache.aries.spifly        | WARN
org.apache.karaf.jaas.modules.audit | INFO
org.apache.sshd                | INFO
```

로거 인수를 사용하여 특정 로거 를 지정할 수 있습니다.

```
karaf@root()> log:get ssh
INFO
```

logger 인수는 **ALL** 키워드를 사용하여 모든 로거의 로그 수준을 목록으로 표시합니다.

예를 들어 **etc/org.ops4j.pax.logging.cfg** 파일에서 다음과 같이 자체 로거를 정의했습니다.

```
log4j2.logger.my.name = MyLogger
log4j2.logger.my.level = DEBUG
```

해당 로그 수준으로 로거 목록을 확인할 수 있습니다.

```
karaf@root()> log:get ALL
Logger                         | Level
-------------------------------------------------------------------------------
MyLogger                       | DEBUG
ROOT                           | INFO
org.apache.aries.spifly        | WARN
org.apache.karaf.jaas.modules.audit | INFO
org.apache.sshd                | INFO
```

log:list 명령은 log:get ALL 의 별칭입니다.

20.2.5. log:log

log:log 명령을 사용하면 로그에 메시지를 수동으로 추가할 수 있습니다. Apache Karaf 스크립트를 만들 때 매우 유용합니다.

```
karaf@root()> log:log "Hello World"
karaf@root()> log:display
12:55:21.706 INFO [pipe-log:log "Hello World"] Hello World
```

기본적으로 로그 수준은 INFO이지만 -l 옵션을 사용하여 다른 로그 수준을 지정할 수 있습니다.

```
karaf@root()> log:clear
karaf@root()> log:log -l ERROR "Hello World"
karaf@root()> log:display
12:55:41.460 ERROR [pipe-log:log "Hello World"] Hello World
```

20.2.6. log:set

log:set 명령은 로거의 로그 수준을 설정합니다.

기본적으로 rootLogger 의 로그 수준을 변경합니다.

```
karaf@root()> log:set DEBUG
karaf@root()> log:get
Logger                    | Level
--------------------------------------------------------------------------------
ROOT                      | DEBUG
...
```

수준 1 뒤에 로거 인수를 사용하여 특정 로거 를 지정할 수 있습니다.

```
karaf@root()> log:set INFO my.logger
karaf@root()> log:get my.logger
Logger    | Level
-----------------
my.logger | INFO
```

level 인수는 Log4j2 로그 수준인 TRACE, DEBUG, INFO, WARN, ERROR, FATAL을 허용합니다.

또한 DEFAULT 특수 키워드도 허용합니다.

DEFAULT 키워드의 목적은 로거 상위 수준의 로거를 사용하기 위해 로거의 현재 수준을 삭제하는 것입니다(및 수준만 수준만, appender와 같은 다른 속성은 삭제되지 않습니다).

예를 들어 다음 로거(etc/org.ops4j.pax.logging.cfg 파일)를 정의했습니다.

```
rootLogger=INFO,out,osgi:*
my.logger=INFO,appender1
my.logger.custom=DEBUG,appender2
```

my.logger.custom 로거의 수준을 변경할 수 있습니다.

```
karaf@root()> log:set INFO my.logger.custom
```

이제 다음을 수행합니다.

```
rootLogger=INFO,out,osgi:*
my.logger=INFO,appender1
my.logger.custom=INFO,appender2
```

my.logger.custom 로거에서 DEFAULT 키워드를 사용하여 수준을 제거할 수 있습니다.

```
karaf@root()> log:set DEFAULT my.logger.custom
```

이제 다음을 수행합니다.

```
rootLogger=INFO,out,osgi:*
my.logger=INFO,appender1
my.logger.custom=appender2
```

즉, 런타임에 my.logger.custom 로거는 상위 my.logger의 수준을 사용하므로 INFO 임을 의미합니다.

*이제 **my.logger** 로거와 함께 **DEFAULT** 키워드를 사용하는 경우:*

```
karaf@root()> log:set DEFAULT my.logger
```

***Red Hat**은 다음과 같습니다.*

```
rootLogger=INFO,out,osgi:*
my.logger=appender1
my.logger.custom=appender2
```

*따라서 **my.logger.custom** 및 **my.logger** 모두 상위 **rootLogger** 의 로그 수준을 사용합니다.*

***rootLogger** 와 함께 **DEFAULT** 키워드를 사용할 수 없으며 부모가 없습니다.*

20.2.7. log:tail

***log:tail** 은 **log:display** 와 정확히 동일하지만 로그 항목을 지속적으로 표시합니다.*

***log:display** 명령과 동일한 옵션 및 인수를 사용할 수 있습니다.*

*기본적으로 **rootLogger** 의 항목을 표시합니다.*

```
karaf@root()> log:tail
2015-07-01 07:40:28,152 | INFO  | FelixStartLevel  | SecurityUtils                    | 16 -
org.apache.sshd.core - 0.9.0 | BouncyCastle not registered, using the default JCE provider
2015-07-01 07:40:28,909 | INFO  | FelixStartLevel  | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Starting JMX OSGi agent
2015-07-01 07:40:28,928 | INFO  | FelixStartLevel  | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering MBean with ObjectName
[osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=b44a44b7-
41cd-498f-936d-3b12d7aafa7b] for service with service.id [13]
2015-07-01 07:40:28,936 | INFO  | JMX OSGi Agent   | core                             | 68 -
org.apache.aries.jmx.core - 1.1.1 | Registering org.osgi.jmx.service.cm.ConfigurationAdminMBean to
MBeanServer com.sun.jmx.mbeanserver.JmxMBeanServer@27cc75cb with name
osgi.compendium:service=cm,version=1.3,framework=org.apache.felix.framework,uuid=b44a44b7-
41cd-498f-936d-3b12d7aafa7b
```

***log:tail** 명령을 종료하려면 **CTRL-C**를 입력합니다.*

20.3. CRYOSTAT LOGMBEAN

log:* 명령으로 수행할 수 있는 모든 작업은 LogMBean을 사용하여 수행할 수 있습니다.

LogMBean 오브젝트 이름은 org.apache.karaf:type=log,name=* 입니다.

20.3.1. 속성

- level 속성은 루트 로거의 수준입니다.

20.3.2. 작업

- getLevel(logger) - 특정 로거의 로그 수준을 가져옵니다. 이 작업은 ALL 키워드를 지원하므로 각 로거의 수준이 포함된 맵을 반환합니다.

- setLevel(level, logger) 을 설정하여 특정 로거의 로그 수준을 설정합니다. 이 작업은 log:set 명령에 대해 DEFAULT 키워드를 지원합니다.

20.4. 고급 구성

20.4.1. SIFT 로깅

Fuse-Karaf는 Log4j2 sift appender의 샘플(기본적으로 설명) 구성과 $FUSE_HOME/etc/org.ops4j.pax.logging.cfg 파일에서 이 appender를 사용하는 로거를 제공합니다.

```
# Sift appender
log4j2.appender.mdc.type = Routing
log4j2.appender.mdc.name = SiftAppender
log4j2.appender.mdc.routes.type = Routes
# see: http://logging.apache.org/log4j/2.x/manual/appenders.html#Routes
log4j2.appender.mdc.routes.pattern = $\\{ctx:bundle.name}
log4j2.appender.mdc.routes.sift.type = Route
log4j2.appender.mdc.routes.sift.appender.type = RollingRandomAccessFile
log4j2.appender.mdc.routes.sift.appender.name = RollingFile
log4j2.appender.mdc.routes.sift.appender.fileName = ${karaf.data}/log/sift-$\\{ctx:bundle.name}.log
log4j2.appender.mdc.routes.sift.appender.filePattern = ${karaf.data}/log/sift-$\\{ctx:bundle.name}-
%i.log.gz
log4j2.appender.mdc.routes.sift.appender.append = true
log4j2.appender.mdc.routes.sift.appender.layout.type = PatternLayout
log4j2.appender.mdc.routes.sift.appender.layout.pattern = ${log4j2.pattern}
log4j2.appender.mdc.routes.sift.appender.policies.type = Policies
```

```
log4j2.appender.mdc.routes.sift.appender.policies.size.type = SizeBasedTriggeringPolicy
log4j2.appender.mdc.routes.sift.appender.policies.size.size = 16MB
log4j2.appender.mdc.routes.sift.appender.strategy.type = DefaultRolloverStrategy
log4j2.appender.mdc.routes.sift.appender.strategy.max = 20
...
# sample logger using Sift appender
#log4j2.logger.example.name = org.apache.camel
#log4j2.logger.example.level = INFO
#log4j2.logger.example.appenderRef.SiftAppender.ref = SiftAppender
```

구성은 http://logging.apache.org/log4j/2.x/manual/appenders.html#RoutingAppender에 설명되어 있습니다.

***SIFT/Routing appender의 pattern** 속성은 로깅의 대상 위치를 구분하는 데 사용할 수 있는 것입니다.*

다음은 다양한 조회를 사용할 수 있으며 여기에 설명되어 있습니다.
http://logging.apache.org/log4j/2.x/manual/lookups.html

*가장 중요한 조회는 **ThreadContext** 맵(**.k.a**)에서 값(**keys**)을 조회하는 **ctx** 입니다. **MDC**).*

***Fuse Karaf**에서 제공하는 기본 구성은 **ctx:bundle.name**을 패턴으로 사용합니다. 즉, 다음과 같습니다.*

```
lookup bundle.name key in MDC
```

***bundle.** 접두사가 지정된 키는 **pax-logging** 자체에서 제공하며 다음 **3**가지 값 중에서 선택할 수 있습니다.*

- ***bundle.name == org.osgi.framework.Bundle.getSymbolicName()***
- ***bundle.id == org.osgi.framework.Bundle.getBundleId()***
- ***bundle.version == org.osgi.framework.Bundle.getVersion().toString()***

*그러나 다음을 사용하여 **MDC** 지원을 통해 **Camel** 컨텍스트가 생성되는 경우(파란트 **XML DSL**).*

```
<camelContext id="my-context" xmlns="http://camel.apache.org/schema/blueprint"
useMDCLogging="true">
```

MDC/ThreadContext에서 사용할 수 있는 더 많은 키가 있으며 **SIFT appender** 구성에서 패턴으로 사용할 수 있습니다.

- **camel.exchangeId - The exchange id**
- **Camel.messageId -** 메시지 **ID**
- **Camel.correlationId -** 상관 관계가 있는 경우 교환의 상관관계 **ID**입니다. 예를 들어 **Splitter EIP**에서 하위 메시지
- **Camel. CryostatKey -** 트랜잭션된 교환에 대한 트랜잭션 **ID**입니다. **id**는 고유하지 않지만 지정된 트랜잭션의 트랜잭션 경계를 표시하는 트랜잭션 템플릿의 **ID**입니다. 따라서 이 사실을 가리키기 위해 **TransactionID**가 아닌 키의 이름을 지정했습니다.
- **Camel.routeId -** 교환이 현재 라우팅되고 있는 경로의 **ID**
- **Camel.breadcrumbId -** 전송 간 메시지를 추적하는 데 사용되는 고유 **ID**입니다.
- **Camel.contextId -** 다른 카멜 컨텍스트의 메시지를 추적하는 데 사용되는 **camel** 컨텍스트 **ID**입니다.

https://people.apache.org/~dkulp/camel/mdc-logging.html에서 참조하십시오.

예를 들어 로깅 대상 파일을 **Camel**의 경로 **ID**로 구분하려면 다음을 사용하십시오.

```
log4j2.appender.mdc.routes.pattern = $\\{ctx:camel.routeId}
...
log4j2.appender.mdc.routes.sift.appender.fileName = ${karaf.data}/log/sift-$\\{ctx:camel.routeId}.log
log4j2.appender.mdc.routes.sift.appender.filePattern = ${karaf.data}/log/sift-$\\{ctx:camel.routeId}-
%i.log.gz
```

한 가지 더 **-** 추가 구성만으로는 충분하지 않습니다. 일부 로거에 연결해야 합니다. 다시 한 번 샘플 구

성에는 다음이 포함됩니다.

```
# sample logger using Sift appender
#log4j2.logger.example.name = org.apache.camel
#log4j2.logger.example.level = INFO
#log4j2.logger.example.appenderRef.SiftAppender.ref = SiftAppender
```

*(**log4j2.logger.example.appenderRef.SiftAppender.ref** 속성의 **SiftAppender** 값은 **appender** 구성의 **log4j2.appender.mdc.name** 값과 일치해야 합니다).*

*여기서 **org.apache.camel**은 로거 이름(또는 카테고리 이름)입니다. 이는 **Camel**의 로그에 사용되는 끝점과 정확히 동일합니다. **If you have (in Camel route):***

```
<to uri="log:org.apache.camel" />
```

로깅이 작동했습니다.

다른 작업 구성은 다음과 같습니다.

```
<to uri="log:my-special-logger" />
```

및 다음을 수행합니다.

```
log4j2.logger.example.name = my-special-logger
log4j2.logger.example.level = DEBUG
log4j2.logger.example.appenderRef.SiftAppender.ref = SiftAppender
```

20.4.2. 필터

***appender**에 필터를 적용할 수 있습니다. 필터는 각 로그 이벤트를 평가하고 로그로 보낼지 여부를 결정합니다.*

***Log4j2**는 필터를 사용할 준비가 되어 있습니다.*

참고

이에 대한 포괄적인 보기는 **Log4J** 사이트의 필터 를 참조하십시오.

20.4.3. 중첩된 appender

중첩된 **appender**는 다른 **appender**를 "내부"하는 특수한 종류의 **appender**입니다. 이를 통해 **appender** 체인 사이에 일종의 "라우팅"을 생성할 수 있습니다.

가장 많이 사용되는 **"nested compliant" appender**는 다음과 같습니다.

- **AsyncAppender(org.apache.log4j2.AsyncAppender)**는 이벤트를 비동기적으로 기록합니다. 이 **appender**는 이벤트를 수집하여 연결된 모든 부록에 전달합니다.
- **RewriteAppender(org.apache.log4j2.rewrite.RewriteAppender)**는 로그 이벤트를 다시 작성한 후 다른 **appender**로 로그 이벤트를 전달합니다.

이러한 종류의 **appender**는 **appender** 정의에서 **appenders** 속성을 허용합니다.

```
log4j2.appender.[appender-name].appenders=[comma-separated-list-of-appender-names]
```

예를 들어 **async** 라는 **Async** 및 비동기적으로 로그 이벤트를 **JMS appender**에 디스패치하는 **AsyncAppender**를 생성할 수 있습니다.

```
log4j2.appender.async=org.apache.log4j2.AsyncAppender
log4j2.appender.async.appenders=jms

log4j2.appender.jms=org.apache.log4j2.net.JMSAppender
...
```

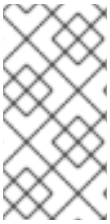
20.4.4. 오류 처리기

경우에 따라 **appender**가 실패할 수 있습니다. 예를 들어 **RollingFileAppender** 는 파일 시스템에 작성하려고 하지만 파일 시스템이 가득거나 **JMS appender**가 메시지를 전송하려고 하지만 **JMS** 브로커를 사용할 수 없습니다.

로그 **appender**가 실패하는지 확인하는 것이 중요하므로 로깅이 중요할 수 있습니다.

각 로그 **appender**는 오류 처리기에 오류 처리를 위임하여 **appender** 오류에 대응할 수 있는 기회를 제공합니다.

- **CryostatAppender(org.apache.log4j2.varia.FailoverAppender)**를 사용하면 기본 추가 기능이 실패할 경우 보조 첨부자가 대신할 수 있습니다. 오류 메시지가 **System.err** 에 출력되고 보조 첨부 파일에 기록됩니다.



참고

Cryostat Appender에 대한 자세한 내용은 **Log4j2의 Apppender 페이지로 이동합니다**.

appender 정의 자체에서 **errorhandler** 속성을 사용하여 각 **appender**에 사용할 오류 처리기 를 정의할 수 있습니다.

```
log4j2.appender.[appender-name].errorhandler=[error-handler-class]
log4j2.appender.[appender-name].errorhandler.root-ref=[true|false]
log4j2.appender.[appender-name].errorhandler.logger-ref=[logger-ref]
log4j2.appender.[appender-name].errorhandler.appender-ref=[appender-ref]
```

20.4.5. OSGi 특정 MDC 속성

라우팅 부록 은 **MDC(Mapped Diagnostic Context)** 속성을 기반으로 로그 이벤트를 분할할 수 있는 **OSGi** 지향 **appender**입니다.

MDC를 사용하면 다양한 로그 이벤트 소스를 구분할 수 있습니다.

라우팅 **appender**는 기본적으로 **OSGi** 지향 **MDC** 속성을 제공합니다.

- **bundle.id** 는 번들 **ID**입니다.
- **bundle.name** 은 번들 심볼릭 이름입니다.

- ***bundle.version*** *은 번들 버전입니다.*

이러한 ***MDC*** *속성을 사용하여 번들당 로그 파일을 생성할 수 있습니다.*

```
log4j2.appender.routing.type = Routing
log4j2.appender.routing.name = Routing
log4j2.appender.routing.routes.type = Routes
log4j2.appender.routing.routes.pattern = $$\\{ctx:bundle.name\\}
log4j2.appender.routing.routes.bundle.type = Route
log4j2.appender.routing.routes.bundle.appender.type = RollingRandomAccessFile
log4j2.appender.routing.routes.bundle.appender.name = Bundle-$\\{ctx:bundle.name\}
log4j2.appender.routing.routes.bundle.appender.fileName = ${karaf.data}/log/bundle-$\\
{ctx:bundle.name\\}.log
log4j2.appender.routing.routes.bundle.appender.filePattern = ${karaf.data}/log/bundle-$\\
{ctx:bundle.name\\}.log.%d{yyyy-MM-dd}
log4j2.appender.routing.routes.bundle.appender.append = true
log4j2.appender.routing.routes.bundle.appender.layout.type = PatternLayout
log4j2.appender.routing.routes.bundle.appender.policies.type = Policies
log4j2.appender.routing.routes.bundle.appender.policies.time.type = TimeBasedTriggeringPolicy
log4j2.appender.routing.routes.bundle.appender.strategy.type = DefaultRolloverStrategy
log4j2.appender.routing.routes.bundle.appender.strategy.max = 31

log4j2.rootLogger.appenderRef.Routing.ref = Routing
```

20.4.6. 향상된 ***OSGi*** *스택 추적 렌더러*

기본적으로 ***Apache Karaf****는 특정 스택 추적 렌더러를 제공하여 일부* ***OSGi*** *특정 정보를 추가합니다.*

스택 추적에서 예외를 ***throw****하는 클래스 외에도 각 스택 추적 라인 끝에* ***[id:name:version]*** *패턴을 찾을 수 있습니다.*

- *번들* ***ID****입니다.*
- *번들 이름입니다.*
- ***version*** *은* ***bundle*** *버전입니다.*

문제의 원인을 진단하는 것이 매우 유용합니다.

*예를 들어 다음 **IllegalArgumentException** 스택 추적에서는 예외 소스에 대한 **OSGi** 세부 정보를 볼 수 있습니다.*

```
java.lang.IllegalArgumentException: Command not found:  *:foo
 at org.apache.felix.gogo.runtime.shell.Closure.execute(Closure.java:225)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.felix.gogo.runtime.shell.Closure.executeStatement(Closure.java:162)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.felix.gogo.runtime.shell.Pipe.run(Pipe.java:101)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.felix.gogo.runtime.shell.Closure.execute(Closure.java:79)
[21:org.apache.karaf.shell.console:4.0.0]
 at
org.apache.felix.gogo.runtime.shell.CommandSessionImpl.execute(CommandSessionImpl.java:71)
[21:org.apache.karaf.shell.console:4.0.0]
 at org.apache.karaf.shell.console.jline.Console.run(Console.java:169)
[21:org.apache.karaf.shell.console:4.0.0]
 at java.lang.Thread.run(Thread.java:637)[:1.7.0_21]
```

20.4.7. 사용자 정의 appender

***Apache Karaf**에서 고유한 **appender**를 사용할 수 있습니다.*

*가장 쉬운 방법은 **appender**를 **OSGi** 번들로 패키지하고 **org.ops4j.pax.logging.pax-logging-service** 번들의 조각으로 연결하는 것입니다.*

*예를 들어 **MyAppender** 를 만듭니다.*

```
public class MyAppender extends AppenderSkeleton {
...
}
```

*다음과 같이 **MANIFEST**가 포함된 **OSGi** 번들로 컴파일하고 패키징합니다.*

```
Manifest:
Bundle-SymbolicName: org.mydomain.myappender
Fragment-Host: org.ops4j.pax.logging.pax-logging-service
...
```

***Apache Karaf** 시스템 폴더에 번들을 복사합니다. 시스템 폴더는 표준 **Maven** 디렉터리 레이아웃인 **groupId/artifactId/version**을 사용합니다.*

etc/startup.properties 구성 파일에서 pax-logging-service 번들 전에 목록에 번들을 정의합니다.

시스템 번들을 다시 로드하려면 새로 실행(데이터 폴더 수행)을 사용하여 Apache Karaf를 다시 시작해야 합니다. 이제 etc/org.ops4j.pax.logging.cfg 구성 파일에서 직접 appender를 사용할 수 있습니다.

21장. 보안

Apache Karaf는 OSGi 호환 방식으로 JAAS(Java Authentication and Authorization Service)를 제공하는 고급적이고 유연한 보안 시스템을 제공합니다.

동적 보안 시스템을 제공합니다.

Apache Karaf 보안 프레임워크는 다음에 대한 액세스를 제어하기 위해 내부적으로 사용됩니다.

- OSGi 서비스(개발 가이드에 설명되어 있음)
- 콘솔 명령
- Cryostat 계층
- WebConsole

애플리케이션은 보안 프레임워크를 사용할 수도 있습니다(자세한 내용은 개발자 가이드 참조).

21.1. REALMS

Apache Karaf는 여러 영역을 관리할 수 있습니다. 영역에는 이 영역에서 인증 및/또는 권한 부여에 사용할 로그인 모듈 정의가 포함되어 있습니다. 로그인 모듈은 영역에 대한 인증 및 권한 부여를 정의합니다.

jaas:realm-list 명령은 현재 정의된 영역을 나열합니다.

```
karaf@root()> jaas:realm-list
Index | Realm Name | Login Module Class Name
---------------------------------------------------------------------------------
1     | karaf      | org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
2     | karaf      | org.apache.karaf.jaas.modules.publickey.PublickeyLoginModule
```

Apache Karaf가 기본 영역 karaf 를 제공하는 것을 확인할 수 있습니다.

이 영역에는 두 가지 로그인 모듈이 있습니다.

- PropertiesLoginModule 은 etc/users.properties 파일을 사용자, 그룹, 역할 및 암호의 백엔드로 사용합니다. 이 로그인 모듈은 사용자를 인증하고 사용자의 역할을 반환합니다.

- PublickeyLoginModule 은 SSHd에서 특히 사용합니다. etc/keys.properties 파일을 사용합니다. 이 파일에는 각 사용자와 연결된 공개 키가 포함되어 있습니다.

Apache Karaf는 추가 로그인 모듈을 제공합니다(자세한 내용은 개발자 가이드 참조).

- JDBCLoginModule은 데이터베이스를 백엔드로 사용

- LdapLoginModule은 LDAP 서버를 백엔드로 사용

- SyncopeLoginModule은 Apache Syncope를 백엔드로 사용

- OsgiConfigLoginModule은 구성을 백엔드로 사용

- Krb5LoginModule은 Kerberos 서버를 백엔드로 사용

- GSSAPILdapLoginModule은 LDAP 서버를 백엔드로 사용하지만 LDAP 서버 인증을 다른 백엔드(일반적으로 Krb5LoginModule)에 위임합니다.

기존 realm, login 모듈을 관리하거나 jaas:realm-manage 명령을 사용하여 고유한 영역을 생성할 수 있습니다.

21.1.1. 사용자, 그룹, 역할 및 암호

*지금까지 본 것처럼 **Apache Karaf**는 **PropertiesLoginModule**을 사용합니다.*

*이 로그인 모듈은 **etc/users.properties** 파일을 사용자, 그룹, 역할 및 암호의 스토리지로 사용합니다.*

*초기 **etc/users.properties** 파일에는 다음이 포함됩니다.*

```
################################################################################
#
#    Licensed to the Apache Software Foundation (ASF) under one or more
#    contributor license agreements.  See the NOTICE file distributed with
#    this work for additional information regarding copyright ownership.
#    The ASF licenses this file to You under the Apache License, Version 2.0
#    (the "License"); you may not use this file except in compliance with
#    the License.  You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#    Unless required by applicable law or agreed to in writing, software
#    distributed under the License is distributed on an "AS IS" BASIS,
#    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#    See the License for the specific language governing permissions and
#    limitations under the License.
#
################################################################################

#
# This file contains the users, groups, and roles.
# Each line has to be of the format:
#
# USER=PASSWORD,ROLE1,ROLE2,...
# USER=PASSWORD,_g_:GROUP,...
# _g_\:GROUP=ROLE1,ROLE2,...
#
# All users, grousp, and roles entered in this file are available after Karaf startup
# and modifiable via the JAAS command group. These users reside in a JAAS domain
# with the name "karaf".
#
karaf = karaf,_g_:admingroup
_g_\:admingroup = group,admin,manager,viewer
```

*이 파일에서는 기본적으로 한 명의 사용자가 있습니다. **karaf**. 기본 암호는 **karaf** 입니다.*

***karaf** 사용자는 하나의 그룹인 **admingroup** 의 멤버입니다.*

그룹은 항상 **g:** 접두사가 지정됩니다. 이 접두사가 없는 항목은 사용자입니다.

그룹은 역할 세트를 정의합니다. 기본적으로 **admin group** 은 그룹 **,admin,manager, viewer** 역할을 정의합니다.

즉, **karaf** 사용자는 **admingroup** 에서 정의한 역할을 갖습니다.

21.1.1.1. 명령

jaas:* 명령은 콘솔에서 영역, 사용자, 그룹, 역할을 관리합니다.

21.1.1.1.1. jaas:realm-list

이전에 이 섹션에서 **jaas:realm-list** 를 사용했습니다.

jaas:realm-list 명령은 각 영역의 영역 및 로그인 모듈을 나열합니다.

```
karaf@root()> jaas:realm-list
Index | Realm Name | Login Module Class Name
---------------------------------------------------------------------------------
1     | karaf      | org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
2     | karaf      | org.apache.karaf.jaas.modules.publickey.PublickeyLoginModule
```

여기에 두 개의 로그인 모듈 **(PropertiesLoginModule** 및 **PublickeyLoginModule)**을 포함하는 하나의 영역 **(karaf)**이 있습니다.

인덱스 는 **jaas:realm-manage** 명령에서 사용할 **realm/login** 모듈을 쉽게 식별할 수 있습니다.

21.1.1.1.2. jaas:realm-manage

jaas:realm-manage 명령은 **login** 모듈에서 사용자, 그룹 및 역할을 관리할 수 있는 **realm/login** 모듈 편집 모드에서 전환합니다.

관리할 **realm** 및 로그인 모듈을 식별하려면 **--index** 옵션을 사용하면 됩니다. 인덱스는 **jaas:realm-list** 명령으로 표시됩니다.

-

```
karaf@root()> jaas:realm-manage --index 1
```

또 다른 방법은 **--realm** 및 **--module** 옵션을 사용하는 것입니다. **--realm** 옵션에는 **realm** 이름이 필요하며 **--module** 옵션에는 로그인 모듈 클래스 이름이 필요합니다.

```
karaf@root()> jaas:realm-manage --realm karaf --module
org.apache.karaf.jaas.modules.properties.PropertiesLoginModule
```

21.1.1.1.3. jaas:user-list

편집 모드에 있는 경우 **jaas:user-list:**을 사용하여 로그인 모듈의 사용자를 나열할 수 있습니다.

```
karaf@root()> jaas:user-list
User Name | Group      | Role
--------------------------------
karaf     | admingroup | admin
karaf     | admingroup | manager
karaf     | admingroup | viewer
```

역할별로 사용자 이름과 그룹을 확인할 수 있습니다.

21.1.1.1.4. jaas:user-add

jaas:user-add 명령은 현재 편집한 로그인 모듈에 새 사용자(및 암호)를 추가합니다.

```
karaf@root()> jaas:user-add foo bar
```

변경 사항(사용자 추가)을 "커밋"하려면 **jaas:update** 명령을 실행해야 합니다.

```
karaf@root()> jaas:update
karaf@root()> jaas:realm-manage --index 1
karaf@root()> jaas:user-list
User Name | Group      | Role
--------------------------------
karaf     | admingroup | admin
karaf     | admingroup | manager
karaf     | admingroup | viewer
foo       |            |
```

반면 사용자 추가를 롤백하려면 **jaas:cancel** 명령을 사용할 수 있습니다.

21.1.1.1.5. jaas:user-delete

jaas:user-delete 명령은 현재 편집된 로그인 모듈에서 사용자를 삭제합니다.

```
karaf@root()> jaas:user-delete foo
```

jaas:user-add 명령과 마찬가지로 **jaas:update** 를 사용하여 변경 사항(또는 **jaas:cancel to rollback)**을 커밋해야 합니다.

```
karaf@root()> jaas:update
karaf@root()> jaas:realm-manage --index 1
karaf@root()> jaas:user-list
User Name | Group      | Role
--------------------------------
karaf     | admingroup | admin
karaf     | admingroup | manager
karaf     | admingroup | viewer
```

21.1.1.1.6. jaas:group-add

jaas:group-add 명령은 현재 편집된 로그인 모듈에서 그룹을 할당하고 결국 그룹을 생성합니다.

```
karaf@root()> jaas:group-add karaf mygroup
```

21.1.1.1.7. JAAS:group-delete

jaas:group-delete 명령은 현재 편집된 로그인 모듈의 그룹에서 사용자를 제거합니다.

```
karaf@root()> jaas:group-delete karaf mygroup
```

21.1.1.1.8. jaas:group-role-add

jaas:group-role-add 명령은 현재 편집된 로그인 모듈의 그룹에 역할을 추가합니다.

```
karaf@root()> jaas:group-role-add mygroup myrole
```

21.1.1.1.9. jaas:group-role-delete

jaas:group-role-delete 명령은 현재 편집된 로그인 모듈의 그룹에서 역할을 제거합니다.

```
karaf@root()> jaas:group-role-delete mygroup myrole
```

21.1.1.1.10. jaas:update

jaas:update 명령은 로그인 모듈 백엔드에서 변경 사항을 커밋합니다. 예를 들어 **PropertiesLoginModule**의 경우 **etc/users.properties** 는 **jaas:update** 명령을 실행한 후에만 업데이트 됩니다.

21.1.1.1.11. jaas:cancel

jaas:cancel 명령은 변경 사항을 롤백하고 로그인 모듈 백엔드를 업데이트하지 않습니다.

21.1.2. 암호 암호화

기본적으로 암호는 **etc/users.properties** 파일에 명확한 형식으로 저장됩니다.

etc/org.apache.karaf.jaas.cfg 구성 파일에서 암호화를 활성화할 수 있습니다.

```
################################################################################
#
#    Licensed to the Apache Software Foundation (ASF) under one or more
#    contributor license agreements.  See the NOTICE file distributed with
#    this work for additional information regarding copyright ownership.
#    The ASF licenses this file to You under the Apache License, Version 2.0
#    (the "License"); you may not use this file except in compliance with
#    the License.  You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
#    Unless required by applicable law or agreed to in writing, software
#    distributed under the License is distributed on an "AS IS" BASIS,
#    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#    See the License for the specific language governing permissions and
#    limitations under the License.
#
################################################################################

#
# Boolean enabling / disabling encrypted passwords
#
encryption.enabled = false

#
# Encryption Service name
#   the default one is 'basic'
#   a more powerful one named 'jasypt' is available
```

```
#       when installing the encryption feature
#
encryption.name =

#
# Encryption prefix
#
encryption.prefix = {CRYPT}

#
# Encryption suffix
#
encryption.suffix = {CRYPT}

#
# Set the encryption algorithm to use in Karaf JAAS login module
# Supported encryption algorithms follow:
#   MD2
#   MD5
#   SHA-1
#   SHA-256
#   SHA-384
#   SHA-512
#
encryption.algorithm = MD5

#
# Encoding of the encrypted password.
# Can be:
#   hexadecimal
#   base64
#
encryption.encoding = hexadecimal
```

encryption.enabled y가 **true**로 설정되면 암호 암호화가 활성화됩니다.

암호화가 활성화되면 사용자가 처음 로그인할 때 암호가 암호화됩니다. 암호화된 암호는 접두사가 지정되고 **\{CRYPT\}**로 접미사가 지정됩니다. 암호를 다시 암호화하려면 **\{CRYPT\}** 접두사 및 접미사 없이 암호를 **clear(etc/users.properties** 파일)으로 재설정할 수 있습니다. **Apache Karaf**는 이 암호가 붙지 않고 **\{CRYPT\}**로 접두사가 지정되어 있음을 감지하고 다시 암호화합니다.

etc/org.apache.karaf.jaas.cfg 구성 파일을 사용하면 고급 암호화 동작을 정의할 수 있습니다.

- **encryption.prefix** 속성은 암호화된 암호로 "플래그"에 대한 접두사를 정의합니다. 기본값은 **\{CRYPT\}**입니다.
- **encryption.suffix** 속성은 암호로 암호화된 "플래그"에 대한 접미사를 정의합니다. 기본값은

\{CRYPT\}입니다.

- *encryption.algorithm 속성은 암호화(digest)에 사용할 알고리즘을 정의합니다. 가능한 값은 MD2,MD5,SHA-1,SHA -256,SHA-384,SHA-512 입니다. 기본값은 MD5 입니다.*

- *encryption.encoding 속성은 암호화된 암호의 인코딩을 정의합니다. 가능한 값은 16진수 또는 base64 입니다. 기본값은 16진수 입니다.*

21.1.3. 키로 인증 관리

SSH 계층의 경우 Karaf는 키로 인증을 지원하므로 암호를 제공하지 않고 로그인할 수 있습니다.

SSH 클라이언트(Karaf 자체에서 제공하는 bin/client 또는 OpenSSH와 같은 ssh 클라이언트)는 Karaf SSHD(서버 측)에서 담당자를 식별하는 공개/개인 키 쌍을 사용합니다.

연결할 수 있는 키는 형식에 따라 etc/keys.properties 파일에 저장됩니다.

```
user=key,role
```

기본적으로 Karaf는 karaf 사용자에 대한 키를 허용합니다.

```
#
karaf=AAAAB3NzaC1kc3MAAACBAP1/U4EddRIpUt9KnC7s5Of2EbdSPO9EAMMeP4C2USZpRV1AIlH
7WT2NWPq/xfW6MPbLm1Vs14E7gB00b/JmYLdrmVClpJ+f6AR7ECLCT7up1/63xhv4O1fnxqimFQ8E+
4P208Uewwl1VBNaFpEy9nXzrith1yrv8iIDGZ3RSAHHAAAAFQCXYFCPFSMLzLKSuYKi64QL8Fgc9Q
AAAIEA9+GghdabPd7LvKtcNrhXuXmUr7v6OuqC+VdMCz0HgmdRWVeOutRZT+ZxBxCBgLRJFnEj6E
woFhO3zwkyjMim4TwWeotUfI0o4KOuHiuzpnWRbqN/C/ohNWLx+2J6ASQ7zKTxvqhRkImog9/hWuWf
BpKLZl6Ae1UlZAFMO/7PSSoAAACBAKKSU2PFl/qOLxIwmBZPPIcJshVe7bVUpFvyl3BbJDow8rXfskl8
wO63OzP/qLmcJM0+JbcRU/53JjTuyk31drV2qxhIOsLDC9dGCWj47Y7TyhPdXh/0dthTRBy6bqGtRPxG
a7gJov1xm/UuYYXPIUR/3x9MAZvZ5xvE0kYXO+rx,admin
```

참고

보안상의 이유로 이 키는 비활성화되어 있습니다. 클라이언트당 키 쌍을 생성하고 etc/keys.properties 파일을 업데이트하는 것이 좋습니다.

키 쌍을 생성하는 가장 쉬운 방법은 OpenSSH를 사용하는 것입니다.

다음을 사용하여 키 쌍을 만들 수 있습니다.

```
ssh-keygen -t dsa -f karaf.id_dsa -N karaf
```

이제 공개 및 개인 키가 있습니다.

```
-rw-------  1 jbonofre jbonofre    771 Jul 25 22:05 karaf.id_dsa
-rw-r--r--  1 jbonofre jbonofre    607 Jul 25 22:05 karaf.id_dsa.pub
```

etc/keys.properties 에 있는 **karaf.id_dsa.pub** 파일의 내용을 복사할 수 있습니다.

```
karaf=AAAAB3NzaC1kc3MAAACBAJLj9vnEhu3/Q9Cvym2jRDaNWkATgQiHZxmErCmiLRuD5Klfv+HT/
+8WoYdnvj0YaXFP80phYhzZ7fbIO2LRFhYhPmGLa9nSeOsQlFuX5A9kY1120yB2kxSIZI0fU2hy1UCg
mTxdTQPSYtdWBJyvO/vczoX/8I3FziEfss07Hj1NAAAAFQD1dKEzkt4e7rBPDokPOMZigBh4kwAAAIEA
LnpbGNbKm8SNLUEc/fJFswg4G4VjjngjbPZAjhkYe4+H2uYmynry6V+GOTS2kaFQGZRf9XhSpSwfdxK
tx7vCCaoH9bZ6S5Pe0voWmeBhJXi/Sww8f2stpitW2Oq7V7lDdDG81+N/D7/rKDD5PjUyMsVqc1n9wCT
mfqmi6XPEw8AAACAHAGwPn/Mv7P9Q9+JZRWtGq+i4pL1zs1OluiStCN9e/Ok96t3gRVKPheQ6IwLac
NjC9KkSKrLtsVyepGA+V5j/N+Cmsl6csZilnLvMUTvL/cmHDEEhTIQnPNrDDv+tED2BFqkajQqYLgMWe
GVqXsBU6IT66itZlYtrq4v6uDQG/o=,admin
```

karaf.id_dsa 개인 키를 사용하도록 클라이언트에 지정합니다.

```
bin/client -k ~/karaf.id_dsa
```

또는 **ssh**로

```
ssh -p 8101 -i ~/karaf.id_dsa karaf@localhost
```

21.1.4. RBAC

Apache Karaf는 역할을 사용하여 **RBAC(**역할 기반 액세스 제어**)** 시스템이라는 리소스에 대한 액세스를 제어합니다.

역할은 다음을 제어하는 데 사용됩니다.

- **OSGi** 서비스에 액세스

- 콘솔에 대한 액세스(명령 실행을 제어)
- Cryostat(MBean 및/또는 작업에 대한 액세스)
- WebConsole 액세스

21.1.4.1. OSGi 서비스

OSGi 서비스 RBAC 지원에 대한 자세한 내용은 개발자 가이드에 설명되어 있습니다.

21.1.4.2. 콘솔

콘솔 RBAC는 OSGi 서비스 RBAC를 지원합니다. 실제로 Apache Karaf에서 모든 콘솔 명령은 OSGi 서비스로 정의됩니다.

콘솔 명령 이름은 scope:name 형식을 따릅니다.

ACL(액세스 목록)은 etc/org.apache.karaf.command.acl.<scope>.cfg 구성 파일에 정의되어 있습니다. 여기서 < scope >는 명령 범위입니다.

예를 들어 etc/org.apache.karaf.command.acl.feature.cfg 구성 파일을 생성하여 feature:* 명령에 대한 ACL을 정의할 수 있습니다. 이 etc/org.apache.karaf.command.acl.feature.cfg 구성 파일에서 다음을 설정할 수 있습니다.

```
list = viewer
info = viewer
install = admin
uninstall = admin
```

여기서는 feature:list 및 feature:info 명령을 뷰어 역할이 있는 사용자가 실행할 수 있는 반면 feature:install 및 feature:uninstall 명령은 관리자 역할이 있는 사용자만 실행할 수 있습니다. 관리 그룹의 사용자에게도 뷰어 역할이 있으므로 모든 작업을 수행할 수 있습니다.

Apache Karaf 명령 ACL은 지정된 명령 범위 내에서 를 사용하여 액세스를 제어할 수 있습니다.

- 명령 이름 **regex (예: name = role)**

- 명령 이름 및 옵션 또는 인수 값 **regex(예: name[/.[0-9][0-9]+./] = 100** 이상의 인수 값으로만 이름을 실행하는 역할 **)**

명령 이름 및 **options/arguments**는 모두 정확히 일치하는 또는 **regex** 일치를 지원합니다.

기본적으로 **Apache Karaf**는 다음 명령 **ACL**을 정의합니다.

- **etc/org.apache.karaf.command.acl.bundle.cfg** 구성 파일은 **bundle:*** 명령에 대한 **ACL**을 정의합니다. 이 **ACL**은 시스템 번들에 대한 **bundle:*** 명령의 실행을 **admin** 역할이 있는 사용자에게만 제한하지만, 비 시스템 번들에 대한 **bundle:*** 명령은 관리자 역할이 있는 사용자가 실행할 수 있습니다.

- **etc/org.apache.karaf.command.acl.config.cfg** 구성 파일은 **config:*** 명령에 대한 **ACL**을 정의합니다. 이 **ACL**은 **jmx.acl.*, org.apache.karaf.command.acl.*, org.apache.karaf.service.acl.*** 구성 **PID**를 **admin** 역할의 사용자로 **config:*** 명령 실행을 제한합니다. 다른 구성 **PID**의 경우 **manager** 역할의 사용자는 **config:*** 명령을 실행할 수 있습니다.

- **etc/org.apache.karaf.command.acl.feature.cfg** 구성 파일은 **feature:*** 명령에 대한 **ACL**을 정의합니다. 관리자 역할이 있는 사용자만 **feature:install** 및 **feature:uninstall** 명령을 실행할 수 있습니다. 기타 기능:* 명령은 모든 사용자가 실행할 수 있습니다.

- **etc/org.apache.karaf.command.acl.jaas.cfg** 구성 파일은 **jaas:*** 명령에 대한 **ACL**을 정의합니다. 관리자 역할이 있는 사용자만 **jaas:update** 명령을 실행할 수 있습니다. 다른 **jaas:*** 명령은 모든 사용자가 실행할 수 있습니다.

- **etc/org.apache.karaf.command.acl.kar.cfg** 구성 파일은 **kar:*** 명령에 대한 **ACL**을 정의합니다. **admin** 역할이 있는 사용자만 **kar:install** 및 **kar:uninstall** 명령을 실행할 수 있습니다. 기타 **kar:*** 명령은 모든 사용자가 실행할 수 있습니다.

- **etc/org.apache.karaf.command.acl.shell.cfg** 구성 파일은 **shell:*** 및 **"direct"** 명령에 대한 **ACL**을 정의합니다. **admin** 역할이 있는 사용자만 **shell:edit,shell:exec,shell:new, shell:java** 명령을 실행할 수 있습니다. 기타 **shell:*** 명령은 모든 사용자가 실행할 수 있습니다.

이러한 기본 **ACL**을 변경하고 추가 명령 범위를 위해 자체 **ACL**을 추가할 수 있습니다(예: **Apache**

Karaf Cellar의 경우 etc/org.apache.karaf.command.acl.cluster.cfg, etc/org.apache.karaf.command.acl.camel.cfg from Apache Camel, …).

etc/system.properties 에서 karaf.secured.services 속성을 편집하여 명령 RBAC 지원을 미세 조정할 수 있습니다.

```
#
# By default, only Karaf shell commands are secured, but additional services can be
# secured by expanding this filter
#
karaf.secured.services = (&(osgi.command.scope=*)(osgi.command.function=*))
```

21.1.4.3. JMX

콘솔 명령과 마찬가지로 ACL(AccessLists)을 Cryostat 계층에 정의할 수 있습니다.

Cryostat ACL은 etc/jmx.acl<ObjectName>.cfg 구성 파일에 정의되어 있습니다. 여기서 < ObjectName >은 object name입니다(예: org.apache.karaf.bundle 은 org.apache.karaf.bundle은 org.apache.karaf;type=Bundle Cryostat).

etc/jmx.acl.cfg 는 가장 일반적인 구성 파일이며 특정 구성 파일이 없는 경우 사용됩니다. "global" ACL 정의가 포함되어 있습니다.

Cryostat ACL은 (inside a Cryostat)를 사용하여 액세스를 제어할 수 있습니다.

- *작업 이름 regex(예: operation* = role)*
- *작업 인수 값 regex (예: operation(java.lang.String, int)[/([1-4])?[0-9]/,/.*/] = role)*

기본적으로 Apache Karaf는 다음 Cryostat ACL을 정의합니다.

- *etc/jmx.acl.org.apache.karaf.bundle.cfg 구성 파일은 org.apache.karaf:type=bundle 에 대한 ACL을 정의합니다. 이 ACL은 admin 역할이 있는 사용자만 시스템 번들에 대한 setStartLevel(), start(), stop(), update() 작업을 제한합니다. 다른 작업은 manager 역할이 있는 사용자가 수행할 수 있습니다.*
-

etc/jmx.acl.org.apache.karaf.config.cfg 구성 파일은 org.apache.karaf:type=config Cryostat에 대한 ACL을 정의합니다. 이 ACL은 admin 역할의 사용자만 jmx.acl*, org.apache.karaf.command.acl* 및 org.apache.karaf.service.acl* 구성 PIDs의 변경을 제한합니다. 다른 작업은 manager 역할이 있는 사용자가 수행할 수 있습니다.

- etc/jmx.acl.org.apache.karaf.security.jmx.cfg 구성 파일은 org.apache.karaf:type=security,area=jmx Cryostat에 대한 ACL을 정의합니다. 이 ACL은 뷰어 역할이 있는 사용자에 대해 canInvoke() 작업의 호출을 제한합니다.

- etc/jmx.acl.osgi.compendium.cm.cfg 구성 파일은 osgi.compendium:type=cm Cryostat에 대한 ACL을 정의합니다. 이 ACL은 admin 역할의 사용자만 jmx.acl*, org.apache.karaf.command.acl* 및 org.apache.karaf.service.acl* 구성 PIDs의 변경 사항을 제한합니다. 다른 작업은 manager 역할이 있는 사용자가 수행할 수 있습니다.

- etc/jmx.acl.java.lang.Memory.cfg 구성 파일은 코어 JVM 메모리에 대한 ACL을 정의합니다. 이 ACL은 manager 역할이 있는 사용자만 gc 작업의 호출을 제한합니다.

- etc/jmx.acl.cfg 구성 파일은 가장 일반적인 파일입니다. 여기에 정의된 ACL은 다른 특정 ACL과 일치하지 않을 때 사용됩니다(특정 ACL에 의해 다른 특정 etc/jmx.acl.*.cfg 구성 파일에 정의된 ACL임). list*(), get*(), is*() 작업은 뷰어 역할의 사용자가 수행할 수 있습니다. set*() 및 기타 모든 *() 작업은 관리자 역할이 있는 사용자가 수행할 수 있습니다.

21.1.4.4. WebConsole

Apache Karaf WebConsole은 기본적으로 사용할 수 없습니다. 이를 활성화하려면 webconsole 기능을 설치해야 합니다.

```
karaf@root()> feature:install webconsole
```

WebConsole은 현재 콘솔 또는 Cryostat와 같은 세분화된 RBAC를 지원하지 않습니다.

admin 역할이 있는 모든 사용자는 WebConsole을 로그인하고 모든 작업을 수행할 수 있습니다.

21.1.5. SecurityMBean

Apache Karaf는 현재 사용자가 지정된 Cryostat 및/또는 작업을 호출할 수 있는지 확인하는 Cryostat를 제공합니다.

canInvoke() 작업은 현재 사용자의 역할을 가져오고, 역할 중 하나가 지정된 인수 값을 사용하여 및/또는 작업을 호출할 수 있는지 확인합니다.

21.1.5.1. 작업

- *현재 사용자가 objectName,false else를 사용하여 Cryostat를 호출할 수 있는 경우 canInvoke(objectName) 가 true 를 반환합니다.*
- *현재 사용자가 objectName,false 를 사용하여 operation methodName 을 호출할 수 있는 경우 canInvoke(objectName, methodName) 가 true 를 반환합니다.*
- *Can Invoke(objectName, methodName, argumentTypes) 는 현재 사용자가 objectName,false 를 사용하여 argumentsTypes 배열을 사용하여 작업 methodName을 호출할 수 있는 경우 true 를 반환합니다.*
- *canInvoke(bulkQuery) 는 canInvoke 가 true 또는 false 인 경우 bulkQuery 테이블 형식 데이터의 각 작업에 대해 포함하는 테이블 형식 데이터를 반환합니다.*

21.1.6. 보안 공급자

일부 애플리케이션에서는 [BouncyCastle|http://www.bouncycastle.org]과 같은 특정 보안 공급자를 사용할 수 있어야 합니다.

JVM은 이러한 safety의 사용에 대해 몇 가지 제한 사항을 적용합니다. 서명하여 부팅 클래스 경로에서 사용할 수 있어야 합니다.

이러한 공급자를 배포하는 한 가지 방법은 $JAVA_HOME/jre/lib/ext 의 JRE 폴더에 배치하고 보안 정책 구성($JAVA_HOME/jre/lib/security/java.security)을 수정하여 이러한 공급자를 등록하는 것입니다.

이 접근 방식은 잘 작동하지만 글로벌 효과가 있으며 모든 서버를 적절하게 구성해야 합니다.

*Apache Karaf는 추가 보안 공급자를 구성하는 간단한 방법을 제공합니다. * 공급자를 lib/ext *에 배치하고 etc/config.properties 구성 파일을 수정하여 다음 속성을 추가합니다.*

```
org.apache.karaf.security.providers = xxx,yyy
```

이 속성의 값은 등록할 공급자 클래스 이름의 쉼표로 구분된 목록입니다.

*예를 들어 **bouncycastle** 보안 공급자를 추가하려면 다음을 정의합니다.*

```
org.apache.karaf.security.providers = org.bouncycastle.jce.provider.BouncyCastleProvider
```

또한 모든 번들이 해당 번들에 액세스할 수 있도록 시스템 번들에서 해당 공급자의 클래스에 대한 액세스를 제공할 수 있습니다.

*동일한 구성 파일에서 **org.osgi.framework.bootdelegation** 속성을 수정하여 수행할 수 있습니다.*

```
org.osgi.framework.bootdelegation = ...,org.bouncycastle*
```