02

Customizando controllers, views e rotas

Entendendo o "Hello World"

No primeiro capítulo criamos uma aplicação Rails com um "Hello World" sendo exibido no navegador, e depois geramos uma estrutura de *CRUD* (*create, read, update, delete*) para os *jobs*. Para relembrarmos, o escopo de nossa aplicação é um "Classificado de Empregos", onde empresas podem cadastrar novas vagas de emprego, e candidatos podem interagir com as vagas.

Como estamos trabalhando com uma aplicação web, vamos construir várias páginas diferentes, e apesar de o Rails nos auxiliar com uma série de geradores de código, não podemos nos basear somente neles. Precisamos entender melhor os conceitos por trás do "Hello World" e do *scaffold* que criamos no primeiro capítulo, revisando o código que o Rails gerou para nós e como ele é executado quando alguém utiliza a aplicação.

Quando alguém acessa */hello/world* através do navegador, a aplicação Rails recebe uma requisição com essa informação, e precisa saber que a aplicação reconhece essa url, e para quem deve delegar o trabalho de responder à ela - um *controller*. Este, por sua vez, vai executar qualquer lógica necessária e devolver a página com o conteúdo que alteramos anteriormente - o "Hello World".

O *generator* que utilizamos para criar o código responsável pelo "Hello World" foi:

```
$ rails generate controller hello world
```

Através deste comando, o Rails passou a reconhecer a url */hello/world*. No Rails, o conceito de *urls* é chamado de **rotas (routes)**. Existe um arquivo que descreve todas as urls que a aplicação conhece: *config/routes.rb*.

Rotas e controllers

Abra *config/routes.rb* em seu editor, ele deve ser algo como:

```
JobBoard::Application.routes.draw do
  resources :jobs

  get "hello/world"
  # ... removido
end
```

Nota: Existe uma série de comentários nesse arquivo, começados por `#`, logo abaixo a linha que diz `get "hello/world"`, e que explicam em detalhes o que é possível fazer utilizando as rotas do Rails. Esses comentários foram removidos do exemplo para torná-lo mais simples de visualizarmos aqui.

A linha que contém `get "hello/world"` foi automaticamente inserida quando rodamos o *controller generator* no capítulo anterior, fazendo com que o Rails entenda que a url *hello/world* existe em nossa aplicação. E a linha `resources :jobs` foi adicionada quando executamos o *scaffold generator*, e faz com que o Rails compreenda todas aquelas rotas relacionadas ao *CRUD* de *jobs* que vimos antes: */jobs*, */jobs/new*, etc. Vamos falar mais sobre o código gerado pelo *scaffold* nos próximos capítulos.

Visualizando as rotas

Para entendermos melhor qual é o destino de cada rota, nós podemos utilizar uma task que o Rails disponibiliza para mostrar as rotas disponíveis:

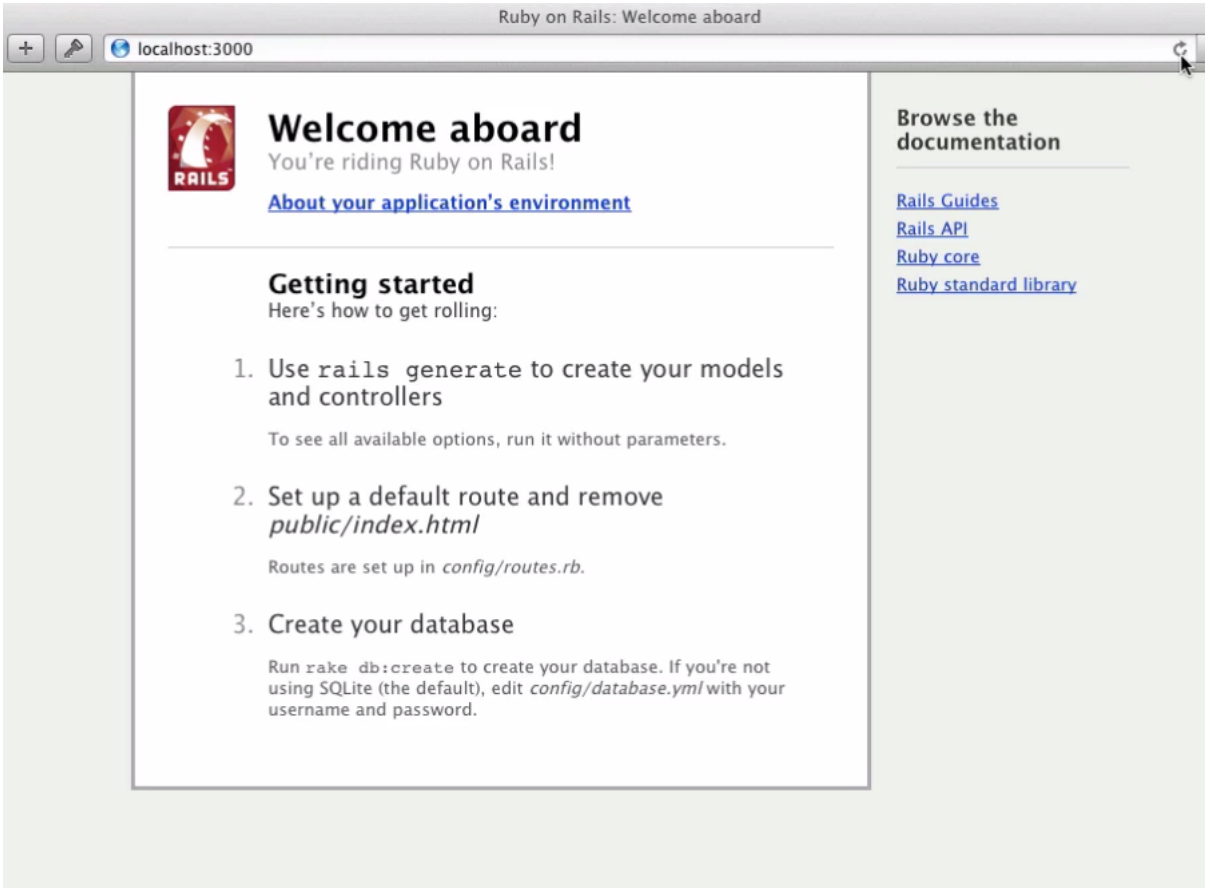
```
$ rake routes
```

Rode-a em seu terminal e você deve ver o seguinte:

```
$ rake routes
       jobs GET    /jobs(.:format)          jobs#index
            POST   /jobs(.:format)          jobs#create
    new_job GET    /jobs/new(.:format)      jobs#new
   edit_job GET    /jobs/:id/edit(.:format) jobs#edit
        job GET    /jobs/:id(.:format)      jobs#show
            PUT    /jobs/:id(.:format)      jobs#update
            DELETE /jobs/:id(.:format)      jobs#destroy
hello_world GET    /hello/world(.:format)   hello#world
```

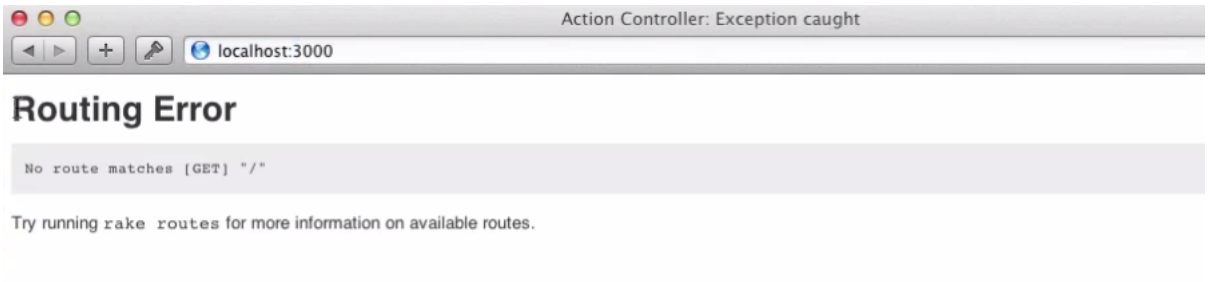
O Rails exibe todas as rotas que ele reconhece na mesma ordem em que aparecem no *config/routes.rb*: as primeiras 7 representam o nosso *CRUD* de *jobs*, e a última é a rota do *hello/world*. As duas últimas colunas são as mais importantes neste momento: elas indicam a url que acessamos no navegador, como por exemplo */jobs* ou */hello/world*, e qual a informação do *controller* que ela está relacionada, como *jobs#index* ou *hello#world*.

Perceba que não existe nenhuma rota para quando acessamos a raiz, também conhecida como *root*, da nossa aplicação. Porém, quando acessamos a página inicial da nossa aplicação, visualizamos a página informativa do Rails que diz *"Welcome aboard"*. De onde que está vindo essa página?



Na verdade, para a página inicial, o Rails utiliza o conteúdo no arquivo *public/index.html*. Para verificar isso, comece novamente a aplicacão no terminal com `rails server` e acesse a página inicial (http://localhost:3000/)http://localhost:3000/ (http://localhost:3000/).

Depois de acessar a página inicial, remova o arquivo *public/index.html* e recarregue a página. Agora o Rails mostra uma página de erro já que a página não pode ser mais encontrada:



Já que removemos a página inicial, agora é uma excelente oportunidade para customizarmos as nossas rotas para mostrar algo mais importante na página inicial, como por exemplo, a lista de jobs (afinal, estamos construindo um classificado de empregos).

Na tabela acima que obtivemos ao rodar `rake routes` , verificamos que a url */jobs* usa *jobs#index* para a mostrar lista de jobs. Logo, queremos usar o mesmo *jobs#index* para mostrar a nossa página inicial. A página inicial é normalmente chamada de **root** e, como todo aplicação possui uma página inicial, o Rails tem um método também chamado **root** para configurar esta rota. Alterando o config/routes.rb, vamos adicionar:

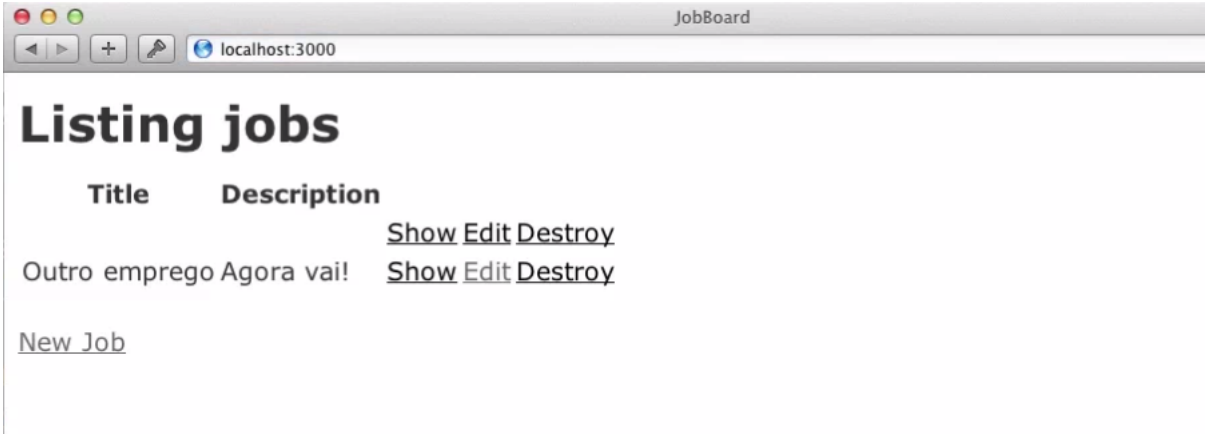
```
root to: "jobs#index"
```

Como aparece aqui:

```
JobBoard::Application.routes.draw do
  root to: "jobs#index"

  resources :jobs
  get "hello/world"
end
```

Agora acesse a url (http://localhost:3000)http://localhost:3000 (http://localhost:3000) novamente no navegador, você deve ver a mesma página de listagem de *jobs* que vimos no capítulo anterior, como essa:



Maravilha, agora a listagem de jobs pode ser acessada diretamente da raiz da nossa aplicação! Porém, como o nome diz, as rotas apenas mandam uma dada url para um controller, que é o responsável por gerar uma resposta para a nossa requisição. E é exatamente sobre eles que vamos falar em seguida.

Controllers e actions

Os *controllers* ficam localizados no diretório *app/controllers*, e são responsáveis por responder a uma requisição. Abra o *hello_controller.rb*:

```
class HelloController < ApplicationController
  def world
  end
end
```

Observando este código, é possível identificar os nomes **hello** e **world**, os mesmos argumentos que passamos ao executar o **generator**. Para entendermos melhor o que este *controller* faz, precisamos compreender primeiro alguns conceitos e conhecer um pouco mais sobre o Ruby.

Ruby

Para podermos testar algumas funcionalidades do Ruby vamos utilizar o **irb** - *Interactive Ruby Shell* - onde podemos escrever qualquer comando Ruby e verificar os resultados. Digite em seu terminal:

```
$ irb
```

Você estará no *shell* do Ruby. Digite o seu nome entre aspas, e pressione *Enter / Return*:

```
>> "carlos"
=> "carlos"
```

Nota: as linhas identificadas com `>>` mostram os comandos digitados, e as linhas identificadas com `=>` são os resultados exibidos pelo *irb* - a cada comando executado o resultado é exibido, algo extremamente útil para fazermos testes rápidos e aprendermos mais sobre a linguagem.

Isso vai criar uma `String`, um texto que podemos manusear de várias formas. Podemos executar algumas operações com essa String, como convertê-la para maiúsculo:

```
>> "carlos".upcase
=> "CARLOS"
```

Essas operações são conhecidas como **métodos (methods)**, ou seja, estamos chamando o *método* **upcase** na String "carlos". Podemos por exemplo usar o **método** `class` para identificar o tipo de objeto que estamos lidando:

```
>> "carlos".class
=> String
```

E é possível por exemplo exibir a data/hora atual desta forma:

```
>> Time.now
=> 2012-07-10 10:37:26 -0300
```

Aqui estamos chamando o método `now` em `Time`. O `Time` é disponibilizado pelo Ruby e engloba todo o comportamento necessário para trabalhar com data/hora, da mesma forma que `String` possui o comportamento para trabalhar com textos. Esse conjunto ou coleção de comportamentos relacionados é conhecido como `classe`, e faz parte do conceito de **Orientação a Objetos** (OO). Não vamos tratar diretamente sobre OO aqui, mas vamos aprender alguns conceitos relacionados e aplicá-los. Utilizamos então as classes `String` e `Time`.

Normalmente precisamos criar nosso próprio conjunto de comportamentos específico, e para isso criamos nossas próprias classes e métodos. Por exemplo, vamos definir uma classe que descreva uma **Pessoa** com um **nome**, e um método ou comportamento que exiba o texto "Olá" e mais esse nome:

```
class Person
  attr_accessor :name

  def say_hello
    "Olá #{name}"
  end
end

person = Person.new
person.name = "Pedro"
person.name #=> "Pedro"
person.say_hello #=> "Ola Pedro"
```

Após definir a classe `Person`, contendo um atributo `name` e o método `say_hello`, criamos uma **instância** (ou objeto) a partir dessa classe usando `Person.new` - uma nova "Pessoa", e dizemos que essa instância recebe o nome *"Pedro"* - que por sua vez é uma instância da classe `String`. Quando chamamos `person.name`, recebemos de volta o nome definido *"Pedro"* - o `#` significa um comentário em ruby, e `#=>` indica a saída do comando que executamos, no caso `person.name`. E quando executamos o método `person.say_hello`, recebemos de volta o "Ola Pedro". Poderíamos criar quantas instâncias / objetos do tipo `Person` quiséssemos, cada um com um nome diferente.

Também podemos dizer que uma classe pode ter como base o comportamento de outra classe, ou seja, pode herdar comportamento, o que é chamado de **herança**. Por exemplo, podemos dizer que um **Usuário** possui o comportamento de uma *Pessoa*, como ter um nome, e também um atributo adicional **login**:

```
class User < Person
  attr_accessor :login
end

user = User.new
user.name = "Maria de Souza"
user.login = "mariadesouza"
user.say_hello #=> "Ola Maria de Souza"
```

`User` é uma **subclasse** de `Person`, como é mostrado com a definição `class User < Person`, e permite que ele herde todas as funcionalidades que `Person` possui, como o atributo `name` e o método `say_hello`, e adicione quaisquer

comportamentos que ele mesmo definir, como ter o atributo `login`.

HelloController

Voltando ao *controller*, agora conseguimos identificar que existe uma classe chamada `HelloController` que define a funcionalidade desse *controller*, e que ele possui um método **world**. No contexto de *controllers*, métodos são também conhecidos como **actions**. Além do mais, podemos notar que o `HelloController` herda de `ApplicationController`.

Um *controller* se preocupa basicamente em responder a requisição após a url ser reconhecida, e então devolver uma resposta, como retornar uma página HTML. Isso é uma funcionalidade em comum para todos os *controllers* da aplicação, e seria impráticavel ter que codificar estes detalhes em todos eles.

Esse é o motivo principal de existir o `ApplicationController`: ele é a classe base para todos os *controllers*, que permite que funcionalidade em comum seja compartilhada através de herança. E ele também herda de um dos componentes do Rails, o `Action Controller`, que gerencia a maior parte da lógica de requisição e resposta para nós, possibilitando que possamos focar no conteúdo e na navegação entre as páginas.

Vamos dar uma olhada no código do `ApplicationController`, abra o arquivo *app/controllers/application_controller.rb*:

```
class ApplicationController < ActionController::Base
  protect_from_forgery
end
```

Em resumo, temos a seguinte estrutura de classes para o `HelloController`:

```
HelloController < ApplicationController < ActionController::Base
```

Ganhamos funcionalidades "de graça" simplesmente usando esse componente do Rails através de herança, ótimo não?

Nota: a linha `protect_from_forgery` é relacionada a uma funcionalidade de segurança do Rails, que será discutida posteriormente.

Views

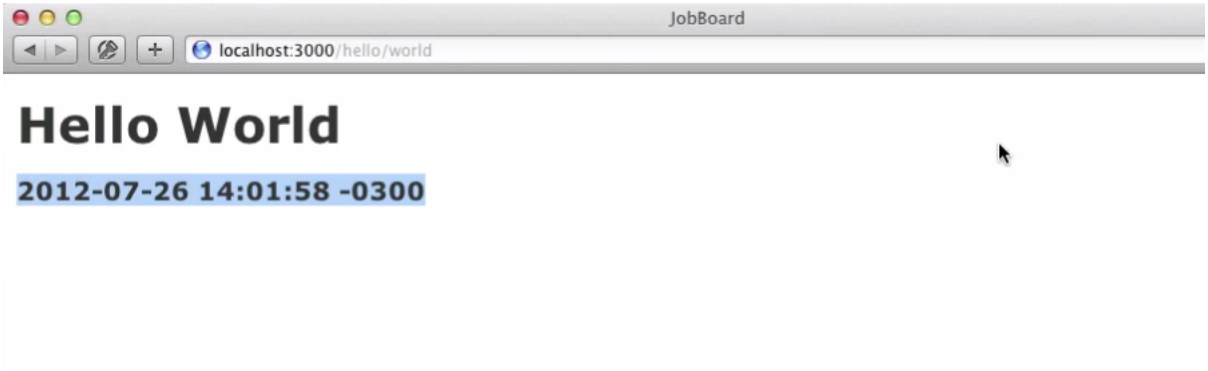
Até agora, verificamos que ao acessar a url */hello/world*, através da definição da rota o *controller Hello* será chamado, e a *action world* será executada. Essa *action* por **padrão** exibirá o arquivo HTML relacionado à ela, que está localizado em *app/views/hello/world.html.erb*. Esses arquivos são conhecidos como **templates** ou **views**, como o próprio nome do diretório diz: *app/views*. Repare como o caminho da *view* também remete a mesma estrutura *hello/world*.

Para entender melhor como as views funcionam e fechar o ciclo do nosso aprendizado, nós vamos fazer duas modicações: adicionar um pouco de conteúdo dinâmico no **hello world**, e logo em seguida adicionar um link na listagem de jobs apontando para essa página.

Para tal, abra o arquivo *app/views/hello/world.html.erb* e adicione o seguinte:

```
<h1>Hello World</h1>
<strong><%= Time.now %></strong>
```

Vamos checar como fica no navegador, acessando a url (http://localhost:3000/hello/world)http://localhost:3000/hello/world (http://localhost:3000/hello/world):



Perceba que o horário atual é mostrado em negrito. Além do mais, se você carregar a mesma página novamente, o tempo mostrado será atualizado, ou seja, a nossa página está dinâmica!

Note que para mostrar o tempo, nós simplesmente usamos código Ruby, neste caso, `Time.now` entre as tags `<%= %>`. Essas tags são chamadas de **ERB**, que significa *Embedded Ruby*, ou *Ruby Embedado*. Observe que a extensão do arquivo também é **ERB** e é dessa forma que o Rails sabe como mostrar o conteúdo dinâmico.

Sucesso! Conseguimos adicionar código dinâmico Ruby no nosso template.

Para finalizar o nosso capítulo, vamos tomar um desafio um pouco maior. Nós já configuramos a nossa aplicação para mostrar a lista de jobs na página inicial, porém, a partir da página inicial, não existe nenhuma forma para acessarmos a nossa página de *hello world*, que acabamos de customizar. É este o problema que vamos resolver em seguida.

Rotas nomeadas

Quando adicionamos a rota *root* da nossa aplicação, nós apontamos essa rota para *jobs#index*. Depois descobrimos que requisições são respondidas por pares *controllers* e *actions*. Nesse caso, a rota *root* é respondida pelo *controller jobs* e a *action index*.

Como tudo no Rails segue uma convenção, da mesma forma que o controller e action *hello#world* mostram o template em *app/views/hello/world.html.erb*, podemos imaginar que *jobs#index* mostrará a view *app/views/jobs/index.html.erb*. Abra esse arquivo no seu editor e você verá o seguinte conteúdo:

```
<h1>Listing jobs</h1>

<table>
  <tr>
    <th>Title</th>
    <th>Description</th>
    <th></th>
    <th></th>
    <th></th>
  </tr>

<% @jobs.each do |job| %>
  <tr>
    <td><%= job.title %></td>
    <td><%= job.description %></td>
    <td><%= link_to 'Show', job %></td>
```

```
    <td><%= link_to 'Edit', edit_job_path(job) %></td>
    <td><%= link_to 'Destroy', job, method: :delete, data: { confirm: 'Are you sure?' } %></td>
  </tr>
<% end %>
</table>

<br />

<%= link_to 'New Job', new_job_path %>
```

Maravilha! O nosso palpite estava correto, essa página é definitivamente a nossa página inicial, como podemos observar pelo cabeçalho *Listing jobs* na primeira linha.

Com o arquivo aberto, para adicionar um link para a página de *hello world*, precisamos apenas adicionar o seguinte ao final do arquivo:

```
<br />
<%= link_to 'Hello World', hello_world_path %>
```

Acesse */* novamente no navegador e você deverá ver agora um link *Hello World*, que aponta de volta para a página *Hello World*.

Note que, para gerar o link, nós utilizamos **ERB** e um método chamado `link_to`. Esse método é definido pelo Rails e ele recebe dois argumentos: o texto do link e o endereço do link. Porém veja que não utilizamos em nenhum momento */hello/world* como endereço, simplesmente utilizamos o método `hello_world_path`. Neste caso, `hello_world_path` é chamado de **rota nomeada**, ou **named route**, e foi criado automaticamente pelo Rails para resolver um problema comum em muitas aplicações web: duplicação de urls.

Imagine que nós precisamos informar a mesma url em vários lugares - como links em páginas diferentes, corpo de emails enviados pelo aplicativo, etc, e depois de algum tempo surge a necessidade de modificar essa url para uma outra qualquer. Com certeza seria um grande problema, pois teríamos que procurar em todo o projeto e possivelmente modificar vários arquivos para corrigir a url. Se esquecermos de modificar algum arquivo (o que é bastante provável dependendo do tamanho da aplicação), a nossa aplicação estaria "quebrada" do ponto de vista de usuário quando ele clicar no link antigo.

Por esse motivo o Rails contém o conceito de **rotas nomeadas**, ou **named routes**. Por exemplo, quando quisermos nos referir a */hello/world*, não vamos digitá-lo manualmente, mas usuremos um método pré-definido, `hello_world_path`. Ou quando quisermos nos referir ao caminho */jobs*, usaremos `jobs_path`. Ao longo do curso veremos diversas conveniências disponibilizadas pelo Rails para evitar que a gente se repita, e por isso dizemos que o Rails adota **DRY - Don't Repeat Yourself** (ou **não se repita**) como filosofia.

Além disso, existe uma diferença importante entre **url** e **path**. Quando nos referimos ao primeiro, sempre queremos dizer a url *completa*, ou seja, algo como *http://localhost:3000/jobs (http://localhost:3000/jobs)* ou *http://localhost:3000/hello/world (http://localhost:3000/hello/world)*. Já com o segundo, falamos sobre o **caminho** dentro de um mesmo domínio, ou seja, apenas na parte da url que está abaixo de *http://localhost:3000 (http://localhost:3000)* (neste exemplo), como */jobs* ou */hello/world*. O Rails também segue esta ideia, e disponibiliza as duas possibilidades de rotas nomeadas: `hello_world_path` e `hello_world_url`, por exemplo. Normalmente dentro das aplicações utilizamos apenas as rotas nomeadas com final `_path`.

Finalmente, é possível verificar quais são as rotas nomeadas na nossa aplicação em qualquer momento usando o comando `rake routes`, que vizualizamos a pouco tempo atrás (execute-o novamente em seu terminal se necessário):

```
$ rake routes
       root GET    /                          jobs#index
       jobs GET    /jobs(.:format)            jobs#index
            POST   /jobs(.:format)            jobs#create
    new_job GET    /jobs/new(.:format)        jobs#new
   edit_job GET    /jobs/:id/edit(.:format)   jobs#edit
        job GET    /jobs/:id(.:format)        jobs#show
            PUT    /jobs/:id(.:format)        jobs#update
            DELETE /jobs/:id(.:format)        jobs#destroy
hello_world GET    /hello/world(.:format)     hello#world
```

Vamos olhar para a primeira coluna agora: ela nos mostra as *named routes* que o Rails gera automaticamente (aqui mostrada sem o `_path` ou `_url`). Ou seja, `hello_world_path` é a rota nomeada para */hello/world*, que aponta para o *controller* e *action* `hello#world`.

Da mesma forma, temos as rotas nomeadas `jobs_path` e `jobs_url`, que geram */jobs* e apontam para o *controller* e *action* `jobs#index`.

Agora estamos prontos para adicionar navegação na nossa aplicação, sem duplicar urls manualmente. E com isso concluímos o segundo capítulo.

Para saber mais

-
- Cheque os comentários do arquivo **config/routes.rb**, para aprender mais sobre as rotas no Rails
-
- Neste capítulo, descobrimos o irb do Ruby. Existe uma versão online do irb do Ruby chamada Try Ruby (http://tryruby.org), que também funciona como um tutorial. Visite o site e aprenda mais sobre o Ruby!