

실행환경 배포도구

- Manual
 - 실행환경 배포도구
 - 배포도구 설치
 - 환경설정
 - 실행환경 배포
 - 메타 파일 작성

배포도구 설치

실행환경 배포 도구는 현재 이클립스 플러그인 형태로 제공되며 전자정부 표준프레임워크 포털에서 다운로드 받아서 직접 설치하거나, 개발환경의 메뉴를 이용해 update site로 설치할 수 있다.
개발자 개발환경(Developer Development Tool) Full Version 2.0.0에 포함되어 있으므로 full 버전을 사용하면 따로 설치 할 필요가 없다.















Customize Development Tool을 이용한 설치

개발자용 개발환경(Implementation Tool) Light Version 2.0.0의 Customize Development Tool 이용하여 설치 할 수 있다. Customize Development Tool의 eGovFrame RTE distribution Tool을 선택하고 설치한다.

- [Customize Development Tool 가이드](#)

다운로드를 이용한 설치

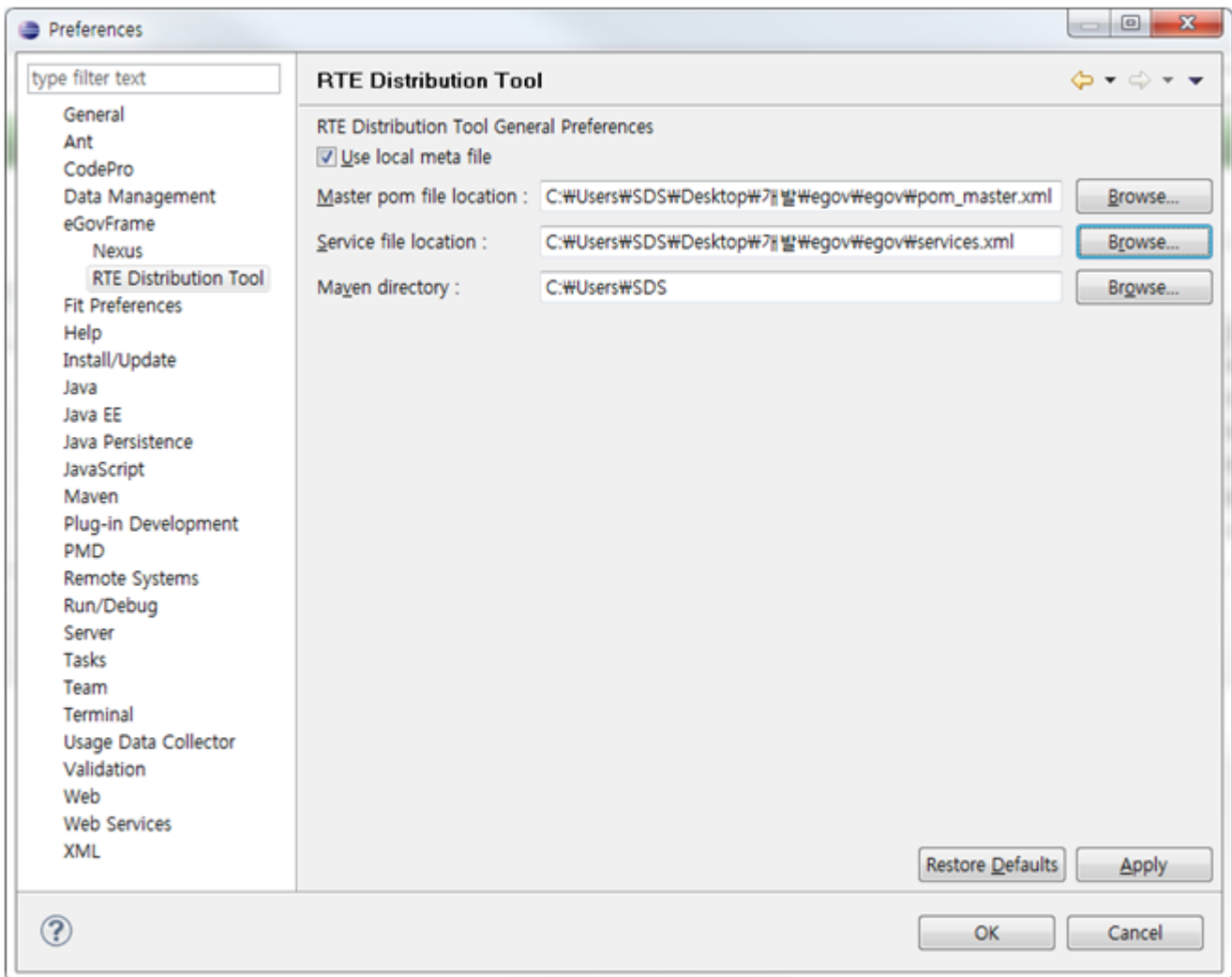
전자정부 표준프레임워크 포털의 실행환경 다운로드페이지에서 실행환경 배포도구를 다운받아 이클립스 설치 폴더의 dropins 폴더에 jar파일을 복사하거나 혹은 plugin 폴더에 압축을 해제하는 방법으로 설치 할 수 있다.

	configuration	2011-10-14 오후...	파일 폴더	
	dropins	2011-08-09 오후...	파일 폴더	
	features	2011-08-09 오전...	파일 폴더	
	p2	2011-06-13 오후...	파일 폴더	
	plugins	2011-08-09 오전...	파일 폴더	
	readme	2011-06-13 오후...	파일 폴더	
	second	2011-08-09 오후...	파일 폴더	
	.eclipseproduct	2010-07-29 오전...	ECLIPSEPRODUCT...	1KB
	artifacts.xml	2011-08-09 오전...	XML 문서	252KB
	eclipse.exe	2010-12-22 오후...	응용 프로그램	52KB
	eclipse.ini	2011-09-22 오전...	구성 설정	1KB
	eclipse.exe	2010-12-22 오후...	응용 프로그램	24KB
	epl-v10.html	2005-02-25 오후...	HTML 문서	17KB
	notice.html	2010-04-27 오후...	HTML 문서	9KB

플러그인을 설치한 뒤, 이클립스를 실행(이미 실행된 경우에는 재기동)하면 배포도구의 설치가 완료된다.

환경설정

실행환경 배포도구는 기본적으로 실행에 필요한 메타정보 파일을 내장하고 있지만, 환경설정을 통해 커스터마이징된 메타 정보 파일을 사용하여 실행할 수 있다.

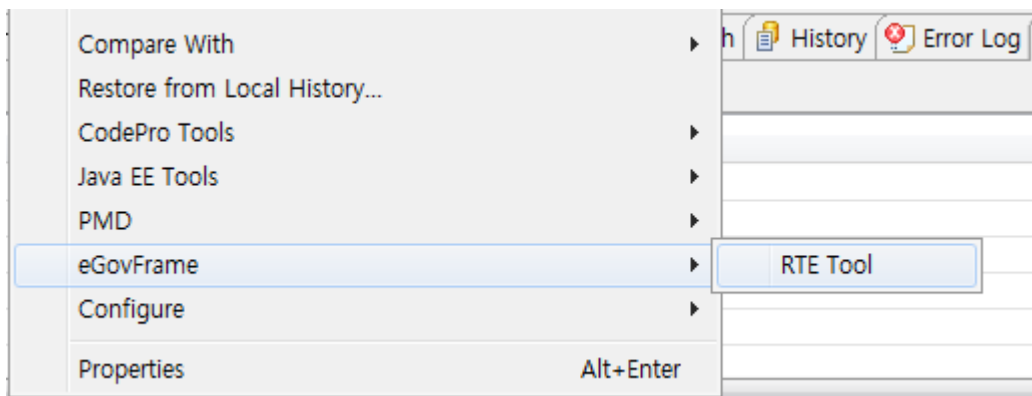


Use local meta file : 직접 작성한 메타정보 파일을 사용하려면 체크한다.
 Master pom file location : 직접 작성한 Master pom 파일의 위치를 지정한다.
 Service file location : 직접 작성한 Service 파일의 위치를 지정한다.
 Maven directory : maven 명령어를 사용하기 위해 maven이 설치된 디렉토리를 지정한다.

실행환경 배포

실행환경 배포도구는 전자정부프레임워크에 포함된 실행환경 라이브러리를 사용하기 위해 사용자가 직접 라이브러리를 다운로드 받아 프로젝트에 포함시키거나 Maven pom.xml에 Dependency를 추가하지 않고, 실행환경 라이브러리를 사용하기 위한 도구이다. 라이브러리를 받아 오기 위해 Maven을 사용한다

실행환경 라이브러리를 사용한 Project를 우클릭을 눌러서 Context Menu를 호출하여 eGovFrame 메뉴 밑의 RTE Tool을 선택해 실행한다.



실행하게 되면 설치된 서비스와 설치 가능한 서비스로 분류되어 표시되고 원하는 서비스를 선택해서 설치하거나 제거, 업데이트할 수 있다.

Distribution Tool
실환경 배포도구

실환경 배포도구

설치된 서비스

	레이어	서비스 명	버전	업데이트
<input type="checkbox"/>	Core	SpringFramework	3.0.5.RELEASE	완료
<input type="checkbox"/>	Core	Foundation Common	2.0.0	업데이트 가능
<input type="checkbox"/>	Foundation	Server Security	2.0.0	업데이트 가능
<input type="checkbox"/>	Foundation	Cache	2.4.1	완료
<input type="checkbox"/>	Presentation	MVC	2.0.0	업데이트 가능
<input type="checkbox"/>	Presentation	Ajax Support	2.6	완료

업데이트 제거

설치가 가능한 서비스

	레이어	서비스 명	버전
<input type="checkbox"/>	Foundation	Compress/Decompress	1.1
<input type="checkbox"/>	Foundation	Encryption/Decryption	2.0.0
<input type="checkbox"/>	Foundation	Excel	2.0.0
<input type="checkbox"/>	Foundation	File Handling	2.0.0
<input type="checkbox"/>	Foundation	File Upload/Download	1.2.2
<input type="checkbox"/>	Foundation	FTP	3.0.1
<input type="checkbox"/>	Foundation	Mail	1.2
<input type="checkbox"/>	Foundation	Marshalling/Unmarshalling	1.3.2
<input type="checkbox"/>	Foundation	Object Pooling	1.5.6

설치 새로고침

☐ 서비스 설치 후 Maven install 실행

Rdt Tool

설치 : 설치 가능 한 서비스를 선택해 설치를 누르면 프로젝트의 pom.xml에 해당 라이브러리가 추가되고 전자정부 개발환경에서 자동으로 maven update가 실행되어 라이브러리를 받는다.

제거 : 설치된 서비스를 선택해서 제거를 할 수 있다.

업데이트 : 설치된 서비스 중 업데이트 가능으로 표시되어 있는 서비스를 선택해서 최신버전으로 업데이트 할 수 있다.

새로고침 : 화면을 새로 고친다.

서비스 설치 후 Maven Install 실행 : 체크하면 서비스 설치후 자동으로 Maven Install 명령을 실행한다

메타 파일 작성

메타파일을 사용자가 직접 작성하고자 하는 경우 pom_master.xml과 services.xml을 추가, 편집, 삭제 할 수 있다.

pom_master.xml 파일은 실제 프로젝트 내 pom.xml에 사용될 dependency들을 정의하고, services.xml 파일은 각 서비스에 포함된 라이브러리들을 정의한다.

pom_master.xml 구조

- 일반 pom파일과 구조가 동일, <dependency>태그를 추가하면 된다.

services.xml 구조

- <services> : root
- <name> : 서비스명
- <layer> : 레이어 종류














- <dependencies> : 서비스에 포함될 라이브러리들이 <dependency>로 정의되어 있다.
- <dependency> : 서비스에 포함될 라이브러리의 dependency ID
- dependency ID : pom_master.xml 에 기술된 라이브러리의 groupId 와 artifactId 의 조합

호환성 점검 도구

- Manual
 - 호환성 점검 도구
 - 점검도구 설치
 - 환경설정
 - 호환성 점검
 - 호환성 점검 기본 룰셋
 - 룰셋 파일 작성

점검도구 설치

호환성 점검 도구는 현재 이클립스 플러그인 형태로 제공되며, 전자정부 표준프레임워크 포털에서 다운로드 받아서 이클립스 설치 폴더의 dropins 폴더에 압축 파일을 집어 넣거나 혹은 plugin 폴더에 압축을 해제하여 설치할 수 있다.

	configuration	2011-10-12 오전...	파일 폴더
	dropins	2011-06-15 오전...	파일 폴더
	features	2011-08-18 오전...	파일 폴더
	p2	2011-07-12 오후...	파일 폴더
	plugins	2011-08-18 오전...	파일 폴더
	readme	2011-07-12 오후...	파일 폴더
	.eclipseproduct	2010-07-29 오전...	ECLIPSEPRODUCT...
	artifacts.xml	2011-08-18 오전...	XML 문서
	eclipse.exe	2011-03-21 오후...	응용 프로그램
	eclipse.ini	2011-08-18 오전...	구성 설정
	eclipsesec.exe	2011-03-21 오후...	응용 프로그램
	epl-v10.html	2005-02-25 오후...	HTML 문서
	notice.html	2011-02-04 오후...	HTML 문서

플러그인을 설치한 뒤, 이클립스를 기동 (이미 실행된 경우에는 재기동) 하면 점검도구의 설치가 완료된다.

만약 표준프레임워크 개발환경이 아닌 독자적인 이클립스에 설치하는 경우, eclipse.ini 파일을 열어서 파일 인코딩이 UTF-8로 지정되었는지 확인하고 없다면 다음 옵션을 추가해준다.

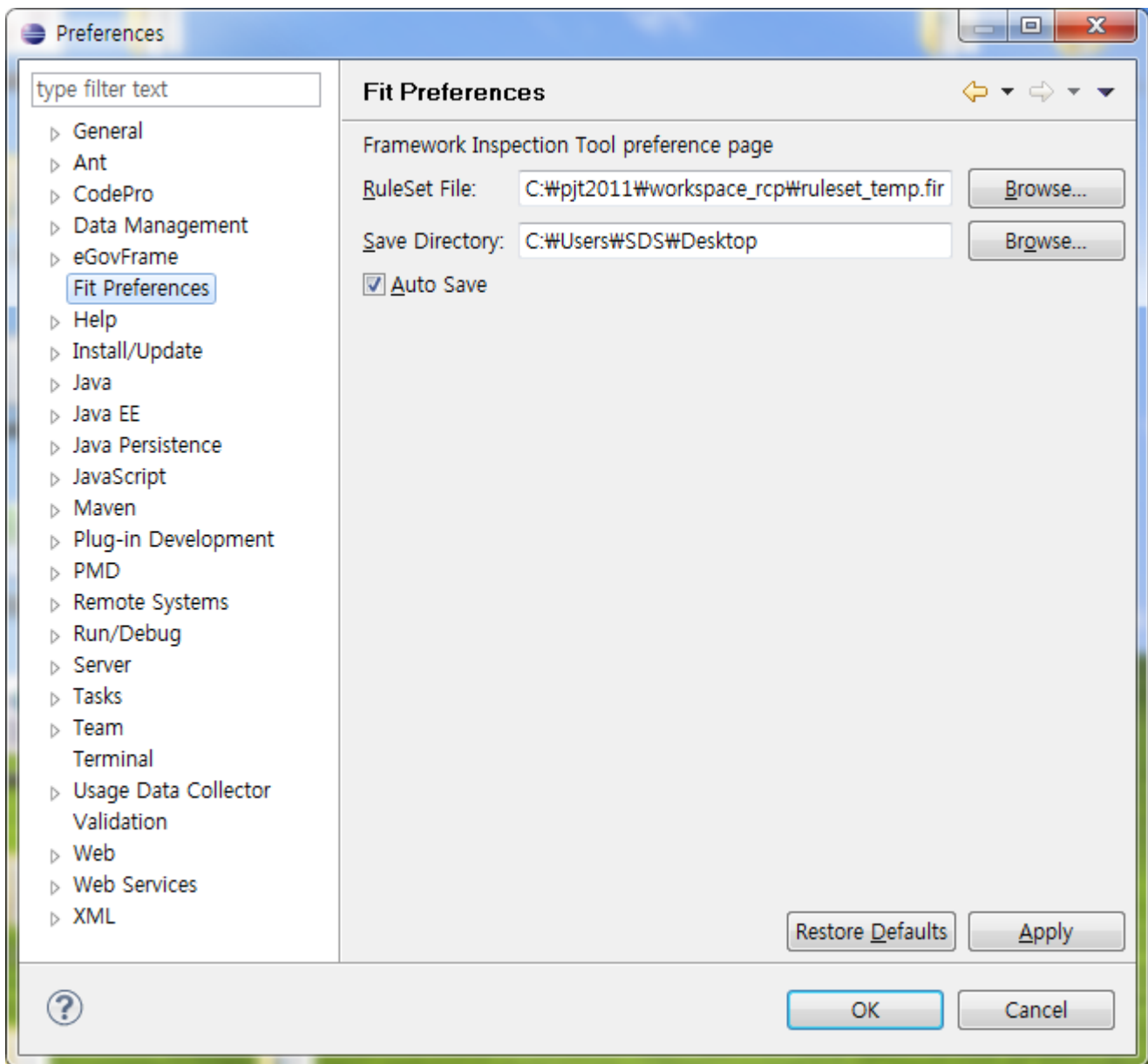
```
-Dfile.encoding=UTF-8
```

환경설정

호환성 점검도구는 호환성 점검 규칙을 정의한 룰셋 파일을 필요로 하며, 환경설정을 통해 룰셋 파일의 위치를 지정해주어야 한다. 또한 환경 설정에서는 결과를 저장할 폴더 및 자동저장 여부를 설정할 수 있다.


- RuleSet File : 호환성 점검을 위한 룰셋 파일의 위치를 지정한다.
- Save Directory : 점검 결과가 저장될 폴더를 지정한다.
- Auto Save : 체크할 경우, 호환성 점검이 이루어질 때 마다 점검 결과가 저장 폴더에 자동으로 저장된다. 체크가 해제되었을 경우, 별도로 저장 버튼을 눌러야만 점검 결과가 저장된다.

호환성 점검



호환성 점검은 점검을 원하는 프로젝트를 Project Explorer 상에서 우클릭을 눌러서 Context Menu를 호출하여 Framework Inspection을 선택하면, 자동으로 이루어지게 된다. 점검에는 일정한 시간이 소요되며, 점검이 완료되면 하단에 Framework Inspection Tool 뷰가 추가되고 결과가 나타나게 된다. 점검 결과는 에러, 메시지, 위반사항 항목으로 구성된다.

Result	
▲ Message (2 items)	
하이버네이트DAO 그룹이 존재하지 않습니다.	
테스트 케이스 그룹이 존재하지 않습니다.	
▲ Violations (2 items)	
실행환경 변경금지 : 전자정부 실행환경 라이브러리는 변경할 수 없습니다.	
[egovframework.rte.ptl.mvc-2.0.0.jar] egovframework.rte.ptl.mvc-2.0.0.jar 파일은 변경될 수 없습니다.	
[egovframework.rte.ptl.mvc-2.0.0.jar] egovframework.rte.ptl.mvc-2.0.0.jar 파일의 크기는 원본과 일치하여야 합니다.	

- Error : 점검을 진행하는 동안 발생한 심각한 오류를 표시한다. 이 항목에는 주로 잘못 정의된 룰셋의 내용, 점검대상 프로젝트의 심각한 컴파일 오류등이 기록되지만, 점검 자체는 그대로 진행하게 된다. 만약 점검을 더 이상 진행할 수 없을 정도의 큰 오류가 발생한 경우 점검이 중단된다.  호환성가이드라인.ppt
- Message : 점검을 진행하는 동안 발생한 메시지들을 표시한다. 이 항목에는 점검에 큰 영향을 주지 않는 메시지들이 표시된다.
- Violations : 점검규칙을 위반한 사항이 있을 경우 이 항목에 각각의 세부 룰 별로 위반한 클래스 및 파일들을 표시해 주게 된다.

저장 버튼을 누르거나 결과 화면에서 우클릭 하여 Context Menu에서 save result를 선택한 경우 검사 결과가 환경설정에서 지정된 디렉토리에 저장된다. 환경 설정에서 자동 저장 (Auto Save)이 체크된 경우 검사를 실시할 때 마다 지정된 디렉토리

에 파일이 저장된다.

호환성 점검 기본 룰셋

전자정부 표준프레임워크 호환성 점검을 위한 표준 룰셋은 표준프레임워크 포털사이트에서 다운로드 할 수 있으며, 해당 룰셋에 대한 가이드 라인은 다음 파일을 참고하도록 한다.

 호환성가이드라인.ppt

룰셋 파일 작성

호환성 점검 규칙을 직접 작성하고자 하는 경우 룰셋 파일을 변경하여 임의의 규칙을 지정하거나 추가, 삭제할 수 있다. 룰셋 파일의 정의는 다음과 같이 한다.

주석문

주석문은 #, , ### 으로 표시한다. 주석으로 표시된 영역은 룰셋 파일의 해석에 영향을 주지 않는다. * # 또는 : 해당 문자의 뒷 부분을 모두 주석처리 한다.

- ### : 해당 라인의 아랫부분을 모두 주석처리 한다.

체크리스트(Checklist), 콜렉터(Collector) 정의

체크리스트 클래스와 콜렉터 클래스를 정의한다. 해당 정의는 룰셋 파일의 최상단에 존재해야 한다. 여기에 정의된 클래스들은 점검도구 실행시 동적으로 로딩되어 점검을 수행하게 된다. 만약 정의된 클래스를 찾을 수 없는 경우 에러메시지를 출력하고 해당 체크리스트의 점검은 수행하지 않는다. 이 경우 점검 결과의 정확성을 보장할 수 없으므로 정의시 반드시 존재하는 체크리스트 클래스를 지정하는 것이 중요하다.

```
Checklist [예약어] : [클래스명] # 주석
Collector [예약어] : [클래스명] # 주석
```

예제 :

```
Checklist has class annotation : egovframework.mgt.fit.inspector.checker.impl.ClassAnnotationChecker
Collector query id collector : egovframework.mgt.fit.inspector.checker.impl.QueryIDCollector
```

예약어는 점검 규칙을 정의할 때 검사항목을 표시하는 용도로 쓰인다. 클래스명은 모든 패키지 경로를 포함하여 작성되어야 한다. 주석은 생략 가능하다.

클래스 별칭 정의

클래스 별칭은 반복적으로 나타나는 클래스의 인용을 위하여 별도의 별칭을 지정하여 룰셋 정의를 간편하게 하기 위한 정의이다.

```
Class [예약어]([설명]) is [클래스명] # 주석
```

예약어는 룰셋 정의에서 사용될 별칭(Alias)을 의미한다. 설명은 해당 Alias의 한글명이 들어갈 수 있으며, 생략 가능하다. 클래스명은 모든 패키지 경로를 포함하여 작성되어야 하며, 주석은 생략 가능하다.

그룹, 리스트 정의

그룹 및 리스트 정의는 검사를 위하여 임의의 조건을 가지는 클래스들의 집합을 묶어두기 위한 규칙이다. 그룹 혹은 리스트 정의 내에서 다른 그룹 혹은 리스트에 대한 참조가 발생할 수 있는데, 이 경우 참조할 그룹 혹은 리스트의 정의가 더 상단에 표시되어서 먼저 해석되도록 해야 한다. 만약 순서가 바뀌어서 정의될 경우 그룹에 대한 참조를 찾을 수 없기 때문에 오류가 발생하고 해당 그룹의 정의는 이루어지지 않게 된다.

```
Group [그룹명]([설명]) is
> [검사대상 정의]
+ [필수체크리스트1]
+ [필수체크리스트2]
- [통과체크리스트1]
- [통과체크리스트2]
....
```

```
List [리스트명]([설명]) is
> [검사대상 정의]
+ [콜렉터명]
```

그룹명은 규칙 정의에서 사용될 그룹의 ID를, 설명은 점검 결과에 표시될 그룹명으로, 설명이 정의되지 않은 경우 그룹명이 설명을 대체하게 된다. 검사대상은 해당 그룹이 점검할 대상으로 검사대상 정의 항목을 참고하여 지정하도록 한다. + 혹은 -로 표시된 부분은 체크리스트 상에 정의된 규칙을 실제로 지정하는 부분으로, 필수물은 +로, 통과물은

-로 지정하게 된다. 필수롤은 위반시 그룹 정의 대상에서 제외되는 룰을, 통과롤은 단 하나라도 통과할 경우 그룹 정의 대상에 포함되는 룰을 지정하게 된다. 단, 통과롤을 모두 통과하지 못할 경우 필수롤을 모두 통과하였다고 하더라도 규칙을 만족하지 못하게 된다. 리스트의 정의 역시 그룹과 동일하게 지정되지만, 리스트는 단 하나의 컬렉터만을 규칙으로 가진다.

검사대상 정의

검사대상 정의는 해당 규칙의 점검 대상이 될 규칙을 정하는 것으로 예약어 혹은 그룹명이 들어갈 수 있다. 검사대상은 > 표시 뒤에 작성되게 된다. 검사대상은 다음 검사대상이 지정되기 전 까지 하위에 속한 모든 검사 규칙에 대해서 유효하다. 그룹 또는 규칙 정의의 첫 줄에는 검사대상 정의가 포함되는 것이 좋으며, 포함되지 않을 경우 java class 가 기본 검사 대상으로 지정되나 추천하지 않는다.

```
> [검사대상 정의]
```

```
java class, xml file, java binary, all group [그룹명], any group [그룹명], just
```

- java class : 모든 자바 클래스를 대상으로 한다.
- xml file : 모든 XML 파일을 대상으로 한다.
- java binary : 모든 자바 바이너리를 대상으로 한다.
- all group [그룹명] : 해당 그룹의 모든 구성요소가 조건을 만족해야 한다. group [그룹명] 형태로 작성하여도 동일한 의미를 가진다.
- any group [그룹명] : 해당 그룹에서 단 하나의 구성요소가 조건을 만족해도 된다.
- just : 검사 대상과 관련 없이 독립적으로 동작하는 규칙에 대해서 사용한다.

규칙 정의

규칙 정의는 전체를 포괄하는 규칙과 해당 규칙에서 지켜야할 세부 체크리스트의 집합으로 이루어진다.

```
Rule [규칙명]([설명]) is # 주석
> [검사대상 정의] is
+ [필수체크리스트1] # 주석
+ [필수체크리스트2] # 주석
- [통과체크리스트1] # 주석
- [통과체크리스트2] # 주석
...
> [검사대상 정의] is # 주석
+ [필수체크리스트3] # 주석
+ [필수체크리스트4] # 주석
- [통과체크리스트3] # 주석
- [통과체크리스트4] # 주석
...
```

일반적으로 주석이 룰셋 파일의 해석에 도움을 주는 용도로만 사용되는 것에 비해서 규칙정의의 주석은 검사 결과에 해당 규칙의 위반 여부를 표시할 때 같이 표시되는 의미를 가지고 있다. 따라서 규칙 정의시 주석으로 해당 규칙의 상세한 설명을 적어주는 것이 좋다. 규칙 정의의 주석문은 위반시 표시될 것을 염두해두어 '~해야 한다.' '~는 할 수 없다.' 식으로 적어주는 것이 자연스럽다.

필수롤과 통과롤의 주석 역시 같은 의미를 가지고 있으며, 검사대상 및 각 룰의 사용법은 그룹 및 리스트 정의시와 동일하다. 만약 해당 항목의 결과가 false로 나와야 통과시키고 싶다면 항목 앞에 not 또는 do not을 붙이면 된다.

XML Query Language 사용

XML 파일을 검사하기 위해서는 기본제공하는 HasQueryResultChecker를 사용하거나 독자적인 XML 검사 체커 클래스를 정의하여 사용할 수 있다. 이 분석도구에서 XML 파일은 Structure의 형태가 아닌 Map의 형태로 저장되며, 해당 맵은 각각의 XML 노드에 대하여 노드의 계층적 위치를 Key값으로, 노드의 내용을 Value로 저장하여 활용하고 있다. 해당 노드에 대한 검사를 위해서 다음과 같이 사용할 수 있다.

- | 연결구분자 : 여러개의 쿼리를 연결하기 위하여 사용할 수 있다. 구분자를 통해 구분되는 각각의 쿼리의 결과를 모두 합쳐서 결과값으로 출력하며, 중복은 제거된다.
- . 명시적 계층구분자 : 각 노드에 명시적으로 접근하기 위하여 . 을 이용하여 노드를 표기할 수 있다.
- .. 암시적 계층구분자 : 노드 구조를 암시적으로 표현하기 위하여 ..을 이용하여 중간 노드의 구분을 모두 생략할 수 있다.
- @ Attribute 검색자 : 해당 Attribute를 가진 노드들을 결과값으로 포함시킨다.
- = Attribute 검색조건 : 해당 Attribute가 조건값과 일치하는 노드들만 결과값으로 포함시킨다.

```
sqlMap..@id
```

위 쿼리는 루트 요소가 sqlMap인 iBatis SQL Map 파일을 기준으로 모든 id 값을 결과값으로 출력한다. 노드의 중간구조를 .. 구분자를 통하여 생략시켰기 때문에 id라는 attribute를 가지는 모든 노드가 결과에 포함된다.

```
sqlMap..@id=goodsDAO.insertGoods
```


위 쿼리는 해당 XML 파일에서 id Attribute를 가지는 모든 노드들 중 그 id값이 goodsDAO.insertGoods인 노드를 찾아낸다. XML 파일에 따라서 결과가 다수 나올 수도, 하나도 나오지 않을 수 있다. 이와 같은 경우 HasQueryResultChecker는 해당 결과값이 하나 이상 나왔을때 true를 반환하는 체커 클래스이다.

```
sqlMap.select..@id|sqlMap.insert..@id|sqlMap.delete..@id|sqlMap.update..@id|sqlMap.procedure..@id
```

위 쿼리는 | 구분자를 이용하여 해당 XML 파일 내의 모든 CRUD 기능을 담당하는 노드의 id값을 수집하고 있다. sqlMap.select 와 같은 형태로 명시적으로 연결되었기 때문에 루트 엘리먼트의 바로 하위에 select, insert, delete, update, procedure 노드가 있는 경우에만 해당하며 그 이후에는 암시적으로 하위 노드를 검색하고 있으므로 해당 노드의 어떤 하위 노드이더라도 id 값을 포함하고 있으면 결과에 포함시킨다.

체크리스트 사용

```
[+/-] [체크리스트명] [검사 값]
또는
[+/-] [체크리스트명] ${클래스 별칭}
또는
[+/-] [체크리스트명] group [그룹명]
```

체크리스트명은 사전에 정의된 체크리스트 명을 그대로 사용한다. 찾을 수 없을 경우 에러메시지를 추가하고, 해당 체크리스트는 검사하지 않는다. 검사값에는 실제 체크리스트 클래스에 전달할 값을 써줄 수 있다. 검사값이 필요없는 클래스의 경우 생략할 수 있다. 만약 특정 클래스명을 전달할 경우 사전 정의된 해당 클래스의 별칭을 지정해 줄 수 있으며, 이는 룰셋 파싱시 해당 클래스의 전체명으로 대체되어 해석된다. 단, \$ 기호 뒤의 단어 (공백이 있기 까지의 단어)는 모두 클래스 별칭으로 인식하므로 클래스 별칭을 주고 .[메소드명] 형태로 연결해서 사용하는 것은 불가능하다. 만약 체크리스트가 그룹을 인자로 받을 경우 그룹명을 직접 부여할 수 있다. 이 경우 모든 검사 대상과 모든 그룹 구성요소가 m*n 형태의 체크리스트를 생성하여 검사를 수행하게 된다. 이 경우 검사속도가 매우 떨어질 가능성이 있으므로 group 형태의 인자를 받는 체크리스트 클래스는 Object[] 형태의 인자를 받는 check 메소드를 overriding 하고 GroupChecker 인터페이스를 implements 해주는 것이 좋다. 이 경우 그룹 구성요소가 한 번에 배열의 인자로 전달되기 때문에 검사 시간을 매우 크게 단축시킬 수 있다.

Basic Checker Class 사용

다음은 분석도구에서 기본적으로 제공하는 Checker 클래스의 사용법이다.

1. Annotation 체커

```
[+/-] has class annotation [어노테이션 명]
[+/-] has annotation [어노테이션 명]
[+/-] has field annotation [어노테이션 명]
[+/-] has local annotation [어노테이션 명]
[+/-] has method annotation [어노테이션 명]
```

- ClassAnnotationChecker : 해당 클래스의 클래스 정의 위의 어노테이션이 있는지 검사한다.
- AnyAnnotationChecker : 해당 클래스의 어느 부분에서든 해당 어노테이션이 있는지 검사한다.
- FieldAnnotationChecker : 필드 정의 위에 해당 어노테이션이 있는지 검사한다.
- LocalAnnotationChecker : 지역변수 정의 위에 어노테이션이 있는지 검사한다.
- MethodAnnotationChecker : 메소드 정의 위에 어노테이션이 있는지 검사한다.

다음 코드는 @Repository 어노테이션이 클래스 정의 위에 선언된 클래스들에 대해서 true를 리턴하는 사용예제이다. 어노테이션 명을 지정할때 @표시는 생략할 수 있다.

```
+ has class annotation @Repository
```

2. 상속관계 체커

```
[+/-] extends [클래스 명]
[+/-] implements [인터페이스 명 / any ]
```

- ExtendsChecker : 해당 클래스가 전달된 클래스를 상속하고 있는지 검사한다. 클래스명은 패키지를 포함한 모든 경로로 전달해야 한다.
- ImplementsChecker : 해당 클래스가 전달된 인터페이스를 구현하고 있는지 검사한다. 인터페이스명은 패키지를 포함한 모든 경로로 전달해야 하며, any를 전달할 경우 인터페이스에 관계없이 임의의 인터페이스를 구현했는지 검사한다.

3. 메소드 호출 체커

```
[+/-] invoke method [메소드 명]
[+/-] invoked by method [메소드 명]
[+/-] invoked by class [클래스 명]
[+/-] invoke class [클래스 명]
[+/-] has parameter [파라미터 명/ignore case] [리스트 명]
```

- InvokeMethodChecekr : 해당 클래스가 전달된 메소드를 호출했는지 여부를 검사한다. 메소드명은 패키지를 포함한 클래스의 전체경로와 메소드명을 점(.)으로 이어서 표기한다. 파라미터 표기는 생략하며 오버로딩된 메소드 간에는 중복이 발생할 수 있다.
- InvokedByMethodChecker : 해당 클래스가 전달된 메소드로 부터 호출 당했는지 여부를 검사한다. 메소드명 표기는 InvokeMethodChecker와 동일하다.
- InvokedByClassChecker : 해당 클래스가 전달된 클래스에 속한 임의의 메소드로 부터 호출 당했는지 여부를 검사한다.
- InvokeClassChecker : 해당 클래스가 전달된 클래스에 속한 임의의 메소드를 호출했는지 여부를 검사한다.
- HasParameterChecker : 해당 클래스의 메소드 호출이 전달된 파라미터를 포함하고 있는지 여부를 검사한다. 여기서 파라미터를 호출시 전달되는 실인자를 기준으로 한다. 이 체커를 사용할때는 검사를 원하는 파라미터를 직접 전달할 수도 있고, 콜렉터를 통하여 수집된 리스트를 전달해줄 수도 있다. 이 때 전달값에 ignore 혹은 ignore case를 전달해주면 리스트를 검사할때 대소문자 검사를 하지 않는다.

다음 코드는 쿼리ID를 수집하여 리스트로 작성하고 해당 클래스가 수집된 쿼리ID를 실인자로 하는 메소드 호출을 시도하는지 여부를 검사한다.

```
List QUERY_ID is
> xml file
+ query id collector

Rule CALL_QUERY(DAO 클래스) is
> java class
+ has parameter $QUERY_ID
```

4. 클래스/파일 위치 체커

```
[+/-] in package [패키지 명]
[+/-] under directory [project:[디렉토리 경로] / 디렉토리 경로]
[+/-] in directory [project:[디렉토리 경로] / 디렉토리 경로]
[+/-] has common super directory [리스트명]
```

- InPackageChecker : 해당 클래스가 지정된 패키지 내에 있는지 검사한다.
- UnderDirectoryChecker : 해당 클래스가 지정된 디렉토리 혹은 그 하위에 위치하는지 검사한다. 디렉토리명은 루트 디렉토리를 기준으로 하는 절대경로 혹은 프로젝트 루트를 기준으로 하는 상대경로로 지정해줄 수 있으며, 상대경로를 사용할 경우 디렉토리 경로 앞에 project: 지시어를 붙여준다.
- IndirectoryChecker : 해당 클래스가 지정된 디렉토리에 위치하는지 검사한다. 디렉토리 경로 표기는 UnderDirectoryChecker와 동일하다.
- has common super directory : 전달된 리스트가 공통적인 상위 디렉토리를 가지고 있는지 검사한다. 검사대상을 따로 지정할 필요가 없으므로 Target을 just로 부여한다.

5. 클래스/파일 검증 체커

```
[+/-] binary equal to [파일명:해시코드]
[+/-] if exist and binary equal to [파일명:해시코드]
[+/-] size equal to [파일명:크기]
[+/-] if exist and size equal to [파일명:크기]
[+/-] file exist [파일명]
[+/-] at least one [group 그룹명]
[+/-] filename starts with [파일명]
[+/-] class name starts with [클래스명]
```

- BinaryEqualToChecker : 전달된 파일의 SHA-1 해시코드가 주어진 값과 일치하는지 확인한다. 파일명은 경로를 생략하며, 콜론으로 구분하여 해시코드를 같이 적어준다.
- ExistBinaryEqualToChecker : 해당 파일이 존재하는 경우에, 전달된 파일의 SHA-1 해시코드가 주어진 값과 일치하는지 확인한다. 해당 파일이 존재하지 않다면 true를 리턴한다.
- SizeEqualToChecker : 전달된 파일의 크기가 주어진 값과 일치하는지 확인한다. 파일의 크기는 바이트 단위이며 전달할때는 단위를 생략하고 숫자만 전달한다.
- ExistSizeEqualToChecker : 전달된 파일이 존재하는 경우에만, 전달된 파일의 크기가 주어진 값과 일치하는지 확인한다.
- FileExistChecker : 주어진 파일이 실제로 존재하는지 검사한다. 검사대상을 지정할 필요가 없으므로 just로 타겟을 지정할 수 있다.
- AtLeastOneChecker : 주어진 그룹에 객체가 존재하는지 검사한다. 비어있는 그룹일 경우 false를 리턴한다. 검사대상을 지정할 필요가 없으므로 just로 타겟을 지정할 수 있다.
- FileNameStartChecker : 주어진 파일이 전달된 파일명으로 시작하는지 검사한다.
- ClassNameStartChecker : 주어진 클래스명이 전달된 이름으로 시작하는지 검사한다.

6. XML 체커

```
[+/-] has attribute [속성명]
[+/-] has bean class [클래스 명]
[+/-] has query result [쿼리]
[+/-] has element [원소명]
```

- HasAttributeChecker : 해당 XML파일이 전달된 속성을 보유하고 있는지 검사한다.
- HasBeanClassChecker : 해당 XML파일에서 전달된 클래스를 bean으로 생성하는지 검사한다. 클래스명은 bean 생성시 지정되는 클래스명과 동일하다.
- HasQueryResultChecker : 임의의 쿼리를 이용하여 XML 검사를 수행한다. 쿼리 작성 규칙에 대해서는 본 가이드의 XML Query Language 사용 부분을 참고한다.
- HasElementChecker : 해당 XML파일이 전달된 원소를 가지고 있는지 검사한다.

룰셋의 검증

룰셋을 임의 정의한 경우 호환성 점검 도구 상의 RulesetParser 클래스를 이용하여 해당 룰셋을 파싱해보고 오류가 발생하지 않는지 테스트 해 볼 수 있다. 만약 임의의 체크리스트 클래스 혹은 컬렉터 클래스를 정의한 경우 실제 샘플 프로젝트의 검사를 수행하여 해당 클래스의 정상 동작여부를 확인하는 것이 좋다. RulesetParser는 룰셋 자체를 인스턴스화 시키면서 문법 검사를 수행할 뿐, 동적으로 체크리스트 클래스를 생성하거나 검사 메소드의 유무를 파악하지는 않는다. 때문에 RulesetParser의 역할은 정적인 룰셋 분석에 한정한다고 생각하는 것이 좋다.