01

Generalizando comportamentos através do method_missing

Transcrição

[00:00] Então repara que os três métodos: livro¬*que*mais_vendeu_por_titulo, livro-*que*mais_vendeu_por_ano, livro¬*que*mais_vendeu_por_editora são muito parecidos. Só muda o campo.

[00:11] Então eu vou adicionar um campo: livro¬*que*mais_vendeu_por um campo determinado. Ao invés de passar sempre o título, passamos sempre o campo que você pedir pra mim.

[00:23] Lembrando que esse campo vai ser um lambda que vamos passar, removemos esses dois métodos e fazemos o mesmo serviço com a revista, refatora pra extrair o único método que recebe o campo, passa esse campo para frente e apaga as duas outras variações do método.

[00:37] Ficamos só com duas versões do método: para o livro e para a revista. Quando invocamos isso no nosso sistema rb, temos que passar a utilizar o parâmetro. Então vamos passar o parâmetro título. Rodamos a aplicação e está tudo rodando direitinho, como esperávamos.

[01:01] Voltando ao nosso estoque, reparem que o método "livro_que_mais_vendeu_por" e "revista_que_mais_vendeu_por" ainda é parecido, mas tudo bem, vamos ver o que acontece agora.

[01:07] Nossa livraria passa a vender também, uma versão eletrônica do livro de "Introdução a Arquitetura e Design de Software".

[01:16] Então eu tenho um livro chamado "online_arquitetura", ele é um livro de "Introdução a Arquitetura e Design de Software", tenho o preço, o ano de lançamento, se ele possui reimpressão, a editora, e ele é um ebook. Não é uma revista e nem um livro: é um ebook.

[01:36] Vamos adicionar ele ao estoque e vamos vender ele. Por fim, eu quero imprimir agora, o ranking, o nosso ranking de ebooks. Vamos lá no estoque e fazer um "copy paste" desse método.

[01:44] Então neste caso, "ebook_que_mais_vendeu_por". Tudo que é muito "copy paste", percebemos que tem algo de estranho. Mas e se eu não criar esse método, o que acontece? Eu vou apagar ele e vou rodar o nosso programa.

[02:07] Repara que o ruby fala para nós: um método, "ebook¬*que*mais_vendeu_por", não existe. "NoMethodError". Ele procurou por um método, mas não encontrou. Quando um método não existe no nosso objeto, o ruby chama um método chamado: "method_missing", isto é, o método não existe. E passa para nós o nome desse método.

[02:26] Então eu vou imprimir nesse caso, o nome do método que ele está passando pra nós. Rodamos o programa novamente e reparem, ele imprimiu: "Não encontrei: ebook_que_mais_vendeu_por".

[02:38] Ele procurou o método "ebook que mais vendeu por", mas não encontrou ele, invocando então o "method missing".

[02:47] Então temos a oportunidade de fazer alguma coisa agora. Vamos pegar esse símbolo, que é o "name.", transformar em uma string com "to_s", e tentar perceber se ele é uma chamada ou método "que_mais_vendeu_por" título, editora ou ano de lançamento. Nesses casos não, por qualquer campo. Eu quero saber se esse método é o método de quem mais vendeu por qualquer coisa.

[03:14] E eu quero saber se ele é revista, ebook, se qualquer coisa que mais vendeu por um campo determinado. Se esse método que está sendo invocado, é desse tipo, eu vou fazer alguma coisa. Senão, eu vou simplesmente delegar, e falar "eu não achei o método".

[[03:37] Se for um método "revista_que_mais_vendeu_por_titulo", eu preciso saber qual é o tipo. É revista, é livro, é ebook, esse é o tipo. E eu preciso saber o campo, se é título, ano de lançamento, é o preço, lembrando que o campo eu vou transformar em um símbolo.

[03:56] O que eu chamo é: o método "que_mais_vendeu_por", esse tipo, esse campo, lembrando de transformar o campo na chamada do lambda.

[04:04] Só recapitulando, eu verifico se o método tem um padrão determinado, que é o padrão que eu estou esperando, extraio o tipo, isto é: livro, revista e ebook. Extraio o campo: título, ano de lançamento, preço e etc.

[04:15] E faço a invocação. Caso o contrário, simplesmente delega, para o object. Pronto, posso apagar o método da revista e o método do livro. Mudo as nossas invocações, para deixar de passar o campo como parâmetro, e simplesmente chamar o método, executo o programa e repara: está tudo funcionando.

[04:32] Reparem que esses dois métodos, "que_mais_vendeu_por" e "quantidade_de_vendas_por", são métodos que não são chamados de fora da minha classe. Eu vou jogar eles para um escopo privado, para não serem invocados por fora.

[04:50] Rodando a aplicação, percebemos que está tudo certo de novo. Mas repare que todo o objeto também tem um método pra saber se ele responde ou não para uma mensagem. Por exemplo, "ebook_que_mais_vendeu_por_titulo".

[05:02] Se imprimirmos isso, no nosso sistema vamos ver que ele responde com "false", como se eu não fosse capaz de responder por esse método. Por que isso? Porque implementamos um método de missing, mas não implementamos o método "respond_to".

[05:20] Toda a vez que implementamos um método de "missing", pra mudar o comportamento de métodos que respondemos, precisamos subscrever também um método "respond_to".

[05:27] No nosso caso, nós vamos verificar se a chamada de método for equivalente ao método que esperamos, ou se a nossa classe pai responde por esse método, então vamos devolver algo que tenha valor verdadeiro.

[05:43] Repara, se rodarmos o programa novamente, ele imprime o nome do método, porque existe sim esse método.

[05:49] Então, quando você gostaria de agir de maneira diferente de acordo com o nome do método que foi invocado, e você não sabe todos os nomes dos métodos, ou você quer suportar infinitos nomes de métodos possíveis, você pode subscrever o "method_missing", lembrando de sempre tomar cuidado e subscrever também o método "respond_to".