로그프레소 앱 개발 가이드

### 로그프레소 앱

로그프레소 플랫폼은 고성능의 스키마리스 빅데이터 엔진을 기반으로 보안 운영에 필요한 로그 수집, 저장, 탐지, 분석, 시각화에 관련된 많은 기능을 제공합니다. 하지만 기본 기능만을 이용하여 실제 운영 환경을 구성하려면 많은 노력이 필요합니다.

연동 대상에 따라 전용 수집기를 개발해야 할 수 있고, 바이너리 패킷을 파싱해야 할 수도 있으며, 연동 장비를 제어하기 위하여 REST API를 호출해야 할 수도 있습니다. 기본 플랫폼 기능만으로 운영 환경을 완전히 구성할 수 있다고 하더라도 여전히 정규표현식을 만들거나, 실시간 혹은 배치 탐지 시나리오를 설정하거나, 대시보드 위젯을 만들어 배치하는 작업이 필요합니다.

로그프레소 파트너는 엔지니어가 이러한 작업을 반복적으로 수행하는 대신, 완전히 패키지화된 앱을 개발하여 재사용성을 극대화 할 수 있습니다. 로그프레소 앱은 아래의 기능을 제공합니다.

**로그프레소 앱 기능**

|  |  |
| --- | --- |
| 항목 | 설명 |
| 로그 스키마 | 표준화된 로그 필드 이름의 집합을 정의합니다. |
| 파서 | 로그를 파싱하여 표준화된 필드 집합을 추출하는 파서를 정의합니다. |
| 수집 모델 | 수집 유형과 정규화 규칙의 조합을 정의합니다. |
| 데이터셋 | 대시보드 위젯이나 피벗에서 데이터 원본으로 사용할 데이터 집합을 정의합니다. |
| 프로시저 | 매개변수화 된 쿼리를 정의합니다. |
| 보고서 | 보고서 서식을 정의합니다. |
| 위협 인텔리전스 피드 | 외부 인텔리전스 서비스의 침해지표를 자동 수집하는 피드를 정의합니다. |
| 실시간 탐지 시나리오 | 밀리초 단위로 탐지하는 실시간 탐지 시나리오를 정의합니다. |
| 배치 탐지 시나리오 | 지정된 주기마다 실행되는 배치 탐지 시나리오를 정의합니다. |
| 네트워크 대역 그룹 | 탐지 시나리오에서 참조하는 네트워크 대역 그룹을 정의합니다. |
| 포트 그룹 | 탐지 시나리오에서 참조하는 포트 그룹을 정의합니다. |
| 패턴 그룹 | 탐지 시나리오에서 참조하는 패턴 그룹을 정의합니다. |
| 차단 연동 모델 | 지정된 IP 주소를 연동된 장비에서 차단 또는 차단 해제하는 차단 연동 모델을 정의합니다. |
| 위젯 | 대시보드에 정보를 시각화하는 재사용 가능한 위젯을 정의합니다. |
| 대시보드 | 여러 개의 위젯을 조합하여 위협 탐지 또는 서비스 모니터링에 필요한 대시보드를 정의합니다. |
| 접속 프로파일 | REST API 주소, API 키 등 접속 설정을 관리합니다. |
| 쿼리 명령어 | 사용자 정의 쿼리 명령어를 확장합니다. |
| 플레이북 | 재사용 가능한 플레이북을 정의합니다. |

즉, 로그프레소 플랫폼에 여러 개의 로그프레소 앱을 설치하면 최소한의 설정으로 운영 환경을 구성할 수 있습니다. 다음 절에서는 로그프레소 앱이 어떻게 로그프레소 플랫폼에서 구동되는지 설명하겠습니다.

### 로그프레소 아키텍처

로그프레소 플랫폼은 OSGi 애플리케이션 서버에서 구동됩니다. OSGi(Open Service Gateway Initiative)는 자바 기반의 동적 모듈 시스템을 위한 표준 프레임워크입니다. OSGi는 애플리케이션을 모듈로 분할하여 개발, 배포, 관리할 수 있는 기능을 제공합니다.

로그프레소 플랫폼은 OSGi 프레임워크 구현체 중 [Apache Felix](https://felix.apache.org/) 프레임워크를 기반으로 사용합니다. 로그프레소 앱은 OSGi 번들로 정의되므로, 런타임에 로그프레소 플랫폼에 설치하거나 제거할 수 있습니다. 로그프레소 앱은 정적 리소스와 자바 바이트코드가 포함된 JAR (Java Archive) 파일입니다. 로그프레소 플랫폼은 로그프레소 앱을 설치하는 시점에 번들에 내장된 정적 리소스를 설치하고 OSGi 서비스를 시작합니다.

로그프레소 플랫폼은 SSH(7022/tcp) 및 텔넷(7004/tcp) 프로토콜을 통해 셸을 제공합니다. 로그프레소 셸에 로그인하여 bundle.list를 입력하면 아래와 같이 수백 개의 OSGi 번들을 확인할 수 있습니다.

logpresso> bundle.list[ ID] Symbolic Name Version Status------------------------------------------------------------------[ 0] org.apache.felix.framework 6.0.5 ACTIVE[ 1] org.araqne.ipojo 1.2.1 ACTIVE[ 2] org.apache.felix.ipojo 1.12.1.asm8 ACTIVE[ 3] org.araqne.msgbus 1.13.1 ACTIVE...

기본 설치된 번들과 마찬가지로 로그프레소 앱 역시 하나의 OSGi 번들입니다. 아래는 샘플 앱 번들에 포함된 파일 목록을 조회한 예입니다.

logpresso> bundle.resources 141Bundle Resources-------------------META-INF/com/

META-INF/MANIFEST.MF 파일은 OSGi 매니페스트를 포함합니다. 아래는 중요한 OSGi 매니페스트 엔트리를 선별한 것입니다.

logpresso> bundle.manifest 141Bnd-LastModified: 1686382446996Bundle-Name: Logpresso Sample AppBundle-SymbolicName: com.logpresso.sonar.sampleBundle-Version: 1.0.2306.0

번들 목록에 표시되는 이름은 Bundle-SymbolicName 항목의 값입니다. 번들 버전은 Bundle-Version 항목으로 정의합니다. 로그프레소 플랫폼에서 OSGi 번들의 이름과 버전은 유일해야 합니다.

아래는 OSGi 번들에 포함된 클래스 바이트코드의 예입니다.

bundle.resources 141 com/logpresso/sonar/sample/queryBundle Resources-------------------com/logpresso/sonar/sample/query/SampleCreateSubnetGroupCommand.classcom/logpresso/sonar/sample/query/SampleCreateSubnetGroupCommandParser.classcom/logpresso/sonar/sample/query/SampleDriverQueryCommand.classcom/logpresso/sonar/sample/query/SampleParams.classcom/logpresso/sonar/sample/query/SampleQueryCommandParser.classcom/logpresso/sonar/sample/query/SampleSubnetGroupsCommand.classcom/logpresso/sonar/sample/query/SampleSubnetGroupsCommandParser.class

로그프레소 앱은 로그프레소 플랫폼이 제공하는 OSGi 서비스를 호출하여 기존 기능을 사용하거나, 로그프레소 플랫폼의 기능을 확장할 수 있습니다.

다음 절에서는 실제 앱 예제를 통해 REST API를 호출하는 쿼리 명령어를 어떻게 확장하는지 설명합니다.

### 개발 환경 준비

#### 로그프레소 설치

로그프레소 앱을 구동하려면 로그프레소 플랫폼이 필요합니다. 로그프레소 플랫폼은 [로그프레소 스토어](https://logpresso.store/)에서 다운로드 할 수 있습니다. 일반 사용자 권한으로는 로그프레소 앱만 다운로드 할 수 있으므로, 패키지를 다운로드 할 수 없는 경우 [로그프레소 지원 포탈](https://support.logpresso.com/)을 통해 파트너 권한을 요청해야 합니다.

* [로그프레소 마에스트로](https://logpresso.store/ko/packages/maestro/releases)
* [MariaDB 10 LTS](https://mariadb.org/download/?t=mariadb&p=mariadb&r=10.11.4)
* SQLite 백엔드를 사용할 경우에는 건너뛸 수 있습니다.
* [Visual C++ Redistributable for Visual Studio 2012 Update 4](https://www.microsoft.com/en-us/download/details.aspx?id=30679)
* 윈도우 개발 환경에서는 이 재배포 패키지를 설치해야 네이티브 코드가 정상적으로 동작합니다.

#### JDK 설치

로그프레소 앱을 빌드하려면 JDK(Java Development Kit)가 필요합니다. 로그프레소는 OpenJDK 기반으로 패키징된 Eclipse Temurin을 권장합니다

* [Eclipse Temurin JDK 11 LTS](https://adoptium.net/temurin/releases/?version=11)

#### Maven 설치

로그프레소는 자바 프로젝트의 빌드 및 관리를 위해 아파치 메이븐을 사용합니다. 로그프레소 샘플 앱을 빌드하려면 아파치 메이븐을 설치해야 합니다.

* [Apache Maven](https://maven.apache.org/download.cgi)

윈도우 환경에서는 시스템 환경변수에 M2\_HOME 및 JAVA\_HOME을 추가하고, PATH를 %M2\_HOME%\bin으로 지정하여 mvn 명령이 정상 동작하도록 합니다.

#### Git 설치

로그프레소는 코드 및 문서의 버전 관리를 위하여 git을 사용합니다. 윈도우 환경에서는 아래 도구를 설치합니다.

* [git for windows](https://gitforwindows.org/)

### 앱 예제 다운로드

[깃허브 저장소](https://github.com/logpresso/logpresso-app-examples)에서 아래와 같이 로그프레소 앱 예제를 clone 하거나 [다운로드](https://github.com/logpresso/logpresso-app-examples/archive/refs/heads/main.zip)합니다.

git clone git@github.com:logpresso/logpresso-app-examples.git

이 예제는 로그프레소 소나 REST API를 호출하는 쿼리 명령어 확장 방법을 소개합니다. 아래의 내용을 학습할 수 있습니다:

* 메이븐 프로젝트 구성: 어떤 과정을 거쳐 로그프레소 앱이 빌드되는지 설명합니다.
* iPOJO 컴포넌트 개발: 선언적으로 OSGi 서비스 컴포넌트를 개발하는 방법을 설명합니다.
* 로그프레소 기능 확장
* 접속 프로파일: REST API를 호출하는데 필요한 API 키와 URL 설정 관리 방법을 설명합니다.
* 쿼리 명령어: 사용자 정의 쿼리 명령어를 어떻게 파싱하고, 쿼리 명령어가 어떤 과정을 거쳐 데이터를 처리하는지 설명합니다.

### 앱 예제 빌드

명령줄에서 logpresso-sample-app 디렉터리로 이동한 후, mvn clean package 명령을 실행하여 빌드합니다.

cmd> mvn clean package[INFO] Scanning for projects...[INFO][INFO] --------------< com.logpresso.sonar:logpresso-sample-app >--------------[INFO] Building Logpresso Sample App 1.0.2306.0[INFO] -------------------------------[ bundle ]-------------------------------[INFO][INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ logpresso-sample-app ---[INFO] Deleting C:\github\logpresso-app-examples\logpresso-sample-app\target[INFO][INFO] --- maven-resources-plugin:3.3.1:resources (default-resources) @ logpresso-sample-app ---[INFO] skip non existing resourceDirectory C:\github\logpresso-app-examples\logpresso-sample-app\src\main\resources[INFO][INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ logpresso-sample-app ---[INFO] Changes detected - recompiling the module![INFO] Compiling 9 source files to C:\github\logpresso-app-examples\logpresso-sample-app\target\classes[INFO][INFO] --- maven-resources-plugin:3.3.1:testResources (default-testResources) @ logpresso-sample-app ---[INFO] skip non existing resourceDirectory C:\github\logpresso-app-examples\logpresso-sample-app\src\test\resources[INFO][INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ logpresso-sample-app ---[INFO] No sources to compile[INFO][INFO] --- maven-surefire-plugin:3.1.2:test (default-test) @ logpresso-sample-app ---[INFO] No tests to run.[INFO][INFO] --- maven-bundle-plugin:5.1.9:bundle (default-bundle) @ logpresso-sample-app ---[WARNING] Bundle com.logpresso.sonar:logpresso-sample-app:bundle:1.0.2306.0 : Unused Import-Package instructions: [com.logpresso.sonar.api.\*][INFO] Building bundle: C:\github\logpresso-app-examples\logpresso-sample-app\target\logpresso-sample-app-1.0.2306.0.jar[INFO] Writing manifest: C:\github\logpresso-app-examples\logpresso-sample-app\target\classes\META-INF\MANIFEST.MF[INFO][INFO] --- maven-ipojo-plugin:1.12.1.asm8:ipojo-bundle (default) @ logpresso-sample-app ---[INFO] Start bundle manipulation[INFO] No metadata file found - trying to use only annotations[INFO] Input Bundle File : C:\github\logpresso-app-examples\logpresso-sample-app\target\logpresso-sample-app-1.0.2306.0.jar[INFO] Apache Felix iPOJO Manipulator - 1.12.1.asm8[INFO] Bundle manipulation - SUCCESS[INFO] ------------------------------------------------------------------------[INFO] BUILD SUCCESS[INFO] ------------------------------------------------------------------------[INFO] Total time: 5.132 s[INFO] Finished at: 2023-06-10T22:38:33+09:00[INFO] ------------------------------------------------------------------------

빌드가 성공하면 target 디렉터리에 logpresso-sample-app-1.0.2306.0.jar 파일이 생성됩니다.

### 앱 예제 설치

SSH를 이용하여 로그프레소 셸에 접속합니다. 아래와 같이 접속할 수 있습니다.

cmd> ssh -p7022 root@localhost

만약 7022 포트가 열려있지 않다면 자바 실행 옵션에 -Daraqne.ssh.server=enabled를 추가하거나, config.sh 파일의 SSH\_OPEN="enabled" 항목을 주석 해제한 후 데몬을 실행합니다.

이제 bundle.install 명령을 이용하여 앱 번들을 설치하고, bundle.start 명령으로 앱 번들을 시작합니다:

logpresso> bundle.install file:///C:\github\logpresso-app-examples\logpresso-sample-app\target\logpresso-sample-app-1.0.2306.0.jarbundle [133] loadedlogpresso> bundle.start 133bundle 133 started.

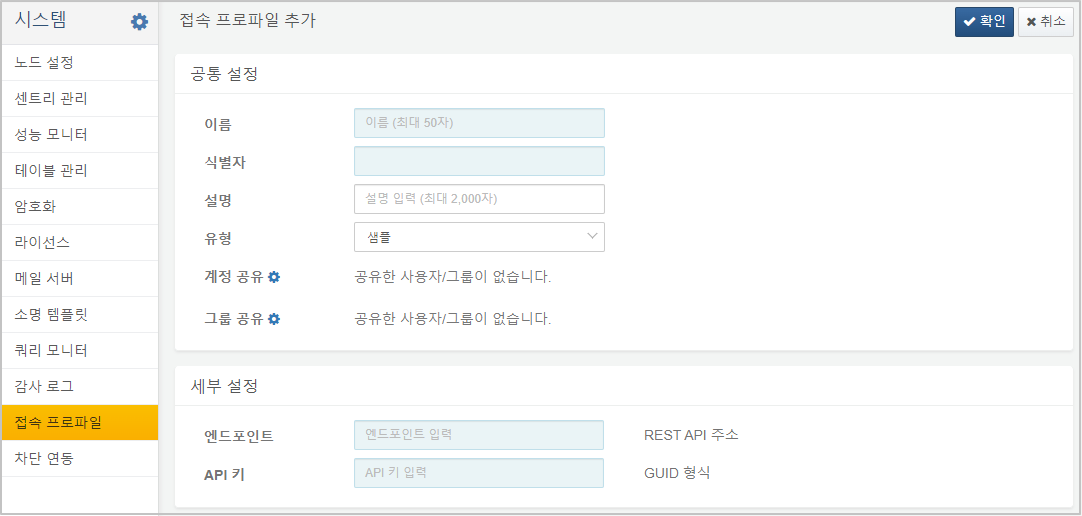
로컬 파일 시스템에서 번들을 설치하는 경우 file:// 스킴을 사용하고, 파일 경로는 /로 시작해야 한다는 점에 유의합니다.

앱 기능 테스트와 관련하여 REST API를 사용할 예정이므로, sonar.generateApiKey 명령을 실행하여 API 키를 추가로 발급합니다.

logpresso> sonar.generateApiKey xeraphnew api key is f1382075-2be1-451b-93bf-e5d52297c9e9

#### 앱 접속 프로파일 설정

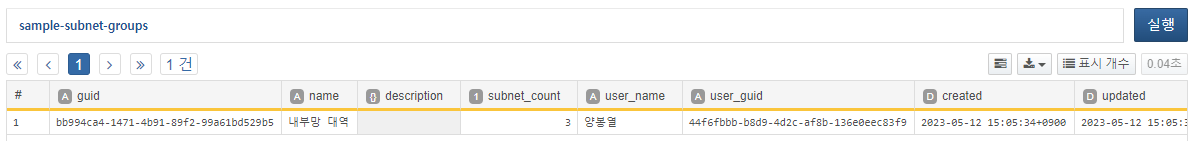
접속 프로파일 메뉴로 이동하여 추가 버튼을 클릭하여 이동한 후 아래와 같이 앱으로 확장된 샘플 접속 프로파일 유형을 선택할 수 있습니다.



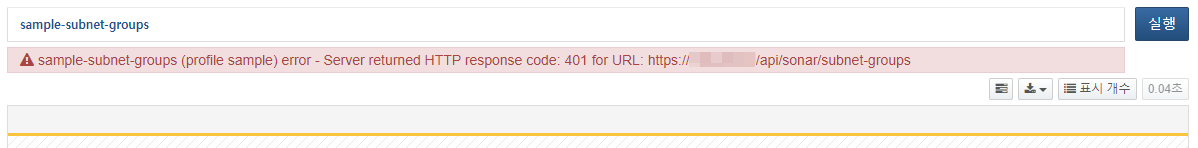
엔드포인트 항목에 https://YOUR\_LOGPRESSO\_IP를 입력하고, API 키 항목에 방금 발급한 API 키를 입력하여 접속 프로파일을 추가합니다.

#### 앱 쿼리 명령어 테스트

이제 쿼리 창에서 sample-subnet-groups 명령어를 실행하면 아래와 같이 1건의 결과를 볼 수 있습니다.



API 키 설정이 잘못된 경우에는 쿼리 실행 시 아래와 같이 401 Unauthorized 오류가 발생합니다.



이제 로그프레소 앱에서 쿼리 명령어를 확장하는 방법에 대해서 알아보도록 하겠습니다.

### 메이븐 프로젝트 구성

이 절에서는 [pom.xml](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/pom.xml) 구성을 항목별로 설명합니다. 전체 내용은 소스를 참고하시기 바랍니다.

#### 아티팩트 정보

<groupId>com.logpresso.sonar</groupId><artifactId>logpresso-sample-app</artifactId><version>1.0.2306.0</version><packaging>bundle</packaging><name>Logpresso Sample App</name>

로그프레소 앱의 그룹 ID는 항상 com.logpresso.sonar로 지정합니다. 아티팩트 ID는 logpresso-VENDOR-MODEL 형식으로 정의하는 것을 권장합니다. 버전은 Major.Minor.YYMM.REV 형식으로 정의합니다. 앱 버전에 배포 시점을 표기하는 것은 호환되는 로그프레소 플랫폼을 추정하기 수월하기 때문입니다.

마지막으로 패키징이 bundle로 지정된 것을 특히 주목해야 합니다. 아래에 설명할 maven-bundle-plugin 플러그인이 bundle 패키징을 수행합니다.

#### 환경변수 설정

<properties> <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding> <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding> <jdk.version>11</jdk.version></properties>

운영체제 로케일이 빌드에 영향을 미치지 않도록 파일 인코딩을 UTF-8로 명시하고, JDK 버전을 11로 지정합니다. jdk.version 변수는 아래에서 사용됩니다.

#### 메이븐 저장소

<repositories> <repository> <id>logpresso-repo</id> <name>Logpresso Maven Repository</name> <url>https://maven.logpresso.com/</url> </repository></repositories><pluginRepositories> <pluginRepository> <id>logpresso-plugin-repo</id> <name>Logpresso Maven Repository</name> <url>https://maven.logpresso.com/</url> </pluginRepository></pluginRepositories>

로그프레소는 https://maven.logpresso.com 저장소에서 서드파티 개발에 필요한 라이브러리를 제공합니다. 빌드 단계에서 바이트코드 변경에 필요한 iPOJO 플러그인을 다운로드 받을 수 있도록 <pluginRepository> 설정을 추가하고, 앱 API 및 iPOJO 라이브러리를 다운로드 받을 수 있도록 <repository> 설정을 추가합니다.

#### 빌드 설정

자바 컴파일 시 소스 코드 및 바이트 코드 버전, 디버그 정보 포함 여부, 최적화 여부를 지정합니다.

<plugin> <groupId>org.apache.maven.plugins</groupId> <artifactId>maven-compiler-plugin</artifactId> <version>3.8.1</version> <configuration> <encoding>UTF-8</encoding> <source>${jdk.version}</source> <target>${jdk.version}</target> <debug>true</debug> <optimize>true</optimize> <showDeprecations>true</showDeprecations> </configuration></plugin>

maven-bundle-plugin 플러그인은 JAR 파일에 OSGi 매니페스트를 추가하여 OSGi 번들을 생성합니다. OSGi 번들은 META-INF/MANIFEST.MF 파일에 OSGi 매니페스트가 추가된 JAR 파일을 의미합니다.

<plugin> <groupId>org.apache.felix</groupId> <artifactId>maven-bundle-plugin</artifactId> <version>5.1.4</version> <extensions>true</extensions> <configuration> <instructions> <Bundle-SymbolicName>com.logpresso.sonar.sample</Bundle-SymbolicName> <Export-Package> com.logpresso.sonar.sample;version=${project.version}, </Export-Package> <Import-Package> org.json;version="1.1.0", org.araqne.codec;version="2.2", org.araqne.log.api;version="3.13.0", org.logpresso.api.profile;version="1.1.0", org.logpresso.api.profile.query;version="1.1.0", org.araqne.logdb;version="3.10.0", org.araqne.msgbus;version="1.12.0", com.logpresso.sonar.api.\*;resolution:=optional, \* </Import-Package> <Private-Package> com.logpresso.sonar.sample.impl, com.logpresso.sonar.sample.query, </Private-Package> </instructions> </configuration></plugin>

<instruction>의 4개 항목을 살펴봐야 합니다:

* Bundle-SymbolicName: 번들 식별자를 정의합니다. 자바 패키지명 규칙에 맞춰서 작성합니다.
* Export-Package: 다른 OSGi 번들에게 공개할 패키지 목록을 입력합니다. 일반적으로 인터페이스가 포함된 패키지만 공개합니다. ${project.version} 매크로를 이용하여 공개하는 패키지의 버전을 명시합니다.
* Import-Package: 다른 OSGi 번들에서 가져올 패키지 목록을 입력합니다. 패키지 뒤에 ;version="VERSION"을 정의하여 최소 버전을 명시합니다. 버전을 명시하지 않으면 빌드 플러그인이 <dependencies>에 정의된 메이븐 아티팩트의 버전을 사용하기 때문에, 호환되는 패키지 버전 범위가 의도와 다르게 지정되어 번들 의존성이 해소되지 않을 수 있습니다.
* Private-Package: 다른 OSGi 번들에게 공개하지 않을 내부 패키지 목록을 입력합니다. 일반적으로 구현이 포함된 패키지는 공개하지 않습니다. 외부에는 인터페이스만 공개해야 구현 세부내용을 언제든지 쉽게 변경할 수 있습니다.

Import-Package에는 로그프레소 앱 API 호출에 필요한 패키지의 버전이 미리 정의되어 있습니다. sonar-app-api 라이브러리 버전(현재 4.0.2306.0)에 따라 불러올 수 있는 패키지 버전의 집합은 변경될 수 있습니다.

<plugin> <groupId>org.apache.felix</groupId> <artifactId>maven-ipojo-plugin</artifactId> <version>1.12.1.asm8</version> <executions> <execution> <goals> <goal>ipojo-bundle</goal> </goals> </execution> </executions></plugin>

iPOJO는 선언적으로 OSGi 컴포넌트를 개발할 수 있도록 만들어주는 프레임워크입니다. OSGi는 런타임 플러그인을 가정하기 때문에, 각 번들이 의존하는 기능이 언제든지 새로 설치되거나 삭제될 수 있습니다. 이 때문에 OSGi 인터페이스만을 사용하여 개발하려면 모든 상태 변경에 대해 통지를 받아 연쇄적으로 기능을 비활성화하거나, 활성화해야 해서 구현이 매우 복잡해집니다. iPOJO는 컴포넌트 생명주기 모델을 정의하고 간단한 어노테이션 추가만으로 OSGi 서비스를 선언적으로 개발할 수 있도록 지원합니다.

iPOJO는 빌드 단계에서 iPOJO 어노테이션을 인식하여 바이트코드를 삽입하는 방식으로 자동 변환을 수행합니다. 따라서 반드시 위와 같이 빌드 단계에서 maven-ipojo-plugin이 호출되어야 합니다.

#### 라이브러리 의존성

<dependencies> <dependency> <groupId>junit</groupId> <artifactId>junit</artifactId> <version>4.8.1</version> <scope>test</scope> </dependency> <dependency> <groupId>org.apache.felix</groupId> <artifactId>org.apache.felix.ipojo.annotations</artifactId> <version>1.12.1.asm8</version> </dependency> <dependency> <groupId>org.slf4j</groupId> <artifactId>slf4j-api</artifactId> <version>1.7.12</version> </dependency> <dependency> <groupId>com.logpresso.sonar</groupId> <artifactId>sonar-app-api</artifactId> <version>4.0.2306.0</version> </dependency></dependencies>

로그프레소 앱은 아래의 라이브러리를 사용합니다:

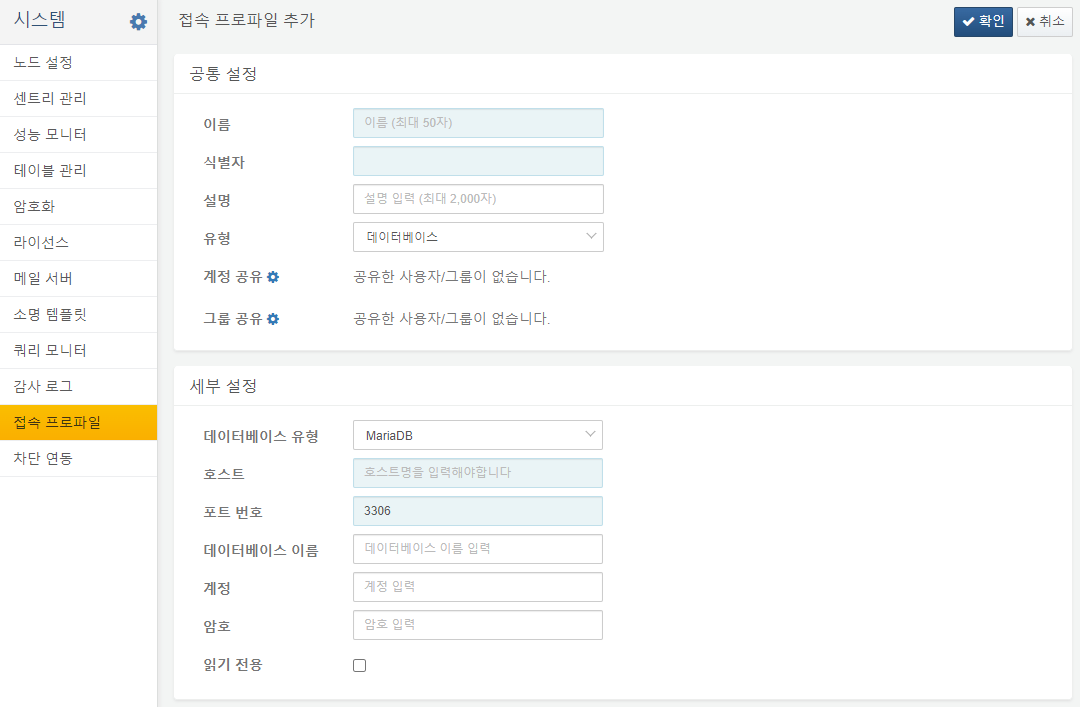
* junit: 단위 테스트 작성에 사용합니다. 단위 테스트 클래스는 src/test/java 디렉터리 아래에 배치합니다.
* org.apache.felix.ipojo.annotations: iPOJO 컴포넌트 어노테이션을 사용합니다.
* slf4j-api: 시스템 로그 기록에 SLF4J를 사용합니다. 로그는 log/araqne.log 파일에 기록되며, 로그 레벨을 런타임에 조정할 수 있습니다.
* sonar-app-api: 로그프레소 앱 API를 사용합니다. 이 라이브러리 버전은 호환되는 로그프레소 플랫폼 버전을 의미합니다.

### 접속 프로파일 구현

이 절에서는 [SampleConnectProfileFactory.java](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/src/main/java/com/logpresso/sonar/sample/impl/SampleConnectProfileFactory.java) 코드를 설명합니다.

#### 접속 프로파일 개요

로그프레소의 쿼리 명령어나 수집기는 외부 시스템과 통신하는 경우가 많습니다. 예를 들어, [dbquery](https://docs.logpresso.comnull) 명령어는 외부 데이터베이스에서 SQL을 실행한 결과를 조회합니다. 외부 데이터베이스에 접속하려면 JDBC 접속 문자열, 계정, 암호가 필수적입니다. 따라서 dbquery 쿼리 명령어는 접속 프로파일을 PROFILE 매개변수로 전달받습니다. dbquery 쿼리 명령어를 실행하려면 아래와 같이 데이터베이스 접속 프로파일을 미리 설정해두어야 합니다.



한편, 접속 프로파일이 설정되었다고 하더라도 아무나 데이터베이스에 SQL 쿼리를 실행하도록 허용해서는 안 됩니다. 계정 또는 계정 그룹에 접속 프로파일의 사용 권한을 부여해야만 해당 계정에서 접속 프로파일을 이용하여 외부 시스템과 통신할 수 있습니다.

데이터베이스 뿐 아니라 외부 시스템은 대부분 인증을 요구합니다. 앱 예제는 로그프레소 소나의 REST API를 호출하는데, REST API를 호출하려면 Authorization 헤더에 API 키를 전달해야 합니다.

외부 시스템에서 요구하는 인증 방식에 따라 설정 항목이 모두 다르기 때문에, 앱은 전용 ConnectProfileFactory를 구현하여 로그프레소 플랫폼의 기능을 확장해야 합니다.

#### ConnectProfileFactory 플러그인

앱 예제에서는 SampleConnectProfileFactory를 구현하여 로그프레소 플랫폼에 등록합니다.

@Component(name = "sample-connect-profile-factory")@Instantiatepublic class SampleConnectProfileFactory extends AbstractConnectProfileFactory { @Requires private ConnectProfileFactoryRegistry connectProfileFactoryRegistry; @Validate public void start() { connectProfileFactoryRegistry.addFactory(this); } @Invalidate public void stop() { if (connectProfileFactoryRegistry != null) connectProfileFactoryRegistry.removeFactory(this); }}

위의 코드는 AbstractConnectProfileFactory를 상속한 SampleConnectProfileFactory 컴포넌트를 정의합니다. 컴포넌트가 시작할 때 ConnectProfileFactoryRegistry OSGi 서비스에 addFactory()를 호출하여 자기 자신을 등록하고, 컴포넌트가 종료될 때 removeFactory()를 호출하여 자기 자신을 제거합니다.

@Component, @Instantiate, @Requires, @Validate, @Invalidate는 모두 iPOJO 어노테이션입니다. 각 어노테이션에 대한 설명은 아래와 같습니다:

* @Component: 이 클래스가 iPOJO 컴포넌트임을 선언합니다. iPOJO 컴포넌트는 유일한 이름을 지정해야 합니다.
* @Instantiate: iPOJO 컴포넌트 인스턴스를 생성하도록 지시합니다. 만약 @Instantiate를 선언하지 않으면, 앱 번들을 시작하더라도 컴포넌트 인스턴스가 생성되지 않습니다.
* @Requires: 서비스 인터페이스 의존성을 선언합니다. 별도의 할당(assignment) 구문이 없지만, 컴포넌트가 시작될 때 iPOJO 프레임워크가 자동으로 ConnectProfileFactoryRegistry 인터페이스를 제공하는 OSGi 서비스 객체를 주입할 것입니다. 이를 injected POJO(Plain Old Java Object)라고 표현하는데, 여기에서 iPOJO라는 프레임워크 이름이 나오게 되었습니다.
* @Validate: 컴포넌트가 시작될 때 호출될 콜백을 선언합니다. @Requires로 선언된 모든 의존성이 충족되어야만 컴포넌트가 시작됩니다.
* @Invalidate: 컴포넌트가 중지될 때 호출될 콜백을 선언합니다. @Requires로 선언된 의존성 중 하나라도 충족되지 않으면 컴포넌트가 중지됩니다. OSGi 번들을 중지하거나 삭제하면 서비스가 제거되면서 이 서비스에 의존하던 컴포넌트들도 중지됩니다.

#### ConnectProfileFactory 구현

**식별자**

getType()은 ConnectProfileFactoryRegistry에 등록된 각 ConnectProfileFactory 객체를 구분하는데 필요한 식별자를 반환합니다. 일반적으로 알파벳 소문자와 하이픈을 사용하여 정의합니다.

public String getType() { return "sample";}

**표시 이름**

getDisplayName()은 사용자 인터페이스에 표시할 접속 프로파일 유형의 이름을 반환합니다. 로케일에 따라 적합한 문자열을 반환해야 합니다.

public String getDisplayName(Locale locale) { if (locale != null && locale.equals(Locale.KOREAN)) return "샘플"; return "Sample";}

**설명**

getDescription()은 사용자 인터페이스에 표시할 접속 프로파일 유형에 대한 설명을 반환합니다. 로케일에 따라 적합한 문자열을 반환해야 합니다.

public String getDescription(Locale locale) { if (locale != null && locale.equals(Locale.KOREAN)) return "샘플 접속 설정을 관리합니다."; return "Manage connection properties of sample app.";}

**보호 설정 항목 목록**

getProtectedConfigKeys()는 사용자 인터페이스에 표시해서는 안 되는 설정 항목의 식별자 목록을 반환합니다. 예를 들어, 암호나 API 키는 관리자라 하더라도 화면에서 조회할 수 없어야 합니다. 보호되는 설정 항목은 재설정만 가능하고 조회가 불가능하며, 로그프레소 플랫폼 내에서도 암호화되어 보관됩니다.

public Set<String> getProtectedConfigKeys() { return Set.of("api\_key");}

**설정 명세 목록**

getConfigOptions()은 접속 프로파일의 설정 양식(Form)을 정의합니다. 접속 프로파일의 설정 명세인데 LoggerConfigOption 이름의 타입을 사용하는 것이 다소 어색하겠지만, 수집기의 설정 양식을 정의할 때에도 동일한 타입을 사용하여 동적으로 설정 양식을 정의합니다.

public List<LoggerConfigOption> getConfigOptions() { LoggerConfigOption endpoint = new MutableStringConfigType("endpoint", t("Endpoint", "엔드포인트"), t("REST API URL", "REST API 주소"), true); LoggerConfigOption apiKey = new MutableStringConfigType("api\_key", t("API key", "API 키"), t("GUID format", "GUID 형식"), true); return Arrays.asList(endpoint, apiKey);}

아래 3개 타입을 사용할 수 있습니다:

* MutableStringConfigType: 변경 가능한 문자열 설정 항목
* MutableIntegerConfigType: 변경 가능한 32비트 정수 설정 항목
* MutableBooleanConfigType: 변경 가능한 불린 설정 항목

리스트로 반환한 설정 항목의 순서대로 화면에 표시됩니다.

**설정 요약**

접속 프로파일 목록에서 빠르게 설정 세부사항을 확인할 수 있도록 핵심적인 설정만 요약하여 반환합니다. 로케일에 따라 적절한 문자열을 반환해야 합니다.

public String describeConfigs(ConnectProfile profile, Locale locale) { String endpoint = profile.getString("endpoint"); if (locale != null && locale.equals(Locale.KOREAN)) return "엔드포인트: " + endpoint; return "Endpoint: " + endpoint;}

### 쿼리 명령어 파서 구현

이 절에서는 [SampleQueryCommandParser.java](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/src/main/java/com/logpresso/sonar/sample/query/SampleQueryCommandParser.java) 코드와 [SampleSubnetGroupsCommandParser.java](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/src/main/java/com/logpresso/sonar/sample/query/SampleSubnetGroupsCommandParser.java) 코드를 설명합니다.

#### 쿼리 명령어 파서 개요

로그프레소의 쿼리는 파이프로 구분된 쿼리 명령어의 조합입니다. 로그프레소 쿼리 엔진은 쿼리 문자열을 파이프 문자로 분할한 다음, 공백으로 구분되는 첫번째 토큰을 쿼리 명령어의 이름으로 인식합니다. 쿼리 엔진은 쿼리 명령어의 이름으로 QueryCommandParser 객체를 검색하여 쿼리 명령어 파싱을 요청합니다.

쿼리 명령어 파서는 QueryCommandParser 인터페이스를 구현합니다:

public interface QueryCommandParser { String getCommandName(); QueryCommand parse(QueryContext context, String commandString); Map<String, QueryErrorMessage> getErrorMessages(); QueryCommandHelp getCommandHelp(); QueryParserService getQueryParserService(); void setQueryParserService(QueryParserService queryParserService);}

각 메소드는 아래의 기능을 구현합니다:

* getCommandName(): 로그프레소 플랫폼에서 유일한 쿼리 명령어 이름을 반환합니다. 쿼리 명령어 이름은 쿼리 창에서 자동으로 문법 강조됩니다.
* parse(): 쿼리 명령어 구문을 파싱하여 QueryCommand 객체를 반환합니다.
* getErrorMessages(): 쿼리 파싱 중에 발생하는 오류 코드에 대한 지역화된 메시지를 제공합니다.
* getCommandHelp(): 쿼리 명령어에 대한 설명, 옵션, 출력 필드 세부사항을 제공합니다. 이 메소드는 기본 구현이 제공됩니다.
* getQueryParserService(): 쿼리 파서 서비스 객체를 반환합니다. 파서에서 서브 쿼리 등 재귀적으로 파싱을 요청해야 하는 경우에 사용합니다.
* setQueryParserService(): 쿼리 파서 서비스 객체를 설정합니다. 단위 테스트 외에는 이 메소드를 직접 호출하면 안 됩니다.

#### 접속 프로파일 쿼리 명령어 파서

앱에서 확장하는 쿼리 명령어는 대부분 외부 시스템과 상호작용하게 되므로 접속 프로파일을 사용합니다. 접속 프로파일을 사용하는 쿼리 명령어 파서는 ConnectProfileQueryCommandParser 클래스를 상속하여 간단하게 구현할 수 있습니다. SampleQueryCommandParser는 앱에서 구현하는 여러 개의 쿼리 명령어 파서가 상속할 기본 구현을 제공합니다.

public abstract class SampleQueryCommandParser extends ConnectProfileQueryCommandParser { protected static final String ERR\_SERVICE\_UNAVAILABLE = "204000"; protected static final String ERR\_PROFILE\_REQUIRED = "204001"; protected static final String ERR\_NAME\_REQUIRED = "204002"; public SampleQueryCommandParser() { super("sample", ERR\_SERVICE\_UNAVAILABLE, ERR\_PROFILE\_REQUIRED); }}

생성자의 매개변수는 아래와 같습니다:

* profileType: 쿼리 명령어가 사용하는 접속 프로파일 유형의 식별자입니다. 즉, ConnectProfileFactory의 getType() 반환 값과 일치해야 합니다.
* emptyProfileErrorCode: 현재 세션에 허용된 접속 프로파일이 하나도 없는 경우 발생시킬 오류 코드입니다.
* missingProfileErrorCode: 쿼리 명령어의 profile 매개변수가 필수인데 profile 값이 지정되지 않은 경우에 발생시킬 오류 코드입니다.

protected List<String> getSupportedOptions() { return new ArrayList<String>(getCommandHelp().getOptions().keySet());}

쿼리 명령어는 COMMAND-NAME KEY1=VALUE1 KEY2=VALUE2의 형태로 정의됩니다. getSupportedOptions()은 사용 가능한 매개변수의 목록을 반환합니다. 이 코드는 도움말에 정의된 모든 매개변수를 반환하도록 구현되어 있습니다.

protected abstract QueryCommand parse(QueryContext context, SampleParams params);

SampleQueryCommandParser를 상속하는 쿼리 명령어 파서는 이미 파싱되어 SampleParams로 전달되는 매개변수를 이용할 수 있습니다. 각 쿼리 명령어 파서는 필수 매개변수 검사 등 추가적인 검증을 수행합니다.

protected ConnectProfileParams parseParams(QueryContext context, Map<String, String> opts) { SampleParams params = new SampleParams(); params.setName(opts.get("name")); return params;}protected QueryCommand parse(QueryContext context, ConnectProfileParams params, String commandString) { return parse(context, (SampleParams) params);}

* parseParams(): 모든 확장 쿼리 명령어의 매개변수를 일괄적으로 파싱합니다. 개별 쿼리 명령어마다 매개변수를 파싱할 수도 있지만, 한 지점에 매개변수 파싱을 모아두면 공통 매개변수 파싱이 더 수월합니다.
* parse(): ConnectProfileQueryCommandParser 클래스는 쿼리 명령어 구문에서 키=값 형식의 매개변수 패턴을 파싱하여 맵을 구성한 후 parseParams()를 호출하고, 그 결과로 반환되는 ConnectProfileParams 객체를 parse() 메소드에 다시 전달합니다. 이미 파싱된 쿼리 명령어의 매개변수를 활용하기만 하면 됩니다.

public Map<String, QueryErrorMessage> getErrorMessages() { Map<String, QueryErrorMessage> errors = new HashMap<>(); errors.put(ERR\_SERVICE\_UNAVAILABLE, newMsg("No available sample profile found.", "사용 가능한 샘플 프로파일이 없습니다.")); errors.put(ERR\_PROFILE\_REQUIRED, newMsg("Specify valid sample profile.", "샘플 프로파일 이름을 입력해주세요.")); errors.put(ERR\_NAME\_REQUIRED, newMsg("Specify name option in the sample-create-subnet-group command.", "sample-create-subnet-group 명령어에 name 옵션을 지정하세요.")); return errors;}

getErrorMessage()는 에러 코드와 지역화된 메시지의 쌍을 반환합니다. 쿼리 명령어 파서는 QueryParseException 예외 객체를 던질 때 에러 코드를 지정합니다. 로그프레소 쿼리 엔진은 에러 코드를 다시 지역화된 메시지로 변환하는 과정에서 이 메소드의 반환 값을 참조합니다.

#### 쿼리 명령어 파서 구현

SampleSubnetGroupsCommandParser 클래스는 위에서 정의한 SampleQueryCommandParser 클래스를 상속합니다.

public class SampleSubnetGroupsCommandParser extends SampleQueryCommandParser { public String getCommandName() { return "sample-subnet-groups"; } protected QueryCommand parse(QueryContext context, SampleParams params) { return new SampleSubnetGroupsCommand(params); }}

이 파서는 sample-subnet-groups라는 이름의 쿼리 명령어를 정의하는데, 별도의 매개변수가 없으므로 parse()는 별다른 과정 없이 바로 SampleSubnetGroupsCommand 객체를 생성하여 반환하게 됩니다.

이 파서에서 주목할 부분은 생성자의 메타데이터입니다.

setDescription(Locale.ENGLISH, "Get subnet groups from the Logpresso server.");setOutput("guid", ValueType.STRING, Locale.ENGLISH, "GUID", "");

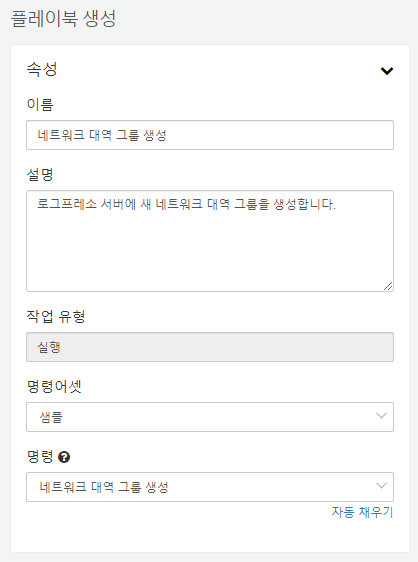
* setDescrpition(): 쿼리 도움말에 표시할 로케일별 명령어 설명을 설정합니다.
* setOutput(): 쿼리 도움말에 표시할 출력 필드의 이름, 타입, 설명을 설정합니다.

SampleCreateSubnetGroupCommandParser 클래스에서는 다른 메타데이터도 확인할 수 있습니다.

setDisplayGroup(Locale.KOREAN, "샘플");setDisplayName(Locale.KOREAN, "네트워크 대역 그룹 생성");setDescription(Locale.KOREAN, "로그프레소 서버에 새 네트워크 대역 그룹을 생성합니다.");setOption("name", REQUIRED, Locale.KOREAN, "이름", "새 네트워크 대역 그룹 이름");

* setDisplayGroup(): 쿼리 명령어 그룹. 플레이북의 명령어셋 항목에 매핑됩니다.
* setDisplayName(): 쿼리 명령어 표시 이름. 플레이북의 명령 항목에 매핑됩니다.
* setOption(): 명령어 매개변수를 정의합니다. Ctrl+Space 입력 시 매개변수 도움말과 자동 완성에 활용됩니다.

예를 들어, 플레이북 디자이너의 실행 작업을 생성할 때 샘플 명령어셋을 선택하면, 아래와 같이 네트워크 대역 그룹 생성 항목을 확인할 수 있습니다.



쿼리 명령어 파싱 중 예외는 아래와 같이 발생시킵니다. create-subnet-group 명령어는 name 매개변수가 필수인데, 이 값을 지정하지 않은 경우 QueryParseException 예외를 발생시켜서 파싱이 실패하도록 합니다.

protected QueryCommand parse(QueryContext context, SampleParams params) { if (params.getName() == null) throw new QueryParseException(ERR\_NAME\_REQUIRED, -1); return new SampleCreateSubnetGroupCommand(params);}

다음 절에서는 이렇게 구현된 쿼리 명령어 파서를 등록하는 부분을 알아보겠습니다.

### 쿼리 명령어 파서 등록

이 절에서는 [SampleServiceImpl](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/src/main/java/com/logpresso/sonar/sample/impl/SampleServiceImpl.java) 코드를 통해 런타임에 쿼리 명령어 파서를 추가하거나 삭제하는 과정을 알아봅니다.

#### 쿼리 파서 서비스

QueryParserService 서비스는 아래 2개의 메소드를 통해 파서를 추가하거나 삭제할 수 있도록 지원합니다.

* addCommandParser(): 새 쿼리 명령어 파서를 추가합니다.
* removeCommandParser(): 기존 쿼리 명령어 파서를 삭제합니다.

#### 파서 추가 및 삭제

SampleServiceImpl 컴포넌트는 시작할 때 QueryParserService에 모든 쿼리 명령어 파서를 추가하고, 정지할 때 QueryParserService에서 모든 쿼리 명령어를 삭제하도록 구현되어 있습니다.

@Component(name = "sample-service")@Instantiatepublic class SampleServiceImpl { @Requires private QueryParserService queryParserService; @Requires private ConnectProfileService connectProfileService; private SampleQueryCommandParser subnetGroupsParser = new SampleSubnetGroupsCommandParser(); private SampleQueryCommandParser createSubnetGroupParser = new SampleCreateSubnetGroupCommandParser(); @Validate public void start() { for (SampleQueryCommandParser parser : getParsers()) { parser.setConnectProfileService(connectProfileService); queryParserService.addCommandParser(parser); } } @Invalidate public void stop() { if (queryParserService == null) return; for (SampleQueryCommandParser parser : getParsers()) { parser.setConnectProfileService(null); queryParserService.removeCommandParser(parser); } } private List<SampleQueryCommandParser> getParsers() { return Arrays.asList(subnetGroupsParser, createSubnetGroupParser); }}

SampleQueryCommandParser 클래스는 ConnectProfileQueryCommandParser를 상속하기 때문에, 각 파서를 초기화하는 시점에 반드시 ConnectProfileService 객체를 설정해야 하는 점에 유의합니다.

로그프레소 셸에서 bundle.start와 bundle.stop을 반복하면서 sample-subnet-groups 쿼리가 어떻게 실행되는지 확인하시기 바랍니다. 예를 들어, com.logpresso.sonar.sample 번들이 133번이라면 아래와 같이 정지할 수 있습니다.

logpresso> bundle.stop 133bundle 133 stopped.

### 쿼리 명령어 구현

이 절에서는 [SampleDriverQueryCommand.java](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/src/main/java/com/logpresso/sonar/sample/query/SampleDriverQueryCommand.java) 코드와 [SampleSubnetGroupsCommand.java](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/src/main/java/com/logpresso/sonar/sample/query/SampleSubnetGroupsCommand.java) 코드를 통해 쿼리 명령어를 구현하는 방법을 설명합니다.

#### 드라이버 커맨드

SampleDriverQueryCommand 클래스는 DriverQueryCommand 클래스를 상속하고, FieldOrdering 인터페이스를 구현합니다.

public abstract class SampleDriverQueryCommand extends DriverQueryCommand implements FieldOrdering { protected SampleParams params; public SampleDriverQueryCommand(SampleParams params) { this.params = params; } protected abstract void run(ConnectProfile profile) throws IOException; @Override public void run() { for (ConnectProfile profile : params.getProfiles()) { if (isCancelRequested()) return; try { run(profile); } catch (Throwable t) { String msg = String.format("%s (profile %s) error - %s", getName(), profile.getName(), t.getMessage()); throw new IllegalStateException(msg, t); } } }

쿼리 명령어 중에서 위와 같이 run()을 구현하여 스레드를 직접 실행하는 명령어를 드라이버 커맨드라고 지칭합니다. 하나의 쿼리에서 2개 이상의 드라이버 커맨드가 존재할 수 없습니다. 드라이버 커맨드는 가장 앞에 배치되고, 파이프로 이어지는 다른 명령어는 드라이버 커맨드의 스레드에 의해 onPush()가 호출되면서 실행됩니다. join 혹은 union 등 서브 쿼리에서 드라이버 커맨드를 추가로 사용할 수 있습니다.

쿼리 명령어는 취소 구현을 누락하지 않도록 주의해야 합니다. 루프(Loop)에서는 isCancelRequested()를 호출하여 현재 사용자나 시스템에 의해 취소가 요청된 상태인지 지속적으로 확인하고, 취소된 경우에는 즉시 수행을 중단해야 합니다.

생성자의 매개변수로 SampleParams 객체가 전달되는데, 이는 ConnectProfileParams를 상속한 클래스이기 때문에 접속 프로파일 객체 목록을 가져올 수 있습니다.

public String toString() { String s = getName(); if (params.getName() != null) s += Strings.doubleQuote(params.getName()); return s;}

모든 쿼리 명령어(QueryCommand)는 반드시 toString()을 구현해야 합니다. 쿼리 명령어의 문자열 표현은 쿼리 파서가 다시 파싱할 수 있는 정규화된 형태로 쿼리 명령어 구문을 반환해야 합니다. 이는 분산 쿼리 실행 계획을 생성하는 단계에서 쿼리 명령어 객체를 다시 쿼리 문자열로 변환하는 과정이 존재하기 때문입니다. 이 구현을 누락하면 실행 계획이 정상적으로 표시되지 않고, 분산 쿼리가 실패할 수 있습니다.

#### 쿼리 명령어의 구현

SampleSubnetGroupsCommand는 SampleDriverQueryCommand를 상속하고 아래의 메소드를 구현합니다:

* getName(): 쿼리 명령어의 이름
* getFieldOrder(): 필드 표시 순서. 쿼리 결과에 필드가 존재하지 않는 경우에는 해당 필드가 무시됩니다. 즉, 여기에서는 출력될 수 있는 모든 필드 이름 집합에 대하여 완전한 필드 순서를 정의하면 됩니다. 화면에서는 쿼리 명령어의 순서 상 FieldOrdering 인터페이스를 구현하는 마지막 명령어의 필드 출력 순서가 적용됩니다.
* run(): 드라이버 커맨드의 스레드가 실행할 기능을 구현합니다.

예제 코드는 RestApiClient 클래스를 이용하여 REST API를 호출하고, pushPipe() 메소드를 호출하여 명령어의 결과를 출력합니다. pushPipe()의 매개변수는 Row 클래스를 사용하는데, 이는 Map 인터페이스의 구현이므로 가변적인 키/값 쌍을 전달할 수 있습니다. 단, 값에는 아래의 타입만 허용됩니다:

|  |  |  |
| --- | --- | --- |
| 자바 타입 | 로그프레소 타입 | 설명 |
| java.lang.String | string | 문자열 |
| java.lang.Short | short | 16비트 정수 |
| java.lang.Integer | int | 32비트 정수 |
| java.lang.Long | long | 64비트 정수 |
| java.lang.Float | float | 32비트 실수 |
| java.lang.Double | double | 64비트 실수 |
| java.lang.Boolean | bool | 불리언 |
| java.util.Date | date | 날짜 및 시각 |
| java.net.Inet4Address | ipv4 | IPv4 주소 |
| java.net.Inet6Address | ipv6 | IPv6 주소 |
| java.util.List | list | 리스트 |
| java.util.Map | map | 맵. 키는 문자열 타입만 허용됨 |
| byte | binary | 바이트 배열 |

1초에 수십 만 건 이상의 처리 성능이 요구되는 경우에는 pushPipe(VectorizedRowBatch) 메소드를 호출하여 데이터를 출력하는 것을 권장합니다. VectorizedRowBatch의 값에는 아래의 벡터 타입을 사용할 수 있습니다.

|  |  |  |
| --- | --- | --- |
| 벡터 타입 | 로그프레소 타입 | 설명 |
| org.araqne.log.api.ObjectVector | object | 문자열 등 모든 타입 허용 |
| org.araqne.log.api.IntVector | int | 32비트 정수 배열을 최적화 처리 |
| org.araqne.log.api.LongVector | long | 64비트 정수 배열을 최적화 처리 |
| org.araqne.log.api.DoubleVector | double | 64비트 실수 배열을 최적화 처리 |
| org.araqne.log.api.BooleanVector | bool | 불리언 배열을 최적화 처리 |

약 1000건 단위로 레코드를 벡터화하여 배치 처리하면 pushPipe() 함수 호출 오버헤드가 1000배 감소하고, 잠금 경쟁으로 인한 성능 저하가 최소화되며, 가비지 컬렉션이 감소하므로 높은 성능을 구현할 수 있습니다.

### 마무리

지금까지 앱 예제 프로젝트를 모두 살펴보았습니다. 이제 아래와 같은 로그프레소 앱 API을 활용할 수 있을 것입니다:

|  |  |  |
| --- | --- | --- |
| 분류 | 이름 | 클래스 |
| 접속 프로파일 API | 접속 프로파일 서비스 | ConnectProfileService |
| 접속 프로파일 API | 접속 프로파일 | ConnectProfile |
| 접속 프로파일 API | 접속 프로파일 유형 레지스트리 | ConnectProfileFactoryRegistry |
| 접속 프로파일 API | 접속 프로파일 유형 | ConnectProfileFactory |
| 접속 프로파일 API | 접속 프로파일 설정 명세 | LoggerConfigOption |
| 접속 프로파일 API | 접속 프로파일 문자열 설정 명세 | MutableStringConfigType |
| 쿼리 API | 쿼리 파서 서비스 | QueryParserService |
| 쿼리 API | 쿼리 명령어 파서 | QueryCommandParser |
| 쿼리 API | 쿼리 오류 메시지 | QueryErrorMessage |
| 쿼리 API | 쿼리 파싱 예외 | QueryParseException |
| 쿼리 API | 접속 프로파일 쿼리 명령어 파서 | ConnectProfileQueryCommandParser |
| 쿼리 API | 접속 프로파일 쿼리 명령어 매개변수 | ConnectProfileParams |
| 쿼리 API | 드라이버 쿼리 명령어 | DriverQueryCommand |
| 쿼리 API | 쿼리 명령어 | QueryCommand |
| 쿼리 API | 필드 순서 정의 | FieldOrdering |
| 쿼리 API | 레코드 | Row |
| 유틸리티 API | REST API 클라이언트 | RestApiClient |
| 유틸리티 API | REST API 응답 | RestApiResponse |