Brot und Spiele

Schnelle Webentwicklung in reinem Java mit Play

Niels Stargardt

Webframeworks gibt es wie Sand am Meer. Ein weiteres zu evaluieren oder vorzustellen erscheint daher müßig, doch Play verspricht eine sehr hohe Produktivität (analog zu Grails oder Ruby on Rails) – und das bei reinem Java. Es ist auch eines der wenigen Frameworks, die das aufkommende Architekturmuster REST konsequent unterstützen. Bei Play sind URLs ein zentrales Element, und der Server ist zustandslos. Es lohnt sich also, Play einmal näher zu betrachten.

Webentwicklung kann Spaß machen

Betrachtet man die aktuelle Entwicklung im Java-Umfeld, so sind Webanwendungen einfach nicht mehr wegzudenken. Die Problematik dabei ist, dass weder Browser noch HTML speziell für die Entwicklung von Anwendungen entworfen wurden.

Zahlreiche Frameworks treten an, den Entwickler vor unterschiedlichem Browser-Verhalten und den Problematiken von Webanwendungen zu bewahren. Insbesondere komponentenbasierte Frameworks versuchen, die Probleme von HTML, HTTP-Protokoll und JavaScript zu verbergen. Um dies zu erreichen, werden zahlreiche Daten in der Session abgelegt. Dadurch wird der Server zustandsbehaftet, was zur Folge hat, dass bei einem Redeployment der alte Zustand erst wiederhergestellt werden muss und die Round-Trip-Zeiten länger werden. Außerdem wird es schwierig, die Ziele von RESTful http [Tilk08] zu erreichen.

Mein Eindruck ist, dass das Entwickeln von Webanwendungen mit Java trotz aller Verbesserungen durch verschiedene Frameworks nach wie vor eher unbeliebt ist. Play ist ein Framework, bei dem die Entwicklung wieder Spaß macht. Dies liegt vor allem an folgenden Punkten:

- Fix the bug and hit reload! Fehlermeldungen zeigen klar und deutlich, wo das Problem liegt. Ist der Fehler behoben, muss lediglich die Seite im Browser neu geladen werden und der korrigierte Code wird ausgeführt. Es ist kein Kompilieren und kein Deployment notwendig.
- Play-Server sind zustandslos. Damit kann der Entwickler direkt die Seite aufrufen, an der er gerade arbeitet.
- Play verfügt über ein effizientes Template-System. Es basiert auf der Groovy Template Engine und bietet damit eine ausdruckstarke Sprachsyntax. Es unterstützt Vererbung, Inkludieren und Tags.
- Play-Anwendungen bestehen aus reinem Java-Code. Es stehen also die meisten* Java-Bibliotheken zur Verfügung; die Wahl der Entwicklungsumgebung ist nicht eingeschränkt.
- Lösungen in Play sind einfach und leicht zu verstehen.
- Play-Anwendungen starten schnell und laufen performant.

* Da Play einen eigenen Classloader benutzt und bei einigen Klassen ein Bytecode-Enhancing vornimmt, kann es Probleme mit Bibliotheken geben, die ebenfalls den Classloader oder Bytecode verändern.

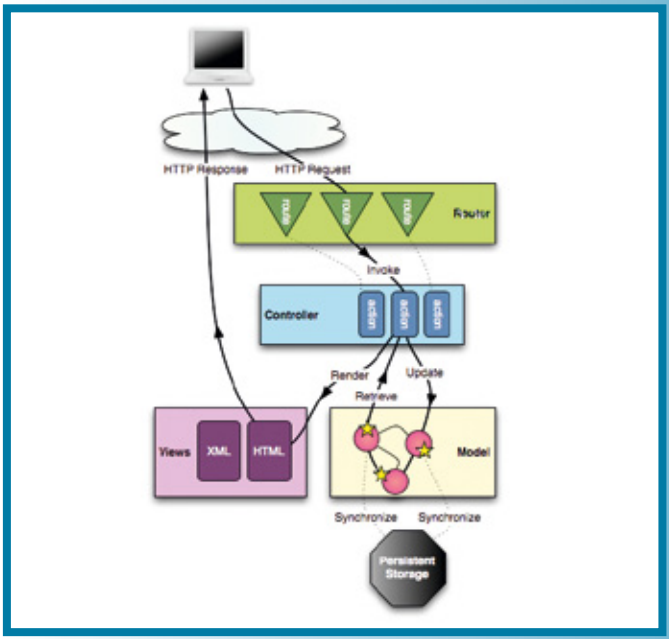

Abb. 1: Ablauf eines HTTP-Requests (übernommen aus der Play-Dokumentation)

Architektur von Play

Play setzt nicht auf der Servlet-API auf, sondern direkt auf dem HTTP-Protokoll. Es stützt sich dabei seit Version 1.1 auf das Network-Application-Framework netty von jboss [Netty]. Die genauen Gründe dafür erläutert der Hauptentwickler Guillaume Bort in seinem Blog [Bort]. Grundsätzlich erfordert das direkte Aufsetzen auf dem HTTP-Protokoll, sich von den gewohnten Servlet-Mechanismen wie Filter, Session usw. zu lösen. Dies ist aber keine relevante Einschränkung.

Im Folgenden wird der Ablauf eines HTTP-Request/Response-Zyklus bei Play beschrieben (s. Abb. 1). Grundlage ist ein Beispiel, bei dem eine Tabelle **`test`** mit den Spalten **`id`** und **`text`** editiert wird.

Jeder Request wird zunächst vom Router analysiert, der durch die **`routes`**-Datei** konfiguriert wird. Listing 1 zeigt eine beispielhafte **`routes`**-Datei.

```
# Home page
GET  /                  Application.index
GET  /edit/{<[0-9]+>id} Application.edit
POST /edit/{<[0-9]+>id} Application.save
*    /admin             module:crud
* /{controller}/{action} {controller}.{action}
```

Listing 1: Inhalt der routes-Datei

Die **`routes`**-Datei besteht aus drei Spalten: In der ersten Spalte wird – typisch für REST – zunächst nach den HTTP-Methoden **`GET`**, **`PUT`**, **`DELETE`** oder **`POST`** unterschieden, oder man wählt **`*`**, um alle HTTP-Methoden anzusprechen. Durch Leerzeichen getrennt folgt die URL, die reguläre Ausdrücke umfassen darf und bei der man Parameter setzen kann. In Listing 1 folgt dem **`edit`** eine Nummer, die der Action als Parameter **`id`** übergeben wird. In der dritten Spalte ist die Action definiert durch Angabe des

** Die **`routes`**-Datei ist eine von zwei Konfigurationsdateien bei Play. Die andere ist die **`application.conf`**, in der unter anderem der Datenbankzugang konfiguriert ist.

Klassen- und Methodennamens. Die Datei wird von oben nach unten abgearbeitet, und die erste Zeile, bei der HTTP-Methode und die URL zu dem Request passen, definiert die Action.

Controller sind Klassen, die im Package `controllers` liegen und die Klasse `Controller` erweitern. Die Actions sind statische Methoden ohne Rückgabewert. Listing 2 zeigt eine Edit- und eine Save-Action. In der Edit-Action wird das Objekt zu der übergebenen ID gelesen. Es folgt eine Prüfung, ob das Objekt gefunden wurde, und anschließend der Aufruf der Methode `render`, um die Darstellung auszulösen.

```
public class Application extends Controller {
  public static void edit(Long id) {
    Test testObj = Test.findById(id);
    notFoundIfNull(testObj);
    render(testObj);
  }
  public static void save(long id, Test testObj) {
    if (!testObj.isPersistent()) {
        throw NotFound("object.not_found");
    }
    if (testObj.validateAndSave()) {
      index();
    } else {
      render("Application/edit.html", testObj);
}}}
```

Listing 2: Edit- und Save-Action

Für die Rückgabe sind verschiedene Ausgabeformate möglich. Der Standardfall ist die HTML-Darstellung. Dazu nutzt Play die Groovy Template Engine, die eine ausdrucksstarke Expression Language bietet. Templates können dabei ineinander geschachtelt werden.

```
#{extends 'main.html' /}
#{set title:'Edit Test' /}
#{ifErrors}
  <h1>Oops...</h1>
  #{errors}
    <li>${error}</li>
  #{/errors}
#{/ifErrors}

#{form @save(testObj?.id) , id:'creationForm',
    enctype:'multipart/form-data' }
  <input type="hidden" name="testObj.id"
    value="${testObj.id}">
  <p>
    <label>Text</label>
    <input type="text" id="testObj_text" name="testObj.text"
      value="${testObj?.text}" class=
      "${errors.forKey('testObj.text') ? 'has_error' : ''}">
  <span class="error">${errors.forKey('testObj.text')}</span>
  </p>
    <input type="submit" value="Save">
#{/form}
```

Listing 3: Edit-Template

Listing 3 zeigt das zugehörige Edit-Template. Es besteht aus HTML, erweitert mit speziellen Tags. Es beginnt im Allgemeinen mit dem `extends`-Tag, der angibt, in welchen äußeren Rahmen der folgende Teil eingebettet ist (in der Datei `main.html` ist die Stelle mit `#{doLayout /}` gekennzeichnet). Anschließend werden Variablen für das `main.html` gesetzt, in diesem Fall der Titel der Seite. Interessant ist, wie die Variablen für das Template definiert werden, in dem Beispiel die Variable `testObj`: In Listing 2 findet sich lediglich der Aufruf `render(testObj)`. Play merkt sich beim Laden der Klasse die Variablennamen innerhalb der Action und ordnet der Variablen `testObj` aus dem Template die gleichnamige Variable aus der Action zu.

Mit der Anzeige der HTML-Seite durch den Browser ist der Request-Response-Zyklus abgeschlossen. Der Vollständigkeit halber sei hier noch der Code für die Save-Action aus Listing 2 betrachtet. Der Code der Save-Action ist komplett. Er behandelt auch das Binding der HTTP-Parameter, die automatisch an das Objekt gebunden werden. Auch dort erfolgt die Zuordnung wieder allein durch Namensgleichheit. In diesem Fall ist der Parameter `id` durch `/{<[0-9]+>id}` in der `routes`-Datei als Teil der URL definiert. Das Objekt `Test` ist ein `Model`, welches via JPA an die Datenbank angebunden ist. Bei der Persistenz wird Standard-JPA mit Hibernate verwendet.

Interessant ist, dass das `testObj` allein aufgrund der HTTP-Parameter (`testobj.id` und `testobj.text`) komplett aus der Datenbank gelesen und dann verändert wurde. Der Entwickler muss sich lediglich um die Validierung und das Speichern kümmern und wird dabei von einigen Komfortfunktionen unterstützt, im Beispiel `validateAndSave`. Um den JPA-Kontext kümmert sich das Framework, genau genommen das JPA-Plug-in. Es gibt allerdings auch andere Plug-ins, um zum Beispiel für die Google App Engine zu programmieren und den dortigen Speichermechanismus zu nutzen.

Ablauf einer Entwicklung

Um einen Eindruck zu vermitteln, wie eine Entwicklung abläuft, ist im Folgenden ein Projektplan dargestellt:

1. *Installation:* ZIP-Archiv von der Homepage [Play] herunterladen und entpacken, zum Beispiel nach `playframework`.
2. *Projekt anlegen:* Mit dem Aufruf `playframework/play-1.2/play new example` ein neues Projekt anlegen und mit `playframework play-1.2/play run example` starten (der Debugger-Port 8000 und der Anwendungs-Port 9000 müssen dafür frei sein). Im Browser die URL `http://localhost:9000/` öffnen. Die Anwendung startet und zeigt eine Seite mit weiteren Erläuterungen.
3. *Implementierung:* Im Ordner `example` liegt die Projektstruktur. Dort Datenmodell- und Controller-Klassen anlegen. Die Zwischenergebnisse im Browser überprüfen, indem man die Seite neu lädt (der Server bleibt gestartet).
4. *Produktivsetzen:* In der Produktion kann der gleiche Server genutzt werden wie bei der Entwicklung. Alternativ eine WAR-Datei erstellen und auf einem Servlet-Container deployen.
   Für den produktiven Betrieb den Anwendungsmodus auf Produktion setzen durch `application.mode=PROD`. Im Produktionsmodus wird der Code einmal zu Beginn komplett kompiliert, und es wird danach nicht geprüft, ob sich die zugehörigen Quelldateien geändert haben, während die Anwendung läuft. Fehler werden im Produktionsmodus ohne technische Details angezeigt.

Validierung

Ein komplettes Anwendungsframework benötigt Validierungsmechanismen. Play stellt dafür einige Validatoren sowie eine Basisklasse zum Erstellen eigener Validatoren zur Verfügung. Allen Validatoren ist gemeinsam, dass sie auf dem Framework Object Validation for Java [OVal] aufbauen. Das Schreiben und Testen eigener Validatoren ist damit leicht zu erledigen [Checks].

CRUD

Play bietet die Möglichkeit, ohne großen Aufwand eine einfache Admin-Oberfläche (die *C*reate-*R*ead-*U*pdate-*D*elete ermög-

licht) zu erstellen. Für jede zu administrierende Tabelle wird ein Controller erstellt, der den CRUD-Controller erweitert. Die Möglichkeiten, das Verhalten oder das Aussehen zu beeinflussen, sind dabei vielfältig und im Vergleich zu Grails deutlich einfacher. Allerdings ist die Grundfunktionalität etwas geringer als bei Grails; so gibt es beispielsweise keine Ansicht zur reinen Anzeige der Daten, sondern nur eine zum Bearbeiten.

Erweiterungsmöglichkeiten

Gute Frameworks zeichnen sich unter anderem durch Erweiterungsmöglichkeiten aus. Play bietet dafür zwei Ansatzpunkte:

- *Moduls* bieten die Möglichkeit, Module zu separieren und so größere Anwendungen zu strukturieren oder allgemeine Funktionalitäten zu kapseln und anderen zur Verfügung zu stellen.
- *Plug-ins* sind Klassen, die von der Klasse `PlayPlugin` abgeleitet sind. Sie können (müssen aber nicht) in Modulen enthalten sein. Man erhält damit die Möglichkeit, eigene Logik in den Request-Ablauf, aber auch in den Server-Start oder das Neuladen von Klassen einzubinden.

Auch dabei ist der Aufbau einfach, und die Anforderungen sind gering, sodass auch Anfänger grundlegende Erweiterungen effizient entwickeln können.

Test

Grundlage für die erfolgreiche agile Entwicklung sind Tests, angefangen bei Unit-Tests über funktionale Tests bis hin zu Oberflächentests. Alle drei Ebenen werden gut unterstützt. Für den eigentlichen Oberflächentest ist Selenium [KaLie07] sehr elegant integriert.

Tests zu schreiben, ist einfach, da die Funktionalitäten auch ohne Mock-Objekte gut zu isolieren sind. Benötigt man eine bestimmte Datenbankkonstellation, beispielsweise für die Oberflächentests, so hilft die Klasse `Fixtures`. Sie ermöglicht es, Tabellen zu löschen oder Daten einzuspielen. Die Daten werden dazu intuitiv in YML [YML] definiert. Ein einfaches Beispiel zum Anlegen von Rechten, Rollen und Benutzern zeigt Listing 4.

```
# Test data (Referenznamen stehen in Klammern)
Permission(example1):
  name:   'example1'
  target: 'target1'
  actions: 'action1.3, action1.2'

Permission(example2):
  name:   'example2'
  target: 'target2'
  actions: 'action2.2, action2.2'

Role(exampleRole):
  name:  'exampleRole'
  permissions: [example1, example2]

User(test):
  loginname: 'test'
  roles: [exampleRole]
  password: "Secret"
```

Listing 4: Inhalt-YML-Datei zum Anlegen von Benutzer, Rollen und Rechte

Im Allgemeinen werden die Tests über eine spezielle Weboberfläche gestartet. Durch die Reload-Fähigkeit geht dies angenehm schnell. Seit Version 1.2 von Play ist es alternativ möglich, die Tests als JUnit-Tests zu starten, beispielsweise in Eclipse oder im Continous-Integration-Server Jenkins.

Dokumentation

Gute Dokumentation ist wichtig. Bei Play findet man eine sehr gut zu bearbeitende Einführung und *The essential documentation* als Handbuch und Nachschlagewerk. Teilweise nutzen die Entwickler auch Beispielanwendungen, um die Möglichkeiten zu dokumentieren. Die Dokumentation weist allerdings noch Lücken auf. Die Play-Community arbeitet aber an dem Thema. In den letzten Monaten waren bereits deutliche Verbesserungen erkennbar. Ergänzend kann auf das Buch „Introducing the Play Framework" von Wayne Ellis [Ell11] zurückgegriffen werden, welches in gedruckter und elektronischer Form erhältlich ist.

Die Lücken in der Dokumentation sind insgesamt kein großes Problem, da Play auf einem sehr aufgeräumten und übersichtlichen Code basiert. So lassen sich viele Fragen schnell durch einen Blick in den Code klären. Alternativ wendet man sich an die hilfsbereite Community.

Erfahrungen

Im Rahmen der Einführung wird eine Blog-Anwendung entwickelt. Dabei ergeben sich viele Möglichkeiten, Speziallfälle auszuprobieren, und man bekommt schnell Lust auf mehr. Die Aussagekraft der Fehlermeldungen und die extrem schnellen Roundtrip-Zeiten machen einfach Spaß.

Fragen zu REST, insbesondere zur fehlenden Session, gehören neben Fragen zu JPA zu den Hauptanfragen an die Play-Community. Wem REST und JPA vertraut sind, der wird Play sehr schnell lernen. Nach einer kurzen Einarbeitungszeit erlebt man beim Arbeiten keine großen Überraschungen mit stundenlanger Fehlersuche. Man kommt schnell zu einem zügigen und kontinuierlichen Arbeitsfortschritt.

Wer steht hinter dem Framework?

Vor dem Einsatz eines Frameworks ist stets dessen Zukunftsfähigkeit zu bewerten. Hinter Play steckt im Wesentlichen die kleine französische Firma Zenexity, die von dem niederländischen Unternehmen Lunatech Research unterstützt wird. Die Entwicklung treiben eine Handvoll Mitarbeiter voran.

Insgesamt gibt es auch zahlreiche Beiträge seitens der Community, und es sieht so aus, als ob einige Firmen schon intern eigene Branches pflegen und nutzen. Es kann daher davon ausgegangen werden, dass die Entwicklung nicht abbricht, sollte die Firma Zenexity die Unterstützung einstellen. Dieser Fall ist zudem unwahrscheinlich, da einige größere Projekte bei Zenexity auf Play aufbauen.

Kommerzieller Support ist erhältlich, jedoch werden im Allgemeinen Bugs und echte Problemfälle auch ohne Kosten in sehr kurzer Zeit (teilweise innerhalb von Stunden) behoben. Einige Anfragen und Verbesserungen bleiben aber manchmal länger liegen.

Fazit

Play ist ein relativ junges Framework (2008 gestartet) und hat daher noch Verbesserungspotenzial. Es besitzt allerdings ein gutes Fundament, zum Beispiel beim Threadhandling oder bei der Unterstützung von Authentifizierung, beim Mail-Versand sowie beim Ansprechen von Webservices. Oberstes Designprinzip ist KISS (Keep it simple and smart), was zu einer inhärenten Stabilität führt. Die kleineren Schwächen wie die aus-

schließliche Unterstützung von Sprachkürzeln (`de`) anstelle von Sprach-/Länderkürzeln (`de_DE`) bei der Internationalisierung sind daher leicht zu verschmerzen.

Play ermöglicht eine ähnlich hohe Produktivität wie Grails oder Ruby on Rails und baut dabei auf bewährter Java-Technologie mit vollem IDE-Support auf. So ist keine neue Sprache zu lernen. Im Vergleich zu Grails begeistern die Einfachheit und Schlichtheit des Frameworks sowie der äußerst geringe Ressourcenverbrauch. Aktuell ist die Version 1.2 fertiggestellt. Version 1.3 ist schon in Arbeit, parallel wird ein stabiler 1.2.x-Zweig gepflegt.

Wer sich nicht scheut, vereinzelt kleinere Fehler in einem gut strukturierten Quellcode zu beheben, dem kann Play schon heute empfohlen werden. Alle anderen sollten Play für die Zukunft im Auge behalten.

Links und Literatur

[Bort] G. Bort, Why there is no servlets in Play, 29.4.2010, `http://iam.guillaume.bort.fr/post/558830013/why-there-is-no-servlets-in-play`

[Checks] How to write custom checks/validation for the play-framework, `http://stackoverflow.com/questions/3690461/howto-write-custom-checks-validation-for-the-play-framework`

[Ell11] W. Ellis, Introducing the Play Framework, 2011, s. z. B. `http://www.the-play-book.co.uk/books.php`

[KaLie07] M. Kain, Th. Lieder, Testen vereinfachen mit Selenium, in: JavaSPEKTRUM, 04/2007

[Netty] Netty – the Java NIO Client Server Socket Framework, `www.jboss.org/netty`

[OVal] Object VALidation Framework for Java, `http://oval.sourceforge.net/`

[Play] `http://www.playframework.org`

[Tilk08] St. Tilkov, Eine kurze Einführung in REST, in: JavaSPEKTRUM, 03/2008

[YML] V. Birk, YML – Why a Markup Language, `http://fdik.org/yml/`

Niels Stargardt ist seit vierzehn Jahren bei der PPI AG Informationstechnologie als Projektleiter und Softwarearchitekt tätig. Er ist darüber hinaus Java-Themenbeauftragter und kümmert sich in diesem Zusammenhang um verschiedene Themen im Java-Umfeld.
E-Mail: niels.stargardt@ppi.de