실전 강의실 2

Play! 프레임워크와 함께 하는 즐거운 웹 개발

CoC(Convention over Configuration) 맛보기

필자가 2011년 초 웹 API를 개발하면서 처음 접하게 된 Play! 프레임워크는 배우기 쉽고 빠르게 결과물을 얻을 수 있게 해줬다. 이후 몇 번의 웹서비스 개발을 Play! 프레임워크로 진행하면서 그 편리함과 생산성에 흠뻑 빠지게 됐다. 이에 보다 많은 개발자들이 Play! 프레임워크로 즐겁게 웹서비스를 개발할 수 있도록 지식을 공유하고자 한다. 그 첫 번째 시간으로 간단한 회원관리 CRUD(Create, Read, Update, Delete) 개발을 통해 Play! 프레임워크의 대표적인 특징인 CoC를 알아보겠다.

김용웅 lifeandsong@gmail.com, 노윤희 lovebabyyh@naver.com, 박경우 sencell @naver.com, 김도훈 kosatba@empal.com, 이상준 u2love@naver.com, 유재헌 jheon2@nate.com, 복정식 sik8012@nate.com, 장승균 zeromad@nate.com, 정보라 nizyuichi@nate.com | 프로그래머가 된 것을 행운이라 여기며 Play! 프레임워크와 함께하는 즐겁고 행복한 개발 문화 정착에 노력하고 있다.

Play! 프레임워크는 자바 또는 스칼라(Scala) 언어로 개발하는 MVC 프레임워크다. 필자에게 Play! 프레임워크의 가장 큰 특징을 말하라고 한다면 망설임 없이 CoC라고 대답한다. CoC는 개발자들의 특별한 설정 없이도 프레임워크가 개발 관련 내용을 그동안 쌓여 있는 관례에 따라 미리 정해 놓는 것을 말한다. 따라서 개발자는 일정한 규칙 내에서 개발할 수 있는 환경을 확보할 수 있다. 이런 CoC 패러다임을 잘 반영한 대표적인 프레임워크가 루비 온 레일즈(RoR)인데, 이런 점 때문에 많은 사람들이 Play! 프레임워크를 RoR과 닮았다고 한다.

개발 환경 구축

Play! 프레임워크 개발을 위해서는 제일 먼저 버전을 선택해야 한다. 현재 Play! 프레임워크는 2.0.3 버전까지 나와 있지만, 자바 개발자 입장에서 봤을 때 2.x 버전은 스칼라에 더 친근한 환경이다. 그래서 필자는 1.x 버전 중 자바 7을 지원하는 1.2.4 버전을 사용할 것을 권장한다. 세부 설치과정은 다음과 같다.

1. JDK 7을 다운로드해 설치하고 JAVA_HOME, CLASSPATH, PATH 등 환경변수를 설정한다(www.oracle.com/technetwork/java/javase/downloads/java-se-jdk-7-download-432154.html).
2. 자바 EE 개발자용 이클립스 IDE(Eclipse Helios Sr2 Packages)도 다운로드한다(www.eclipse.org/downloads/packages/release/helios/sr2).
3. Play! 1.2.4를 다운로드한 후(www.playframework.org/download) 어디서나 실행할 수 있도록 PATH를 추가한다. 필자는 Play! 프레임워크를 D:\play\1.2.4에 복사했고, PATH에 해당 위치를 추가했다.
4. 이클립스에서 Play! 프레임워크를 사용자 라이브러리(User Library)에 등록한다. 그리고 난 다음 이클립스를 실행해 남은 설치과정을 수행한다.
   1) Windows → Preference → Java → Build Path → User Libraries → New
   2) User library name:에 Play! 1.2.4를 입력한 뒤 OK
   3) Play! 1.2.4를 클릭한 다음 Add JARs...를 선택
   4) Play! 설치 디렉터리 아래 /framework/play-1.2.4.jar와 /framework/lib/ 아래의 모든 .jar 파일을 선택한 후 열기를 선택

모든 과정을 마치고 나면 개발 준비가 완료된다. 참고로 이 방법으로 다른 Play! 프레임워크 버전들을 추가할 수 있다.

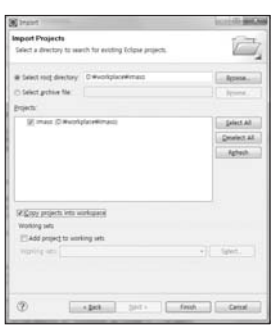
〈그림 1〉
이클립스에 Play! 프레임워크의
여러 버전이 등록된 모습

프로젝트 생성과 실행

지금부터 Play! 프레임워크를 활용해 회원 관리 웹 애플리케이션을 개발해 보자. 먼저 프로젝트를 생성한다. 명령 창에서 이클립스 개발 환경으로 이동한 다음 play new imaso라고 입력하는데, 여기서 imaso는 프로젝트 이름이다. 이렇게 생성된 프로젝트를 이클립스에서 읽기 위해 imaso 디렉터리로 이동한 다음 play eclipsify라고 입력 후 엔터 키를 누른다.

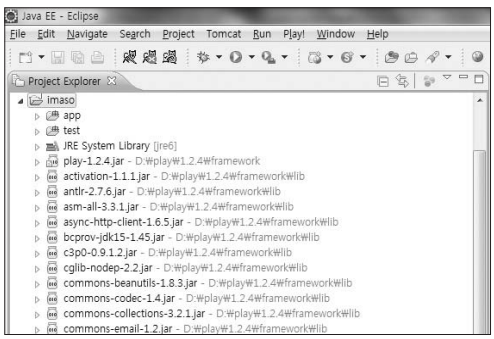
〈그림 2〉
생성한 Play! 프로젝트를
이클립스에서도 읽을 수 있게 실행

그 다음 이클립스를 실행하고 File → import → General → Existing Projects into Workplace → Next → Select root directory 과정을 거쳐 생성한 imaso 디렉터리를 지정하고 Finish를 클릭한다.



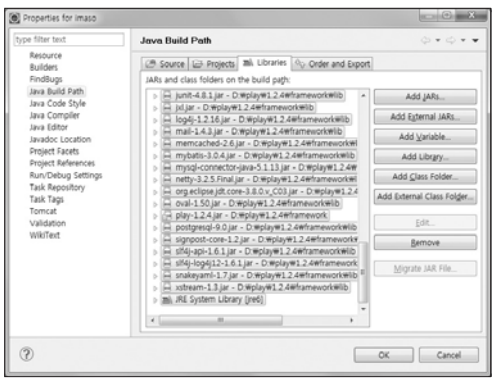
〈그림 3〉
이클립스로 생성한 프로젝트를
가져온 화면

그러면 생성한 imaso 프로젝트가 이클립스의 프로젝트 익스플로러(Project Explorer)에 있는 것을 확인할 수 있다. 하지만 .jar 파일들이 뒤죽박죽 나열돼 있어 추가적인 정리가 필요하다(〈그림 4〉 참조). 라이브러리 파일을 정리하는 과정은 다음과 같다.



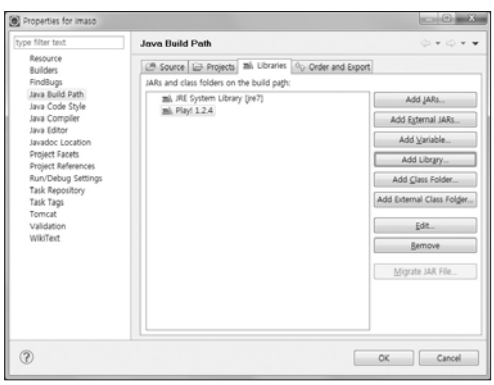
〈그림 4〉
정리되지 않은
프로젝트

Build Path → Configure Build Path... → Libraries 탭에서 모든 라이브러리를 선택한 다음 Remove 선택(〈그림 5〉 참조).



〈그림 5〉
라이브러리 삭제

Add Library... 클릭 → JRE System Library 선택 후 Next → Alternate JRE 선택 후 jre7 이어서 Finish 선택 → Add Library... 클릭 → Play! 1.2.4 체크 후 Finish(〈그림 6〉 참조).



〈그림 6〉
JRE 및 Play!
라이브러리 정리

모든 과정을 완료하면 프로젝트가 깔끔하게 정리된 것을 확인할 수 있다(〈그림 7〉 참조).

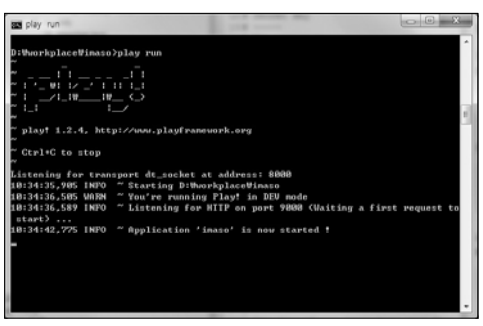
〈그림 7〉
정리된 Play! 프로젝트의 구조

생성된 imaso 프로젝트는 app, test, conf, public이란 디렉터리로 구성돼 있다. app 디렉터리는 MVC 프레임워크의 기본이 되는 controllers, models, views 패키지 또는 디렉터리로 구성된다. 개발자들은 각 패키지(또는 디렉터리)에 필요한 클래스나 HTML 파일을 추가하면 된다. test 디렉터리에는 app 디렉터리 안에 있는 클래스들을 테스트하는 코드가 있다. 이 말을 듣고 눈치 챈 독자들도 있겠지만 Play! 프레임워크는 테스트 주도 개발(TDD) 도구인 JUnit과 Selenium을 이용한 테스트를 지원한다.

conf 디렉터리에는 application.conf와 routes 등 주로 환경설정과 관련된 파일들이 포함돼 있다. public 디렉터리에는 자바스크립트, 스타일시트, 이미지 등 정적 자원(Static Resource)들이 저장돼 있다. 만일 정적 서버를 별도로 구성해 자원들을 관리하고 있다면 해당 public 디렉터리를 정적 서버로 배포하면 된다.

이렇듯 Play! 프레임워크에서는 웹 개발을 위해 필요한 기본적인 설계가 완료돼 있다. 그래서 개발자는 이와 관련된 내용을 특별히 고민하지 않아도 된다. 뿐만 아니라 모델, 컨트롤러, 뷰 그리고 정적 자원들이 네이밍 규칙을 통해 유기적으로 연결돼 있는데 이에 대한 설명은 회원 관리 CRUD를 개발하면서 확인해 본다.

이제 생성한 프로젝트를 실행해 보자. imaso 디렉터리에서 play run을 입력한 뒤 엔터 키를 누른다(〈그림 8〉 참조). 그리고 웹브라우저를 열고 주소 창에 http://localhost:9000을 입력하면 〈그림 9〉와 같은 화면을 볼 수 있다.



〈그림 8〉
프로젝트를
실행한 화면

〈그림 9〉
실행된 프로젝트를
웹브라우저에서
확인한 화면

프로젝트를 중지하려면 Ctrl+C를 클릭한 다음 Y를 입력하면 된다. 하지만 우리는 코드를 수정하는 동안 Play! 프로젝트를 중단시키지 않을 것이다. Play! 프로젝트는 개발 모드에서 코드를 수정하고 저장만 해주면 변경 내용이 곧바로 반영된다. 바로 이것이 Play! 프레임워크가 다른 자바 웹 프레임워크와 다른 점이다.

CoC에 기반을 둔 회원 관리 CRUD 개발

이제 회원 관리를 위한 CRUD 기능을 구현하는 웹 애플리케이션을 개발해 보자. 매우 기초적인 웹 애플리케이션이지만 전형적인 Play! 프레임워크의 개발 순서와 방법을 배울 수 있는 좋은 예제다.

먼저 사용할 DB를 결정한다. Play! 프레임워크는 MySQL, 오라클, MS-SQL 같은 DBMS와의 커넥션 정보를 Play! 프레임워크의 환경 파일인 application.conf에서 db.-로 시작하는 변수를 통해 선언할 수 있다. 또한 모델 클래스에 기반을 두고 있어서 실제 DBMS를 연결하지 않아도 메모리나 파일에 테이블 구조를 생성 및 저장할 수 있다.

단 메모리 DB를 사용할 경우 실행을 멈추면 모든 데이터가 사라지므로 예제에서는 파일에 데이터를 저장할 것이다. application.conf에서 #db=mem 부분에 있는 주석(#)을 제거하고 db=fs와 같이 수정한 다음 저장한다. 이때 실제 데이터는 Play! 프로젝트 아래에 있는 db/h2 디렉터리에 저장된다.

그 다음 모델 클래스를 생성한다. 아이디, 비밀번호, 이름, 이메일 정보를 저장할 멤버 클래스를 models 패키지 내에 생성한다.

〈리스트 1〉 회원 정보를 담을 모델 클래스

```
@Entity
public class Member extends GenericModel {
@Id
public String id;

@NotNull
public String password;
```

```
public String name;

public String email;
}
```

Play! 프레임워크는 ORM(Object-Relational Mapping)에 기반을 두고 DB와 자바 모델 클래스를 매핑한다. 그래서 모델 클래스를 생성하고 저장하면 JPA(Java Persistence API)의 DDL(Data Definition Language) 설정에 따라 DB에 모델 클래스와 똑같은 이름의 테이블이 생성된다.

이때 기존 데이터가 있는 테이블과 매핑하고 모델 클래스 구조를 변경해야 한다면 데이터가 모두 삭제될 수도 있으니 주의한다. 만일 기존 데이터가 들어있는 테이블을 매핑해야 한다면 application.conf를 jpa.ddl=none으로 설정해 실제 매핑된 테이블과 다를 경우 발생할 수 있는 테이블 구조 변경이나 데이터 삭제를 방지한다. 또 Play! 프레임워크는 기본 개발 환경에서는 jpa.ddl=update로 돼 있기 때문에 DB 테이블 구조가 변경될 수 있으니 조심한다.

JPA나 ORM에 익숙하지 않은 독자를 위해 추가로 설명하면, 모델 클래스를 상속하게 되면 Long 형식의 id 프로퍼티가 포함된다(〈리스트 2〉 참조). 이 프로퍼티는 레코드가 추가될 때마다 자동으로 증가하므로 id 값이 필요 없을 때는 GenericModel 클래스를 상속한다.

이 경우 반드시 유니크한 프로퍼티 1개가 있어야 하며, 해당 프로퍼티에는 @Id라는 어노테이션을 붙여야 한다. JPA 모델 클래스에는 단순히 데이터를 보관하는 VO(Variable Object) 기능뿐만 아니라 데이터 추가, 조회, 수정, 삭제 등의 메소드들도 포함돼 있다. 이 기능에 대해서는 컨트롤러를 구현할 때 자세히 살펴본다.

〈리스트 2〉 모델 클래스 구조

```
@MappedSuperclass
public class Model extends GenericModel {

@Id
@GeneratedValue
public Long id;

public Long getId() {
            return id;
        }
        @Override
        public Object _key() {
            return getId();
        }
}
```

모델 클래스를 생성했다면 회원 관리 CRUD 기능에 맞는 RESTful URI를 routes 파일에 생성한다. 이때 HTTP 메소드(POST, GET, PUT, DELETE)와 URI 그리고 컨트롤러 클래스와 메소드를 〈리스트 3〉처럼 차례로 지정하는데, 편의상 클라이언트에 전송될 데이터 형식(HTML, JSON, XML 등)을 URI에 확장자로 붙인다. 또한 routes 파일에서는 /members/{id}.json의 {id}처럼 URI 일부를 인자로 받은 다음 컨트롤러에서 사용할 수 있다.

〈리스트 3〉 RESTful URI와 컨트롤러를 매핑하는 routes 파일

```
# 회원 관리 CRUD
POST      /members.json
MemberController.create
GET       /members/{id}.json      MemberController.read
GET       /members                MemberController.list
PUT       /members/{id}.json      MemberController.update
DELETE    /members/{id}.json      MemberController.delete
```

routes까지 생성했다면 이제 컨트롤러를 만들어볼 차례다. 패키지 내에 MemberController.java를 생성하고 routes 파일에서 지정한 대로 create( ), read( ), list( ), update( ), delete( ) 메소드들을 〈리스트 4〉와 같이 만든다.

MemberController.create( ) 메소드를 살펴보면, Member member를 인자로 받는다는 것을 알 수 있다. 이렇게 객체를 인자로 받으려면 클라이언트에서 member.id, member.password, member.name, member.email 등의 형식으로 인자를 전송해야 한다.

〈리스트 4〉 회원 정보를 CRUD하는 컨트롤러 코드

```
public class MemberController extends Controller {

    public static void create(Member member) {
        if (member == null ||
StringUtils.isBlank(member.id)) {
                renderJSON(new JsonResult(404));
                return;
        }

        member.save();
        renderJSON(new JsonResult(true));
    }

    public static void read(String id) {
        if (StringUtils.isBlank(id)) {
                renderJSON(new JsonResult(404));
                return;
        }
```

```
        Member member = Member.findById(id);
        if (member == null) {
                renderJSON(new JsonResult(404));
                return;
        }

        Map<String, Object> result = new HashMap<String,
Object>();
        result.put("result", true);
        result.put("member", member);
        renderJSON(result);
    }

    public static void list() {
                List<Member> members = Member.findAll();
                if (members == null)
                        members = new
ArrayList<Member>();

                render(members);
    }

    public static void update(Member member) {
        if (member == null ||
StringUtils.isBlank(member.id)) {
                renderJSON(new JsonResult(404));
                return;
        }

        member.save();
        renderJSON(new JsonResult(true));
    }

    public static void delete(String id) {
        if (StringUtils.isBlank(id)) {
                renderJSON(new JsonResult(404));
                return;
        }

        Member member = Member.findById(id);
        if (member == null) {
                renderJSON(new JsonResult(404));
                return;
        }

        member.delete();
        renderJSON(new JsonResult(true));
    }

}
```

앞서 모델 클래스를 소개할 때 설명한 대로 멤버 클래스에는 데이터 생성 및 수정(member.save( )), 삭제(member.delete( )) 그리고 조회(Member.findById(id), Member.findAll( ))를 위한 다양한 메소드가 있다. 자세한 내용은 Play! 프레임워크 홈페이지나 API를 참고하기 바란다.

처리 결과를 클라이언트로 전송할 때는 render( ) 메소드를 사용한다. MemberController.list( )의 경우 /views/MemberControllers/list.html에 필요한 값을 담아 클라이언트에 전송한다. 그리고 처리 결과를 뷰에 전달할 때는 render( ) 메소드의 객체를 인자로 넘기면 뷰에서 이를 받아 사용한다. 출력되는 파일과 형식에 따라 renderTemplate(특정 HTML 파일 지정), render JSON( ), renderXML( ), renderText( ) 등도 사용할 수 있다. 한편 Play! 프레임워크에서는 StringUtils.isBlank(…)처럼 아파치 커먼 라이브러리(Apache Common Library)가 내장돼 있어 해당 라이브러리에 포함된 다양한 기능을 사용할 수 있다. 아파치 커먼 라이브러리의 다양하고 편리한 기능에 관심이 있다면 마소 6월호 '효과적인 자바 개발을 위한 최고의 선택 Apache Common Library(p 250)'를 참조하기 바란다.

마지막으로 컨트롤러에서 처리한 결과를 출력하는 뷰를 개발해 보자. Play! 프레임워크에서 사용하는 뷰 템플릿은 1.x 버전의 경우 그루비(Groovy) 표현법을 사용했다. 그러나 2.x 버전부터는 스칼라 표현법을 사용하는 것으로 변경됐다. 우리는 1.2.4 버전으로 개발하고 있으므로 그루비 표현법을 사용한다. FreeMarker같은 템플릿 엔진을 사용해 본 경험이 있다면 Play! 프레임워크 홈페이지에 나온 설명만으로도 쉽게 사용할 수 있다.

먼저 HTML 파일을 뷰 디렉터리에 작성한다.

〈리스트 5〉 기본 HTML 형식을 담은 main.html

```
<!DOCTYPE html>

<html>
    <head>
        <title>#{get 'title' /}</title>
        <meta charset="${_response_encoding}">
        <link rel="stylesheet" media="screen"
href="@{'/public/stylesheets/main.css'}">
        #{get 'moreStyles' /}
        <link rel="shortcut icon" type="image/png"
href="@{'/public/images/favicon.png'}">
        <script src="@{'/public/javascripts/jquery-
1.6.4.min.js'}" type="text/javascript"
charset="${_response_encoding}"></script>
        #{get 'moreScripts' /}
    </head>
    <body>
        #{doLayout /}
    </body>
</html>
```

이때 컨트롤러 클래스 이름으로 된 디렉터리에 있는 메소드와 동일한 이름으로 HTML 파일을 만들면 해당 메소드 내에서 render( ) 메소드를 이용해 해당 HTML 파일을 출력할 수 있다.

예를 들면 Member Controller.list( ) 메소드에서 사용할 수 있는 HTML은 /app/views/MemberController/list.html이다.

뷰 개발 과정은 먼저 main.html에 전체 HTML에서 사용하는 공통 요소를 넣은 다음 각 HTML에서 해당 파일을 확장시킨 후 필요한 부분만을 추가한다. 이때 페이지 제목과 스타일시트 그리고 자바스크립트는 #{set title:' ~' /}, #{set 'moreStyles'} ~ #{/set}, #{set 'moreScripts'} ~ #{/set}을 사용해 추가할 수 있다. 컨트롤러에서 뷰로 넘긴 인자들은 ${member.id}와 같이 ${객체명}으로 참조할 수 있다. 그리고 제어문과 반복문 등은 #{list items:members, as:'member'} ~ #{/list}처럼 #{명령문} 형태로 이뤄져 있다.

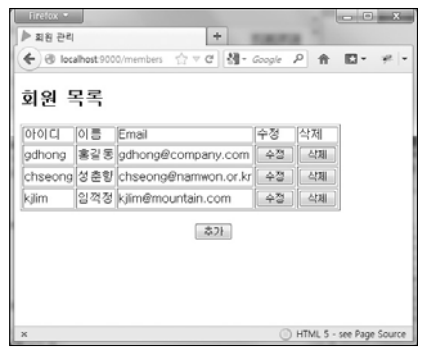
뷰에서 사용하는 명령문을 좀더 빠르게 익히고 싶다면 Play 프레임워크 Cheetsheet(www.playframework.org/documentation/1.2.4/cheatsheet/templates)을 참고하기 바란다. 이미지 파일, 스타일시트, 자바스크립트 등 정적 파일은 퍼블릭 디렉터리 아래에 있는 해당 디렉터리에 생성하면 된다.

〈리스트 6〉 회원 목록을 출력하는 뷰(list.html)

```
#{extends 'main.html' /}
#{set title:'회원 관리' /}
#{set 'moreStyles'}
<link rel="stylesheet" type="text/css"
href="@{'/public/stylesheets/member.css'}" />
#{/set}
#{set 'moreScripts'}
<script src="@{'/public/javascripts/trimpath-template-
1.1.22-jei.js'}" type="text/javascript"
charset="${_response_encoding}"></script>
<script src="@{'/public/javascripts/member.js'}" type=
"text/javascript" charset="${_response_encoding}"></script>
#{/set}

<div id="content">
        <p align="center"><h2>회원 목록</h2></p>
        <table border="1">
                <tr>
                        <td>아이디</td>
                        <td>이름</td>
                        <td>Email</td>
                        <td>수정</td>
                        <td>삭제</td>
                </tr>
                #{list items:members, as:'member' }
                <tr>
                        <td>${member.id}</td>
                        <td>${member.name}</td>
                        <td>${member.email}</td>
                        <td><input type="button"
class="update" memberId="${member.id}" value="수정" /></td>
                        <td><input type="button"
class="delete" memberId="${member.id}" value="삭제" /></td>
                </tr>
                #{/list}
        </table>
        <p align="center">
                <input type="button" id="listCreate"
value="추가" />
        </p>
</div>
… 중간 코드 생략 …
```

완성된 코드를 실행해 보자. 웹브라우저에서 http://localhost:9000/members로 접속하면 된다(〈그림 10〉 참조). 전체 코드는 이달의 디스켓을 확인하자.



〈그림 10〉
완성된 회원 관리 CRUD

정리하며

이상으로 Play! 프레임워크로 개발 과정을 차례로 살펴봤다. 개발자들은 프레임워크가 미리 정해놓은 규칙에 따라 꼭 필요한 핵심 기능을 개발하는 것에만 집중하면 된다는 게 Play! 프레임워크의 가장 큰 장점이며 이것이 CoC 패러다임의 핵심이다.

다음 시간에는 Play! 프레임워크를 이용해 실제 상용 서비스를 개발하면서 팀이 직면했던 문제들과 그 해결책들을 중심으로 보다 완성도 있는 웹 애플리케이션 개발을 위해 필요한 요소들을 살펴보기로 한다.

참고자료

1. Play! 프레임워크 홈페이지 : playframework.org/
2. Play! 프레임워크 튜토리얼 : playframework.org/documentation/1.2.4/home
3. Play! 프레임워크 Cheetsheet : playframework.org/documentation/1.2.4/cheatsheet/templates
4. Play! Framework API : playframework.org/documentation/api/1.2.4/index.html
5. CoC : en.wikipedia.org/wiki/Convention_over_configuration
6. Play! 프레임워크 페이스북 페이지 : www.facebook.com/pages/Play-Framework/50884087887
7. Play! 프레임워크 트위터 계정 : @playframework
8. Play! 프레임워크 한국 개발자 그룹 페이스북 페이지: www.facebook.com/groups/294888493919737

이달의 디스켓 : playframework_1.zip

다운로드 방법 안내 – 196p 참고