mitp

Jens-Christian Fischer

Professionelle Webentwicklung mit Ruby on Rails 2

Das Praxisbuch für Entwickler

- Test und Behaviour Driven Development mit RSpec und User Stories
- REST, AJAX mit jQuery, HAML und SASS
- Suchfunktionen integrieren, Deployment mit Capistrano

Teil I

Rails praktisch anwenden

In diesem Teil:

- **Kapitel 1**
  Eine Anwendung entsteht 27
- **Kapitel 2**
  Einführung ins Testen 63
- **Kapitel 3**
  Weitere einfache Funktionen der Anwendung . . . 83
- **Kapitel 4**
  ActiveRecord: Mit Modellen arbeiten 117
- **Kapitel 5**
  Relationen zwischen Objekten herstellen 145
- **Kapitel 6**
  Navigation und Layout 173

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Eine Anwendung entsteht

In diesem Buch entsteht eine einfache Rails-Anwendung: Ein Referenzhandbuch, das in verschiedenen Kategorien Text bereithält. Sie werden diese Anwendung von Kapitel zu Kapitel erweitern und neue Fähigkeiten hinzufügen. Am Ende haben Sie eine Anwendung, in der Sie strukturierte Informationen im Internet veröffentlichen können. Leser können Kommentare und Ergänzungen anbringen. Unten sehen Sie einen Screenshot der fertigen Anwendung. Sie finden sie auch unter der Adresse `http://www.rails-praxis.de` im Internet. Gleichzeitig ist diese Anwendung auch eine Rails-Referenz, die in Kurzform den Umgang mit Rails dokumentiert. Im Verlauf des Buchs werde ich immer wieder darauf verweisen.

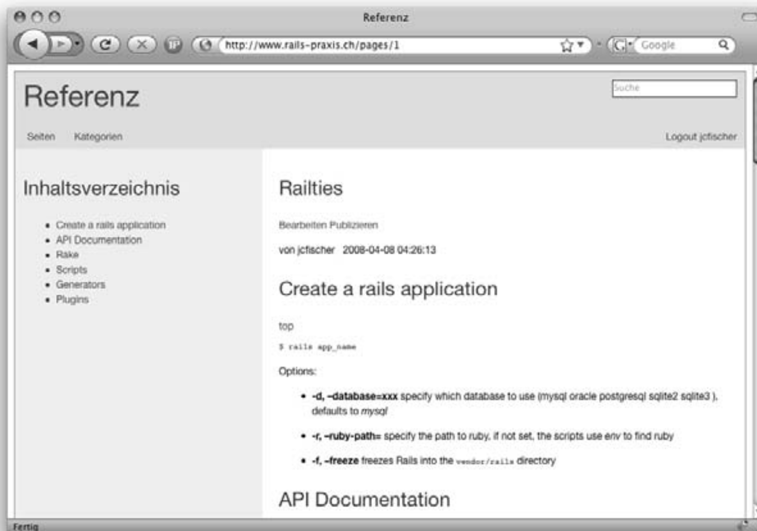

Abb. 1.1: Die fertige Referenz im Internet

1.1 Anforderungen

Das Referenzhandbuch besteht anfangs aus ganz einfachen Seiten, die über einen Titel und Inhalt verfügen. Jeder Anwender kann neue Seiten anlegen und bestehende Seiten verändern.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

So viel mal für den Anfang. Sie werden mit diesem Handbuch keinem gestandenen System ernsthaft Konkurrenz machen, dafür haben Sie in der nächsten Stunde eine funktionierende Website – und das ist doch auch etwas.

1.2 Voraussetzungen

Damit es im nächsten Abschnitt überhaupt weitergeht, müssen sie Ruby, Rails und eine Datenbank installiert haben. Wie das für Ihr jeweiliges Betriebssystem geht, lesen Sie im Anhang A.

1.3 Achtung, fertig, Rails!

Am Anfang steht die Kommandozeile. Trotz aller bunten Fenster, der Maus und ähnlichen modernen Hilfsmitteln, ist das eine Fenster mit dem Terminal (oder der Kommandozeile in Windows) immer noch von Bedeutung – vor allem, wenn Sie entwickeln. Öffnen Sie also ein Terminal, wechseln Sie in ein Verzeichnis, in dem Sie arbeiten können und geben Sie den `rails`-Befehl ein:

```
$ cd /Users/jcf/dev/work
$ rails referenz
```

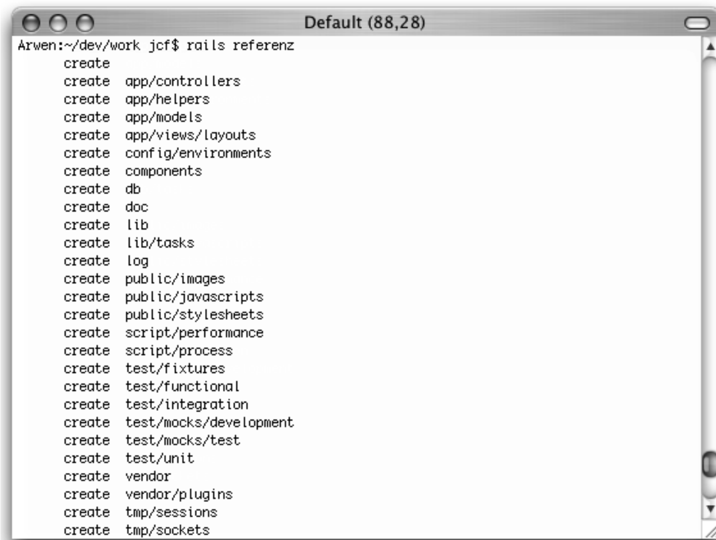

Abb. 1.2: Eine neue Rails-Anwendung entsteht

Sie sehen, dass der `rails`-Befehl erst das Verzeichnis für die Anwendung `referenz` erzeugt, und darin dann eine ganze Reihe von Unterverzeichnissen. Sollten

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Sie eine Fehlermeldung bekommen, dass der Befehl `rails` unbekannt ist, dann müssen Sie nochmals in den Anhang wechseln und die Installation von Rails (und gegebenenfalls Ruby) überprüfen.

Die erzeugte Struktur ist fest vorgegeben und das hat eine ganze Reihe von Vorteilen:

- Jede Rails-Anwendung ist genau gleich strukturiert – haben Sie eine gesehen, finden Sie sich in jeder weiteren sofort zurecht.
- Sie müssen sich keine Gedanken machen, wie Sie anfangen sollen. Das Gegenstück zum weißen Blatt Papier desjenigen, der ein Buch beginnt; der leere Bildschirm bleibt Ihnen erspart.

Schauen Sie sich diese Struktur einmal etwas genauer an:

Pfad	Beschreibung
`app`	Das Applikations-Verzeichnis. Den größten Teil Ihrer Anwendung finden Sie hier bzw. in den Unterverzeichnissen.
`app/controllers`	Hier liegen die Controller. Sie steuern Ihre Applikation.
`app/helpers`	Die Helper sind kleine Hilfsfunktionen, die Sie vor allem für die verschiedenen Ansichten brauchen.
`app/models`	Die Models sind das Herzstück der Anwendung. Sie sind die Schnittstelle zur Datenbank und beinhalten die so genannte Businesslogik.
`app/views`	Die Views dienen dem Anzeigen von Daten.
`config`	Sämtliche Konfigurationsdateien (so viele sind das jedoch gar nicht)
`db`	Definition der Datenbankstruktur und der Datenbankmigrationen
`doc`	Die Dokumentation
`lib`	Ruby-Bibliotheken für zusätzliche Funktionen
`log`	Die Logdateien
`public`	Das öffentliche Webverzeichnis. Hier liegen alle statischen Dateien (z.B. Bilder, CSS-Stylesheets und JavaScripts)
`script`	Verschiedene Scripts. Server starten, Generieren von Models, Controllern und anderen Komponenten, interaktive Konsole etc.
`test`	Die verschiedenen Tests (Unit, Functional und Integration sowie die Testdaten)
`tmp`	Temporäre Dateien
`vendor`	Von externen Quellen stammende Teile der Anwendung. z.B. Plugins

Tabelle 1.1: Standard-Verzeichnisstruktur

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Wenn Sie ganz neu zu Rails gestoßen sind, dann schauen Sie sich einmal in dieser Verzeichnisstruktur um. Sie finden im Prinzip eine fertige Rails-Anwendung – nur tut diese noch nicht so schrecklich viel.

1.4 Rails das erste mal starten

Sie können schon eine ganze Menge anstellen, obwohl Sie noch nicht so viel Schreibarbeit vorgenommen haben. Wechseln Sie in das neu erstellte `referenz`-Verzeichnis und geben Sie dort `script/about` ein:

```
$ ruby script/about
About your application's environment
Ruby version              1.8.6 (universal-darwin9.0)
RubyGems version          0.9.5
Rails version             2.0.2
Active Record version     2.0.2
Action Pack version       2.0.2
Active Resource version   2.0.2
Action Mailer version     2.0.2
Active Support version    2.0.2
Application root          /Users/jcf/dev/work/referenz
Environment               development
Database adapter          mysql
Database schema version   1
```

Rails gibt Ihnen eine Auflistung aller beteiligten Programme – von der Ruby-Version über die Versionsnummern der verschiedenen Rails-Komponenten etc. Das ist immer dann ganz praktisch, wenn Sie sich versichern wollen, dass Sie z.B. tatsächlich die Rails-Version verwenden, die Sie meinen zu verwenden.

Das Script-Verzeichnis bietet aber noch mehr:

```
Arwen:~/dev/referenz jcf$ script/server
=> Booting Mongrel (use 'script/server webrick' to force WEBrick)
=> Rails application starting on http://0.0.0.0:3000
[...]
** Mongrel available at 0.0.0.0:3000
** Use CTRL-C to stop.
```


© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Mit dem Befehl `script/server` starten Sie den mit Rails mitgelieferten Webserver. Es wird Zeit, dass Sie sich ihre erste Anwendung auch einmal anschauen. Öffnen Sie Ihren Webbrowser und geben Sie `http://127.0.0.1:3000` als URL ein. Wenn alles funktioniert hat, sehen Sie folgende Startseite:

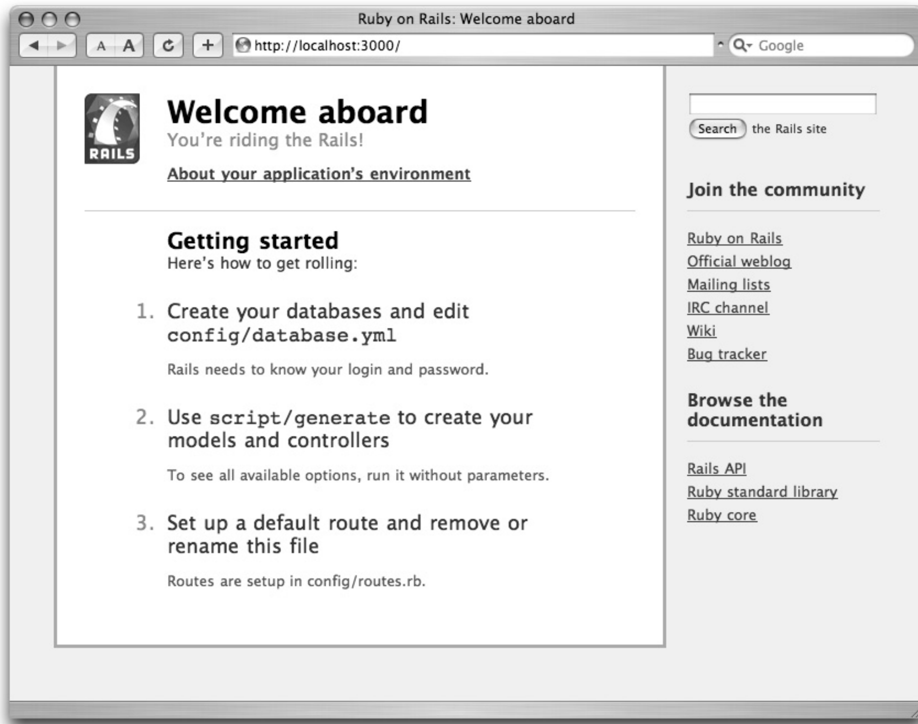

Abb. 1.3: Rails ist richtig installiert

Die nächsten Schritte sind auf dieser Startseite bereits beschrieben. Ab Version 2.0.2 verwendet Rails standardmäßig SQLite3 als Datenbank. Das hat den Vorteil, dass Sie keine weitere Datenbank wie MySQL oder PostgreSQL benötigen, wenn Sie mit einem Linux- oder Mac-OS-X-System arbeiten. SQLite ist dort bereits installiert und Sie können direkt loslegen.

Hinweis

Sie können bereits am Anfang festlegen, welche Datenbank Sie verwenden möchten. Dazu verwenden Sie den Parameter `-d`:

```
$ rails referenz -d mysql
```

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010 by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Verwenden Sie SQLite, dann hat Rails die benötigten Datenbanken bereits angelegt, so dass Sie gleich loslegen können. Verwenden Sie eine andere Datenbank, müssen Sie zuerst noch die Datenbanken erzeugen:

```
$ cd referenz
$ rake db:create:all
```

Die einzige Konfiguration, die Sie für eine lauffähige Rails-Anwendung benötigen, ist die Angabe der verwendeten Datenbank sowie des Datenbankbenutzers.

Sie haben jetzt drei Datenbanken (eine für die Entwicklung, eine für die Tests und einen produktive). Standardmäßig ist kein spezieller Datenbankbenutzer angelegt. Dies müssen Sie noch von Hand nachholen.

Je nach verwendetem Datenbanksystem sind die verwendeten Befehle anders – im Beispiel hier ist es MySQL. Verwenden Sie SQLite, dann entfällt dieser Schritt, denn SQLite kennt keine Benutzer.

```
$ mysql -uroot
mysql> grant all on referenz _development.* to 'referenz'@'localhost'
   identified by 'geheim';
mysql> grant all on referenz _test.* to 'referenz'@'localhost'
   identified by 'geheim';
mysql> quit
$
```

Wie die Datenbanken und der Datenbank-Benutzer heißen, müssen Sie jetzt auch noch in der Rails-Anwendung konfigurieren. Erinnern Sie sich, in welchem Verzeichnis Konfigurationsdateien stehen?

Es ist das `config` Verzeichnis. Dort finden sie die Datei `database.yml` die Sie nun mit einem Editor folgendermaßen anpassen:

```
development:
  adapter: mysql
  database: referenz _development
  username: referenz
  password: geheim
  host: localhost
  encoding: utf8

# Warning: The database defined as 'test' will be erased and
```


© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

```
# re-generated from your development database when you run 'rake'.
# Do not set this db to the same as development or production.
test:
  adapter: mysql
  database: referenz _test
  username: referenz
  password: geheim
  host: localhost
  encoding: utf8
```

Den Rest der Datei lassen Sie unverändert. Sie haben gesehen, dass außer dem Benutzernamen und dem Passwort bereits alles andere richtig eingetragen war. Zusätzlich haben Sie angegeben, dass die Datenbank in UTF-8 kodiert ist. Das interessiert allerdings erst einiges später, wenn Sie sich um Lokalisierung kümmern müssen.

Sie werden dieses »Mitdenken« von Rails noch an einigen Stellen erleben. Hier sehen Sie jedoch bereits, was ich in der Einleitung mit »opinionated software« gemeint habe. Rails hat sehr genaue Vorstellungen davon, wie die Datenbanken zu heißen haben und dass sie sowohl eine Entwicklungs-, eine Test- und eine Produktivumgebung verwenden werden.

Wichtig

Wenn Sie Änderungen an `database.yml` vornehmen, müssen Sie den Rails-Applikationsserver neu starten, damit er diese Änderung auch mitbekommt. Brechen Sie also mit `Strg`+`C` ab, und starten ihn dann mit `script/server` neu.

YAML

Der Dateityp `yml` steht für Konfigurationsdateien nach dem YAML Standard[1]. Der Standard trägt den Titel: »YAML Ain't Markup Language«. YAML ist deutlich leichter lesbar als XML, dem de-facto-Standard für Konfigurationsdateien. Vor allem die Notwendigkeit, alle Tags in XML sauber zu schließen, erhöht bei XML-Konfigurationsdateien den Schreibaufwand erheblich und verschlechtert die Lesbarkeit. XML mag für Maschinen gut lesbar sein, für Menschen wurde dieses Format aber wirklich nicht geschaffen. YAML ist ähnlich flexibel wie XML, aber viel leichter für Menschen lesbar. Es gibt für beinahe alle Programmiersprachen YAML-Bibliotheken. Unter Rails hat es sich als bevorzugtes Format für Konfigurationsdateien durchgesetzt.

1 http://yaml.org

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

1.5 Endlich: Die erste Seite

Um etwas anzuzeigen, benötigen wir in Rails eine View. Damit der Benutzer der Anwendung die Seite aufrufen kann, einen Controller. Sie wissen noch aus der Übersicht über die Verzeichnisstruktur, wo diese beiden Teile hingehören: in `app/controllers` und `app/views` respektive. Aber wie sehen diese Dateien aus? Rails bietet uns (natürlich) auch hier Vorlagen. Views und Controller haben in Rails eine sehr enge Verwandtschaft. Geben Sie Folgendes im Terminal ein:

```
$ script/generate controller Pages index
```

Es werden einige Dateien angelegt: Der `pages_controller.rb`, der dazugehörende funktionale Test, eine entsprechende Helper-Datei sowie die Datei `index.html.erb` im Verzeichnis `app/views/pages`.

Rufen Sie jetzt die URL `http://localhost:3000/pages` in Ihrem Browser auf. Sie sehen den Inhalt der Datei `app/views/pages/index.html.erb`.

Öffnen Sie diese Datei in Ihrem Editor und ändern Sie sie wie folgt ab:

```
<h1>Seiten Index</h1>
<p>Hallo Welt! - Heute ist der <%= Time.now.strftime("%d.%m.%Y") %></p>
<p>
  <% diff = Date.today - DateTime.new(1997, 1, 13) %>
  Vor <%= diff %>
  Tagen wurde HAL 9000 geboren
</p>
```

Nach dem Speichern laden Sie die Seite im Browser neu:

Abb. 1.4: Kubrik- und-Clarke Fans aufgepasst – am 13.1.2007 war der 10. Geburtstag des HAL 9000

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

`html.erb`-Dateien sind eine Mischung aus Ruby und HTML. ERB steht für Embedded Ruby. Neben dem ganz normalen HTML-Code (**h1** für Titel, **p** für Absätze usw.) können Sie mittels der speziellen Tags <% ... %> sowie <%= ... %> Ruby-Code ausführen lassen.

Während im ersten Fall nichts auf der Webseite ausgegeben wird, wird im zweiten Fall das Resultat des Ruby-Codes in die Seite eingefügt. Im Beispiel wird eine lokale Variable `diff` erzeugt, in der die Differenz in Tagen zwischen Heute und dem Geburtstag von HAL 9000 abgelegt ist. Diese Zahl geben Sie dann in der darauf folgenden Zeile aus.

1.6 URLs verstehen

Wieso ist die URL `/pages` plötzlich vorhanden? Das ist ein Teil der Rails-Magie. Sie erinnern sich, dass Sie einen Controller mit dem Namen Pages (der dann zu einer Datei `pages_controller.rb` wurde) angelegt haben? Rails prüft jede URL, ob der erste Teil (nach dem Hostnamen bis zum nächsten /) einem vorhandenen Controller entspricht. Wenn ja, wird dieser aufgerufen. Wenn nichts Weiteres in der URL steht, wird die Aktion `index` dieses Controllers aufgerufen. Und der wiederum zeigt die View `index.html.erb` an.

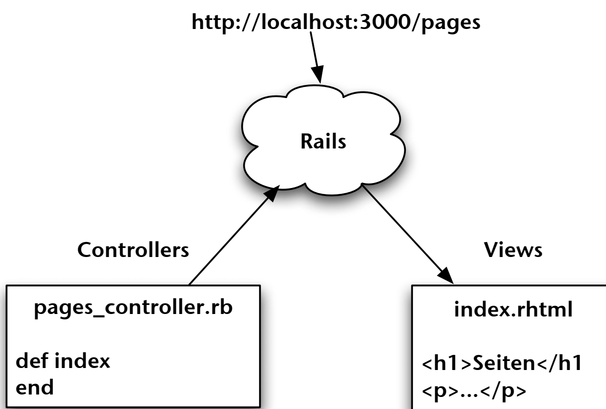

Abb. 1.5: Umsetzung von URL -> Controller -> View

Versuchen Sie einmal eine andere URL einzutippen, etwa `http://localhost:3000/xyz`. Was geschieht?

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

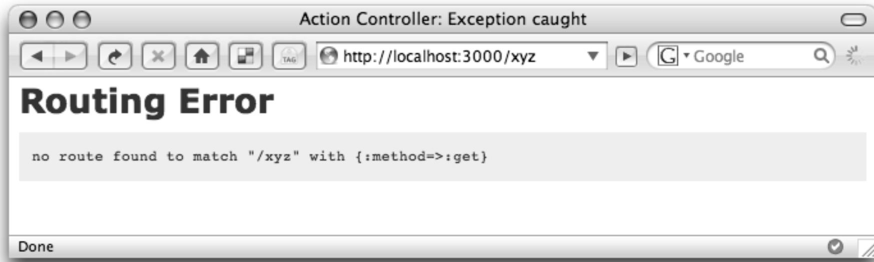

Abb. 1.6: Rails weiß nicht, was es mit der URL http://localhost:3000/`xyz` anfangen soll.

Sie bekommen eine (aussagekräftige) Fehlermeldung serviert. Rails findet keine Route, die der URI `/xyz` entspricht. Und das entspricht genau den Erwartungen, denn es gibt keinen xyz-Controller.

Wie sieht es mit komplexeren URIs aus? Was passiert denn, wenn Sie folgende URL eingeben `http://localhost:3000/pages/show/1`?

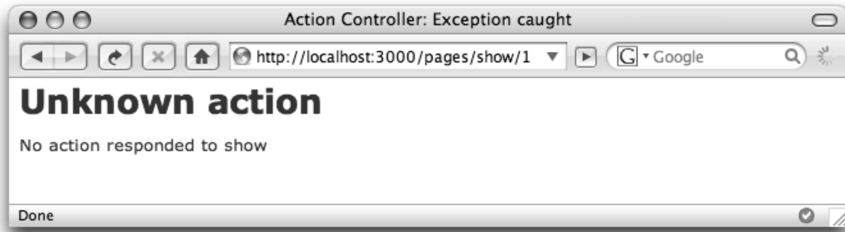

Abb. 1.7: Keine Aktion definiert

Die Fehlermeldung ist eine ganz andere. Rails beschwert sich jetzt darüber, dass die Aktion `show` unbekannt sei. Aktionen werden im Controller definiert. Öffnen Sie also die Datei `pages_controller.rb` und ergänzen Sie folgendermaßen:

```
class PagesController < ApplicationController

  def index
  end

 def show
  end
end
```


© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Speichern Sie die Datei und laden dann die Seite neu:

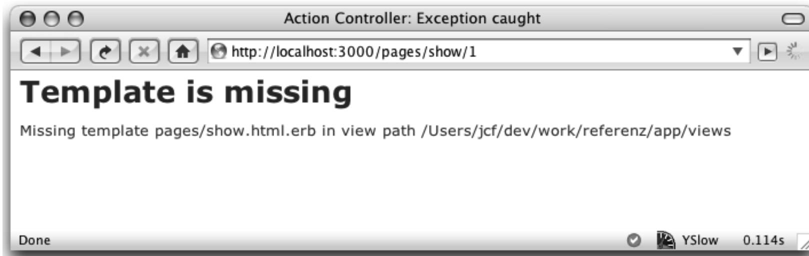

Abb. 1.8: Jetzt fehlt das Template.

Sie sind wieder einen Schritt weiter gekommen, denn jetzt beklagt sich Rails über das fehlende View-Template. Erstellen Sie die Datei `app/views/pages/show.html.erb` und fügen dann Folgendes ein:

```
<h1>Zeigen</h1>
<p>Ich wurde durch /pages/show/1 aufgerufen<p>
```

Was erwarten Sie nach dem Neuladen im Browser?

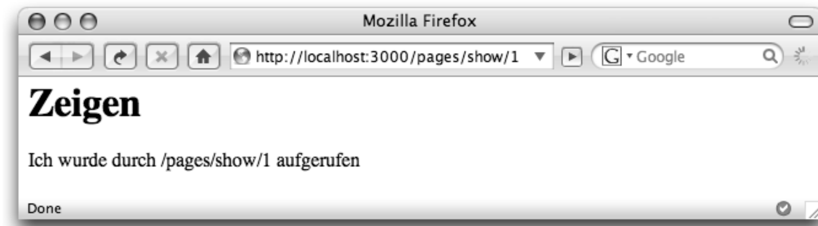

Abb. 1.9: Endlich sind alle Fehlermeldungen beseitigt.

Bleibt eigentlich nur noch eines zu klären: Die Zahl am Ende. Sie ist in diesem Fall ein Parameter, den sie im Controller verwenden können. Rails packt ihn (auch das ist wieder Magie, die Sie später noch genauer anschauen werden) mit allfälligen weiteren Parametern in einen Hash der `params` heißt. Dort können Sie ihn unter Angabe seines Namens abholen. In diesem Fall heißt der Parameter `id`. Ändern Sie die `show`-Methode im Pages-Controller folgendermaßen:

```
def show
  @page = params[:id]
end
```

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

und die Datei `show.erb.html` wie folgt:

```
<h1>Zeigen</h1>
<p>Ich wurde durch /pages/show/<%= @page %> aufgerufen<p>
```

Speichern sie die beiden Dateien und rufen dann die Seite mit verschiedenen URLs auf:

```
http://localhost:3000/pages/show/42
http://localhost:3000/pages/show/abc
```

Verhält sich alles so, wie Sie es erwarten?

Der Rails-Dispatcher schaut sich wieder die URL an, die er bekommt. `pages` sagt, dass der `pages_controller` verwendet werden soll. `show` zeigt an, dass die Methode `show` in diesem Controller aufgerufen wird. Der Parameter wird im `params`-Hash gespeichert und im Controller in der Instanzvariablen `@page` abgelegt. Dann wird in den Views die Datei `show.erb.html` verwendet um das Resultat der Aktion darzustellen. Und innerhalb dieser View haben Sie Zugriff auf die Instanzvariablen, die Sie im Controller definiert hatten.

Ruby

Variablenarten: Ruby kennt eine Reihe verschiedener Variablen. Die Art legt die »Sichtbarkeit« der Variablen fest:

Lokale Variablen: Eine Variable ohne weitere Auszeichnung (`foo`, `i`, `tmp`) ist eine lokale Variable. Sie ist nur in dem Block oder der Methode sichtbar, in der sie definiert ist. Eine Variable beginnt dann zu existieren, wenn das erste Mal eine Zuweisung auf sie stattfindet. Eine explizite Deklaration wie in C oder Java ist nicht nötig.

Instanzvariablen: Eine Instanzvariable ist durch ein vorangestelltes `@` ausgezeichnet (`@page`). Eine Instanzvariable ist innerhalb eines Objektes sichtbar.

Klassenvariablen: Klassenvariablen besitzen zwei `@@` vor dem Variablennamen (`@@counter`). Sie sind in der gesamten Klasse und allen instantiierten Objekten sichtbar.

Globale Variablen: Ein `$` vor dem Variablennamen zeichnet eine globale Variable aus (`$global`). Diese sind im gesamten Programm sichtbar. In der Regel werden Sie keine globalen Variablen verwenden.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Ruby

Hash: Der Hash ist eine Datenstruktur, ähnlich einem Array. Während in einem Array die Indizierung der verschiedenen Elemente über eine Zahl (dem Index) geschieht (`foo[42] = "hallo"`), können Sie in einem Hash beliebige Objekte zur Indexierung verwenden. Bei einem Hash spricht man allerdings nicht von Index, sondern von Schlüssel und einem dazugehörenden Wert (key und value im Englischen). Der Zugriff geschieht syntaktisch gleich wie beim Array:

```
a = Hash.new
a["hallo"] = "Welt"
a[42] = "6*9 ?"
```

Um einen ganzen Hash zu definieren, kann man (wie oben) `Hash.new` verwenden. Üblich ist allerdings die abgekürzte Form `{}`:

```
a = {}
```

Es lassen sich auch gleich Schlüssel und Wertpaare vorgeben:

```
a = { "hallo" => "Welt", 42 => "6*9 ?"}
```

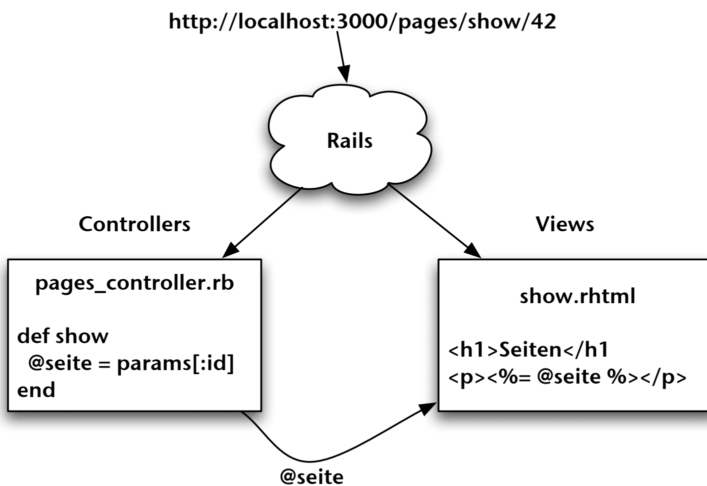

Abb. 1.10: Komplexere URL und wie sie auf Controller/Action zeigt

Bleibt noch ein Rätsel zu lösen: Woher weiß Rails, wie es mit Controllern, Actions und Parametern umgehen soll? Die Antwort liegt in der Datei `config/routes.rb`. In dieser Datei wird definiert, wie URLs bearbeitet werden sollen. Wenn Sie die Datei öffnen, sehen Sie gegen Ende folgende Zeile:

```
map.connect ':controller/:action/:id'
```


© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Ich denke, Sie haben eine Ahnung, was da definiert wird: Jede URL, die dem Muster `/etwas/sowas/irgendwas` entspricht (wobei etwas, sowas und irgendwas beliebige Strings sein können) wird folgendermaßen interpretiert: Der erste Teil (etwas) bezeichnet den **Controller**. Der zweite Teil (sowas) die **Action**. Der letzte Teil schließlich wird dem Parameter **id** übergeben. Auch hier gilt das oben zu den Hashs gesagte: Wenn Ihnen unklar ist, wieso hier überall ein Doppelpunkt steht, dann schauen sie Abschnitt 1.7.2 nach, dort sind *Symbole* erklärt.

1.7 Das Model einer Seite

Sie haben bereits jetzt die Möglichkeit, eine statische Website mit den bisher vorgestellten Techniken zu erstellen. Einfach für jede Seite eine eigene Methode im Controller definieren und die entsprechende `erb.html`-Seite anlegen. Nur wäre es wahrscheinlich einfacher gewesen, Sie hätten direkt HTML-Dateien geschrieben, wie anno dazumal, im letzten Jahrtausend.

Es wird Zeit, die Datenbank ins Spiel zu bringen.

Sie haben bereits Controller und Views kennen gelernt. Mit den Models kommen Sie zum Dritten im MVC-Bund. Die Komponente **ActiveRecord** von Rails ist für die Models und ihre Verbindung zur Datenbank zuständig. Im Grunde macht ActiveRecord nichts anderes, als Ruby-Objekte in eine SQL-Datenbank zu speichern und sie von dort wieder zu laden. Dazu implementiert ActiveRecord ein Pattern aus dem Buch »Patterns für Enterprise Application-Architekturen« [FRF02]. Das Pattern beschreibt eine relativ einfache Art des Objekt-Relationship-Mappings (ORM) und es heißt: ActiveRecord. ActiveRecord schirmt Sie von der Datenbank und SQL ab. Praktisch alle Operationen auf den Daten (und damit der Datenbank) sind in Ruby formuliert.

Wie soll denn jetzt das Model einer Seite aussehen? Am Anfang des Kapitels sind die Anforderungen formuliert: Eine Seite besteht aus einem Titel, einem Textfeld und einer Kategorie. Sie müssen jetzt also nur die entsprechenden Felder in der entsprechenden Tabelle in der Datenbank anlegen und die Datei für das Model erstellen. Oder Sie lassen Rails das machen:

```
$ script/generate model Page title:string body:text category:string
```

Sie sehen wieder die übliche Anzeige der neu angelegten Dateien: Es sind dies das Model selber `app/model/page.rb`, der Unit-Test in `test/unit/page_test.rb` mit dazugehörigem Fixture `test/fixtures/pages.yml` und die Migration `db/migrate/001_create_pages.rb`. Was es mit den Tests auf sich hat, werden Sie im nächsten Kapitel lernen. Zuerst interessiert, wie die Daten der Seite abgebildet sind.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Weiter oben haben Sie gesehen, dass ActiveRecord die Schnittstelle zwischen Datenbank und Objekten ist. Mehr oder weniger das gesamte Mapping (also die Abbildung zwischen den beiden Repräsentationen) kann ActiveRecord selbständig aus der Datenbank ablesen. Aber dazu muss erst mal die Struktur der Datenbank definiert sein.

1.7.1 Migration

Üblicherweise würden Sie jetzt zu einem von zwei bewährten Werkzeugen greifen: einem Texteditor, um die Tabellendefinition in SQL zu schreiben, oder einem SQL-Frontend für Ihre Datenbank, in der Sie Definitionen erfassen können. Mit Rails gibt es einen dritten Weg – die Migrationen.

Migrationen sind kleine Ruby-Programme, die das Datenbankschema verändern. Entweder fügen sie neue Tabellen oder Spalten hinzu oder löschen sie wieder. Warum der Aufwand? Mit allergrößter Wahrscheinlichkeit (vielmehr: mit Sicherheit) werden sich im Lauf der Entwicklung Änderungen am Datenbank-Schema ergeben. Wenn Sie bereits Erfahrung damit haben, solche Erweiterungen an der Datenbank vorzunehmen, wissen Sie, dass das unter Umständen ein heikler Prozess ist. Sie müssen sicherstellen, dass in jeder Umgebung (Entwicklung, Test, Produktion) alle Änderungen nachgezogen werden. Allenfalls müssen auch noch Daten verändert werden (etwa wenn eine neue Tabelle Daten aus einer bestehenden übernimmt). Gar nicht so einfach, das zu koordinieren. Und was geschieht, wenn Sie einen Fehler gemacht haben und wieder einen Schritt zurück müssen? Viel Arbeit!

Die Entwickler von Rails sind dieses Problem relativ früh mithilfe der oben genannten Migrationen angegangen. Stellen Sie sich vor, dass Ihre Datenbank (bzw. das Schema der Datenbank) zum Zeitpunkt X eine bestimmte Versionsnummer hat. Für Änderungen führen Sie gewisse Operationen auf der Datenbank durch, z.B. eine neue Tabelle einfügen. Die Datenbank hat nach dieser Operation eine um eins höhere Versionsnummer. Muss die Änderung rückgängig gemacht werden, dann können Sie sozusagen einen Schritt zurück gehen und die neu eingefügte Tabelle löschen.

Genau dieses Konzept wird durch die Migrationen abgedeckt. Es sind kleine Ruby-Programme, die eine Datenbank von einer Version in die nächst höhere oder niedrigere »befördern«. Selbstverständlich geht das auch, wenn die Datenbank noch komplett leer ist – in Version 0 sozusagen. Die erste Migration (die durch den Aufruf von `script/generate model` als leere Datei erzeugt wurde) ist also die Initialisierung der Datenbank.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Öffnen Sie die Datei `db/migrate/001_create_pages.rb` in Ihrem Editor und sehen Sie nach, was dort steht:

```
class CreatePages < ActiveRecord::Migration
  def self.up
    create_table :pages do |t|
      t.string :title
      t.text :body
      t.string :category

      t.timestamps
    end
  end

  def self.down
    drop_table :pages
  end
end
```

Listing 1.1: Migration, um die pages-Tabelle zu erzeugen

`CreatePages` ist eine Subklasse der Migrationsklasse von Rails. Sie hat zwei Klassenmethoden (`up` und `down`), mit der das Datenbankschema um eine Version nach oben oder unten geändert wird. Durch die Angabe der beiden Felder `title` und `body`, die Sie beim Aufruf der `generate`-Methode mit übergeben hatten, sind die entsprechenden Spaltendefinitionen auch bereits eingetragen.

Jetzt geht es darum, die `pages`-Tabelle zu erstellen – ergänzen Sie die beiden Methoden wie folgt:

```
class CreatePages < ActiveRecord::Migration
  def self.up
    create_table :pages do |t|
      t.string :title, :size => 80
      t.text :body
      t.string :category, :size => 80

      t.timestamps
    end
  end
```


© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

```
  def self.down
    drop_table :pages
  end
end
```

Listing 1.2: Um Größenangaben ergänzte Migration

Sie sehen die Definition der Tabelle pages. Diese Tabelle hat zwei Spalten (title, body und category), die jeweils als String (mit der Länge 80 Zeichen) oder Text (mit unbeschränkter Länge) definiert sind. Später lernen Sie noch weitere Optionen für Migrationen kennen, im Moment soll das reichen.

Die Methode down löscht diese Tabelle wieder.

Um die Änderungen in der Datenbank zu machen, müssen Sie die Migration aufrufen. Dazu verwenden Sie folgenden Befehl:

```
$ rake db:migrate
== CreatePages: migrating
   ======================================================
-- create_table(:pages)
   -> 0.0124s
== CreatePages: migrated (0.0126s)
   =============================================
```

Sie sehen, was die Migration macht, und wie lange sie gedauert hat. Wenn Sie sich die Entwicklungsdatenbank jetzt im SQL-Werkzeug Ihrer Wahl anschauen, dann sehen Sie, dass tatsächlich eine neue Tabelle pages mit den entsprechenden Spalten entstanden ist.

```
mysql> show tables from referenz_development;
+----------------------------------+
| Tables_in_referenz_development |
+----------------------------------+
| pages                            |
| schema_info                      |
+----------------------------------+
2 rows in set (0.00 sec)
```

pages? Moment – Sie hatten doch ein »Page«-Model bestellt, wieso heißt die Tabelle in der Datenbank jetzt plötzlich pages? Das ist eine der Konventionen, auf die Sie noch häufiger stoßen werden. Rails verwendet immer die Pluralisierung

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

des Modelnamens als Tabellenname. Einer der Gründe dafür ist die bessere Lesbarkeit des Rails-Codes.

Hinweis

Tabellennamen sind immer der Plural des Modelnamens. Aus `page` wird `pages`, aus `index` wird `indices` und so weiter. Rails ist ziemlich geschickt, wenn es darum geht, die richtige Pluralisierung auszuwählen – mindestens solange der Modellname Englisch ist.

Sie können das gut oder schlecht finden, besser ist es, Sie gewöhnen sich daran. Es ist Teil der Rails-Magie. Außerdem wurden auf den einschlägigen Mailinglisten bereits genügend Beiträge dafür oder dagegen verfasst.

Weiter sehen Sie die Tabelle `schema_info`. Diese wird von Rails automatisch erstellt. In ihr ist die Versionsnummer der letzten erfolgreich durchgeführten Migration gespeichert. Rails liest sie aus, um zu entscheiden, ob und wenn ja welche Migrationen auszuführen sind, um die Datenbank auf den aktuellen Stand zu bringen.

```
mysql> use referenz_development;
mysql> select * from schema_info;
+---------+
| version |
+---------+
|       1 |
+---------+
1 row in set (0.00 sec)
```

Solange die Datenbank noch leer ist, können Sie versuchen, auf die ursprüngliche Version (also die leere Datenbank) zurückzumigrieren:

```
Arwen:~/dev/workwiki jcf$ rake db:migrate VERSION=0
== CreatePages: reverting
   ======================================================
-- drop_table(:pages)
   -> 0.6227s
== CreatePages: reverted (0.6229s)
   =============================================
```

Wenn Sie sich jetzt die Tabellen bzw. die Versionsnummer des Schemas in der Datenbank ansehen, dann sehen Sie, dass der Urzustand wieder hergestellt wurde

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

(die Tabelle `schema_info` ist allerdings bestehen geblieben – aber das scheint mir auch sinnvoll zu sein).

Geben Sie jetzt nochmals:

```
rake db:migrate
```

ein, um die Datenbank bereit zu machen.

Tipp

Rails verwaltet zwar die `schema_info`-Tabelle selber, Sie können aber durchaus den Wert von `version` hinter dem Rücken von Rails verändern.

Ich verwende das gerne, wenn eine komplexe Migration nicht auf Anhieb funktioniert hat und ein erster Teil durchgeführt wurde, während ein zweiter Teil fehlschlug. In diesem Fall können Sie nicht auf die ursprüngliche Version zurückmigrieren, weil die Versionsnummer noch gar nicht aktualisiert wurde. Sie können die Migration aber auch nicht nochmals starten, da sich der erste Teil (der erfolgreiche) in der Regel nicht noch einmal wiederholen lässt.

In diesem Falle ist Handarbeit angesagt: In der Datenbank machen Sie den ersten Teil rückgängig und führen dann die Migration nochmals durch. Sie könnten natürlich auch die fehlenden Schritte von Hand ausführen und die Versionsnummer dann erhöhen. Davon rate ich jedoch ab, denn so ist nicht sichergestellt, dass sämtliche Migrationen reibungslos durchlaufen.

Wichtig ist, dass Sie wissen, dass Sie im Falle eines Falles diese Daten ändern können, wenn Sie sich mit fehlerhaften Migrationen in eine Sackgasse manövriert haben.

1.7.2 Datenpflege per Hand

Nachdem die Datenbank vorbereitet ist, geht es als nächstes darum, Daten einzugeben. Natürlich soll es am Schluss ein schickes Web-Interface dafür geben, aber der Weg dorthin dauert noch einen Moment. Daher werden die Daten erstmals von Hand eingetragen – und Sie bekommen ein Gefühl dafür, was in Ihrem Model eigentlich passiert. Wenn Sie bereits mit Ruby gearbeitet haben, dann kennen Sie `irb`, den interaktiven Ruby-Interpreter. Wenn Sie bereits etwas älter sind, dann kennen Sie ganz ähnliches von den Basic-Interpretern auf den guten alten Brotkästen und Türmatten (C64, Sinclair Spectrum). `irb` erlaubt es Ihnen, auf der Kommandozeile mit Ruby zu experimentieren. Als Rails-Entwickler steht Ihnen die aufgebohrte Version, in Form der `console`, zur Verfügung.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Sie starten die Konsole durch `script/console` und sehen sich Folgendem gegenüber:

```
Arwen:~/dev/referenz jcf$ script/console
Loading development environment.
>>
```

Jetzt können Sie normalen Ruby-Code eintippen, mit den Models arbeiten und sehen, wie diese sich verhalten.

Erstellen

Für den Moment reicht es, wenn Sie ein paar neue Seiten anlegen, die nachher im Web gezeigt werden:

```
>> p = Page.new            # neues Page Objekt erzeugen
=> #<Page id: nil, title: nil, body: nil, category: nil, created_at: nil,
   updated_at: nil>
>> p.title = "Rails Referenz"   # einen Titel zuweisen
=> "Rails Referenz"
>> p.body = "Diese Seite beschreibt Rails"
=> "Diese Seite beschreibt Rails"
>> p.category = "rails"
=> "rails"
>> p.save                  # Objekt in der Datenbank speichern
=> true
>> p
=> => #<Page id: 1, title: "Rails Referenz", body: "Diese Seite beschreibt
   Rails", category: "rails", created_at: "2007-08-21 00:42:35",
   updated_at: "2007-08-21 00:42:35">
>> p.title
=> "Rails Referenz"
```

So – Sie haben eben Ihr erstes Modelobjekt erzeugt und gespeichert. Die lange und komplizierte Ausgabe in der zweiten Zeile (`#<Page:....`) zeigt eine Repräsentation des Objektes. Als erstes ist der Name der Klasse (`Page`) gezeigt und dann die drei Attribute, die den Spalten in der Datenbanktabelle entsprechen. Alle drei sind mit `nil` belegt, haben also den Nullwert. Die beiden Attribute `created_at` sowie `updated_at` wurden durch die Definition `t.timestamps` in der Migration erzeugt. Es sind »magische« Spalten, die von Rails automatisch mit dem Erstellungs- beziehungsweise dem Bearbeitungszeitpunkt des Objektes gefüllt werden.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Danach weisen Sie der Reihe nach die drei Attribute zu und speichern das Objekt. Rails hat automatisch die entsprechenden Methoden (`title`, `title=`, `body`, `body=` etc.) erzeugt. Andere Methoden (wie `save`, `new` oder `create`) sind Teil der Basisklasse von ActiveRecord und stehen allen Models zur Verfügung.

Der Aufruf von `p.save` liefert `true` zurück, was besagt, dass die Daten in die Datenbank geschrieben wurden. Das dem tatsächlich so ist, zeigt ein Blick in die Datei `log/development.log`.

```
Page Columns (0.002634)  SHOW FIELDS FROM pages
SQL (0.000324)  BEGIN
SQL (0.000377)  INSERT INTO pages (`category`, `title`, `body`)
   VALUES('rails', 'Rails Referenz', 'Diese Seite beschreibt Rails')
SQL (0.010295)  COMMIT
```

Hier sehen Sie unter anderem, welche SQL-Befehle Rails generiert hat, als Sie die Methode `save` aufgerufen haben. Vielleicht fragen Sie sich, was der Befehl `SHOW FIELDS FROM pages;` ganz zu Beginn der Logdatei hier sucht. Das ist der Moment, in dem Rails die Datenbank nach der Struktur der Tabelle befragt hat, um danach die Methoden zu erzeugen, mit denen Sie auf die Attribute zugreifen. Diese Methoden existieren also bis zu dem Zeitpunkt gar nicht, sondern sie werden erst zu diesem Zeitpunkt dynamisch erzeugt. Diese »Zauberei« wird durch die Sprache Ruby ermöglicht und macht einen großen Teil des Reizes von Rails aus.

In der vorletzten Zeile im oberen Listing, sehen Sie den Zustand des Objektes nachdem es gespeichert wurde. Wichtig sind im Moment zwei Dinge: `new_record` ist `false`, und Sie sehen ein neues, zusätzliches Attribut: `id`. Rails vergibt automatisch jedem gespeicherten Objekt eine fortlaufende Identifikationsnummer – eben `id`. Mithilfe dieser Nummer kann das Objekte eindeutig identifiziert werden.

Hinweis

Wenn Sie nochmals zurückblättern und die Migrationsdatei anschauen, mit der die `pages`-Tabelle erzeugt wurde, sehen Sie, dass dort keine Spalte `id` definiert ist. Da `id` ein zentraler Bestandteil jeder Datenbanktabelle ist, erzeugt Rails sie einfach automatisch. Sie können das verhindern, indem Sie dem Aufruf von `create_table` den Parameter `:id => false` übergeben.

Der letzte Befehl `p.title` gibt Ihnen den Inhalt des Attributes `title` dieser Seite zurück.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Erzeugen Sie jetzt noch einige solcher Seiten in der Konsole. Wenn Sie das neue Objekt in einem Durchgang erzeugen möchten, dann können Sie die einzelnen Attribute direkt als Parameter übergeben:

```
>> p = Page.new(:title => "Ruby Referenz", :body => "Ruby etc.", :category
   => "ruby")
=> #<Page id: nil, title: "Ruby Referenz", body: "Ruby etc.", category:
   "ruby", created_at: nil, updated_at: nil>
>> p.save
=> true
```

Am Attribut `id`, das `nil` ist, erkennen Sie, dass das Objekt noch nicht gespeichert wurde. Schließlich lässt sich auch das noch etwas vereinfachen – indem Sie `create` statt `new` verwenden, speichern Sie das Objekt auch gleich in der Datenbank:

```
>> Page.create(:title => "HTML Referenz", :body => "Alles über HTML",
   :category => "web")
=> #<Page id: 3, title: "HTML Referenz", body: "Alles uber HTML", cate-
   gory: "web", created_at: "2007-08-21 00:52:46", updated_at: "2007-08-
   21 00:52:46">
```

Ruby

Wie soll man eigentlich die seltsamen Parameter für `new` oder `create` lesen? Und was sind sie überhaupt?

Die => deuten darauf hin, dass es sich um einen Hash handelt, der da als Parameter an die entsprechende Methode übergeben wird. Aber wo sind die geschweiften Klammern geblieben? Ruby erlaubt es Ihnen, diese Klammer wegzulassen, wenn die Parameter, die Sie einer Methode geben, alle zum gleichen Hash gehören. Ruby sammelt sie und erzeugt den Hash selbstständig.

Sie müssen die Klammern nur dann verwenden, wenn nach dem Hash noch weitere Parameter übergeben werden sollen:

```
def foo(options, bar)
    ...
   end
foo { :a => 1, :b => 2}, "hallo"
```

Das Weglassen von runden und geschweiften Klammern führt zu lesbarerem Code:

```
Page.create({:title => "HTML Referenz", :body => "Alles über HTML",
             :category => "web"})
Page.create :title => "HTML Referenz", :body => "Alles über HTML",
            :category => "web"
```


© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Es bleibt zu klären, was `:title` oder `:body` sind. Es handelt sich dabei nicht um Variablen oder Strings, sondern um Symbole. Symbole sind Namen oder Bezeichnungen. Ein Symbol hat keinen anderen Wert als seinen Namen. Es eignet sich unter anderem deshalb gut als Schlüssel in einem Hash. Ein weiterer Vorteil von Symbolen ist, dass sie nur ein einziges Mal im Speicher vorhanden sind. Man könnte den Hash oben gut auch mit Strings schreiben:

```
Page.create "title" => "HTML Referenz", "body" => "Alles über HTML",
            "category" => "web"
```

Das hat aber zwei Nachteile: Erstens ist es etwas schwieriger lesbar, und zweitens wird für jeden String ein neues Objekt im Speicher angelegt. Das nächste Mal, wenn Sie `"title"` als Schlüssel in einem Hash verwenden, wird ein neues Objekt angelegt. Verwenden Sie ein Symbol, ist dieses nur einmal vorhanden.

Finden

Erstellen von neuen Objekten ist nur die halbe Miete, Sie müssen auch in der Lage sein, diese Objekte später wiederzufinden und zu laden. Rails stellt Ihnen die mächtige `find`-Methode zur Verfügung, mit deren Hilfe Sie Ihre Objekte wieder aus der Datenbank laden können:

```
>> Page.find 1
=> #<Page id: 1, title: "Rails Referenz", body: "Diese Seite beschreibt
   Rails", category: "rails", created_at: "2007-08-21 00:42:35",
   updated_at: "2007-08-21 00:45:51">
```

`find` nimmt eine ganze Reihe unterschiedlicher Parameter entgegen. In der einfachsten Form einfach eine Zahl. Diese Zahl ist, wie oben erwähnt, die interne `id` des Objektes in der Datenbank.

```
>> Page.find 99
ActiveRecord::RecordNotFound: Couldn't find Page with ID=99
        from /usr/local/lib/ruby/gems/1.8/gems/activerecord-1.15.1/lib/
   active_record/base.rb:1028:in `find_one'
        from /usr/local/lib/ruby/gems/1.8/gems/activerecord-1.15.1/lib/
   active_record/base.rb:1011:in `find_from_ids'
        from /usr/local/lib/ruby/gems/1.8/gems/activerecord-1.15.1/lib/
   active_record/base.rb:416:in `find'
        from (irb):5
```

Geben Sie eine nicht vorhandene ID ein, bekommen Sie eine Exception von ActiveRecord. Die Idee dahinter ist, dass Sie genau wissen sollten, dass ein entsprechendes Objekt vorhanden ist, wenn Sie es direkt mit der ID laden wollen.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Sind Sie hingegen nicht sicher, dann verwenden Sie z.B. folgende Form:

```
>> Page.find_by_id 99
=> nil
```

Anstelle der Exception erhalten Sie jetzt einfach den leeren Wert `nil` zurück. Sie können so nach Werten in jeder der definierten Spalten suchen:

```
>> Page.find_by_category "rails"
=> #<Page id: 1, title: "Rails Referenz", body: "Diese Seite beschreibt
   Rails", category: "rails", created_at: "2007-08-21 00:42:35",
   updated_at: "2007-08-21 00:45:51">
```

Speziell bei diesem Methodenaufruf ist, dass das Page-Model gar keine Methode hat, die `find_by_category` heißt. Diese Methode wird erst dann dynamisch erzeugt, wenn sie das erste mal verwendet wird.

Falls Sie vorhin mehrere Seiten mit der Kategorie »main« angelegt hatten, dann wundern Sie sich vielleicht, wieso Sie nur ein Exemplar davon zurückbekommen. Die `find_by...`-Methoden suchen nur nach dem ersten passenden Eintrag. Möchten Sie alle Objekte finden, die der entsprechenden Kategorie angehören, verwenden Sie entweder `find_all_by_category` oder die folgende Form von `find`:

```
>> Page.find :all, :conditions => [ "category = ?", "rails" ]
=> [#<Page id: 1, title: "<em><blink>Rails Referenz</blink></em>", body:
   "Diese Seite beschreibt Rails", category: "rails", created_at: "2007-
   08-21 00:42:35", updated_at: "2007-08-21 00:45:51">, #<Page id: 4,
   title: "ActiveRecord", body: "Datenbanken und Objekte", category:
   "rails", created_at: "2007-08-21 00:56:19", updated_at: "2007-08-21
   00:56:19">]
```

Jetzt erhalten Sie ein Array (sichtbar an den eckigen Klammern `[ ]`) aller Objekte zurück, die der Bedingung `category = 'main'` entsprechen.

Genau das gleiche Resultat erhalten Sie durch folgenden Aufruf:

```
>> Page.find :all, :conditions => { :category => 'rails' }
```

Welche Form Sie verwenden, ist weitgehend Geschmackssache.

Ein so gefundenes und geladenes Objekt können Sie nach Belieben weiterverwenden. Im Fall des Referenzhandbuches wollen Sie es wahrscheinlich im Browser anzeigen.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Wichtig

Auf keinen Fall sollten Sie Folgendes verwenden:

```
Page.find :all, :conditions => "category = 'main'"
```

Obwohl dieser Aufruf funktioniert (probieren Sie es aus), ist er sehr problematisch. Denn jetzt sind Sie für das richtige »Quoten« der Parameter verantwortlich und müssen Anführungszeichen innerhalb des Suchbegriffs durch die Form ersetzen, die ihre Datenbank erwartet. Schlimmer noch ist der Fall aber, wenn Sie Eingaben ihrer Benutzer direkt weitergeben:

```
suchbegriff = params[:search]
Page.find :all, :conditions => "catgegory = '#{suchbegriff}'"
```

So öffnen Sie einem Angriff durch SQL-Injection Tür und Tor, denn wenn Ihr Benutzer als Suchbegriff `1'; drop table pages;` eingibt, dann dürfen Sie die Datenbank von einem Backup zurückspielen.

Verwenden Sie **immer** die Formen, bei denen Rails das Argument an die Datenbank übergibt. Rails verhindert solche SQL-Injection-Angriffe und setzt auch die richtigen Anführungszeichen...

Löschen

Der Lebenszyklus eines Objektes wäre nicht vollständig, wenn nicht auch noch das Löschen betrachtet werden würde.

```
> p = Page.create( :title => 'kurzlebig', :category => 'main', :body =>
   'ich bin gleich wieder weg')
=> #<Page id: 5, title: "kurzlebig", body: "ich bin gleich wieder weg",
   category: "main", created_at: "2007-08-21 00:57:23", updated_at:
   "2007-08-21 00:57:23">
>> p.destroy
=> #<Page id: 5, title: "kurzlebig", body: "ich bin gleich wieder weg",
   category: "main", created_at: "2007-08-21 00:57:23", updated_at:
   "2007-08-21 00:57:23">
>> p.title = "lebt es noch?"
TypeError: can't modify frozen hash
```

Der Aufruf der Methode `destroy` eines Objektes löscht das Objekt aus der Datenbank (suchen Sie den entsprechenden Log-Eintrag) und »friert« das Objekt ein, das noch im Speicher ist. Es lassen sich – wie Sie in der letzten Zeile sehen – keine Veränderungen mehr daran vornehmen.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

`destroy` hat einen Nachteil, wenn Sie viele Objekte löschen. Denn jedes dieser Objekte muss zuerst geladen und instantiiert werden:

```
>> pages = Page.find :all
=> [#<Page......>, #<Page....>]
>> pages.each { |page| page.destroy }
```

Dies ist also ein ineffizienter Weg, alle Seiten zu löschen. Schneller und schonender im Umgang mit dem Speicher geht es folgendermaßen:

```
>> Page.delete_all
=> 5
```

Bei dieser Form werden die Objekte nicht geladen, sondern der entsprechende SQL-Befehl wird direkt ausgeführt. Als Rückgabewert (5 im Beispiel) bekommen Sie die Anzahl `rows`, die aus der Datenbank gelöscht wurden.

1.8 Darstellung einzelner Seiten

1.8.1 Einzelne Seite

Nachdem Sie jetzt bereits einige Seiten gespeichert haben, geht es im nächsten Schritt darum, diese anzuzeigen. Sie erinnern sich: Der `pages`-Controller lädt die gewünschte Seite in der Methode `show` und übergibt sie danach der View `show.rhtml`.

Mit dem Wissen vom letzten Abschnitt, können Sie die Methode um den `find`-Aufruf ergänzen. Die Instanzvariable `@page` enthält jetzt das `page`-Objekt aus der Datenbank.

```
# in app/controllers/pages_controller.rb
def show
  @page = Page.find params[:id]
end
```

Nun können Sie auch die View (`app/views/pages/show.html.erb`) erweitern:

```
<h1><%= h @page.title %></h1>
<p>Kategorie: <%= h @page.category %></p>
<p><%= h @page.body %></p>
```


© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Wenn Sie jetzt in Ihrem Browser die URL `http://localhost:3000/pages/show/1` aufrufen, sollten Sie etwa folgendes Bild sehen:

Abb. 1.11: Die erste Webseite aus der Datenbank

Schauen Sie sich die `show.html.erb`-Datei nochmals an. In ihr wird in einer Mischung aus HTML und Ruby festgelegt, was angezeigt wird. Die <%= %>-Tags fügen das Resultat des Rubycodes direkt an der entsprechenden Stelle in die HTML-Seite ein. Sie sehen, dass nichts weiter geschieht, als dass die drei Attribute der Seite ausgegeben werden.

Unbekannt ist aber das `h` vor dem Aufruf der Methode. Was geschieht da? Wenn Sie bereits Webanwendungen geschrieben haben, wissen Sie, dass es gefährlich ist, beliebige Texte einfach auszugeben. Vor allem dann, wenn diese Texte vorher von den Benutzern Ihrer Anwendung erfasst wurden. Denn die Schlaumeier unter den Benutzern könnten in Versuchung geraten, HTML zu erfassen. Machen Sie doch einmal folgendes Experiment: In der Konsole ändern Sie den Inhalt der ersten Seite wie folgt:

```
>> p = Page.find 1
=> #<Page:0x2567740 @attributes={"category"=>"rails", "title"=>"Rails
   Referenz", "id"=>"1", "body "=>"Diese Seite beschreibt Rails"}
```

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

```
>> p.title = "<em><blink>Rails Referenz</blink></em>"
=> "<em><blink>Rails Referenu</blink></em>"
>> p.save
```

Danach duplizieren sie die Titelzeile in `show.html.erb` und lassen das `h` weg:

```
<h1><%= h @page.title %></h1>
<h1><%= @page.title %></h1>
<p>Kategorie: <%= h @page.category %></p>
<p><%= h @page.content %></p>
```

Laden Sie dann die Seite im Browser neu:

Abb. 1.12: Mit und ohne HTML-Codierung

Das Blinken bleibt Ihnen hier im Buch glücklicherweise erspart, aber Sie erkennen das Problem. Die HTML-Tags werden in der zweiten Version (die ohne `h`) direkt vom Browser interpretiert, während sie in der ersten Version ausgeschrieben erscheinen. Beim Betrachten des Quelltextes der Seite erkennen Sie, was geschehen ist: Der Aufruf von `h` führt ein sogenanntes HTML-Encoding durch. Aus einem < wird ein `<` während ein > in ein `>` gewandelt wird. Damit ist die Gefahr gebannt, dass sich unerwünschter HTML-Code in Ihre Webseiten schleicht. Hinter den Kulissen wird die Funktion `HTMLEncode` aus dem `CGI`-Modul aufgerufen. `h` ist ein Aliasname dieser Methode.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Selbstverständlich müssen Sie nicht jede Ausgabe durch `h` filtern lassen. Sie sollten sich aber von Fall zu Fall überlegen, ob es sinnvoll ist, diesen Filter zu verwenden. Eingaben von Benutzern, die Sie nicht überprüfen können, sollten auf jeden Fall so bereinigt werden.

1.8.2 Fehlerhaft!

Experimentieren Sie mit weiteren IDs und irgendwann werden Sie die Exception wiedersehen, die Ihnen bereits vorher in der Konsole begegnet ist:

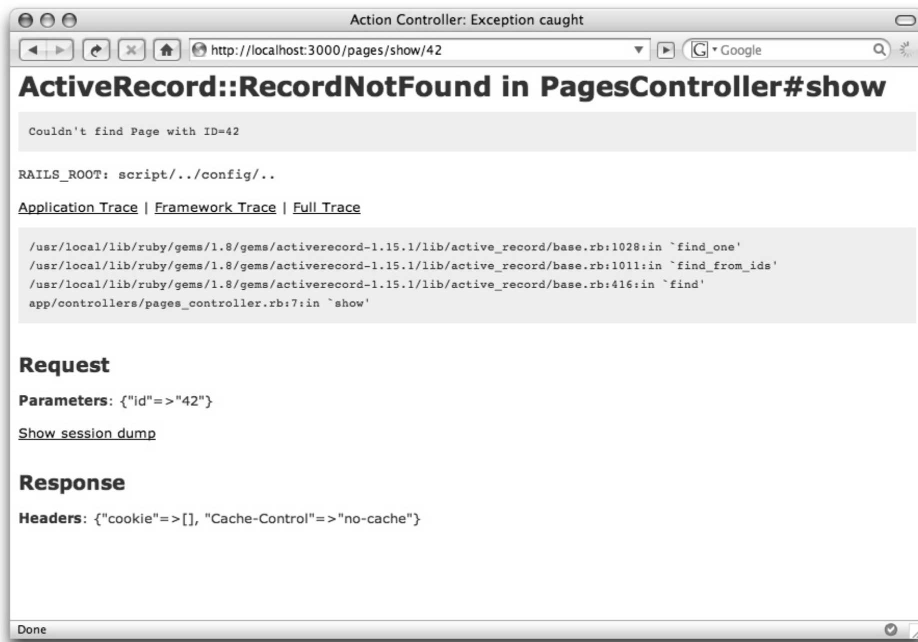

Abb. 1.13: Keine Seite gefunden

Wenn Sie sich die Seite einmal genauer ansehen, stellen Sie fest, dass Rails Ihnen sehr viele Informationen darüber gibt, was wo schief gelaufen ist. Es lohnt sich zu verstehen, was für Informationen gezeigt werden, denn mit etwas Erfahrung können Sie so Fehler blitzschnell lokalisieren (und manchmal sogar blitzschnell korrigieren).

Dem Titel der Seite entnehmen Sie die Exception (`RecordNotFound`) und den Ort (im `PagesController` in der Methode `show`). Als nächstes folgt eine genauere Beschreibung der Fehlerursache (es konnte keine Page mit der ID 42 gefunden werden).

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Der nächste größere Block ist der Stacktrace, mit dem Sie zurückverfolgen können, welche Kette von Aufrufen zu dem Fehler geführt haben. Die eigentliche Exception ist in `active_record/base.rb` auf Zeile 1028 in der Methode `find_one` aufgetreten. `find_one` wurde aufgerufen von `find_from_ids`, diese Methode wiederum von `find` und `find` schließlich aus dem `pages_controller`, Zeile 7.

Danach folgen zwei Abschnitte, die den Request einerseits und die Response des Servers andererseits beschreiben. In diesem Beispiel ist da noch nicht viel zu sehen, aber Sie sehen auf jeden Fall, dass die `show`-Methode mit dem Parameter `id` mit dem Wert `42` aufgerufen wurde.

Ein solcher Fehler ist zwar im Kontext absolut verständlich und nachvollziehbar, Ihren Benutzern wollen Sie das jedoch nicht zumuten. Sie müssen sich überlegen, wie Sie mit diesem Fehlerfall umgehen möchten. Sie könnten z.B. einfach auf die Hauptseite umleiten lassen. Klassischerweise handelt es sich bei diesem Fall aber um einen «Seite nicht gefunden»-Fehler, oder einem Fehlercode 404. 404 steht dabei für den Antwortcode des Webservers. Genau das können Sie erreichen, indem Sie den Controller wie folgt ergänzen:

```
def show
  @seite = Page.find params[:id]
rescue
  render :file => 'public/404.html', :status => 404
end
```

Der Befehl `rescue` leitet einen Exceptionhandler ein. Der Code wird genau dann ausgeführt, wenn im vorangehenden Code eine Exception ausgelöst wurde. Auf die Methode `render` werden Sie immer wieder stoßen, sie weist Rails an, gewisse Dinge an den Browser zu übergeben. In diesem Fall eine HTML-Datei (die im Verzeichnis `public` liegt). Sie können (und sollen) diese Datei natürlich Ihren Bedürfnissen anpassen. Der Parameter `status` weist `render` an, einen Fehlercode 404 an den Browser zu übergeben.

Es ist allerdings besser, wenn Sie die Behandlung dieser Fehler an zentraler Stelle abhandeln. Ansonsten schreiben Sie den gleichen Code immer und immer wieder – und das ist nicht DRY.

Hinweis

DRY? Eines der Rails-Mantras: Don't Repeat Yourself. Wann immer möglich, soll keine Information und kein Code dupliziert werden. Das erhöht die Wartbarkeit der Anwendung.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Um die RecordNotFound-Exception an zentraler Stelle zu behandeln, wechseln Sie in die Datei app/controllers/application.rb.

```
class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time

  # See ActionController::RequestForgeryProtection for details
  # Uncomment the :secret if you're not using the cookie session store
  protect_from_forgery # :secret => 'a27a513a9ea7c44da0cd55caa0b5a30f'
end
```

Hier finden Sie die Superklasse aller Controller (eigentlich müsste die Datei application_controller.rb heißen, so zumindest die Meinung, die momentan hitzig diskutiert wird). Jede Methode, die Sie hier einfügen, wird allen weiteren Controllern vererbt.

Rails 2.0 gibt Ihnen die Möglichkeit, hier auf Exceptions zu reagieren.

```
class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time

  rescue_from ActiveRecord::RecordNotFound, :with => :show_404

  def show_404
    render :file => 'public/404.html', :status => 404
  end

 ...
end
```

Im Controller können Sie jetzt den Rescue-Zweig löschen; jeder Zugriff, der die RecordNotFound-Exception auslöst, wird durch diese Fehlerbehandlung abgedeckt.

Sie können mehrere rescue_from-Anweisungen definieren, und Sie können diese in den einzelnen Controllern auch spezialisieren.

1.8.3 Liste aller Seiten anzeigen

Nachdem Sie eine einzelne Seite darstellen können, geht es als nächstes darum, eine Liste aller Seiten anzuzeigen. Der Benutzer soll dann aus der Liste eine Seite

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

anzeigen können. Das ist zwar noch einigermaßen weit von einer richtigen Referenz entfernt, aber Sie nähern sich Schritt für Schritt daran an.

Überlegen Sie mal, was Sie denn für Teile benötigen, damit Sie eine Liste aller Seiten anzeigen können. Sie benötigen einen Controller und eine View. Der Controller muss alle Seiten laden und an die View übergeben. Die View bekommt diese Objekte und soll sie der Reihe nach anzeigen. Mit dem Wissen über Models und die `find`-Methode ist es einfach, die Controller-Methode zu schreiben.

```
def index
  @pages = Page.find :all
end
```

Es werden alle Seiten geladen und in einem Array in der Variablen `@pages` abgelegt. Danach wird die View `list.html.erb` angezeigt.

Diese View muss nun jedes Element dieses Arrays mit einem Link auf die entsprechende Seite anzeigen. Löschen Sie den Inhalt der bestehenden `index.html.erb`-Datei und ändern Sie sie wie folgt:

```
<h1>Alle Seiten</h1>

<% @pages.each do |page| %>
  <%= link_to(h(page.title), :action => 'show', :id => page) %><br/>
<% end %>
```

Listing 1.3: `app/vies/pages/index.html.erb`

Was passiert hier? Im ersten Ruby-Block verwenden Sie den `each`-Iterator, der jedes Element des Arrays `@pages` durchläuft und dieses Element in der lokalen Variablen `page` speichert. Das Kernstück dieses Blocks ist der Aufruf einer Hilfsfunktion `link_to`, die einen HTML-Link erzeugt. Als ersten Parameter erwartet sie den Titel des Links (in diesem Falle also den Titel der Seite), als zweiten Parameter eine Anweisung dafür, wie die URL aufgebaut ist, auf die der Link zeigt.

Hier übergeben Sie die gewünschte Aktion (`show`) und die ID der Seite, die angezeigt wird. Korrekterweise müssen Sie die ID des Objektes übergeben, das Sie anzeigen wollen (also `page.id`), aber die `link_to`-Methode nimmt ebenso das gesamte Objekt und findet selbstständig die ID davon.

Sie können jetzt Ihre Seiten mit der URL `http://localhost:3000/pages/list` anzeigen:

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

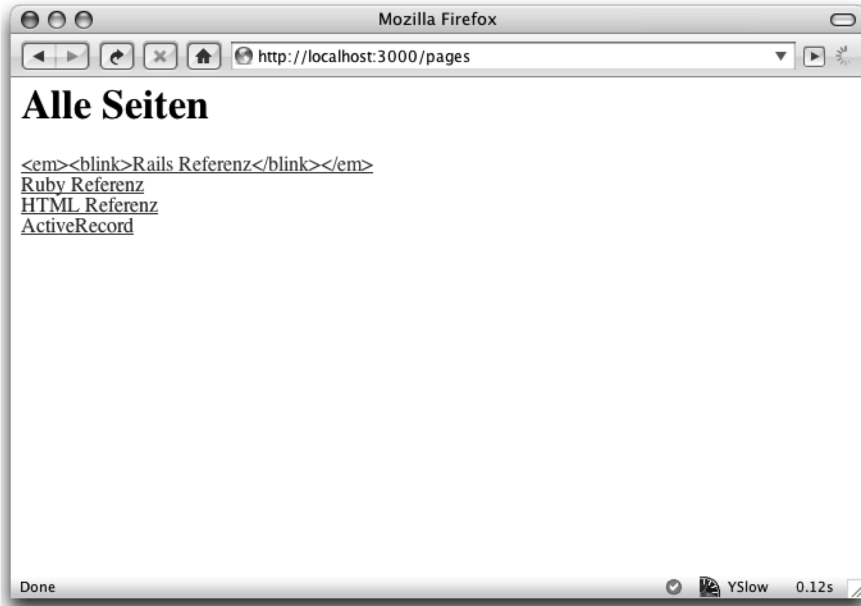

Abb. 1.14: Die Liste aller Seiten in der Datenbank

1.9 Ein einfaches Layout verwenden

Wenn Sie sich den HTML-Sourcecode der Seiten anschauen, die Sie bis jetzt erzeugt haben, dann werden Sie feststellen, dass das Resultat zwar von Ihrem Browser angezeigt wird, es mit korrektem HTML aber nicht viel zu tun hat:

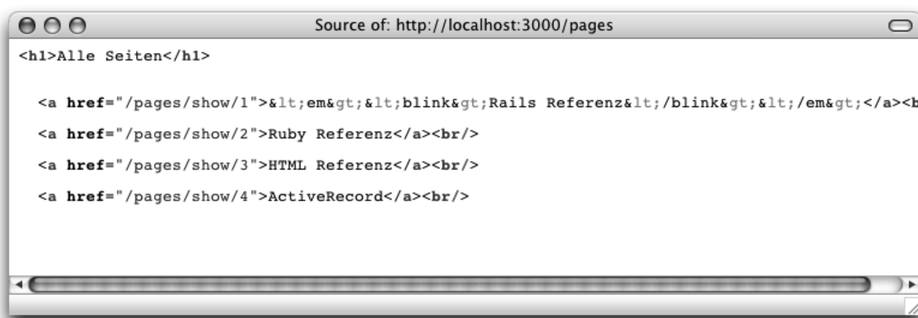

Abb. 1.15: Source-Ansicht einer Rails-Seite

Es fehlt an allen Ecken und Enden: Kein Doctype, keine `<html>`-, `<head>`- oder `<body>`-Tags, nur der nackte Code aus Ihren Views.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

Sie könnten natürlich die fehlenden Elemente in Ihre verschiedenen `.html.erb`-Dateien einfügen. Das ist aber nicht viel besser als wie in guten alten Zeiten, wo man ganze Websites von Hand gepflegt hat, vor allem ist es nicht DRY (schon wieder dieses Wort!).

Natürlich gilt das auch für die HTML-Definition ihrer Seiten. Im `views`-Verzeichnis finden Sie neben dem Verzeichnis `pages` ein weiteres, das `layouts` heißt. In diesem Verzeichnis liegen ein oder mehrere Layouts. Legen Sie eine Datei `pages.html.erb` an, die folgenden Inhalt hat:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title><%= @title || "Referenz" %></title>
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

Listing 1.4: `app/views/layout/pages.rhtml`

Laden Sie jetzt die Seite in Ihrem Browser neu und betrachten Sie den HTML-Code nochmals. Jetzt sehen sie korrektes und vollständiges HTML.

Sie sehen, dass im Layout zwei Zeilen mit Ruby-Code vorhanden sind. In der ersten wird der Titel des Browserfensters gesetzt. Dazu schaut Rails nach, ob eine Instanzvariable namens `@title` vorhanden ist. Wenn ja, wird ihr Inhalt angezeigt. Wenn nicht, wird einfach nur `Referenz` dargestellt.

Die zweite Zeile ist im `<body>`-Tag. Dort steht nur `yield`. Damit wird Rails aufgefordert, an dieser Stelle den Inhalt der View, die momentan angezeigt werden soll, einzufügen.

1.10 Was Sie gelernt haben

In diesem ersten Kapitel haben Sie einen großen und schnellen Überblick über verschiedene Komponenten von Rails erhalten. Dabei habe ich bewusst vermieden, zu sehr in die Tiefe zu gehen. Einiges kann man anders lösen, aber das ist Thema von späteren Kapiteln. Es geht mir darum, dass Sie sehen, welche Teile von Rails es gibt und wie diese zusammenhängen. Wenn Sie dieses Grundgerüst ver-

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025

standen haben, ist es für sie deutlich einfacher, die weitergehenden Konzepte »an die richtige Stelle zu rücken«.

Schauen Sie sich einmal an, was sie alles bereits in diesem ersten Kapitel gelernt haben:

- Eine neue Rails-Anwendung erstellen (`rails`)
- Die Verzeichnisstruktur einer Rails-Anwendung verstehen
- Die Versionsnummern aller Komponenten bestimmen (`script/about`)
- Den eingebauten Webserver starten (`script/server`)
- Die benötigten Datenbanken (Entwicklung und Test) anlegen und in `config/database.yml` konfigurieren
- Was das Model View Controller Pattern ist und wie es in Rails umgesetzt ist
- Wie man Controller und Views generiert (`script/generate controller Name aktion`)
- Wie man Ruby in einer HTML-Seite einbettet (in einer RHTML-View mit den <% %> und <%= %> Tags)
- Wie Rails URLs auflöst (via der Datei `config/routes.rb`)
- Wie ein Model erzeugt wird (`script/generate Model Name`)
- Was eine Datenbankmigration ist
- Wie man eine Tabelle in der Datenbank anlegt (via Migration)
- Wie man die Version der Datenbank verändert (`rake db:migrate`)
- Die Rails-Konsole starten (`script/console`)
- Objekte erzeugen (`new`, `create`)
- Objekte finden (`find`, `find_by_xxx`)
- Objekte löschen (`destroy`, `delete`)
- Eine einfache View schreiben
- Ausgaben durch HTML Encoding bereinigen (`h`)
- Mit Fehlern und Exceptions umgehen
- Eine View schreiben, die über eine Sammlung von Objekten iteriert
- Ein Layout für ihre Seiten verwenden
- Uff – ich denke, Sie haben eine Pause mehr als verdient.

© des Titels »Professionelle Webentwicklung mit Ruby on Rails 2« (ISBN 978-3-8266-9025-9) 2010
by Verlagsgruppe Hüthig Jehle Rehm GmbH, Heidelberg
Nähere Informationen unter: http://www.it-fachportal.de/9025