Webanwendungen mit Ruby und Rails

Wie auf Schienen

Die meisten Webbibliotheken machen 90 Prozent der Aufgaben einfach, den Rest schlichtweg unmöglich. Das in Ruby geschriebene Open-Source-Framework Rails bewahrt sich die notwendige Flexibilität, um auch die restlichen zehn Prozent auf die Bahn zu bringen. Armin Röhrl, Stefan Schmiedl

Webentwicklung ist oft von wüsten Hacks beherrscht, in denen Programmcode und HTML bunt gemischt sind. Dabei gibt es mittlerweile Ansätze, die auch in diesem Bereich strukturierte Entwicklung ermöglichen. So implementiert etwa Rails [1] in Ruby ein klassisches Model-View-Controller-Framework (MVC), wie es beispielsweise die Smalltalk-Welt schon lange verwendet. Es isoliert die eigentliche Datenverarbeitung im Modell weitgehend vom GUI-bezogenen Manipulationscode (Controller) und dem reinen Darstellungscode (View), siehe Abbildung 1.

Das Modell kapselt die Daten und die Methoden, die sie bearbeiteten. Das kann vom Arbeiten mit einer Datenbanktabelle bis zu einer Modellierung des Steuerungsprozesses eines Großkonzerns reichen. Im View wird das Modell ganz oder teilweise visualisiert. Die Controller zeichnen dafür verantwortlich, die Zustandsänderungen am Modell auszulösen, indem sie zum Beispiel auf Benutzereingaben reagieren.

Durch diese Aufteilung wird die Applikationslogik von der äußeren Form der Darstellung und Art der Benutzerführung unabhängig. Einige Pakete unterstützen zwar angeblich MVC, was sich bei näherer Betrachtung aber oft nur als Drittelwahrheit entpuppt, siehe Kasten „Rails und Java-Struts …“.

Software aus der Praxis

Rails bringt fürs Modell das Paket Active Record mit, das objektrelationales Mapping realisiert. Das Modul Action Pack teilt Webrequests in Controller- und View-spezifische Teile auf. Damit ist Rails bestens geeignet für die Entwicklung datenbankbasierter Webapplikationen wie der Projektmanagement-Software Basecamp [2], die mit Rails programmiert wurde, siehe Kasten „Interview mit dem Rails-Autor“. Diese Geschichte aus der Praxis merkt man Rails beim Arbeiten an, da es viele Probleme vermeidet, die bei Frameworks vom grünen Tisch unschön auffallen.

Rails baut auf das DRY-Konzept (Don’t repeat yourself – Wiederhol dich nicht, auch bekannt als Once and only once). Denn schreiben Entwickler den mehr oder weniger gleichen Code innerhalb eines Programms immer wieder, ist das ein Zeichen schlechten Designs.

Konfigurationsdateien ersetzt Rails weitgehend durch Reflexion und Laufzeit-Erweiterungen, weswegen die sonst übliche XML-Konfigurationsorgie entfällt. Ein weiterer Effekt ist, dass die Auswirkungen von Änderungen sofort sichtbar werden, da zeitaufwändigen Kompiliervorgänge wegfallen.

Rails verfügt auch über eingebaute Funktionalität für allgemeine und Unit-Tests, zur Dokumentation dient RubyDoc. Das größte Plus ist, dass alle drei Bestandteile des MVC-Entwurfsmusters in Ruby geschrieben sind.

Wenn sich Entwickler an die Rails-Konventionen halten (Zuordnung zwischen Datenbank- und Objektstrukturen), bringt das so genannte Scaffolding (Gerüst bauen) schnell ein Projekt online. Auf Knopfdruck stehen die nötigen Funktionen zur Verfügung, beispielsweise das Hinzufügen, Bearbeiten und Löschen von Objekten.

Objekte in die Datenbank

Die Beispielanwendung orientiert sich am Tutorial der Ruby-Homepage, um eine einfache Webapplikation im Ordner »/var/www/railsdemo« zu erstellen. Zunächst legt Rails das Grundgerüst von Dateien an:

```
rails /var/www/railsdemo
```

Dieser Befehl überschreibt eine bereits existierende Rails-Anwendung in dem

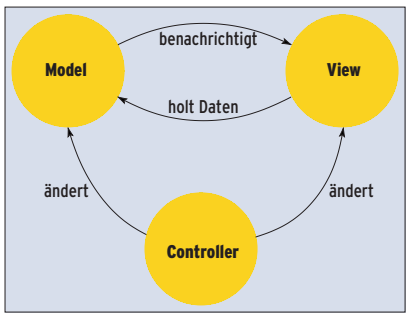

Abbildung 1: Schema der Model-View-Controller-Idee. Die Datenflüsse sind dabei nicht für alle Anwendungen gleich, sondern können auch etwas anders aussehen als hier.

Rails und Java-Struts im Vergleich

Obwohl das Framework Struts [3] hauptsächlich für Controller vorgesehen ist, ziehen es Java-Fans gerne als Paradebeispiel für MVC-Programmierung heran. Rails dagegen bietet etwas für alle drei Begriffe: Model, View und Controller.

In Struts sind alle Aktionen in einer XML-Datei zu notieren (und damit auch indirekt die Rückgabewerte). Rails erledigt das selbstständig durch Reflexion. Daneben versucht der ActionController von Rails lesbare URLs wie »/customers/show/154« zu erzeugen. Alle Aktionen in Struts müssen Klassen sein, bei Rails genügen Methoden, die in der erwähnten Basecamp-Applikation durchschnittlich fünf Zeilen lang sind. Der »ActionController« in Rails unterstützt Layouts direkt, während Struts dafür weitere XML-Dateien braucht.

Der »ActionController« ist überhaupt hilfreich dafür, mit Scaffolding eine Modellklasse online zu bringen. Er unterstützt außerdem Filter und Interzeptoren, für Struts gibt es in dieser Richtung lediglich ein Forschungsprojekt namens SAIF [4]. Wer umsteigen will, findet in Tabelle 1 die wichtigen Java-Bibliotheken entsprechenden Ruby-Projekte.

genannten Verzeichnis, will also mit Vorsicht verwendet sein. Eine MySQL-Datenbank nimmt die Beispieltabelle auf, siehe Listing 1.

Den Zugriff auf die Datenbank legt die Yaml-Datei »config/database.yml« fest, die in einem sprachunabhängigen Format Daten serialisiert:

```
production:
  adapter: mysql
  database: rails_production
  host: localhost
  username: root
  password:
```

Der Eintrag für die Testdatenbank sieht analog aus. Im wirklichen Leben sollte man allerdings weder als Administrator noch ohne Kennwort arbeiten.

Das Skript »new_controller« erzeugt einen neuen Controller, dessen Name mit einem Großbuchstaben beginnen soll. Nach dem Controller-Namen folgen die gewünschten Views:

```
./script/new_controller Friends list➥
   display new edit
```

Der View-Name »process« ist dabei zu vermeiden, da Rails ihn für interne Zwecke verwendet. Auf ähnliche Weise entsteht ein neues Modell:

```
./script/new_model Person
```

In der Regel ermittelt Rails die zugrunde liegende Datenbanktabelle selbst. Bei Bedarf akzeptiert das Skript den Tabellennamen hinter dem Modell:

```
./script/new_model Person people
```

Um die Konfiguration der Rails-Anwendung zu testen, führt der Programmierer im Applikationsverzeichnis »rake« aus, das Ruby-Make. Dabei können je nach Systemkonfiguration verschiedene Probleme auftreten:

- Wenn sich »mysql.sock« an einer unüblichen Position befindet, kann man entweder »mysql.rb« anpassen (in einem Verzeichnis der Art »/usr/lib/ruby/gems/1.8/gems/active-record-0.9.5/lib/active_record/vendor«) oder die Umgebungsvariable »MYSQL_UNIX_PORT« setzen.
- Die Zugangsberechtigungen für die Datenbank reichen möglicherweise nicht aus.
- Die Skripte erwarten als Ruby-Pfad »/usr/local/bin/ruby«. Befindet es sich an einem anderen Ort, legt man entweder einen Symlink an oder ersetzt den Pfadnamen in den Skripten.

Installation

Für Benutzer des Mac OS X (ab 10.3) gibt es zwei Videos [5], die den Installationsprozess vorführen und auch für Linux-User sehenswert sind. Sehr bequem ist die Installation über Rubygems [6] (mindestens Version 0.7), einem Paketmanagement-Tool für Ruby:

```
gem install rails
```

Wenn eine neue Version von Action Pack oder Active Record herauskommt, reicht für die Aktualisierung ein einfaches »gem update« aus. Zu beachten ist in jedem Fall, dass für das Backend die Datenbankanbindungen vorhanden sind, wobei MySQL und PostgreSQL in Frage kommen. Für die Installation unter Debian sind wegen der fein unterteilten Pakete viele Voraussetzungen zu erfüllen (siehe auch [7]): irb1.8, libbigdecimal-ruby1.8, libdbm-ruby1.8, libdl-ruby1.8, libdrb-ruby1.8, liberb-ruby1.8, libgdbm-ruby1.8, libiconv-ruby1.8, libopenssl-ruby1.8, libpty-ruby1.8, libracc-runtime-ruby1.8, libreadline-ruby1.8, librexml-ruby1.8, libruby1.8, libruby1.8-dbg, libsdbm-ruby1.8, libsoap-ruby1.8, libstrscan-ruby1.8, libsyslog-ruby1.8, libtest-unit-ruby1.8, libweb-rick-ruby1.8, libxmlrpc-ruby1.8, libyaml-ruby1.8, libzlib-ruby1.8, rdoc1., ri1.8 ruby1.8, ruby1.8-dev, ruby1.8-elisp und zu guter Letzt ruby1.8-examples.

Apache einstellen

Falls noch nicht geschehen, ist das Rewrite-Modul für Apache zu aktivieren:

```
# a2enmod
Which module would you like to enable?
Your choices are: ... rewrite ...
Module name? rewrite
Module rewrite installed; run ➥
/etc/init.d/apache2 force-reload to enable.
```

Um eine Rails-Applikation in »/var/www/myapp« laufen zu lassen, fehlt in der Apache-Konfigurationsdatei ein Eintrag für den virtuellen Host, der ungefähr so aussieht:

```
<VirtualHost *:80>
    ServerName rails
    DocumentRoot /var/www/railsdemo/public
    ErrorLog /var/www/railsdemo/log/➥
             apache.log
    <Directory /var/www/railsdemo/public/>
      Options ExecCGI FollowSymLinks
            AllowOverride all
      Order allow,deny
      Allow from all
    </Directory>
</VirtualHost>
```

Weitere Einstellungen finden sich in der Htaccess-Datei der Webapplikation, im Beispiel »/var/www/railsdemo/public/.htaccess«. Nun muss Root noch Apache aktualisieren und die Konfiguration damit neu laden:

```
/etc/init.d/apache2 force-reload
```

Um die Webapplikation fertig zu stellen, ist noch »app/views/friends/display.rhtml« entsprechend auszufüllen:

```
<html>
<body>
<h1>Friends#display</h1>
<p>Zeigt einen Personen-Eintrag an</p>
<p>
<%= @person.name %><br />
<%= @person.street1 %><br />
<%= @person.street2 %><br />
<%= @person.city %><br />
<%= @person.state %><br />
<%= @person.zip %><br />
</p>
</body>
</html>
```

Das sieht ähnlich aus wie in Erb, der in HTML eingebetteten Ruby-Variante. Die HTML-Seite zeigt die Daten an, die das Modell zur Verfügung stellt. Das bezeichnete Objekt »@person« muss der Controller definieren und erzeugen. Dies geschieht in der Datei »app/controllers/friends_controller.rb«, die schon die Zugriffsmethoden enthält. Das Modell selbst bindet man mit der Anweisung »require« ein:

```
require 'person'
```

Nun kann die Webapplikation über die Active-Record-Klasse auf die Datenbank zugreifen. Wird die Methode »display« durch folgenden Code ersetzt, holt sie den ersten Eintrag aus der Tabelle und schreibt ihn in die Instanzvariable »@person« der View-Klasse:

```
def display
  @person = Person.find(1)
end
```

Auch die generierte Definition der Model-Klasse in »app/models/person.rb« ist einen Blick wert:

```
require 'active_record'
class Person < ActiveRecord::Base
end
```

Die Oberklasse »ActiveRecord::Base« übernimmt die meiste Arbeit, indem sie einfache Zugriffsmethoden auf die Datenbank definiert. Der Tabellenname ergibt sich dabei als die Pluralform des Klassennamens, in englischer Sprache natürlich. Das ist ganz ernst zu nehmen und bezieht sich auch auf unregelmäßige Pluralformen, zum Beispiel Men zur Einzahl Man oder eben People zu Person! Die Klassendokumentation geht ausführlich darauf ein [8].
Nun ist unter »http://localhost/rails/friends/display« das Ergebnis der Bemühungen zu sehen. Diese URL enthält sowohl den Controller (»friends«) als auch die Action (»display«).

Verknüpfte Tabellen

Auch für die Verknüpfung von Tabellen bietet Rails eine Lösung. Als Testobjekt dient eine zweite Tabelle mit Telefonnummern, siehe Listing 2.
Rails verwendet als Kennzeichnung für einen Fremdschlüssel das Suffix »_id« wie zum Beispiel in »person_id«. Wie schon erwähnt nimmt der Tabellenname

Interview mit dem Rails-Entwickler David Heinemeier Hansson

Linux Magazin: Wieso haben Sie Rails überhaupt entwickelt?
Heinemeier Hansson: Ich habe einige Jahre in PHP programmiert und Java-Projekte fürs Web beobachtet. Ich war von PHP wegen der Unmittelbarkeit der Reaktion begeistert und inspiriert von den Java-Frameworks. Also wollte ich weder die kurzen Feedback-Zyklen von PHP aufgeben noch unter den Problemen großer objektorientierter Systeme leiden.
Als ich heftige Probleme mit PHP hatte, kurz nach einem ernüchternden Kontakt mit Java (ein halbes Jahr J2EE) las ich, dass Martin Fowler und die „Pragmatischen Programmierer" von einer Nischensprache ganz begeistert seien. Das war natürlich Ruby und nachdem ich die Anfangsschwierigkeiten überwunden hatte, war es nur noch Liebe.
Es war für mich ganz selbstverständlich, die Java-Framework-Inspirationen auf Ruby zu übertragen. Musste es auch sein, denn wir erstellten die Web-basierte Projektsoftware Basecamp in nur zwei Mannmonaten.

Linux Magazin: Was ist denn so toll an Rails?
Heinemeier Hansson: Die saubere Unmittelbarkeit. Man kann unglaublich schnell Applikationen erstellen, die nicht in ein Codechaos ausarten, und diese Applikationen dann auch behalten. Eine Orientierung an bewährten Mustern wie Test-getriebener Entwicklung zusammen mit der MVC-Dreiteilung hilft ungemein.

Linux Magazin: Was ist die größte Schwäche von Rails?
Heinemeier Hansson: Es kann schwer sein, künftige Kunden (oder den Manager) davon zu überzeugen, dass man in einer relativ unbekannten Sprache produktiv sein kann, wenn sie immer nur von Java und C# hören.

Linux Magazin: Wie sieht die Zukunft von Rails wahrscheinlich aus?
Heinemeier Hansson: Hoffentlich gibt es noch in 2004 die Release 1.0. Ich denke, 2005 könnte wirklich das Ruby-Jahr werden, und hoffe, dass die Rails-Releases dazu beitragen. Abgesehen davon habe ich sehr bescheidene Ziele bei der weiteren Entwicklung des Frameworks. Rails ist im Moment ungefähr 2500 Codezeilen groß und ich möchte nicht, dass es sehr viel größer wird.

Linux Magazin: Kann man mit einer Software wie Basecamp eigentlich Geld verdienen?
Heinemeier Hansson: Basecamp ist ein Werkzeug für Web-basiertes Projektmanagement, das aus Weblogs, Meilensteinen und Todo-Listen besteht. Es ist eine sehr einfache Anwendung, die einen großen Einfluss darauf hat, wie viele Menschen ihre Projekte managen (und dabei mehr als nur E-Mail einsetzen, was sicher das am meisten genutzte Projektmanagement-Werkzeug ist). Es ist eine gehostete Anwendung, für die Firmen zwischen 19 und 59 Dollar monatlich bezahlen. Es gibt sogar ein zeitlich unbefristetes, kostenloses Projekt zum Ausprobieren.

Linux Magazin: Und was soll aus Basecamp werden?
Heinemeier Hansson: Wir verbessern es ständig. Wir haben gerade einige neue Features aufgesetzt, um Verantwortlichkeiten in einem Projekt besser zu verwalten. Ich glaube, dass wir erst einen winzigen Anteil eines riesigen Marktes erreicht haben. Projektmanagement ist überall. Neben den offensichtlichen Businessanwendungen wird Basecamp überall eingesetzt: von Renovierungsplänen bis zu Hochzeiten. Basecamp passt zu so verschiedenen Projekten, gerade weil es so einfach ist. Projektmanagement ist kein eigenes Projekt.

Linux Magazin: Was wird aus Web-basiertem Programmieren?
Heinemeier Hansson: Ich hoffe, es erholt sich von der Komplexität, die ihm von J2EE, .NET und anderen Schwergewichten aufgedrückt wurde. Dynamische Sprachen wie Ruby mit ihren Frameworks wie Rails werden zum Renner werden. Kunden und Entwickler werden erkennen, dass die meisten Webapplikationen nicht die Feuerkraft eines imperialen Sternzerstörers brauchen, sondern vielmehr die Wendigkeit eines X-Wing-Jägers – ja, ich habe gerade Star Wars auf DVD erlebt.

Tabelle 1: Java- und Ruby-Pakete

Anwendung	Java	Ruby
Applikations-GUI	Swing	TK, QT, Fox
Web-GUI	JSP, XSLT, XMLC ...	Iowa, Rails
Persistente Domain-Objekte und Transaktions-Management	Entity EJBs, JDO, per Hand erzeugte DAOs ...	Kansas
RPC-Mechanismus (Remote Procedure Call) zur Kommunikation zwischen Frontend und Backend	Session EJBs, Servlets	DRb
Infrastruktur für asynchrone Kommunikation	Message-Driven EJBs	DRb/Rinda/Tupleserver

eine Mehrzahlform an, während das Modell wie oben im Singular steht:

```
./script/new_model Phone
```

Die vorher generierte Datei »person.rb« erfordert zwei Änderungen:

```
require 'active_record'
require 'phone'

class Person < ActiveRecord::Base
    has_many :phones
end
```

Der Aufruf »has_many :phones« legt die 1:n-Beziehung fest, dass also eine Person mehrere Telefonnummern haben kann. Die folgenden Zeilen, zu »display.html« hinzugefügt, passen die Darstellung an:

```
<% for phone in @person.phones %>
<%= phone.phone %><br/>
<% end %>
```

Existiert das Modell, lassen sich mit dem Scaffolding-Verfahren in Rails schnell die Anzeige und Bearbeitung der Daten realisieren. Die erzeugten Methoden verfeinert der Webentwickler dann nach und nach.

Gerüstbau automatisch

Das folgende Beispiel geht von einem frischen Rails-Applikationsverzeichnis aus. Nur ein Controller und ein Modell sind erst mal anzulegen (»./script/new_controller Friend« und »./script_new_model Person«, siehe oben). Entscheidend ist, dass die Aktion »list« noch nicht existiert. Die bestehende Datenbank nutzt Rails weiter. Eine Änderung am Friends-Controller genügt:

```
scaffold :person
```

Diese Ergänzung erzeugt neue Methoden, deren Standardverhalten in der Dokumentation [9] beschrieben ist: »list«, »show«, »destroy«, »new«, »create«, »edit« und »update«.
Scaffolding funktioniert nun erst mit dem Modell »Person«, es fehlen noch die Telefonnummern. Um bei der Definition der Telefon-Aktionen Namenskollisionen zu vermeiden, hängt Rails den Modellnamen an die Aktionsbezeichnung, wenn die passenden Flags gesetzt sind:

```
scaffold :person, :suffix => true
scaffold :phone, :suffix => true
```

Damit stehen beispielsweise die folgenden URLs zur Verfügung:

```
http://rails/friends/list_person
http://rails/friends/list_phone
```

Ohne großen Aufwand ist so eine strukturierte Webanwendung entstanden, die ihre Objekte in einer relationalen Datenbank pflegt. Ruby setzt das MVC-Modell sauber um und erzeugt sogar selbst den grundlegenden Code dafür.

Logout

Rails erleichtert die Entwicklung von Standardanwendungen für alle, die sich an die vorgegebenen Regeln halten. Als Einstieg in die Programmierung von Webapplikationen ist das System sehr gelungen. Wer noch weitere Starthilfe zu Ruby braucht, findet sie im Linux-Magazin-Sonderheft [10] oder in der deutschen Version des Online-Ruby-Buchs unter [11]. *(ofr)* ■

Infos

[1] Ruby on Rails: [http://www.rubyonrails.org]
[2] Basecamp: [http://www.basecamphq.com]
[3] Apache-Struts: [http://struts.apache.org]
[4] Struts Action Invocation Framework (SAIF): [http://struts.sourceforge.net/saif/]
[5] Tutorial-Videos: [http://www.rubyonrails.org/show/HomePage]
[6] Rubygems: [http://rubygems.rubyforge.org/wiki/wiki.pl]
[7] Rails für Debian: [http://www.rubyonrails.org/show/RailsOnDebian]
[8] Regeln zur Namensgebung mit Pluralbildung: [http://api.rubyonrails.org/classes/ActiveRecord/Base.html]
[9] Scaffolding-Methoden: [http://api.rubyonrails.org/classes/ActionController/Scaffolding/ClassMethods.html]
[10] Linux-Magazin-Sonderheft „Skriptsprachen"
[11] Projekt Kansas: [http://home-web.de/juergen.katins/ruby/buch]
[12] Weblog des Rails-Autors: [http://loudthinking.com]

Die Autoren

Armin Röhrl und Stefan Schmiedl leiten die agile Softwareschmiede Approximity GmbH, setzen seit fünf Jahren Ruby und andere hyperproduktive Lösungen bei Projekten ein und haben immer noch Spaß daran. Produktivität ist ihnen wichtiger als der aktuelle Hype.

Listing 1: SQL für Personen

```
01 CREATE DATABASE rails_production;
02 USE rails_production;
03
04 CREATE TABLE people (
05   id int(10) unsigned NOT NULL auto_increment,
06   name varchar(50) NOT NULL default '',
07   street1 varchar(70) NOT NULL default '',
08   street2 varchar(70) NOT NULL default '',
09   city varchar(70) NOT NULL default '',
10   state char(2) NOT NULL default '',
11   zip varchar(10) NOT NULL default '',
12   PRIMARY KEY (id),
13   KEY name (name)
14 ) TYPE=MyISAM AUTO_INCREMENT=2 ;
15
16 INSERT INTO people VALUES (1, 'Superman',
   '123 Somewhere', '', 'Smallville', 'KS', '123456')
```

Listing 2: SQL für Telefonnummern

```
01 USE rails_production;
02 CREATE TABLE phones (
03   id int(10) unsigned NOT NULL auto_increment,
04   person_id int(10) unsigned NOT NULL default '0',
05   phone varchar(15) NOT NULL default '',
06   PRIMARY KEY  (id),
07   KEY people_id (person_id),
08   KEY phone (phone)
09 ) TYPE=MyISAM AUTO_INCREMENT=3 ;
10
11 INSERT INTO phones VALUES (1, 1, '1234567890');
12 INSERT INTO phones VALUES (2, 1, '1122334455');
```