KaffeeKlatsch
Das Magazin rund um Software-Entwicklung

ISSN 1865-682X
08/2014

I did it my way

Sinatra – Web-Entwicklung einmal anders

von Werner Eberling

I did it my way

Sinatra – Web-Entwicklung einmal anders

von Werner Eberling

Aus der Java-Welt sind wir es gewöhnt, Web-Anwendungen auf Basis komplexer Frameworks zu entwickeln und diese in schwergewichtige Server-Prozesse zu deployen. Dass es auch anders gehen kann, zeigt u. a. die Ruby-Gemeinde mit *Ruby on Rails* oder der Web-DSL *Sinatra*. Dieser Artikel gibt einen ersten Einblick in *Sinatra* und möchte damit Lust machen, einmal in die andere Art der Web-Entwicklung hinein zu schnuppern.

Zugegeben, Web-Frameworks gibt es in der Welt der Software-Entwicklung wirklich mehr als genug. Speziell in der Java-Welt kann der geneigte Web-Entwickler hier aus dem Vollen schöpfen. Doch viele dieser Frameworks leiden an einer nicht zu leugnenden Schwergewichtigkeit, speziell wenn es um das Thema Laufzeitumgebung geht. Hier schweifen die neidischen Blicke schnell in die Welt anderer Programmiersprachen.

Mit Sinatra soll hier ein Vertreter dieser Gattung aus dem Ruby-Ökosystem vorgestellt werden. Eigentlich prädestiniert für die Entwicklung von *RESTful-Web-Services*, lassen sich damit auch einfache Web-Seiten mit fast minimalistischem Aufwand entwickeln.

Installation und Hello World

Als Ruby-basierte, domänenspezifische Sprache (DSL) sind sowohl Ruby [1] als auch der *Ruby Gem Installer* [2] Voraussetzung für die eigentliche Installation von Sinatra. Ist beides auf dem Rechner der Wahl vorhanden, so lässt sich das Framework einfach über den Aufruf *gem install sinatra* installieren.

Hat der Installer seine Arbeit vollbracht, steht der ersten Sinatra-Anwendung nichts mehr im Wege. Diese wird in Form einer Ruby-Klasse implementiert.

```
require "sinatra"
get "/hello" do
   "Hello MATHEMA"
end
```

Listing 1: hello.rb

Die Syntax macht bereits deutlich, dass Sinatra auf die Entwicklung von *RESTful-HTTP-Services* zugeschnitten ist und sich dabei gleichzeitig auf die wirklich notwendigen Angaben konzentriert. Zuerst erfolgt die Angabe der unterstützten *HTTP*-Methode und des URL-Kontexts auf den die Anwendung reagieren soll. Nach dieser sogenannten Route wird direkt das zurückzuliefernde Ergebnis beschrieben.

Um das ganze auch im Browser zu testen, muss die Anwendung noch gestartet werden. Hier liefert Sinatra (oder besser Ruby) den passenden Server gleich mit. Somit lässt sich die gerade geschriebene Anwendung mittels *ruby hello.rb* direkt starten.

```
INFO  WEBrick 1.3.1
INFO  ruby 2.0.0 (2014-02-24) [universal.x86_64-
darwin13]
== Sinatra/1.4.5 has taken the stage on 4567 for
development with backup from WEBrick
INFO  WEBrick::HTTPServer#start: pid=1765
port=4567
```

Startausgaben des WEBrick-Servers

Sobald sich der Server auf der Konsole als arbeitsbereit meldet, kann die Anwendung unter *http://localhost:4567/hello* getestet werden.

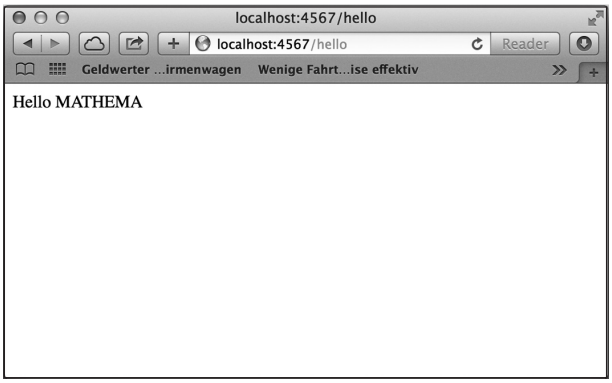

Abbildung 1: hello.rb im Browser

Ein RESTful-Service

Sollen verschiedene URL-Methoden und/oder -Pfade unterstützt werden, so kann dies durch die Definition

unterschiedlicher Routen in der selben Anwendung erreicht werden. Dabei ist es auch möglich komplexe Pfade mit Hilfe von regulären Ausdrücken zu definieren oder mit Parametern auf Eingaben des Clients zu reagieren.

So lässt sich auf einfache Art ein RESTful-Web-Service implementieren. Dabei können zusätzlich benötigte Hilfsmethoden entweder innerhalb der selben Klasse als *top-level helpers*-Methode oder innerhalb von unterstützenden Ruby-Modulen implementiert werden.

```
get "/kunden" do
  @all = findAll()
  #Alle Kunden zurueckliefern ...
end

get "/kunden/:id" do
  #Pfad-Parameter
  id = params[:id].to_i
  @customer = find(id)
  #Kunden zurueckliefern ...
end

post "/kunden/:id" do
  #Pfad-Parameter
  id = params[:id]
  #Query-Parameter
  name = params['name']
  firstname = params['firstname']
  #Kunden erzeugen...
end

# Weitere REST-Methoden...

helpers do
 # Hilfsmethoden und Persistenz...
end
```

Listing 2: kunde.rb

Das Frontend

Grundsätzlich zielt Sinatra auf die Entwicklung von RESTful-HTTP-Services ab. Um nun eine vollständige Web-Anwendung zu erhalten, muss noch ein *HTML-Frontend* vor die so entstandenen Services gesetzt werden. Eine sich hier anbietende Kombination wäre beispielsweise eine Client-seitige Web-Anwendung auf Basis von *Angular.js* [3], die zur Abfrage/Verarbeitung von Daten auf das *Sinatra-Backend* zugreift. Da hier ein zusätzliches Framework zum Einsatz kommt, soll an dieser Stelle auf eine tiefere Betrachtung dieses Ansatzes verzichtet werden. Dem geneigten Leser seien in diesem Zusammenhang die zahlreichen Sinatra-Beispiele [4] im Internet ans Herz gelegt.

HTML-Templating

Es muss aber nicht zwingend eine Client-seitige Anwendung sein. Sinatra bietet ebenfalls die Möglichkeit beliebige *Templating-Frameworks* einzubinden [5], um komplett aufbereitetes HTML zurückzuliefern. Ohne weitere Hilfsmittel bietet Sinatra die Möglichkeit, mittels *EmbeddedRuby* (ERB) HTML-Templates mit eingebettetem Ruby-Code zu hinterlegen und den Aufbau der Rückgabe diesen Templates zu überlassen. Dabei hat das Template Zugriff auf Variablen, die innerhalb der Sinatra-Methoden entsprechend definiert wurden (*@all*).

```
get "/kunden" do
  @all = findAll()
  # Einbindung des erb Templates
  # ./views/hello.erb
  erb:all
end
#...
```

Listing 3: Ansprechen eines ERB-Templates

```
<html>
<body>
 <h1>Alle Kunden</h1>
 <ul>
  <% @all.each do | customer | %>
  <li>
   <a href="/kunden/<%= customer.id %>"><%=
customer.name %></a>
  </li>
  <% end %>
 </ul>
</body>
</html>
```

Listing 4: ERB-Template mit Variablen-Zugriff

Den Java-affinen Leser mag dieser Ansatz an *Java ServerPages* (JSP) erinnern und wie bei JSPs ist auch hier darauf zu achten, dass das HTML-Template lediglich *View*-Informationen enthält und alles weitere an Logik im Rahmen der Sinatra- bzw. Ruby-Implementierung von statten geht. Ist eine deutlichere Trennung zwischen Darstellungslogik und Verarbeitungslogik gewünscht, stellt die bereits erwähnte Trennung in client-seitige Frontend-Anwendung und RESTful-Service-Backend die bessere Alternative dar.

Ergebnis des Templatings im Browser

Resümee

Alles in allem bietet Sinatra ein praktisches Toolset zur Implementierung sowohl von RESTful-HTTP-Services wie auch von einfachen Web-Anwendungen. In beiden Fällen überzeugt die DSL durch ihre Einfachheit. Gleichzeitig bietet sie über die Kombinationsmöglichkeit mit anderen Frameworks (speziell im Bereich der HTML-Generierung), sowie das sie umgebende Ruby-Ökosystem ein Potential, das deutlich über einfache CRUD[1]-Anwendungen hinaus geht.

Im Rahmen dieses Artikels konnte nur an der Oberfläche von Sinatra gekratzt werden. Dem tiefer interessierten Leser sei hier die Sinatra-Homepage [6] zur Lektüre empfohlen. Und vielleicht bietet sich ja bald einmal die Gelegenheit eine einfache, schlanke Web-Applikation mal nicht mit den großen Standard-Frameworks, sondern ausnahmsweise auf einem anderen, leichtgewichtigeren Wege zu implementieren – „it's up to you".

Referenzen

[1] Ruby, https://www.ruby-lang.org

[2] Ruby Gems *Guides*, http://guides.rubygems.org

[3] Angular.js, https://angularjs.org

[4] Sinatra *Beispiele für Sinatra-Anwendungen* http://www.sinatrarb.com/wild.html

[5] Sinatra *Getting Started*, http://www.sinatrarb.com/intro.html

[6] Sinatra, http://www.sinatrarb.com

Kurzbiografie

Werner Eberling ist Principal Consultant und Technical Lead bei der MATHEMA Software GmbH in Erlangen. Seit 1999 beschäftigt er sich mit verteilten Systemen, sein Fokus liegt dabei auf CORBA und der JEE. Neben seiner Projekttätigkeit hält er Technologie-Trainings und ist als Sprecher auf nationalen und internationalen Konferenzen anzutreffen. Sein aktuelles Steckenpferd ist die Anwendungsentwicklung für mobile Endgeräte bzw. deren Anbindung an Enterprise Systeme. Er ist u. a. Autor des Buches *Enterprise JavaBeans 3.1. Das EJB-Praxisbuch für Ein- und Umsteiger*, erschienen im Hanser Verlag.

[1] CRUD: Create, Read, Update, Delete.