**Sqoop 개발자 가이드 (v1.3.0-cdh3u3)**

**목차**

[1. 소개](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_introduction)

[2. 지원 배포판](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_supported_releases)

[3. Sqoop 배포판](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_sqoop_releases)

[4. 사전지식](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_prerequisites)

[5. 소스에서 Sqoop 컴파일하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_compiling_sqoop_from_source)

[6. 개발자 API 참조](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_developer_api_reference)

[6.1. 외부 API](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_the_external_api)

[6.2. 확장 API](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_the_extension_api)

[6.2.1. HBase 직렬화 확장](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_hbase_serialization_extensions)

[6.3. Sqoop 내부](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_sqoop_internals)

[6.3.1. 일반적인 프로그램 흐름](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_general_program_flow)

[6.3.2. 서브패키지](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_subpackages)

[6.3.3. MapReduce와 인터페이스하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_interfacing_with_mapreduce)

**1. 소개**

당신이 만약 Sqoop을 수정하거나 Sqoop의 내부 API를 사용하여 확장을 빌드하려고 하는 개발자나 어플리케이션 프로그래머이라면, 이 문서를 읽어야 한다. 다음의 섹션에서는 각 API의 목적, 내부 API가 어디에 쓰이는지, 추가적인 데이터베이스를 지원하고자 할 때 어느 API가 필요한지를 설명할 것이다.

**2. 지원 배포판**

이 문서는 Sqoop v1.3.0-cdh3u3에 적용된다.

**3. Sqoop 배포판**

Sqoop은 아파치 소프트웨어 재단의 오픈소스 소프트웨어 제품이다.

Sqoop 소프트웨어 개발은<http://svn.apache.org/repos/asf/incubator/sqoop/trunk>에서 진행한다. 이 사이트에서 다음 정보들을 얻을 수 있다. :

* Sqoop의 새 릴리즈와 가장 최근의 소스코드
* 이슈 추적기
* Sqoop 문서를 포함한 위키

**4. 사전지식**

Sqoop을 위하여 다음의 사전지식이 필요하다. :

* Java 소프트웨어 개발
* JDBC에 대한 친숙함
* ("새로운" MapReduce API of 0.20+를 포함하는) Hadoop의 API에 대한 친숙함
* 관계형 데이터베이스 관리 시스템과 SQL

이 문서는 당신이 리눅스 또는 리눅스와 같은 환경을 사용하고 있다고 가정한다. 만약 윈도우즈를 사용하고 있다면, 뒤따르는 대부분의 작업들을 수행하기 위하여 cygwin을 사용할 수도 있다. 만약 Mac OS X를 사용하고 있다면, 일부의 호환성 에러들을 보게 될 것이다. Sqoop은 리눅스상에서 거의 사용되고 시험되었다.

**5. 소스로부터 Sqoop 컴파일하기**

다음의 위치에서 Sqoop의 소스코드를 얻을 수 있다.: <http://svn.apache.org/repos/asf/incubator/sqoop/trunk>

Sqoop 소스코드는 git 저장소에 유지되고 있다. 저장소에서 소스를 조회하는 방법은 다음 사이트에서 제공된다. : TODO provide a page in the web site.

컴파일 방법은 소스 저장소의 루트에 있는 COMPILING.txt 파일에서 제공된다.

**6. 개발자 API 참조**

[6.1. 외부 API](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_the_external_api)

[6.2. 확장 API](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_the_extension_api)

[6.2.1. HBase 직렬화 확장](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_hbase_serialization_extensions)

[6.3. Sqoop 내부](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_sqoop_internals)

[6.3.1. 일반적인 프로그램 흐름](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_general_program_flow)

[6.3.2. 서브패키지](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_subpackages)

[6.3.3. MapReduce와 인터페이스하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_interfacing_with_mapreduce)

이 섹션은 Sqoop을 통합하고자 하는 어플리케이션 작성자와 Sqoop을 수정하고자 하는 사람들이 사용할 수 있는 API를 명시한다.

다음 세 개의 서브섹션들은 다음의 유스케이스들을 위하여 작성되었다.:

* Sqoop에 의하여 생성된 클래스와 공용 라이브러리 사용하기
* Sqoop 확장을 작성하기 (여기에서는, 더 많은 데이터베이스와 연동하는 추가적인 ConnManager 구현체)
* Sqoop 내부 수정하기

각 섹션은 시스템을 연속적으로 더 깊이 있게 설명할 것이다.

**6.1. 외부 API**

Sqoop은 Hadoop Distributed File System (HDFS)으로 가져온 테이블들에 해당하는 클래스들을 자동으로 생성한다. 그 클래스는 가져온 테이블의 각 컬럼에 대한 멤버필드들을 포함한다. 생성된 클래스들은 Hadoop에서 사용되는 직렬화 API(*Writable*과 *DBWritable* 인터페이스)를 구현한다. 그리고 다른 편의성이 있는 메서드들도 포함하고 있다.:

* 구분된 텍스트 필드를 해석하는 parse() 메서드
* 사용자가 선택한 구분자를 보존하는 toString() 메서드

자동생성된 클래스에 존재하도록 되어 있는 메서드들의 전체 집합은 추상 클래스인 com.cloudera.sqoop.lib.SqoopRecord에 명시된다.

SqoopRecord의 인스턴스들은 Sqoop의 공용 API에 의존적일 것이다. 이 API는 com.cloudera.sqoop.lib 패키지에 있는 모든 클래스들이다. 이 클래스들은 아래에 간단히 설명될 것이다. Sqoop의 클라이언트들은 이 클래스들 중 어떠한 것과도 직접 연동될 필요가 없다. Sqoop이 생성한 클래스들이 API 클래스들에 의존한다고 하더라도 말이다. 그러므로 이 API는 공용으로 인식되며, 점진적으로 발전시키는 데에 주의가 필요할 것이다.

* RecordParser 클래스는 제어가능한 구분자와 묶음(quote) 문자들을 이용하여 텍스트의 하나의 줄을 필드의 목록으로 구문분석할 것이다.
* 정적 FieldFormatter 클래스는 qoopRecord.toString()의 구현에서 쓰일 필드 내의 묶음 문자와 회피(escape) 문자를 처리하는 메서드이다.
* *ResultSet,* *PreparedStatement* 객체와 *SqoopRecords* 사이에서 데이터를 마샬링하는 것은 JdbcWritableBridge를 통해 이루어진다.
* BigDecimalSerializer는 *Writable* 인터페이스 상에서 BigDecimal 객체의 직렬화를 가능하게 하는 메서드 쌍을 포함한다.

공용 API에 대한 전체적인 상세 설명서는 Sqoop Development Wiki의 [SIP-4](http://wiki.github.com/cloudera/sqoop/sip-4)에서 참고한다.

**6.2. 확장 API**

[6.2.1. HBase 직렬화 확장](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_hbase_serialization_extensions)

이 섹션은 Sqoop이 다른 데이터베이스 벤더들과 인터페이스하게 하는 Sqoop의 확장에 사용되는 API와 주 클래스들에 대한 내용을 담고 있다.

Sqoop이 데이터베이스로부터 읽기 위하여 JDBC와 DataDrivenDBInputFormat를 사용하는 반면, JDBC 메타데이터 뿐만 아니라 서로 다른 벤더에 의해서 지원되는 SQL의 차이점들 때문에 대부분의 데이터베이스에 대한 벤더에 특정한 코드경로(codepath)가 필요하다. Sqoop은 이 문제를 ConnManager API (com.cloudera.sqoop.manager.ConnMananger)를 소개함으로써 해결한다.

ConnManager는 데이터베이스와 연동할 수 있는 모든 메서드를 정의하는 추상 클래스이다. ConnManager는 대부분의 액션을 수행하기 위해 표준 SQL을 사용하는 com.cloudera.sqoop.manager.SqlManager 추상 클래스를 확장하여 구현한다. 데이터베이스로의 실제 JDBC 커넥션을 반환하는 getConnection() 메서드를 구현하는 데에 서브클래스들이 필요하다. 서브클래스들은 모든 다른 메서드들도 마음대로 오버라이드할 수 있다. SqlManager 클래스는 개발자가 선택적으로 행위를 오버라이드할 수 있도록 protected API를 노출한다. 예를 들면, getColNamesQuery() 메서드는 getColNames()가 사용하는 SQL 쿼리가 getColNames()의 대부분의 내용을 재작성하지 않고도 수정될 수 있도록 한다.

ConnManager 구현체는 Sqoop에 특정한 클래스인 SqoopOptions로부터 환경설정 데이터를 받는다. SqoopOptions은 가변 객체이다. SqoopOptions은 특정한 매니저 별 옵션을 직접적으로 저장하지 않는다. 그 대신에 GenericOptionsParser로 명령줄 인자들을 구문분석한 후 Tool.getConf()에 의해서 반환되는 Configuration에 대한 레퍼런스를 포함한다. 이것은 옵션의 계층적인 구문분석이나 SqoopOptions의 수정 없이 "-D any.specific.param=any.value" 를 통한 인자 확장 인자를 가능하게 한다. 이 Configuration은 워크플로우 내에서 작동되는 어떠한 MapReduce 잡에게 넘겨지는 Configuration의 기초를 형성한다. 그래서 사용자들은 명령줄에서 Hadoop 상태와 관련한 필요한 맞춤 요소를 설정할 수 있다.

모든 기존 ConnManager 구현체은 상태를 보존하지 않는다. 그리하여 ConnManagers를 인스턴스화하는 시스템은 Sqoop의 동작시간에 걸쳐서 동일한 ConnMananger 클래스의 인스턴스 여러 개를 구현할 수 있을 것이다. 현재로서는 ConnManager를 인스턴스화하는 것이 가벼운 작업이며 합리성이 있게 자주 일어나지 않는다는 것을 가정한다. 그러므로 ConnManagers는 작업들이나 다른 것들 간에 캐싱되지 않는다. 현재 ConnManagers는 추상 클래스인 ManagerFactory의 인스턴스에 의해서 생성된다 (<http://issues.apache.org/jira/browse/MAPREDUCE-750>를 참고하라). 하나의 ManagerFactory 구현체(com.cloudera.sqoop.manager.DefaultManagerFactory)가 Sqoop의 모든 부분에 대하여 서비스 되고 있다. 확장은 DefaultManagerFactory를 수정할 수 없다. 그 대신 확장에 특정적인 ManagerFactory 구현체가 ConnManager와 함께 제공되어야 한다. ManagerFactory는 accept()라고 하는 단일 메서드를 가진다. 이 메서드는 사용자의 SqoopOptions를 위하여 ConnManager를 인스턴스화할 수 있는지를 결정할 것이다. 만약 그렇다면, ConnManager 인스턴스를 반환하고, 그렇지 않다면 null을 반환한다.

사용되는 ManagerFactory 구현체은 sqoop-site.xml에 설정되는 sqoop.connection.factories에 의해 결정된다. 확장 라이브러리의 사용자는 새로운 ManagerFactory와 ConnManager(들)을 포함하는 써드파티 라이브러리를 설치할 수 있으며 새로운ManagerFactory를 사용하기 위하여 sqoop-site.xml을 설정한다. DefaultManagerFactory는 SqoopOptions에 저장되는 커넥트 문자열을 구문분석함으로써 데이터베이스들을 원칙적으로 구별한다.

확장 저작자는 com.cloudera.sqoop.io, mapreduce, util 패키지의 클래스들을 구현에 활용할 수 있다. 이 패키지들은 다음 섹션에서 좀 더 자세하게 설명된다.

***6.2.1. HBase 직렬화 확장***

Sqoop은 데이터베이스에서 HBase로 가져오기를 지원한다. 데이터를 HBase로 복사할 때, HBase가 받아들일 수 있는 포맷으로 변형되어야 한다. 특별히 아래와 같이 말이다.:

* HBase의 하나(이상)의 테이블에 데이터가 위치되어야 한다.
* 입력 데이터의 컬럼은 컬럼 군에 위치되어야 한다.
* 값은 셀에 넣어질 수 있도록 바이트 배열로 직렬화되어야 한다.

위의 항목들은 HBase 클라이언트 API의 Put문을 통하여 이루어진다. HBase과의 연동은 com.cloudera.sqoop.hbase 패키지에서 수행된다. 레코드는 데이터베이스로부터 역직렬화되고 매퍼에서 내어진다. OutputFormat는 HBase로 결과를 삽입하는 데에 쓰인다. 이것은 PutTransformer 인터페이스를 통하여 이루어진다. PutTransformer는 데이터셋의 필드를 대표하는 Map<String, Object>이 입력인 getPutCommand() 메서드를 가진다. 이 메서드는 셀들을 HBase에 삽입하는 법을 설명하는List<Put>을 반환한다. 기본 PutTransformer 구현체는 ToStringPutTransformer이며 이것은 필드를 HBase로 직렬화하기 위하여 각 필드를 문자열 기반으로 표현한다.

당신만의 PutTransformer를 구현하고 그것을 맵 태스크를 위한 클래스패스에 (예를 들면, -libjars 옵션으로) 추가함으로써 이 구현체를 오버라이드할 수 있다. Sqoop의 당신의 구현체를 사용하게 하려면, -D로 그 클래스를sqoop.hbase.insert.put.transformer.class 속성에 지정하라.

당신의 PutTransformer 구현체 내부에서, 명시된 행 키 컬럼과 컬럼 군은 getRowKeyColumn() , getColumnFamily() 메서드를 통해서 사용 가능하다. 이러한 제약의 범위 밖에서 추가 Put 작업을 마음대로 만들 수 있다. 예를 들면, 부 인덱스를 대표하는 추가적인 행을 주입하는 것들을 말이다. 그러나 Sqoop은 --hbase-table로 명시된 테이블에 대하여 모든 Put 작업을 실행할 것이다.

**6.3. Sqoop 내부**

[6.3.1. 일반적인 프로그램 흐름](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_general_program_flow)

[6.3.2. 서브패키지](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_subpackages)

[6.3.3. MapReduce와 인터페이스하기](http://archive.cloudera.com/cdh/3/sqoop/SqoopDevGuide.html#_interfacing_with_mapreduce)

이 섹션은 Sqoop의 내부 아키텍처를 설명한다.

Sqoop 프로그램은 com.cloudera.sqoop.Sqoop 메인 클래스에 의해서 동작된다. 제한된 수의 추가 클래스들은 동일한 패키지에 있다. (이전에 설명한) SqoopOptions, (ManagerFactory 인스턴스를 관리하는) ConnFactory.

***6.3.1. 일반적인 프로그램 흐름***

일반적인 프로그램 흐름은 다음과 같다.:

com.cloudera.sqoop.Sqoop은 메인 클래스이며 *Tool*을 구현한다. 새 인스턴스는 ToolRunner로 시작된다. Sqoop의 첫번째 인자는 실행할 SqoopTool의 명칭을 식별하는 문자열이다. SqoopTool은 사용자의 요청 작업(예: import, export, codegen 등)의 실행을 동작시킨다.

SqoopTool API은 [SIP-1](http://wiki.github.com/cloudera/sqoop/sip-1)에서 완전히 설명하고 있다.

선택된 SqoopTool은 나머지 인자들을 구문분석하여 SqoopOptions 클래스의 적절한 필드로 설정할 것이다. 그 다음 SqoopOptions의 본체를 실행할 것이다.

그리고 나서 SqoopTool의 run() 메서드에서 가져오기나 내보내기 또는 다른 액션들이 적절히 실행된다. 전형적으로 ConnManager는 SqoopOptions 내의 데이터에 기반하여 인스턴스화 된다. ConnFactory는 ManagerFactory로부터 ConnManager를 얻기 위해 사용된다. 이 메카니즘은 이전 섹션에서 설명되었다. 가져오기, 내보내기, 다른 데이터 이동과 관련된 큰 작업은 병렬적이고 신뢰할 수 있는 방법으로 작동되도록 주로 MapReduce 잡을 실행한다. 가져오기는 MapReduce 잡을 통해서 특별히 실행될 필요는 없다. ConnManager.importTable() 메서드가 가져오기를 어떻게 최적으로 실행할 지를 결정하도록 한다. 각 주 액션은 실제로 에 의해서 제어된다. 여기에서 (com.cloudera.sqoop.orm 패키지에 있는) CompilationManager와 ClassWriter로 제어되는 코드 생성은 예외이다. importTable()가 끝난 후 Hive로 가져오기 작업은 com.cloudera.sqoop.hive.HiveImport를 통해서 관리된다. 이것은 ConnManager 구현체가 관여되지 않는다.

ConnManager의 importTable() 메서드는 파라미터를 포함하는 ImportJobContext 타입의 단일 인자를 받는다. 이 클래스는 향후 파라미터를 추가하여 확장될 것이며 더 나아가 가져오기 작업을 총괄한다. 비슷한 방식으로 exportTable() 메서드는ExportJobContext 타입의 인자를 받는다. 이 클래스들은 가져오기/내보내기를 수행할 테이블의 이름, SqoopOptions 객체에 대한 레퍼런스, 다른 관련 데이터를 포함한다.

***6.3.2. 서브패키지***

다음의 서브패키지들은 com.cloudera.sqoop 하에 존재한다.:

* hive - 데이터를 Hive로 가져오기 하는 것을 가능하게 한다.
* io - java.io.\* 인퍼테이스의 구현체들 (즉, *OutputStream*과 *Writer*).
* lib - (이전에 설명한) 외부 공용 API
* manager - ConnManager,ManagerFactory 인터페이스와 구현체들
* mapreduce - 새로운 (0.20+) MapReduce API와 인터페이스하는 클래스들
* orm - 코드 자동 생성
* tool - SqoopTool의 구현체
* util - 기타 유틸리티 클래스들

io 패키지는 HDFS에 직접 쓸 때 사용되는 *OutputStream*과 *BufferedWriter* 구현체를 포함한다. SplittableBufferedWriter는 단일 BufferdWriter가 클라이언트에게 열리게 한다. 그 클라이언트 내부적으로는 파일의 대상 임계 크기에 도달시킴으로써 연속적으로 다수의 파일들을 기록하게 된다. 이것은 차후의 MapReduce 잡이 데이터셋 당 다수의 입력 스플릿들을 여전히 사용하도록 하는 환경에서 분할되지 않는 압축 라이브러리들(예: gzip)이 Sqoop 가져오기와 함께 사용될 수 있도록 한다. 커다란 오브젝트 파일 스토리지 ([SIP-3](http://wiki.github.com/cloudera/sqoop/sip-3) 참조) 시스템의 코드도 io 패키지 내에 있다.

mapreduce 패키지는 Hadoop MapReduce 와 직접 인터페이스하는 코드를 포함한다. 이 패키지의 내용은 다음 섹션에서 더 자세히 설명된다.

orm 패키지는 클래스 생성에 사용되는 코드를 포함한다. 이것은 com.sun.tools.javac 패키지를 제공하는 JDK의 tools.jar에 의존한다.

util 패키지는 Sqoop에서 쓰이는 다양한 유티리티를 포함한다.:

* ClassLoaderStack는 현재 스레드에서 사용되는 ClassLoader 인스턴스들의 스택을 관리한다. 이것은 로컬 (스탠드얼론) 모드에서 MapReduce를 실행할 때, 현재 스레드 로 자동 생성된 코드를 로딩하기 위해서 원칙적으로 사용된다.
* DirectImportUtils는 HDFS로 직접 가져오기를 수행할 때 쓰이는 편리한 메서드를 포함한다.
* Executor는 외부 프로세스들을 시작하고, AsyncSink(아래의 상세 내용 참조)에 의해서 생성된 스트림 핸들러에 이 프로세스들을 연결시킨다.
* ExportException는 내보내기가 실패할 때 ConnManagers가 던진다.
* ImportException는 가져오기가 실패할 때 ConnManagers가 던진다.
* JdbcUrl는 커넥트 문자열의 구문분석을 처리한다. 커넥트 문자열은 URL형식이지만 사양에 일치하지는 않는다. (특별히 JDBC 커넥트 문자열은 multi:part:scheme:// 컴포넌트를 가질 수 있다.)
* PerfCounters는 사용자에게 표시하기 위한 목적으로 전송율을 산정하는 데에 사용된다.
* ResultSetPrinter는 *ResultSet*을 보기 좋게 출력한다.

Sqoop이 외부 프로세스들로부터 stdout 읽도록 하는 경우도 있다. 가장 직관적인 경우는 LocalMySQLManager와 DirectPostgresqlManager에 의해서 수행되는 직접모드 가져오기이다. Runtime.exec()에 의해서 프로세스가 만들어진 후, 프로세스의 stdout (Process.getInputStream())과 잠재적으로 stderr (Process.getErrorStream())은 처리(handle)되어야 한다. 이 스트림들로부터 충분한 데이터를 읽어들이는 데에 실패하면 쓰기가 더 진행되기 전에 외부 프로세스는 차단 당할 것이다. 결과적으로 이 프로세스들은 가급적이면 비동기적으로 처리되어야 한다.

Sqoop 용어에서 "async sink"는 InputStream을 얻고 끝까지 읽어들이는 스레드이다. 이것은 AsyncSink 구현체를 통해서 이루어진다. com.cloudera.sqoop.util.AsyncSink 추상 클래스는 이 팩토리 클래스가 수행해야 하는 작업들을 정의한다.processStream()는 InputStream 인자로부터의 데이터 처리를 즉시 시작하기 위하여 또 다른 스레드를 만들어낼 것이다. 그리고 이 스트림을 끝까지 읽어야 한다. join() 메서드는 외부 스레드가 이 처리가 완료될 때까지 기다리도록 한다.

몇몇의 “비축" AsyncSink 구현체들도 제공된다. LoggingAsyncSink는 log4j INFO 문으로 InputStream 상에 모든 것을 거듭하여 쓸 것이다. NullAsyncSink은 모든 입력을 소비하지만 아무것도 하지 않는다.

외부 프로세스들을 사용하는 다양한 ConnManagers은 내부 클래스로서 고유의 AsyncSink 구현체를 가진다. 그 구현체는 데이터베이스 도구로부터 데이터를 조회하여 HDFS를 향하여 보내며 그 사이에 포맷팅 변환을 수행할 것이다.

***6.3.3. MapReduce와 인터페이스하기***

Sqoop은 가져오기와 내보내기를 일으키는 MapReduce 잡을 스케줄링한다. MapReduce 잡의 구성과 실행은 몇 가지 공통 단계를 따른다. (InputFormat 구성, OutputFormat 구성, Mapper 구현체, 등). 이 단계는com.cloudera.sqoop.mapreduce.JobBase 클래스 내부에 형식화 되어 있다. JobBase는 사용자가 사용할 InputFormat, OutputFormat, Mapper를 명시하도록 한다.

JobBase는 가져오기나 내보내기 각각에 관련된 잡에 공통적으로 구성 단계를 더 잘 지원하는 ImportJobBase, ExportJobBase로 서브클래스화 된다. ImportJobBase.runImport()는 테이블을 HDFS에 가져오기 위하여 설정 단계를 호출하고 잡을 실행할 것이다.

이 기본 클래스들의 서브클래스들 역시 존재한다. 예를 들면, DataDrivenImportJob는 가져오기를 실행하기 위해 DataDrivenDBInputFormat을 사용한다. 이것은 다양한 ConnManager 구현체들이 사용하는 가장 공통적인 가져오기 형식이다. MySQL은 직접모드 가져오기를 실행하기 위해 다른 클래스(MySQLDumpImportJob)를 사용한다. 그 클래스의 커스텀 Mapper, InputFormat 구현체도 이 패키지 내에 있다.