Play Framework

A java web framework looking less like java

Alexander Reelsen

@spinscale

alexander@reelsen.net

Table of contents

About me - Alexander Reelsen

- Studied information systems
- 10 years linux system engineering, converted to software engineering
- Web framework enthusiast, fed up with complex java environment for simple webapps
- Other interests: Scaling web architectures, Web 2.0 (nosql, search)
- Started with play 1.0 in 2009, started writing cookbook in dec 2010
- Working at Lusini GmbH, building an ecommerce platform
- Streetball/Basketball

The web is taking over

- Web as ubiquitous platform
- Providing data, not web pages
- Browser as the operating system
- JavaScript gains steam, toolkits as well as engines
- Web **applications** are alive

Play in 10 seconds - Quick overview

- No compile, deploy, restart cycle - Fix the bug and hit reload!
- Share nothing system - no state in the server
- Clean templating system - using groovy
- Exact errors (including line numbers, even for templates)
- Lots of builtin features for fast development
- Pure Java
- Starts fast, runs fast
- Extensible by modules
- WebSocket support, **no servlet support**

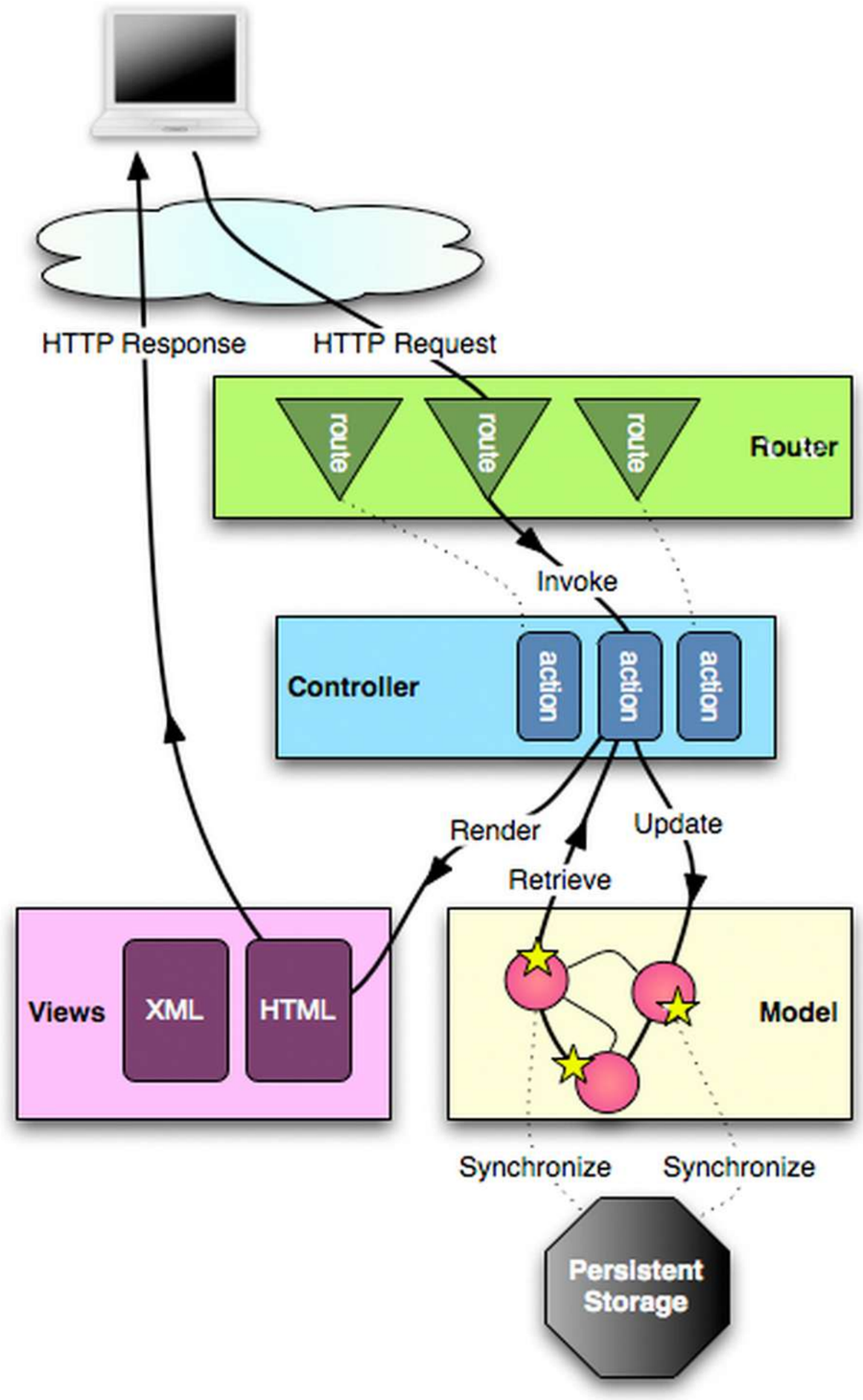
History

- Exists since 2008, by Guillaume Bort from zenexity
- Releases
  - 1.0 in October 2009
  - 1.1 in November 2010
  - 1.2 in April 2011
- Current: 1.2.4 + development tree + play-scala
- Play 2.0 preview is available

A web framework imposes its own architecture style

- REST as architectural paradigm for ressources
- URLs are the entry point (and implicit interface) to your application
- Do not work against HTTP (stateless protocol)
- Convention over configuration
- Only fractions of differences between development and production mode

Play framework architecture



Play is a lot of glue code

- Hibernate, JBoss Netty, log4j
- Oval, Lucene, Gson
- Eclipse compiler, ivy, xstream
- Apache Commons: fileupload, httpclient, email, logging, beanutils
- EhCache, JAMon, Groovy, javassist
- c3p0, joda time, htmlunit

Play application layout

```
play new myapp
```

- conf
- app
- test
- lib
- public
- modules

conf/application.conf

- Configure database access (via JDBC): `db=fs` , `db=mem` `db=mysql:user:pwd@database_name`
- I18N
- Logging, Caching, Mail configuration
- Node & system specific settings
- Module specific configuration

conf/routes

```
GET     /                       Application.index
GET     /user/{username}        Application.showUser
POST    /user                   Application.createUser
DELETE  /user/{username}        Application.deleteUser
GET     /public                 staticDir:public
```

Controller

```
public class Application extends Controller {
  public static void index() {
    List<User> users = User.findAll();
    render(users);
  }
}
```

Controller - Showing a user

```
public static void showUser(String username) {
  User user = User.find("byUsername", username).first();
  notFoundIfNull(user);
  render(user);
}
```

Controller - Deleting a user

```
public static void deleteUser(String username) {
  User user = User.find("byUsername", username).first();
  notFoundIfNull(user);
  user.delete();
  index();
}
```

Controller - Creating a user

```
public static void createUser(@Valid User user) {
  if (validation.hasErrors()) {
    flash.error("Invalid user data");
    Application.index();
  }
  user = user.save();
  Application.showUser(user.username);
}
```

Controller - Accessing the session

```
public class AuthController extends Controller {


@Before(unless = "login")
public static void checkSession() {
  if (!session.contains("username")) {
    forbidden("You are not authorized");
  }
}
```

Controller - Accessing the session

```
public void login(String username, String password) {
  User user = User.find("byUsernameAndPassword",
    username, Crypto.passwordHash(password)).first();
  notFoundIfNull(user);
  session.put("username", user);
  Application.index();
}
```

Template: app/views/Application/index.html

```
#{extends 'main.html' /}
<ul>
#{list items:users, as:'user'}
  <li>
    #{a @Application.showUser(user.username)}
      ${user.username}
    #{/a}
  </li>
#{/list}
</ul>
```

Template: Creating complex objects

```
#{form @Application.createUser()}
Username: <input type="text" name="user.username" />
Password: <input type="pass" name="user.password" />
Email: <input type="text" name="user.email" />
<input type="submit" value="Add user" />
#{/form}
```

Templating: More tags

- `doLayout`, `extends`, `include`
- `if`, `ifnot`, `else`, `elseif`
- `&{'i18nVariable'}` out of conf/messages.de
- Always access to: `session`, `flash`, `request`, `params`, `lang`, `messages`, `out`, `play`

Official templating documentation

Using mixins for better templating

```
public class SqrtExtension extends JavaExtensions {
  public static Double sqrt(Number number) {
    return Math.sqrt(number.doubleValue());
  }
}
```

```
<div>
Square root of ${x} value is: ${x.sqrt()}
</div>
```

Model

```
@Entity public class User extends Model {
  @Required @Column(unique = true)
  public String username;

  @Required @Email
  public String email;


  @Required
  public String password;
}
```

Model - Finders

```
User user = User.find("byUsername", username).first();
user.password = "foobar"; // setter call
user.save();

List<User> users = User.findAll();
users.get(0).delete();

List<String> names = User.find("select u.username from User u
 order by u.username desc").fetch();
```

Jobs - execution flow outside requests

- Scheduled jobs (housekeeping)
- Bootstrap jobs (initial data initialization)

```
/* @Every("1h"), @OnApplicationStop, @On("0 0 12 * * ?") */
@OnApplicationStart
public class LoadDataJob extends Job {
  public void doJob() {
    /* .. do whatever you want */
  }
}
```

Promise is the new Future

- Asynchonous operations allows to reuse the connection thread pool while the request is not closed
- Inherits from `Future`, `onRedeem()` is executed on arrival
- `play.libs.F`: `Promise.waitAll()`, `Promise.waitAny()`

```
public static void renderPdf() {
  InputStream is = await(new PdfRenderJob().now());
  renderBinary(is);
}
```

Promises for asynchronous Requests

```
public static void getResponses() {
  Promise r1 = WS.url("http://www.google.de").getAsync();
  Promise r2 = WS.url("http://www.lycos.de").getAsync();
  List<HttpResponse> responses =
      Promise.waitAll(r1, r2).getOrNull();
  render(responses);
}
```

Thread is suspended at `Promise.waitAll(r1, r2)`

Caching

- `String f = Cache.get("id", String.class);`
- `Cache.set("id", "someId", "5mn");`
- Writing your own implementation is implementing one interface
- Existing implementations: EhCache (default), memcached, hazelcast

```
memcached=enabled

memcached.host=127.0.0.1:11211
```

Caching inside the controller

```
/* caches the whole result */
@CacheFor("5s")
public static void time() {
    /* HTTP header Cache-Control: max-age: 1 */
    response.cacheFor("1s");
    renderText(new Date());
}
```

Caching inside templates

- Make sure no expensive computation happened in the controller
- Fetching 1000 entities in the controller, then using the cached version in the template is useless

```
#{cache 'currentTime', for:'3s'}
   ${new java.util.Date()}
#{/cache}
```

Modules & Plugins

- Modules provide arbitrary extensions (HTML, CSS, Java, deployment helpers, i18n, URLs)
- Modules are mini applications inside your application
- Packaged as zip file, fetched from repository
- Currently there are more than 100 modules, but still far from the rich grails eco system
- Plugins are java classes inheriting from `PlayPlugin` and extending functionality of an application via hooks

Providing hooks to extend functionality

- `play.PlayPlugin` is the key class, has a priority
- Startup: `onLoad()`, `onApplicationStart()`, `afterApplicationStart()`, `onApplicationStop()`, `onConfigurationRead()`, `onRoutesLoaded()`, `onApplicationReady()`, `afterFixturesLoad()`
- Status: `getStatus()`, `getJsonStatus()`
- Objects: `enhance()`, `bind()`, `unbind()`, `getMessage()`
- Events: `onEvent`, `PlayPlugin.postEvent()`

PlayPlugin hooks

- Invocation: `rawInvocation()`, `beforeInvocation()`, `afterInvocation()`, `onInvocationException()`, `invocationFinally()`, `onInvocationSuccess()`
- Action Invocation: `beforeActionInvocation()`, `onActionInvocationResult()`, `afterActionInvocation()`
- Template: `loadTemplate()`, `overrideTemplateSource()`, `addTemplateExtensions()`, `addMimeTypes()`

PlayPlugin hooks

- Development: `detectChange()`, `detectClassesChange()`, `beforeDetectingChanges()`, `onClassesChange()`
- Requests & routing: `serveStatic()`, `onRequestRouting()`, `routeRequest()`

Dependency & module management

- Based on Apache Ivy, but using YAML
- Configuration file: `conf/dependencies.yml`
- `play deps --sync`

```
require:
    - play
    - play -> crud
    - play -> cobertura 2.1
    - org.elasticsearch -> elasticsearch 0.17.2
```

Dependency Management - Repositories

- Maven or custom repositories can be added (URL layout can be configured)

```
repositories:
    - sonatype:
        type: iBiblio
        root: "http://oss.sonatype.org/content/repositories/releases"
        contains:
            - org.elasticsearch -> *
```

Testing

- **Unit tests**

  Standard junit tests extending from UnitTest - need a JPA environment

- **Functional tests**

  Integration tests checking the HTTP responses and their contents

- **Selenium tests**

  GUI tests. Very nice controllable, in-browser playback recorder

  Possibility of doing step-by-step slow debugging

- **Cobertura module**, **spock plugin**

Using YAML to provide test data

```
User(aleree):
  username: alr
  password: test
  email   : alexander@reelsen.net
```

```
@Before public void setUp() {
  Fixtures.deleteAll();
  Fixtures.load("data.yml");
}
```

Unit testing

```
public class BasicTest extends UnitTest {
    @Test
    public void testThatSomethingInTheModelWorks() {
        // Model tests go here
    }
}
```

Starts a play environment including JPA if needed (when started inside the IDE)

Functional testing

```
public class ApplicationTest extends FunctionalTest {
    @Test
    public void testThatIndexPageWorks() {
        Response response = GET("/");
        assertIsOk(response);
        assertContentType("text/html", response);
        assertCharset("UTF-8", response);
    }
}
```

Selenium testing

- Own methods to receive emails and access the cache in tests
- Play Selenium Plugin for Firefox

```
#{fixture delete:'all', load:'data.yml' /}


#{selenium}
    open('/')
    assertNotTitle('Application error')
#{/selenium}
```

Tests runner

Select the tests to run, then click [Start] and pray

Start ! 3 tests to run (Bookmark this link to save this configuration) - Unselect all

There is 1 unit test,

+ **BasicTest**

1 functional test,

+ **ApplicationTest**

and 1 selenium test,

+ **Application**

Running the tests at localhost:9000/@tests

Play is running a selenium test... Stop it

Google Custom Search Engine

Tests	
// Open the home page, and check that no error occured	
open	/
pause	10000
assertNotTitle	Application error

Selenium TestRunner

Execute Tests

Fast — Slow

Highlight elements

Elapsed: 00:01

Tests	Commands
0 run	0 passed
0 failed	0 failed
	0 incomplete

Tools

View DOM | Show Log

Your new application is ready!

Congratulation, you've just created a new play application. This page will help you in the few next steps.

Why do you see this page?

The **conf/routes** file defines a route that tell play to invoke the **Application.index** action when a browser requests the / URI using the **GET** method:

```
# Application home page
GET     /         Application.index
```

So play has invoked the **controllers.Application.index()** method:

```
public static void index() {
    render();
}
```

Using the **render()** call, this action asks play to display a template. By convention play has displayed the **app/views/Application/index.html** template:

```
#{extends 'main.html' /}
#{set title:'Home' /}

#{welcome /}
```

This template extends the **app/views/main.html**, and uses the **#{welcome /}** tag to display this welcome page.

Play!

Play 1.2.2

Browse

- Local documentation
- Browse Java API

Contents

1. Why do you see this page?
2. Need to set up a Java IDE?
3. Need to connect to a database?
4. Need more help?

Search

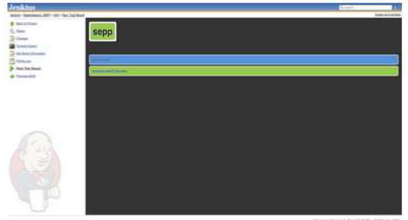
Get help with google

Running the tests at localhost:9000/@tests

Continouos integration

- Calimoucho, runs also on integration.playframework.org
- Hudson plugin

  Runs `play auto-test`



Scala test cases
1.1Final-536-gd2eed28-84985f

Yabe with Scala
1.1Final-536-gd2eed28-84985f

Zencontact
1.1Final-536-gd2eed28

Jobboard
1.1Final-534-g7c96c18

Yabe with Scala
1.1Final-531-gda81ccc-84985f

Zencontact
1.1Final-531-gda81ccc

Test cases
1.1Final-531-gda81ccc

Scala test cases
1.1Final-529-geadab91-84985f

Jobboard
1.1Final-529-geadab91

Zencontact
1.1Final-526-g14de1db

Forum
1.1Final-526-g14de1db

Test cases
1.1Final-526-g14de1db

Yabe with Scala
1.1Final-524-ga84bba9-84985f

Forum
1.1Final-524-ga84bba9

Jobboard
1.1Final-524-ga84bba9

Scala test cases
1.1Final-521-ga86456c-84985f

Back to Project
Status
Changes
Console Output
Edit Build Information
Polling Log
Play! Test Result
Previous Build

sepp

application.log

controllers.SearchTest.class

Running in production

- No servlet container or application server needed
- Deploying into a servlet container is supported but not recommended
- SSL support built-in
- Serving static files is faster with apache or nginx
- Several environments in one configuration file possible
- Support for automatic deploying into GAE, Heroku, Stax, capistrano, dotcloud, cloudbees, playapps, cloudfoundry

Setting environments in application.conf

```
application.name=My application
%test.application.name=Test application
%prod.application.name=Production application
```

```
play start --%prod
```

Security

- The session cookie is signed but not encrypted!
- The template engine automatically escapes strings
- SQL injection is only possible when writing own JPA queries
- CSRF: Use of authenticity token

See more at the security documentation

Monitoring via `play status`

- Provides information about your application state
- JVM infos, Play version, running threads and application configuration information
- Loaded modules, loaded plugins (order)
- Datasource, Jobs, execution pool information
- Detailed information about controller runtimes and template rendering times, including min/max/avg values

Adding arbitrary monitoring points in your code

Adding a simple counting monitor

```
MonitorFactory.add("CacheHitsFoo", "cnt", 1);
```

Adding a monitor which calculates min/max/avg values

```
Monitor monitor = MonitorFactory.start("RemoteApiCall");
/// … logic goes here
monitor.stop();
```

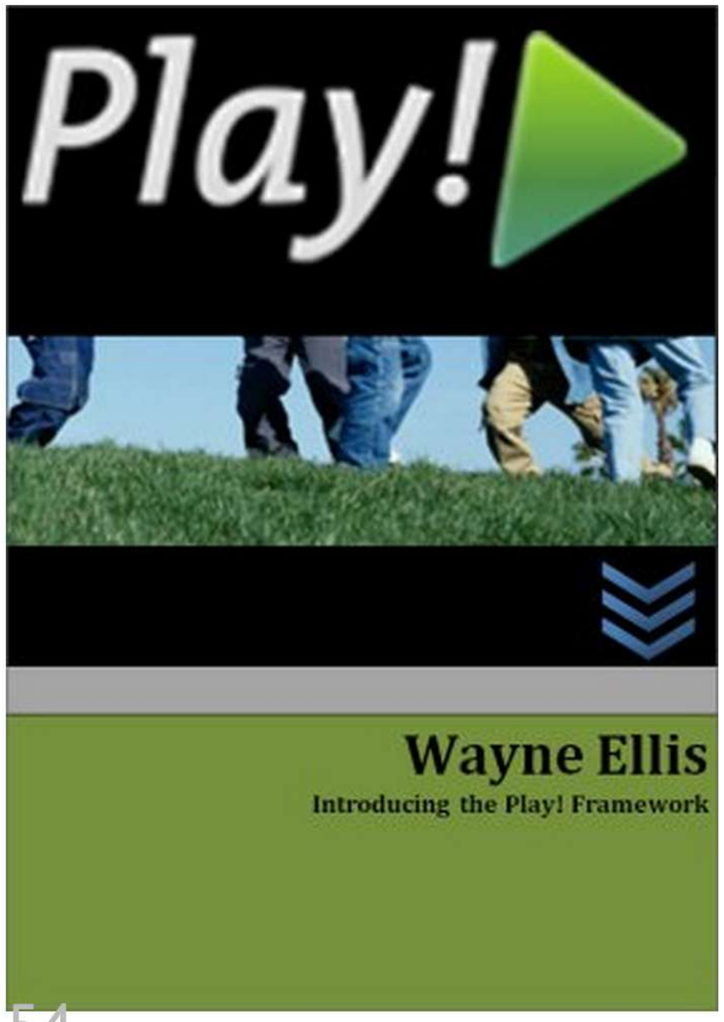
All monitors are automatically added to the status output

Roadmap

- Play 2.0 is coming, available as preview
- 1.x will be supported and developed as well
- The core will be more scala, but Java and Scala will be used
- Typesafe template engine, typesafe router (more performance)
- No more Hibernate (ebean as replacement)
- SBT as build system
- Native Akka support
- Built-in LESS and coffeescript support

Questions?

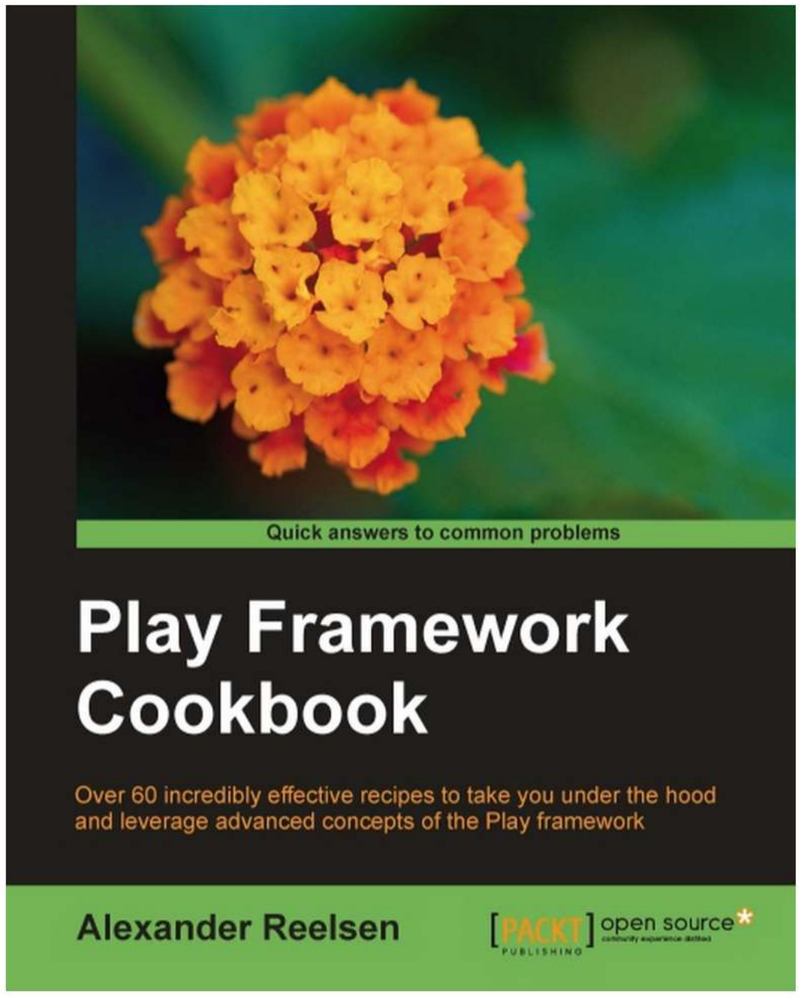
Books: Introducing the Play! framework



Introducing the Play! framework

by Wayne Ellis

Books: Play Framework Cookbook



Play Framework Cookbook

by Alexander Reelsen

Resources

- Playframework
- Twitter: @playframework
- My presentations at slideshare
- German: Articles from 2009 in kaffeeklatsch magazine (part 1, part 2)
- Play framework cookbook (example sources on github)
- Presentation has been done with shower

Workshop - Bidding application using WebSockets

You need a newer version than play 1.2.3 for this

```
git clone git://github.com/playframework/play play
cd play
git checkout remotes/origin/1.2.x
cd framework
ant
```

```
git clone git://github.com/spinscale/play-bidding-sample
```