Web development with the Play! framework

Java web development is fun again

Tomche Delev

Java User Group
Macedonia
www.jug.mk

October 05, 2011

Outline

Outline

Quick overview

- Fix the bug and hit reload!
  - No compile, deploy or restart the server cycle. Edit your Java files, save, refresh your browser and see the results immediately!
- Efficient template system
  - A clean template system based on **Groovy** as an expression language (template inheritance, includes and tags)
- Full stack
  - Integration with Hibernate, OpenID, Memcached... And a plugin system.
- Resolve errors quickly
  - When an error occurs, play shows you the source code and the exact line containing the problem. Even in templates.

Quick overview

- Fix the bug and hit reload!
  - No compile, deploy or restart the server cycle. Edit your Java files, save, refresh your browser and see the results immediately!
- Efficient template system
  - A clean template system based on **Groovy** as an expression language (template inheritance, includes and tags)
- Full stack
  - Integration with Hibernate, OpenID, Memcached... And a plugin system.
- Resolve errors quickly
  - When an error occurs, play shows you the source code and the exact line containing the problem. Even in templates.

Quick overview

- Fix the bug and hit reload!
  - No compile, deploy or restart the server cycle. Edit your Java files, save, refresh your browser and see the results immediately!
- Efficient template system
  - A clean template system based on **Groovy** as an expression language (template inheritance, includes and tags)
- Full stack
  - Integration with Hibernate, OpenID, Memcached... And a plugin system.
- Resolve errors quickly
  - When an error occurs, play shows you the source code and the exact line containing the problem. Even in templates.

Quick overview

- Fix the bug and hit reload!
  - No compile, deploy or restart the server cycle. Edit your Java files, save, refresh your browser and see the results immediately!
- Efficient template system
  - A clean template system based on **Groovy** as an expression language (template inheritance, includes and tags)
- Full stack
  - Integration with Hibernate, OpenID, Memcached... And a plugin system.
- Resolve errors quickly
  - When an error occurs, play shows you the source code and the exact line containing the problem. Even in templates.

Quick overview

- Fix the bug and hit reload!
  - No compile, deploy or restart the server cycle. Edit your Java files, save, refresh your browser and see the results immediately!
- Efficient template system
  - A clean template system based on **Groovy** as an expression language (template inheritance, includes and tags)
- Full stack
  - Integration with Hibernate, OpenID, Memcached... And a plugin system.
- Resolve errors quickly
  - When an error occurs, play shows you the source code and the exact line containing the problem. Even in templates.

Quick overview

- Stateless model
  - Play is a real "Share nothing" system. Ready for REST, it is easily scaled by running multiple instances of the same application on several servers.
- Asynchronous
  - Based on an Non blocking IO model, it allows to create modern web applications based on long polling and WebSockets.
- Pure Java
  - Code with Java, use any Java library and develop with your preferred IDE. Integrates nicely with Eclipse or NetBeans.
- Fun & Productive
  - Cut out the time you spend waiting for your Java application to restart, increase your productivity complete your projects faster.

Quick overview

- Stateless model
  - Play is a real "Share nothing" system. Ready for REST, it is easily scaled by running multiple instances of the same application on several servers.
- Asynchronous
  - Based on an Non blocking IO model, it allows to create modern web applications based on long polling and WebSockets.
- Pure Java
  - Code with Java, use any Java library and develop with your preferred IDE. Integrates nicely with Eclipse or NetBeans.
- Fun & Productive
  - Cut out the time you spend waiting for your Java application to restart, increase your productivity and complete your projects faster.

Quick overview

- Stateless model
  - Play is a real "Share nothing" system. Ready for REST, it is easily scaled by running multiple instances of the same application on several servers.
- Asynchronous
  - Based on an Non blocking IO model, it allows to create modern web applications based on long polling and WebSockets.
- Pure Java
  - Code with Java, use any Java library and develop with your preferred IDE. Integrates nicely with Eclipse or NetBeans.
- Fun & Productive
  - Cut out the time you spend waiting for your Java application to restart, increase your productivity and complete your projects faster.

Quick overview

- Stateless model
  - Play is a real "Share nothing" system. Ready for REST, it is easily scaled by running multiple instances of the same application on several servers.
- Asynchronous
  - Based on an Non blocking IO model, it allows to create modern web applications based on long polling and WebSockets.
- Pure Java
  - Code with Java, use any Java library and develop with your preferred IDE. Integrates nicely with Eclipse or NetBeans.
- Fun & Productive
  - Cut out the time you spend waiting for your Java application to restart, increase your productivity and complete your projects faster.

Quick overview

- Stateless model
  - Play is a real "Share nothing" system. Ready for REST, it is easily scaled by running multiple instances of the same application on several servers.
- Asynchronous
  - Based on an Non blocking IO model, it allows to create modern web applications based on long polling and WebSockets.
- Pure Java
  - Code with Java, use any Java library and develop with your preferred IDE. Integrates nicely with Eclipse or NetBeans.
- Fun & Productive
  - Cut out the time you spend waiting for your Java application to restart, increase your productivity and complete your projects faster.

Outline

Roots and fundamentals

History

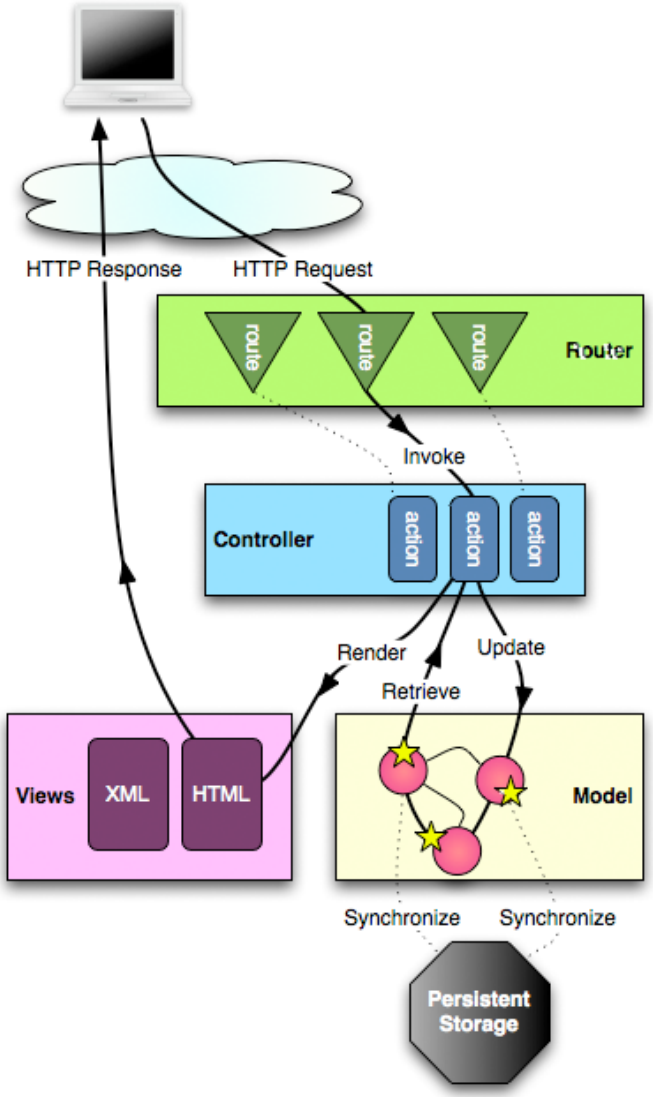
- Exists since 2008, by Guillaume Bort from Zenexity
- Release 1.0 was in October 2009
- Current: 1.2.3 (24 Aug 2011) + development tree

Own architectural style

- REST as architectural paradigm for resources
- 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

Outline

Play architecture overview



Play is a lot of glue code

- Hibernate (Persistence)
- OVal (Validation)
- Lucene (Searching)
- Google gson (JSON)
- Eclipse compiler (Compiling and building)
- Apache Commons (FileUpload, HttpClient, Email, Logging, BeanUtils)
- Apache MINA and Asyncweb (Asynchronius programming)
- Ehcache (Caching)
- JAMon (Monitoring)
- Groovy (Dynamic language)

Play specialties

- No support for servlet API (yes, in a web framework)
- Sharing objects via memcached through several nodes
- Everything is UTF-8
- Full text indexing with 2 annotations
- No anemic domain model - logic is in the object
- DAOs and finders are not external
- Textmate, Eclipse bundles, also support for IDEA IntelliJ and NetBeans

Application layout

Creating a new app

```
play new myapp
```

Application structure

```
./conf
./conf/routes
./conf/application.conf
./conf/messages
./test
./lib
./public
./app
./app/models
./app/controllers
./app/views
```

Application config file

conf/application.conf

- Configure database access
  - `db=fs, db=mem`
  - `db=mysql:user:pwd@database_name`
  - Any JDBC connection
- Specify modules
- Supported languages
- Logger
- memcached setup
- mail configuration
- mode/system specific settings

The conf/routes file

Interface contract to the outer world

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

Outline

Designing a domain model

A simple user

```
@Entity
public class User extends Model {
    @Required
    @Column(unique = true)
    public String username;
    @Required
    @Email
    public String email;
    @Required
    public String password;
    public void setPassword(String password) {
        this.pass = Crypto.passwordHash(pass);
    }
}
```

The Model class is a helper

Finders and Entity actions

```
User user = User.find("byUsername", username).first();

user.password = "foobar";
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();
```

Calling business logic

User controller

```
public class Application extends Controller {
    public static void index() {
    List<User> users = User.findAll();
    render(users);
}
public static void showUser(String username) {
    User user = User.find("byUsername", username)
    .first();
    notFoundIfNull(user);
    render(user);
}
...
```

Calling business logic

User controller (ctd.)

```
..
 public static void deleteUser(String username) {
    User user = User.find("byUsername", username).first();
    notFoundIfNull(user);
    user.delete();
    Application.index();
 }
 public static void createUser(@Valid User user) {
    if (validation.hasErrors()) {
        flash.error("Invalid user data");
        Application.index();
    }
    user = user.save();
    Application.showUser(user.username)
 }
}
```

Calling business logic

Example: Accessing the session

```
public class AuthController extends Controller {
    @Before(unless = "login")
    public static void checkSession() {
        if (!request.session.contains("username")) {
            forbidden("You are not authorized");
        }
    }
    public void login(String username, String password) {
        String pass = Crypto.passwordHash(password);
        User user = User.find("byUsernameAndPassword",
            username, pass).first();
        notFoundIfNull(user);
        request.session.put("username", user);
        Application.index();
    }
}
```

The templating system

List users (app/views/Application/index.html)

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

The templating system

Add user (createUser.html)

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

The templating system

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

The templating system

Extending objects using mixins

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

The template code

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

Outline

Testing

Providing test data

YAML formatted file provides testdata

```
User (Tomche):
 - username : tdelev
 - password : test
 - email : tomche.delev@finki.ukim.mk
```

Loading test data...

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

Testing

Unit tests

- Standard junit tests
- Extend from UnitTest, which needs a JPA environment

Functional tests

- Integration tests
- Checks the external responses (http response)

Selenium tests

- GUI tests
- Very nice controllable, playback recorder
- Possibility of doing step-by-step slow debugging

JUGMK Play!

Testing

CI with Calimoucho

- Poor mans hudson
- Checks out the project and runs play auto-test, which needs a graphical layer for selenium tests
- Check it under http://integration.playframework.org

Code coverage with cobertura

- Enable the cobertura module in application.conf
- Run the tests, check the results

Jobs - being asynchronous

Doing the right thing at the right time

- Scheduled jobs (housekeeping)
- Bootstrap jobs (initial data providing)
- Suspendable requests (rendering a PDF report without blocking the connection thread pool)

```
@OnApplicationStart
public class LoadDataJob extends Job {
   public void doJob() {

   }
}
```

Putting play into production

The setup

- A redirector like nginx or apache is preferred
- Also eliminates the need to serve static files
- Redirect to different nodes would be the main task
- Profile per nodes possible (very useful for server farms)

JUGMK Play!

Monitoring play application

Partial Output of play status

```
Monitors:

Application.showLatestRecipesRss(), ms. ->
4120 hits; 41.0 avg; 20.0 min; 260.0 max;
/app/views/Application/showLatestRecipesRss.html, ms. ->
4120 hits; 34.9 avg; 19.0 min; 235.0 max;

Datasource:
Job execution pool:
Scheduled jobs:
```

Outline

Play 2.0 is on its way

the next major version of Play framework

- Brand new build system
- More asynchronous features
- All native Java and Scala support
- More modules through module repository

JUGMK Play!

Useful modules

Slowly but steadily growing

- Scala, Scalate, Akka
- PDF, Excel modules
- Guice and Spring modules
- Netty and Grizzly support
- GWT support, GAE support
- Extended CSS, SASS
- Ivy and Maven support
- Siena, Ebean ORM, MongoDB
- Database migration module
- Hosting: stax, playapps

JUGMK Play!

TODO

Open issues

- NoSQL support (Siena, MongoDB)
- Amazon Cloud Integration
- Hosting platform (playapps.net has just launched)
- Lucene Solr Support for shared environments
- Tighter integration with JavaScript Toolkits like Dojo
- Far more modules - check out the rich grails ecosystem

Done!

Thanks for listening.
Questions?