Если вы используете систему управления версиями, создайте рабочую ветку, как обычно:

```
$ git checkout master
$ git checkout -b sign-up
```

7.1.1. Отладка и окружение Rails

Профили в этом разделе станут первыми, по-настоящему динамическими страницами в нашем приложении. Хотя представление будет существовать в виде единственной страницы, в каждом профиле будет использована его собственная информация, полученная из базы данных приложения. Готовясь к созданию динамических страниц, добавим в шаблон сайта вывод отладочной информации (листинг 7.1). Вывод этой полезной информации о каждой странице будет осуществляться с помощью встроенного метода `debug` и переменной `params` (мы узнаем о ней больше в разделе 7.1.2).

Листинг 7.1 ❖ Добавление вывода отладочной информации в шаблон сайта (app/views/layouts/application.html.erb)

```
<!DOCTYPE html>
<html>
  .
  .
  .
  <body>
    <%= render 'layouts/header' %>
    <div class="container">
      <%= yield %>
      <%= render 'layouts/footer' %>
      <%= debug(params) if Rails.env.development? %>
    </div>
  </body>
</html>
```

Чтобы отладочная информация не отображалась перед пользователями развернутого приложения, в листинге 7.1 имеется инструкция

```
if Rails.env.development?
```

разрешающая вывод отладочной информации только в *окружении разработки* – одном из трех окружений, по умолчанию определенных в Rails (блок 7.1)[1]. В частности, `Rails.env.development?` возвращает `true` только в окружении разработки, поэтому код на встроенном Ruby

```
<%= debug(params) if Rails.env.development? %>
```

---

[1] Также можно определить собственное окружение; подробности можно найти на сайте RailsCasts (http://railscasts.com/episodes/72-adding-an-environment).

не будет добавляться в развернутое приложение или в тесты. (Вывод отладочной информации в тестах не навредит, но и не даст ничего хорошего, поэтому лучше ограничиться ее выводом только в окружении разработки.)

Блок 7.1 ❖ Окружения Rails

Rails поставляется с тремя настроенными окружениями: `test` (тестовое), `development` (разработки) и `production` (промышленное). Окружением по умолчанию для консоли Rails является окружение разработки:

```
$ rails console
Loading development environment
>> Rails.env
=> "development"
>> Rails.env.development?
=> true
>> Rails.env.test?
=> false
```

Как видите, в Rails имеется объект `Rails` с атрибутом `env` и логическими методами, среди которых имеется метод `Rails.env.test?`, возвращающий `true` в тестовом окружении и `false` в остальных.

Если понадобится запустить консоль в другом окружении (например, для отладки теста), можно передать окружение сценарию `console` в виде параметра:

```
$ rails console test Loading test environment
>> Rails.env
=> "test"
>> Rails.env.test?
=> true
```

Сервер Rails, так же как консоль, по умолчанию выполняется в окружении разработки, и для него тоже можно изменить окружение:

```
$ rails server --environment production
```

Если попробовать запустить приложение в промышленном окружении, оно не будет работать без настроенной базы данных, которую можно создать, выполнив команду `rake db:migrate` в промышленном окружении:

```
$ bundle exec rake db:migrate RAILS_ENV=production
```

(Я считаю, что три разных взаимоисключающих способа переопределения окружения в консоли, на сервере и для команд миграции могут сбить с толку кого угодно, поэтому я потрудился показать здесь все три.)

Кстати, если вы развернули учебное приложение на Heroku, определить его окружение можно командой `heroku run console`:

```
$ heroku run console
>> Rails.env
=> "production"
```

```
>> Rails.env.production?
=> true
```

Естественно, поскольку Heroku является платформой для развертывания сайтов, она запускает каждое приложение в промышленном окружении.

Чтобы вывод отладочной информации выглядел более опрятно, добавим несколько правил в таблицу стилей, созданную в главе 5 (листинг 7.2).

Листинг 7.2 ❖ Добавление правил оформления блока с отладочной информацией, включая примесь Sass (app/assets/stylesheets/custom.css.scss)

```
@import "bootstrap-sprockets";
@import "bootstrap";

/* примеси, переменные и пр.*/

$gray-medium-light: #eaeaea;

@mixin box_sizing {
  -moz-box-sizing:    border-box;
  -webkit-box-sizing: border-box;
  box-sizing:         border-box;
}
.
.
.
/* разное */

.debug_dump {
  clear: both;
  float: left;
  width: 100%;
  margin-top: 45px;
  @include box_sizing;
}
```

Этот код вводит новый для нас инструмент – Sass-примесь, в данном случае box_sizing. Примесь позволяет сгруппировать CSS-правила, чтобы их могли использовать несколько элементов, превращая

```
.debug_dump {
  .
  .
  .
  @include box_sizing;
}
```

в

```
.debug_dump {
  .
  .
```

```
  .
  -moz-box-sizing:    border-box;
  -webkit-box-sizing: border-box;
  box-sizing:         border-box;
}
```

Мы еще раз задействуем эту примесь в разделе 7.2.1. Результат ее применения к блоку с отладочной информацией показан на рис. 7.3.

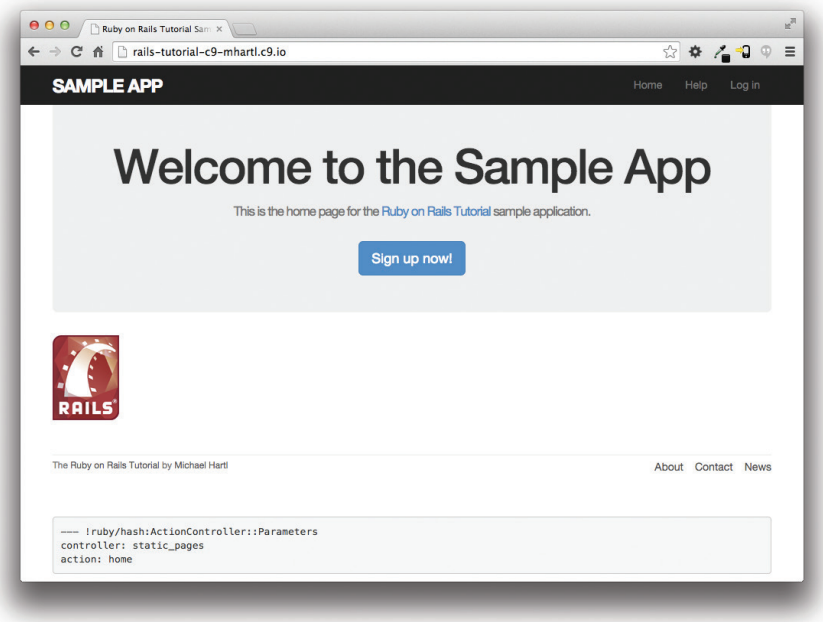

Рис. 7.3 ❖ Главная страница учебного приложения с отладочной информацией

Блок с отладочной информацией на рис. 7.3 содержит потенциально полезные сведения об отображаемой странице:

```
---
controller: static_pages
action: home
```

Это представление `params` на языке YAML[1], который, по сути, является хэшем и в данном случае идентифицирует контроллер и его метод. Еще один пример мы увидим в разделе 7.1.2.

---

[1] Отладочная информация в Rails отображается в формате YAML (рекурсивный акроним от «YAML Ain't Markup Language», – не язык разметки) – дружественного формата данных, удобочитаемого и для компьютеров, и для людей (https://ru.wikipedia.org/wiki/YAML).