실전 강의실 4

Play! 프레임워크와 함께 하는 즐거운 웹 개발

테스트와 멀티 환경에 배포하기

지난 시간에는 Play! 프레임워크에서 제공하는 다양한 기능들을 살펴보며 왜 Play! 프레임워크를 올인원(All in One) 웹 프레임워크라 부르는지 확인했다. Play! 프레임워크 연재의 마지막 시간에는 개발된 Play! 프로젝트를 테스트하고 여러 환경에 배포하기 위한 방법을 설명한다.

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 | Play! 프레임워크와 함께하는 즐겁고 행복한 개발 문화 정착에 노력하고 있다.

소프트웨어 개발 조직에는 개발 프로세스마다 요구사항 분석, 설계 보고서, 코딩 표준 등 각기 정책이 있다. 프레임워크를 사용하는 이유를 생산성의 측면에서도 찾을 수 있지만, 개발 정책을 프레임워크에 반영해 서로 다른 개발자들이 개발에 임해도 일관성 있는 개발을 수행하도록 하는 것이 더 중요한 목적이다. Play! 프레임워크는 모든 개발 과정에 효율적인 개발 정책이 반영돼 있다. 국제화, 테스트, 배포 지원 기능을 살펴보며 각각의 개발 정책들을 확인해보자.

국제화(Internationalization, i18n)

Play! 프레임워크에서 제공하는 국제화 기능을 사용해 다국어 서비스를 구현하려면 먼저 application.conf에서 사용하려는 언어 코드를 설정해야 한다. 영어와 한국어 사용을 위해 application.langs=en,ko 코드를 입력하자.

다음으로 〈표 1〉을 참조해 /conf 디렉터리 내에 언어별 메시지 파일을 생성하고 '메시지 키=메시지' 형태로 메시지를 추가하자. 동일한 메시지 키에 언어별 메시지를 설정하면 언어 설정(locale)에 따라 선택된 언어의 메시지들을 사용할 수 있다.

이제 현재의 언어 설정을 변경해보자. 〈리스트 1〉은 컨트롤러에서 현재의 언어 설정을 조회하고 사용자의 요청에 따라 언어 설정을 변경하는 코드다. 사용자가 특별히 언어 설정을 하지 않았다면 Play! 프레임워크는 사용자의 HTTP 요청 헤더 중 Accept-Language의 첫 번째 값을 기본 언어로 설정한다. 언어 설정이 변경되면 PLAY_LANG 쿠키에 변경된 언어 설정을 저장해 유지한다.

언어	영어	한국어
파일명	message.en	messages.ko

내용	# Authenticate
	signIn=Sign In
	signOut=Sign Out
	id=Id
	password=Password
	cancel=Cancel
	inputPassword=Please, input your password.
		# 인증
		signIn=로그인
		signOut=로그아웃
		id=아이디
		password=비밀번호
		cancel=취소
		inputPassword=비밀번호를 입력하지 않았습니다.

〈표 1〉 언어별 메시지 파일 작성

〈리스트 1〉 컨트롤러에서 언어 설정을 조회, 수정

```
public class I18nController extends Controller {

… 중간 코드 생략 …

    @Before
    public static void changeLanguage(String locale) {
        if (StringUtils.isBlank(locale)) {
```

```
            if (StringUtils.isBlank(Lang.get()))
            Lang.change(getLanguage());
                return;
            }
if (locale.equals(Lang.get()))
                        return;

            Lang.change(locale);
      }
}
```

〈리스트 2〉 HTTP 요청 헤더에 포함된 사용자의 언어 설정 정보

```
[Request Headers]
Accept-Language:ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4
```

언어 설정에 따라 메시지를 출력하려면 코드에 메시지를 사용하지 않고 메시지 키를 사용해야 한다. 컨트롤러에서 메시지를 조회하는 예인 〈리스트 3〉과 뷰에서 메시지를 조회하는 예인 〈리스트 4〉를 보자.

〈리스트 3〉 컨트롤러에서 메시지 키로 메시지 조회

```
@With({ I18nController.class, SignedController.class })
public class AuthController extends Controller {

… 중간 코드 생략 …

      public static void changePassword(String password) {
            if (StringUtils.isBlank(password)) {
                  renderTemplate("/AuthController/expired
Password.html", Messages.get("inputPassword"));
                  return;
            }

… 중간 코드 생략 …

      }
}
```

〈리스트 4〉 뷰에서 메시지 키로 메시지 조회

```
#{extends 'main.html' /}
#{set title: messages.get('member') + ' ' +
messages.get('management') /}

<p align="center">
      <h2>&{'admin'} &{'signIn'}</h2>
      #{if message != null}<p>${message}<p>#{/if}
      <form method="post" action="/signIn">
            <table border="1">
                  <tr>
                        <td>&{'id'}</td>
                        <td><input type="text" name="member.
id" maxlength="16" /></td>
                  </tr>
                  <tr>
                        <td>&{'password'}</td>
                        <td><input type="password" name=
"member.password" maxlength="16" /></td>
                  </tr>
            </table>
            <p>
                  <input type="submit" id="createSubmit"
value="&{'signIn'}" />
                  <input type="reset" id="createReset"
value="&{'cancel'}" />
            </p>
      </form>
</p>

<p>
#{if lang == "ko"}한글#{/if}
#{else}<a href="${request.path}?locale=ko">한글</a>#{/else}
| #{if lang == "en"}English#{/if}
#{else}<a href="${request.path}?locale=en">English</a>#
{/else}
</p>
```

이제 프로젝트를 실행해보면 맨 아래에 언어 설정을 할 수 있는 '한글 | English'가 추가됐음을 확인할 수 있다.

〈그림 1〉
다국어 기능이 적용된
로그인 폼

MVC 테스트 지원

최근 테스트 주도 개발(Test-Driven Development, 이하 TDD)을 개발 정책에 포함한 조직들이 늘어나고 있다. 테스트 케이스를 작성하고 자동화된 테스트를 수행해 점진적으로 코드를 개선해나가는 TDD는 코드의 품질을 향상시키는 좋은 방법이다. Play! 프레임워크도 모델, 컨트롤러, 뷰에 대해 TDD를 적용할 수 있도록 지원한다.

테스트 종류	테스트 대상	사용 도구
단위 테스트(Unit Test)	모델, 유틸리티	JUnit
기능 테스트(Functional Test)	컨트롤러	JUnit
인수 테스트(Acceptance Test)	뷰	Selenium

〈표 2〉 Play! 프레임워크에서 지원하는 테스트

Play! 프로젝트를 테스트 모드로 실행하면 웹상에서 테스트를 실행하고 그 즉시 결과 확인이 가능하다. 테스트 모드로 실행하려면 application.conf의 끝부분에 %test로 시작하는 환경 설정을 개발 환경과 동일하게 수정하면 된다(〈리스트 5〉).

〈리스트 5〉 테스트 모드 실행을 위한 환경 설정

```
%test.application.mode=dev
%test.db.url=jdbc:mysql://localhost:3306/test?useUnicode=t
rue&characterEncoding=utf8
%test.jpa.ddl=none
%test.mail.smtp=mock
```

그 후 Member 모델의 CRUD 기능을 테스트하는 코드를 /test 디렉터리 아래의 models.MemberTest에 작성하자(〈리스트 6〉 참고). UnitTest 클래스를 상속받았고 단위 테스트마다 @Test 어노테이션이 붙어 있음에 유의하자. 사용할 수 있는 어노테이션들을 〈표 3〉에 정리했다. 이 코드는 org.junit.Assert 클래스의 assertEqual(), assertTrue(), assertFalse(), assertNull(), assertNotNull() 메소드들을 사용해 예상 값과 실행 결과를 비교할 수 있다.

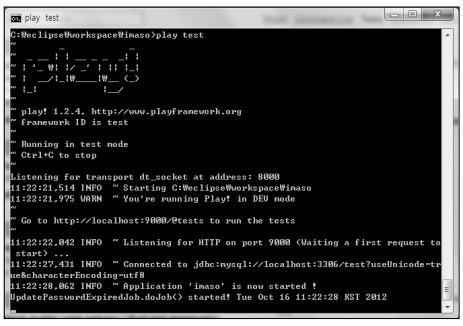
〈리스트 6〉 Member 모델에 대한 단위 테스트

```
public class MemberTest extends UnitTest {
private static long count;
private static Member member;

        @BeforeClass
        public static void oneTimeSetUp() {
                count = Member.count();

        member = new Member();
        member.id = "test";
        member.password = Crypto.passwordHash("1234",
HashType.SHA256);
        member.passwordExpired = "N";
        member.name = "테스터";
        member.email = "test@company.com";
        member.lastSigned = new Timestamp(new
Date().getTime());
        }

        @Before
        public void testSetUp() {
                member.save();
        }

        @Test
        public void testCRUD() {
                testCreate();

… 중간 코드 생략 …

        }
        public void testCreate() {
        assertEquals(count + 1, Member.count());
assertNotNull(Member.findById(member.id));
        }

… 중간 코드 생략 …

        @After
        public void tearDown() {
                Member newMember =
Member.findById(member.id);
                if (newMember != null)
                        newMember.delete();
         }
}
```

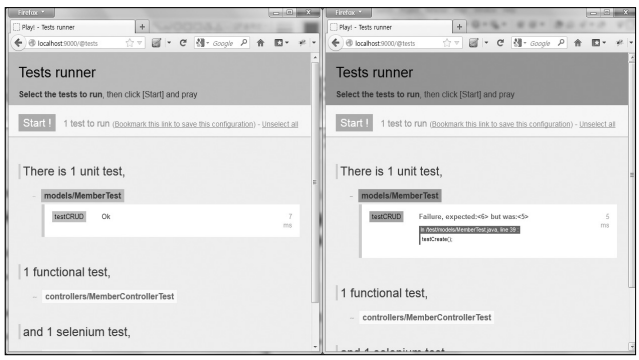
어노테이션	의미
@BeforeClass	클래스가 초기화될 때 1회만 실행
@Before	테스트 메소드 실행 전 매 회 실행
@Test	테스트 메소드
@Ignore	테스트 메소드에서 발생하는 오류를 무시
@After	테스트 메소드들이 실행된 후 매 회 실행
@AfterClass	클래스가 종료될 때 1회 실행

〈표 3〉 단위 테스트에 사용할 수 있는 어노테이션

이제 지금까지 작성한 테스트 코드를 실행해보자. Play! 프로젝트를 테스트 모드로 실행하려면 명령 창에서 프로젝트 디렉터리로 이동한 뒤 play test를 입력한다. 프로젝트가 테스트 모드로 실행되면 〈그림 2〉에서처럼 웹에서 테스트를 실행할 수 있는 주소(http://localhost:9000/@tests)도 보인다. 브라우저를 열고 이 주소로 접속해보자. 파란색 화면의 Tests runner 화면이 나오고 단위 테스트(unit test), 기능 테스트(functional test), 인수 테스트(selenium test)별로 수와 목록이 보인다. 단위 테스트 중 방금 작성한 models/MemberTest를 클릭한 다음 맨 위의 'Start !'를 클릭하자. 단위 테스트가 성공했다면 초록색 화면으로 변하며 models/MemberTest 왼쪽의 +를 클릭해 단위 테스트별 실행 결과를 확인할 수 있다. 만약 단위 테스트에 실패했다면 붉은색 화면으로 바뀌며 실패한 단위 테스트를 보여준다(〈그



〈그림 2〉 테스트 모드로 실행



〈그림 3〉 웹에서 테스트 실행

림 3〉 참조).

이번에는 MemberController에 대한 기능 테스트(Functional Test)를 /test 디렉터리 아래 controllers.MemberControllerTest로 작성해보자(〈리스트 7〉 참조). 기능 테스트는 FunctionalTest 클래스를 상속받는다.

Play! 프레임워크에서 제공하는 FunctionalTest.PUT() 메소드의 경우 if (savedCookies != null) request.cookies = savedCookies;로 코딩돼 쿠키에 저장된 세션이 유지되지 않는 오류가 있다. 이 문제의 해결을 위해 이전 응답의 쿠키를 저장하고 있다가 PUT 메소드로 전송할 때 Request 객체를 생성한 뒤 저장된 쿠키를 추가해주자.

〈리스트 7〉 MemberController에 대한 기능 테스트

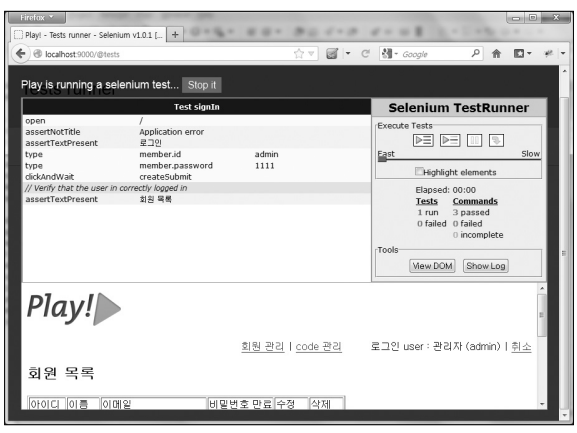
```
public class MemberControllerTest extends FunctionalTest {
      private static Member member;
      private static Response response;
      private static Map<String, Cookie> savedCookies;

      @BeforeClass
      public static void oneTimeSetUp() {
            member = new Member();
            member.id = "test";
            member.password = "1234";
            member.name = "테스트";
            member.email = "test@company.com";
      }

      public void saveCookies(Response response) {
            if (response == null || response.cookies ==
null || response.cookies.get(SIGNED_COOKIE_ID) == null)
                  return;

            savedCookies = response.cookies;
      }

      @Before
      public void setUp() {
            String params = String.format("member.id=
%s&member.password=%s", "admin", "1111");
            response = POST("/signIn",
APPLICATION_X_WWW_FORM_URLENCODED, params);
            saveCookies(response);
            assertStatus(302, response);

            params = String.format("member.id=%s&member.
password=%s&member.name=%s&member.email=%s",
            member.id, member.password, member.name,
member.email);
            response = POST("/members.json",
APPLICATION_X_WWW_FORM_URLENCODED, params);
            saveCookies(response);
      }

      @Test
      public void testCRUD() {
            testCreate();

… 중간 코드 생략 …

            testUpdate();
      }

      public void testCreate() {
            assertStatus(200, response);

            Gson gson = new Gson();
            JsonResult jsonResult = gson.fromJson
(getContent(response), JsonResult.class);
            assertEquals(jsonResult.isResult(), true);
      }

… 중간 코드 생략 …

      public void testUpdate() {
            Member savedMember = Member.findById
(member.id);
            savedMember.password = member.password;
            savedMember.passwordExpired = "Y";
            String params = String.format("member.id=
%s&member.password=%s&member.name=%s&member.email=%s&membe
r.passwordExpired=%s", savedMember.id, savedMember.
password, savedMember.name, savedMember.email,
savedMember.passwordExpired);
            Request request = newRequest();
            request.cookies = savedCookies;
            response = PUT(request, String.format
("/members/%s.json", member.id),
APPLICATION_X_WWW_FORM_URLENCODED, params);
            assertIsOk(response);
            assertContentType("application/json", response);
            assertCharset("utf-8", response);

            Gson gson = new Gson();
            JsonResult jsonResult =
gson.fromJson(getContent(response), JsonResult.class);
            assertEquals(jsonResult.isResult(), true);
      }
```

```
… 중간 코드 생략 …

    @After
    public void oneTimeTearDown() {
        Member savedMember =
Member.findById(member.id);
        if (savedMember != null)
            savedMember.delete();

        Response response = GET("/signOut");
        assertStatus(302, response);
    }
}
```

다음으로 로그인 기능에 대한 인수 테스트를 /test 디렉터리 아래의 Application.signInTest.html에 작성하자. 〈리스트 8〉을 참고해 메인 페이지에 접속 후 아이디와 비밀번호를 입력한 뒤 '회원 목록' 페이지가 나오는 과정을 Selenium 스크립트로 작성하자. 이를 Tests runner 화면에서 실행하면 화면이 Selenium TestRunner로 전환되고 작성한 내용을 실제 화면에서 실행한다. 테스트가 끝나면 다시 Tests runner 화면으로 전환되고 결과를 출력한다.

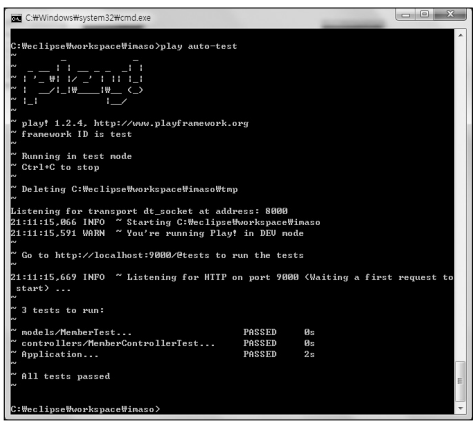
〈리스트 8〉 로그인에 대한 인수 테스트

```
#{selenium 'Test signIn'}

    open('/')
    assertNotTitle('Application error')
    assertTextPresent('로그인')
    type('member.id', 'admin')
    type('member.password', '1111')
    clickAndWait('createSubmit')

    // Verify that the user in correctly logged in
    assertTextPresent('회원 목록')

#{/selenium}
```



〈그림 4〉 Selenium TestRunner로 로그인 인수 테스트를 하는 과정

지금까지 작성한 단위 테스트, 기능 테스트, 인수 테스트들을 한 번에 실행할 수도 있다. 명령 창에서 Play! 프로젝트의 디렉터리로 이동한 다음 play auto-test를 실행하면 프로젝트 내의 모든 테스트들이 차례로 실행되고(〈그림 5〉) /test-result 디렉터리 밑에 테스트 결과가 log와 .html 형태로 저장된다. 이런 테스트 결과로 개발 결과물에 대한 품질 신뢰도를 높일 수 있다.



〈그림 5〉 모든 테스트를 한 번에 실행

위의 세 가지 테스트 외에도 웹 개발 시 도입할 수 있는 테스트 정책들을 〈표 4〉에 정리했다.

테스트명	도구	내용
정적 테스트 (Static Test)	FindBugs	코드에서 잠재적 오류를 조사 이클립스 플러그인으로 설치해 실행 (findbugs.sourceforge.net)
웹 취약점 점검 (Security Test)	WebSecurify	웹 취약점 점검 크롬이나 파이어폭스 플러그인을 설치해 실행 (www.websecurify.com/suite)
속도 테스트	Google Page Speed Insight	웹사이트의 응답 속도가 느린 원인을 분석 (developers.google.com/speed/pagespeed/insights)
부하 테스트 (Stress Test)	Apache JMeter	대규모 트래픽이 발생할 때 인프라와 애플리케이션이 어떤 영향을 받는지 확인 (jmeter.apache.org)

〈표 4〉 그 외에 추가로 도입할 만한 테스트 정책

개발/스테이징/운영 서버에 배포하기

웹 프로젝트는 보통 개발(dev) 환경에서 개발된 후 배포 전 실제 운영 환경과 동일한 스테이징(staging) 환경에서 최종 테스트를 거쳐 운영(live) 환경에 배포된다. 각 환경마다 DB 접속 정보, 정적 자원의 URL 등 코드에 실제로 적용되는 부분들이 달라지게 마련이다. 매번 이 부분들을 찾아 수정하고 배포하는 번거로움과 이 과정에서 생길지도 모르는 실수를 방지하기 위해 play! 프레임워크에서는 applicaiton.conf에서 각 환경별 설정과 실행 시 원하는 환경을 선택할 수 있도록 지원한다. 〈표 5〉의 서로 다

른 구성을 가진 서버 환경이라도 〈리스트 9〉에서 볼 수 있듯 application.conf에 모두 등록할 수 있다.

구분	대표 URL	DB 서버	
		주소	계정(id/pw)
개발	dev.company.com	dev.db.com	did / dpw
스테이징	staging.company.com	staging.db.com	sid / spw
운영	live.company.com	live.db.com	lid / ipw

〈표 5〉 서로 다른 구성을 가진 서버 환경의 예

〈리스트 9〉 개발, 스테이징, 운영 환경을 application.conf에 모두 설정

```
# 모드 설정 : dev(개발 모드), prod(프로덕션 모드)
application.mode=dev
%staging.application.mode=prod
%live.application.mode=prod

# DB 설정 : 개발 모드일 경우에도 반드시 %dev.를 써준다
%dev.db=mysql://did:dpw@dev.db.com/company
%staging.db=mysql://sid:spw@staging.db.com/company
%live.db=mysql://lid:lpw@live.db.com/company

# 기본 URL
application.baseUrl=dev.company.com
%show.application.baseUrl=staging.company.com
%live.application.baseUrl=live.company.com
```

Play! 프레임워크에는 dev 모드와 prod 모드의 2개의 실행 모드(application.mode)가 있다. prod 모드에서는 실행 전 컴파일(Precomplie)되므로 프로젝트 실행까지 약간의 시간이 소요되며 이후 코드를 수정해도 바로 반영되지 않는다. 컴파일 시간은 코드의 분량과 서버 성능에 따라 다르지만(보통 1분 내외) 다중 서버에 차례로 배포할 때는 고려해 둬야 한다. 또 prod 모드는 dev 모드와 성능과 기능에서 차이가 크니 실제 운영 서비스는 반드시 prod 모드로 설정하길 권한다.

최근에는 웹서비스의 속도와 성능 향상을 위해 웹서버와 정적 서버를 분리하고 정적 자원을 CDN(Content Delivery Network)으로 서비스하기도 한다. 이 경우 실행 환경별로 정적 자원의 주소가 달라진다. routes 파일을 수정하고 뷰 파일의 정적 자원 URL의 @ 대신 절대 경로를 표현하는 @@로 변경해 각각의 정적 자원 주소를 매핑할 수도 있다.

〈리스트 10〉 routes 파일에서 실행 환경별로 다른 정적 자원의 주소를 매핑

```
#{if play.Play.id == "live"}
    GET     static.company.com/home
staticDir:public
#{/}
#{elseif play.Play.id == "staging"}
    GET     staging-static.company.com/home
staticDir:public
#{/}
#{else}
    GET     /public/
staticDir:public
#{/}
```

〈리스트 11〉 뷰 파일에서 정적 자원의 주소를 절대 경로로 변경

```
<!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>
        <p><img src="@@{'/public/images/logo.png'}" /></p>

… 중간 코드 생략 …

    </body>
</html>
```

Play! 프로젝트를 서버로 배포할 때는 /app, /conf, /test, /public 4개의 디렉터리만 배포한다. 별도의 정적 서버를 운영하고 있다면 /public 디렉터리만 해당 서버로 따로 배포하면 된다. 그 외에 Java EE 애플리케이션 서버나 클라우드 호스팅 서비스에 배포하는 방법은 Play! 프레임워크 홈페이지를 참고하자.

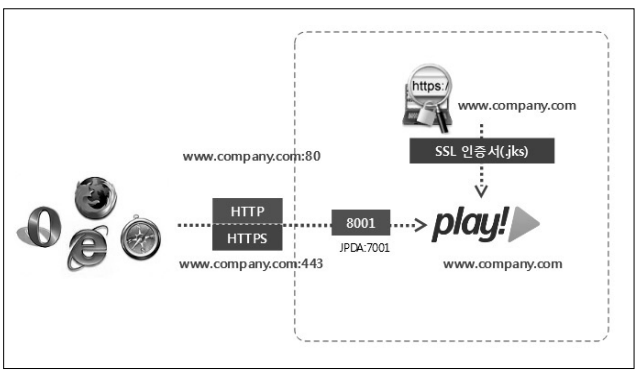
이제 〈표 6〉을 참고해 환경별 형식에 맞춰 실행해보자. 포그라운드(play run)로 실행하면 로그를 화면에서 볼 수 있는 반면 Ctrl+C를 누르면 프로세스가 중단되므로 실제 운영 환경에서는 백그라운드(play start)로 실행하길 바란다.

구분	실행 방법	
	포그라운드 (foreground)	백그라운드 (background)
개발	play run —%dev	play start —%dev
스테이징	play run —%staging	play start —%staging
운영	play run —%live	play start —%live

〈표 6〉 환경별 실행 방법

여러 Play! 프로젝트를 함께 운영하기 위한 아키텍처

물리적인 웹서버 1대에 Play! 프로젝트를 1개만 운영할 경우 별다른 수정 없이 Play! 프로젝트를 그대로 배포하면 된다(〈그림 6〉). 이때 SSL 인증서를 사용해야 한다면 인증서 발급기관으로부터 자바 키스토어(Java KeyStore) 파일 형식(.jks)을 받아 /conf 디렉터리에 저장하고 application.conf에 SSL 인증서를 설정하자.



〈그림 6〉 Play! 프로젝트를 1개만 운영하기 위한 아키텍처

〈리스트 12〉 application.conf에 SSL 인증서 설정

```
ssl.KeyManagerFactory.algorithm=SunX509
trustmanager.algorithm=JKS
keystore.password=dev123
keystore.file=conf/www.company.com.jks

http.port=80
https.port=443
```

그러나 웹서버로 Play! 프로젝트를 1:1로 운영하는 경우는 드물고, 보통 물리적 웹서버 1대에 여러 Play! 프로젝트를 운영하게 된다. 이 경우 Play! 프로젝트 앞단에 다른 웹서버를 놓고 이 웹서버에서 Play! 프로젝트로 리버스 프록시(Reverse Proxy) 방식으로 클라이언트의 요청을 전달하도록 구성한다. 대부분의 웹서버에 적용할 수 있지만 필자는 Nginx(http://nginx.org/)를 추천한다. Nginx는 Play! 프레임워크에 내장된 JBoss Netty처럼 C10K 문제를 해결했고 리버스 프록시나 정적 자원 서비스 같은 간단한 HTTP 응답에 탁월하다고 알려져 있다.

Nginx와 Play! 프로젝트들을 리버스 프록시로 연동하기 위해 각 Play! 프로젝트별로 사용할 서비스 포트와 JPDA 포트를 미리 정해야 한다. JPDA 포트는 기본적으로 8000번 포트가 세팅돼 있는데, 만약 8000번 포트를 사용할 수 없을 경우 임의로 아무 포트나 잡게 된다. 따라서 다른 Play! 프로젝트가 사용하는 포트를 JPDA 포트로 사용하는 일이 없도록 〈표 7〉처럼 미리 Play! 프로젝트의 서비스 포트와 JPDA 포트를 지정하는 정책을 마련하는 게 좋다. 〈리스트 13〉은 지정된 포트들을 각 Play! 프로젝트마다 application.conf에 설정한 것이다. 이 아키텍처에서는 SSL 인증서를 사용해도 사용자와 Nginx 간에만 HTTPS 프로토콜로 통신하고 Nginx와 Play! 프로젝트 사이는 HTTP 프로토콜로 통신하게 된다. 따라서 Play! 프로젝트들의 HTTPS 설정은 사용하지 않으므로 주석으로 처리한다. 그리고 리버스 프록시를 받을 수 있는 서버 IP를 추가한다.

서비스 도메인	Play! 서비스 포트	JPDA 포트	SSL 포트
a.company.com	8001	7001	443
b.company.com	8002	7002	443
www.test.com	8003	7003	8443

〈표 7〉 Play! 프로젝트들의 서비스 포트와 JPDA 포트를 미리 지정한 예

〈리스트 13〉 application.conf에 www.test.com의 서비스 포트 JPDA 포트를 설정

```
http.port=8003
# https.port=8443

jpda.port=8001

XForwardedSupport=127.0.0.1
```

이번에는 Nginx에서 Play! 프로젝트로 리버스 프록시를 설정해보자. Nginx의 환경 파일인 /conf/nginx.conf에 설정하면 된다. SSL 인증서는 .pem, .key 파일 형식을 사용한다. 사용자가 Nginx에 HTTPS 프로토콜로 요청을 하더라도 Nginx는 Play!에 HTTP 프로토콜로 요청하므로, Play! 프로젝트에서는 실제 사용자가 요청한 프로토콜이 HTTPS인지 알 수 없다. 이에 Nginx에서 Play!로 요청을 보낼 때 HTTP 헤더에 x-forwarded-for, x-forwarded-ssl을 추가로 보내도록 SSL 환경 설정을 추가한다. 이 2개의 헤더는 Play!에서 대소문자를 구분하므로 꼭 소문자로 표기해야 한다. 이제 Play! 프로젝트의 콘트롤러에서 request.secure로 사용자가 Nginx로 요청한 프로토콜이 HTTPS인지 확인할 수 있다.

〈리스트 14〉 Play! 프로젝트로 리버스 프록시를 하기 위한 Nginx 환경 설정의 예

```
http {

… 중간 코드 생략 …

        # HTTP : www.test.com
        server {
                listen       80;
                server_name  www.test.com;
                server_tokens off;
```

```
                location / {
                proxy_pass      http://127.0.0.1:8003/;
                        proxy_redirect  off;
                proxy_set_header Host   $host;
                proxy_set_header X-Real_IP
$remote_addr;
                proxy_set_header X-Forwarded_For
$proxy_add_x_forwarded_for;
                client_max_body_size 10m;
                }
        }

        # Redirect : test.com → www.test.com
        server {
                listen       80;
                server_name  test.com;
                server_tokens off;
                rewrite ^(.*) http://www.test.com $1
permanent;
        }

        # HTTPS : www.test.com
        server {
                 listen       8443;
                 server_name  www.test.com;
                 server_tokens off;
                 ssl  on;
                 ssl_certificate
/usr/local/ssl/www.test.com.pem;
                 ssl_certificate_key  /usr/local/ssl/
www.test.com.key;
                 ssl_session_timeout  5m;
                 ssl_protocols  SSLv2 SSLv3 TLSv1;
                 ssl_ciphers  HIGH:!aNULL:!MD5;
                 ssl_prefer_server_ciphers  on;
                 location / {
                        proxy_pass
http://127.0.0.1:8003/;
                proxy_redirect  off;
                proxy_set_header Host   $host;
                        proxy_set_header X-Real_IP
$remote_addr;
proxy_set_header x-forwarded-for
$proxy_add_x_forwarded_for;
                proxy_set_header x-forwarded-ssl  on;
                }
        }
}
```

윈도우 PC 환경에서 Nginx를 설치하려면 Nginx for Windows(http://kevinworthington.com/nginx-for-windows/)를 설치하자. GUI를 지원해 리눅스 버전보다 사용이 편리하다. 이제 Nginx와 Play! 프로젝트를 각각 실행한 뒤 웹브라우저에서 http://localhost/로 접속하면 80포트로도 Play! 프로젝트에 접속할 수 있다. Nginx까지는 80포트로, Nginx에서 Play! 프로젝트로는 8003포트로 리버스 프록시로 접속된다. 아파치 HTTP 서버나 라이티(lighttpd) 등 다른 웹서버와 Play! 프로젝트를 연동하는 방법은 Play! 프레임워크 홈페이지의 예제를 참고하자. 앞서 설명한 Nginx와 환경설정 방법만 다를 뿐 기본적인 개념은 동일하다.

여기까지 Play! 프레임워크의 국제화, 테스트, 배포 지원 기능 등을 살펴봤다. Play! 프레임워크를 도입하면 기술의 생산성과 웹서비스의 품질을 높일 수 있는 개발 정책도 도입할 수 있게 된다. Play! 프레임워크의 이름이 'Play'인 이유는 이런 정책들로 인해 더 즐겁게, 놀 듯이 쉽게 개발할 수 있도록 하겠다는 의지의 표현일 것이다. 오픈소스이므로 부담 없이 고쳐 쓸 수도 있고 커미터(Commiter)로도 활약할 수도 있는 Play! 프레임워크를 더 많은 개발자들이 사용해 보다 많은 가능성에 도전하길 바라면서 3회에 걸친 연재를 여기서 마친다.

이달의 디스켓 : playframework_3.zip

다운로드 방법 안내 – 234p 참고

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/tests
4. Play! 프레임워크 한국 개발자 그룹 페이스북 : facebook.com/groups/294888493919737
5. Tutorial: Play Framework, JPA, JSON, jQuery, & Heroku jamesward.com/2011/12/11/tutorial-play-framework-jpa-json-jquery-heroku
6. Nginx 공식 홈페이지 : nginx.org
7. The C10K problem : en.wikipedia.org/wiki/C10k_problem
8. Play framework으로 블로그 개발하기 1~6 : iam1492.tistory.com/23
9. Play framework에서 Long Polling 구현하기 : blog.outsider.ne.kr/527
10. 오픈소스 첫 경험 : ppassa.wordpress.com/2012/04/21/contribute_to_open_source