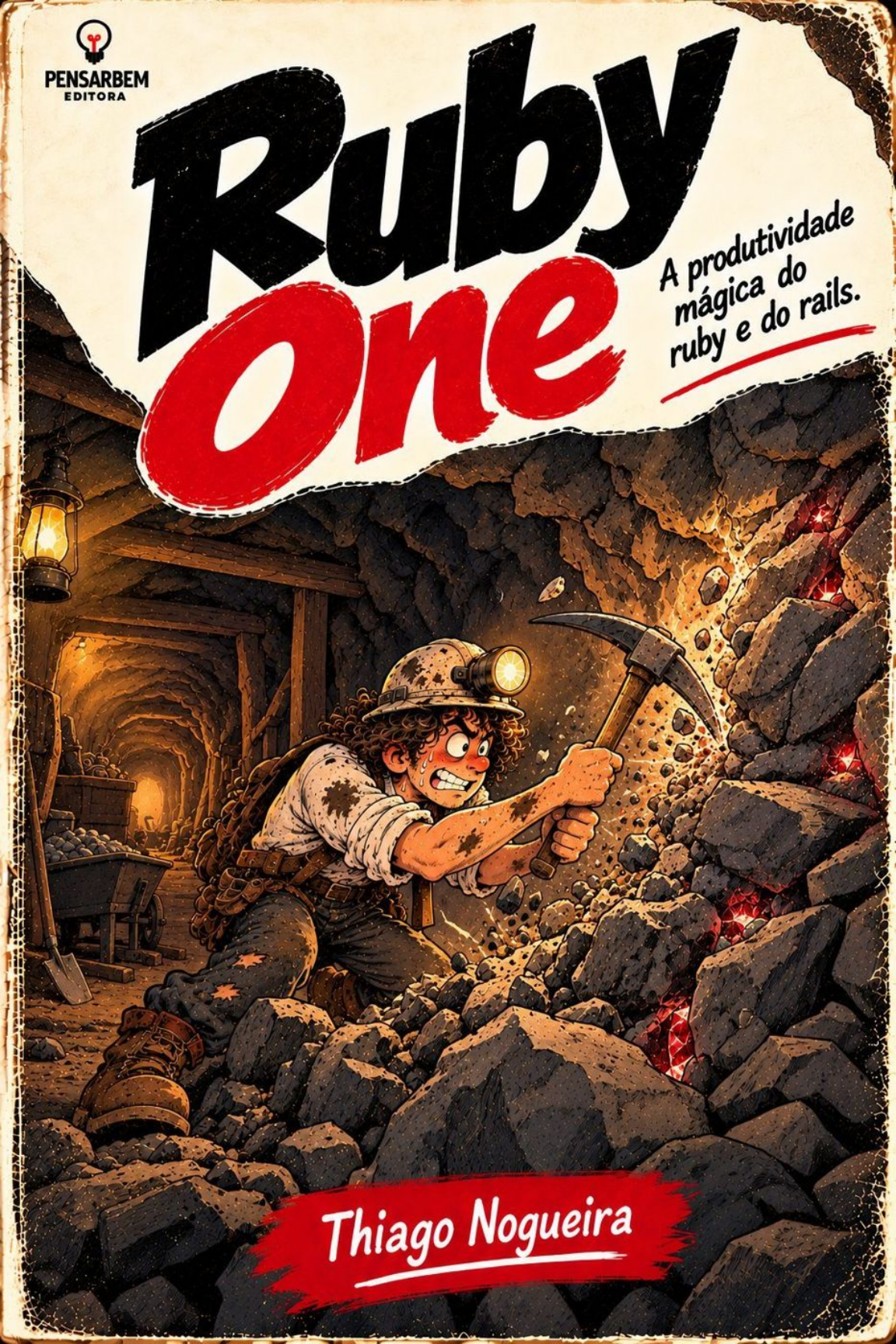



PENSARBEM
EDITORA

Ruby One

*A produtividade
mágica do
ruby e do rails.*



Thiago Nogueira

Ruby One — Prévia

Do primeiro puts ao contrato em produção

Thiago Nogueira

PENSARBEM

Ruby One — Prévia

© 2026 Thiago Nogueira

Pensarbem

Prévia · 1ª edição

Composto em IBM Plex Serif, IBM Plex Sans e IBM Plex Mono. Diagramado por uma pipeline de tipografia programática.

Sumário

	Antes de começar	1
	A Norte e a data	3
	Quem aparece	4
	Como o código aparece	4
1	Fevereiro de 1993	9
1.1	24 de fevereiro de 1993	11
1.2	A saída do Japão	12
1.3	Basecamp, Merb, e duas máquinas virtuais	13
1.4	O calendário de Natal	14
2	O que vamos construir	18
2.1	O que o dia 3 de junho confere	22
2.2	Instalando	22
3	O primeiro programa	25
3.1	O irb	27
3.2	Trocar o valor	27
3.3	Três falhas da primeira hora	27
3.4	Comentário	27
4	Objetos e tipos	31
4.1	Tudo é objeto	33
4.2	Texto somado com texto	33
4.3	Dinheiro não é <code>Float</code>	33
4.4	Divisão entre inteiros	33
4.5	Mostrar em reais	33
5	Métodos	34
5.1	<code>def</code>	37
5.2	O método do contrato, fora do Rails	37
5.3	O <code>?</code> no nome	37
5.4	Parêntese opcional	37
5.5	Valor padrão e argumento com nome	37
5.6	Ausência de chamada no arquivo	37
6	Condicionais	38
6.1	<code>if</code> , <code>elsif</code> , <code>else</code>	41
6.2	Dois <code>if</code> não são uma escolha	41
6.3	O que é falso	41
6.4	<code>unless</code> , e o <code>if</code> no fim da linha	41

6.5	<code>case</code>	41
7	Strings e símbolos	42
7.1	String	45
7.2	O <code>!</code> e a string que muda no lugar	45
7.3	O texto escrito entre aspas vai congelar	45
7.4	Símbolo	45
7.5	<code>:active</code> não é <code>"active"</code>	45
7.6	O ponto de milhar	45
8	Arrays e hashes	46
8.1	Array: a lista da manhã	49
8.2	Hash: o contrato	49
8.3	<code>fetch</code> : a chave que tem de existir	49
8.4	Aninhado, e o <code>dig</code>	49
8.5	Dois contratos, uma forma	49
9	Blocos e enumerables	50
9.1	O bloco	54
9.2	<code>yield</code> : o lado de quem recebe	54
9.3	<code>each</code> : percorrer	55
9.4	<code>map</code> , <code>select</code> , <code>reject</code> , <code>find</code>	55
9.5	Atalhos: <code>&</code> : e <code>it</code>	56
9.6	Somar, contar, ordenar, agrupar	57
9.7	O filtro e o <code>nil</code>	57
10	Classes	63
10.1	<code>class</code> e <code>new</code>	65
10.2	<code>initialize</code> e a variável de instância	65
10.3	Ler de fora: <code>attr_reader</code>	65
10.4	Métodos que perguntam ao objeto	65
10.5	Duas instâncias, mesma classe	65
10.6	Um construtor de classe: <code>self.</code>	65
11	Módulos	66
11.1	Um comportamento, duas classes	69
11.2	Um módulo que o Ruby já inclui	69
11.3	Onde o Ruby procura um método	69
11.4	<code>super</code> : estender em vez de substituir	69
11.5	Módulo como sobrenome	69
12	Exceções	70
12.1	A falha tem tipo	73
12.2	<code>begin</code> , <code>rescue</code>	73
12.3	O <code>rescue</code> sem tipo	73
12.4	<code>raise</code> e a exceção da Nortea	73

12.5	<code>ensure</code>	73
13	Gems e Bundler	74
13.1	Gem	77
13.2	O problema que o <code>gem install</code> não resolve	77
13.3	<code>Gemfile</code> e <code>Gemfile.lock</code>	77
13.4	O <code>~></code>	77
13.5	<code>bundle install</code> e <code>bundle update</code>	77
13.6	Dois Rubys na mesma máquina	77
14	O que é o Rails	81
14.1	Framework	83
14.2	As pastas	84
14.3	Subir o servidor	84
14.4	Um pedido, do endereço à página	85
14.5	O que o nome decide	86
14.6	Ver as rotas	86
15	Modelos	88
15.1	<code>class Contract < ApplicationRecord</code>	91
15.2	As colunas não estão no model	91
15.3	Buscar	91
15.4	A pluralização e a tabela de equipamentos	91
15.5	O que o Caio digitou	91
16	Migrations	92
16.1	O que uma migration é	95
16.2	Rodar, e onde o banco guarda o que já rodou	95
16.3	O <code>schema.rb</code> é gerado	95
16.4	<code>change</code> , <code>up</code> e <code>down</code>	95
16.5	Os 212 sem status	95
16.6	As três datas ficam	95
17	Associações	96
17.1	<code>belongs_to</code> e <code>has_many</code>	99
17.2	Navegar	99
17.3	Escopo: o filtro com nome	99
17.4	Os disponíveis, de novo	99
17.5	E a reserva de maio	99
18	Validações	100
18.1	<code>validates</code>	103
18.2	<code>valid?</code> , <code>errors</code> , e o <code>save</code> que devolve falso	103
18.3	Em português	103
18.4	O período	103
18.5	A mesma máquina, duas vezes	103

18.6	Os caminhos que passam por cima	103
19	Callbacks	104
19.1	O que é um callback	107
19.2	A ordem	107
19.3	Reproduzir antes de corrigir	107
19.4	A correção: observar a mudança	107
19.5	O que não mora num callback	107
20	Rotas	111
20.1	Um pedido é verbo e caminho	113
20.2	<code>config/routes.rb</code>	113
20.3	O cancelamento com nome	113
20.4	<code>collection</code> e rota aninhada	113
20.5	Nome de rota, não endereço montado	113
21	Controllers	114
21.1	O <code>create</code> de 2021	117
21.2	O que é do controller	117
21.3	Primeiro: o cancelamento sai	117
21.4	Segundo: o <code>permit!</code> sai	117
21.5	Terceiro: o <code>begin</code> sai	117
21.6	<code>render</code> e <code>redirect_to</code>	117
21.7	Quarto: o aviso sai	117
21.8	O que sobrou	117
22	Views	118
22.1	ERB	121
22.2	Percorrer a lista	121
22.3	O que o <code><%= %></code> escapa	121
22.4	Partial	121
22.5	A mesma pergunta, num lugar só	121
22.6	Os erros do formulário	121
23	Console	122
23.1	O <code>irb</code> com o <code>nortea</code> dentro	125
23.2	O que lê e o que grava	125
23.3	O log diz o que a linha fez	125
23.4	O registro certo	125
23.5	Sandbox	125
23.6	O que foi feito com o <code>3197</code>	125
24	Testes	129
24.1	O que um teste é	132
24.2	Factory	133
24.3	O vermelho	134

24.4	A factory honesta	134
24.5	A reserva dupla, de novo	136
25	Jobs	138
25.1	O que não cabe no pedido	141
25.2	Job	141
25.3	O worker	141
25.4	Agendar depois do COMMIT	141
25.5	Falhar e tentar de novo	141
25.6	O job que roda duas vezes	141
26	APIs	142
26.1	API	145
26.2	O contrato antes do código	145
26.3	A resposta com nome	145
26.4	O controller da API	145
26.5	A lista, e o 422	145
27	Autenticação	146
27.1	Quem é e o que pode	149
27.2	A Helena entra	149
27.3	A Serra Azul consulta	149
27.4	Só os próprios contratos	149
27.5	O teste do 403	149
28	Deploy	150
28.1	O que tem de ser igual	153
28.2	A esteira	153
28.3	A ordem	153
28.4	Voltar	153
28.5	/up	153
28.6	O horário	153
29	Projeto final	154
29.1	O que o módulo cobre	157
29.2	A lista da manhã	157
29.3	A consulta da Serra Azul	157
29.4	O que fica para depois	157
	Respostas	158
	Glossário	165

Antes de começar

*Três colunas para o começo de um contrato,
uma planilha que ainda manda no pátio, e uma
multa de R\$ 1.800 por dia quando a tela mente.*

Esta é a pasta do sistema da Nortea, no ar desde março de 2016:

```
1  nortea/  
2    Gemfile  
3    Gemfile.lock  
4    app/models/contract.rb  
5    app/models/equipment.rb  
6    app/controllers/contracts_controller.rb  
7    app/views/contracts/  
8    db/schema.rb  
9    spec/factories/contracts.rb
```

O `Gemfile` lista as bibliotecas do projeto e as versões. Uma delas é o Rails, o framework em que esse sistema foi escrito. O `Gemfile.lock` grava a versão exata que foi instalada, para o servidor e a máquina de alguém não “atualizarem” sozinhos no meio do caminho.

O `schema.rb` é o arquivo em que o Rails descreve as tabelas. A tabela `contracts` tem vinte e três colunas. Três delas guardam um começo:

```
start_date  
begin_date  
started_at
```

Uma coluna é um campo da tabela. Três campos para a mesma pergunta — quando o contrato começa — significa que o sistema respondeu essa pergunta três vezes, e não apagou as respostas antigas.

E funciona.

Cerca de quatrocentas e trinta máquinas, o pátio carregando às 6h. A Helena confere a tela. O Seu Nestor confere a planilha `patio_SEMANAL.xlsx`. Quando as duas discordam, o caminhão sai pela planilha. No último trimestre a Nortea pagou cerca de R\$ 46 mil de multa porque uma máquina contratada não estava no canteiro. A cláusula cobra R\$ 1.800 por dia.

Você vai escrever o módulo de contratos que merece ser comparado com essa planilha. A pasta de produção fica onde está.

A Nortea e a data

A **Nortea Equipamentos**, em Betim, aluga máquina para construtora: betoneira, gerador, plataforma elevatória, compactador. A **Construtora Serra Azul**, em Belo Horizonte, nove canteiros, renova o contrato em **quarta-feira, 3 de junho**. São R\$ 3,6 milhões por ano.

O adendo, que o comercial já assinou, pede uma página em que a Serra Azul veja qual equipamento está preso a qual contrato. Hoje essa resposta sai por telefone, ou pela planilha.

Quem aparece

Lívia entrou na Nortea em 2 de março. O acesso ao repositório chegou na sexta, dia 6. É boa tecnicamente. A convenção da casa ela ainda está lendo.

Caio escreve Ruby há anos e já leu o `contract.rb` inteiro, o que explica a calma. Fala baixo, e às vezes resolve o problema sem falar.

Renato é o tech lead. Quer o código simples. Simples, para ele, é o que a Marta consegue conferir sem uma aula.

Marta é a product manager. Conhece o contrato e o pátio. Abstração que não muda a manhã da Helena ela devolve.

Diego é o QA. O caso que ele traz é uma combinação: cancelar no meio da reserva, devolver antes de sair, equipamento sem responsável.

Sérgio mantém pedaços do sistema de 2016. Quando ele diz que uma solução é temporária, pode contar com ela por uns cinco anos.

Helena Noronha toca a operação. Não escreve Ruby. Sabe em qual canteiro está cada plataforma.

Seu Nestor fundou a Nortea. Está no pátio às 6h. A planilha é o sistema em que ele confia.

Rômulo fechou o adendo de 3 de junho. Aparece quando a data que ele assinou precisa virar tela.

E o **Rails de 2016**, com três começos na mesma tabela, é o sistema em produção. Cada coisa nova vai ser medida contra ele e contra a planilha.

Como o código aparece

Código aparece assim, às vezes com o nome do arquivo:

CONTRATO.RB

ruby

```
codigo = "CT-2041"  
puts "Contrato #{codigo}"
```

Um nome em minúscula, com `=`, guarda um valor. `puts` escreve esse valor no terminal e pula a linha. Dentro de aspas duplas, `#{codigo}` é trocado pelo valor do nome. Fora das chaves, o texto sai como foi escrito.

O que o terminal responde aparece sem nome de arquivo e sem realce:

```
Contrato CT-2041
```

Quando o programa quebra, a primeira linha diz o arquivo, a linha e o tipo da falha. É essa que se lê primeiro.

- **O ESSENCIAL**

Comando de terminal aparece com \$ na frente. O \$ representa o prompt e não faz parte do comando: não digite.

Você não precisa instalar nada para começar a ler. Quando o primeiro programa precisar rodar, a instalação vem junto.

A data que ninguém pode empurrar é 3 de junho. Até lá, o pátio carrega às 6h, e o caminhão sai pela planilha quando a tela discorda.

PARTE 1

O que já está no ar

De onde vieram o Ruby e o Rails, o que o dia 3 de junho cobra do contrato, e o primeiro programa na máquina de quem acabou de receber o acesso.

NESTA PARTE

- 1 Fevereiro de 1993
- 2 O que vamos construir
- 3 O primeiro programa

CAPÍTULO

1

Fevereiro de 1993

Um programador no Japão, um critério de leitura, e vinte anos até o framework que a Nortea já usa de manhã.

“I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python. That’s why I decided to design my own language.”

YUKIHIRO MATSUMOTO, ENTREVISTA A BILL VENNERS, 29 DE SETEMBRO DE 2003

AO FIM DESTES CAPÍTULO

Contar a história documentada do Ruby e a do Rails: o desenho de 1993, a saída do Japão, o Basecamp, o Merb, o YARV e o calendário de Natal que ainda marca a versão do servidor da Nortea.

ENQUANTO ISSO, NA EMPRESA Por que Ruby?

O Rômulo tinha almoçado com um fornecedor na quarta. Na quinta parou na sala.

— Esse módulo de contratos. Por que Ruby?

— Porque é o que está no ar — disse a Livia. — O Sérgio me falou. Desde 2016.

— O pessoal do almoço falou que Rails é legado.

O Renato não abriu slide.

— O contrato que a Helena abre de manhã é esse código. A Serra Azul renova em 3 de junho.

O Caio falou sem virar a cadeira.

— Três ponto três. No servidor.

A Livia olhou para ele.

— Três ponto três do quê?

— Do Ruby. Manutenção normal até 1º de abril. Segurança até 31 de março de 2027.

Era a quinta-feira da primeira semana da Livia na Nortea. Ela ainda nem tinha acesso ao repositório — chegava no dia seguinte —, e a resposta que deu ao Rômulo era o que tinha ouvido no corredor. O número exato, o Caio sabia de cabeça, porque tinha olhado no servidor.

Ruby é a linguagem. Rails é um framework escrito nessa linguagem: um conjunto de bibliotecas e de convenções que já decide a forma do programa. Quem escreve preenche o que é da Nortea — contrato, equipamento, data. O sistema de 2016 é Rails. A linguagem em que ele está escrito começa em 1993, no Japão, com outro autor e sem framework nenhum.

1.1 24 de fevereiro de 1993

Yukihiro Matsumoto, conhecido como Matz, começou o Ruby nessa data, no Japão, como projeto pessoal, enquanto trabalhava como programador. O critério ele declarou anos depois, na entrevista que abre este capítulo: uma linguagem de script mais poderosa que Perl e mais orientada a objetos que Python. As influências que ele mesmo listou foram Perl, Smalltalk, Eiffel, Ada e Lisp.

Do Perl ele queria o trato com texto e a vontade de resolver a tarefa. De Smalltalk, o objeto e o bloco. De Lisp, a ideia de que código pode ser

dado: um trecho de programa guardado e entregue a outro trecho. Eiffel e Ada entraram no critério de clareza, não como sintaxe copiada.

O desenho que saiu dali ainda é o núcleo.

Tudo é objeto. O número `2` é um objeto. `nil`, o valor que significa ausência, é um objeto. Um método é uma mensagem enviada a um objeto. Não há um tipo especial de “valor cru” ao lado dos objetos, como havia nas linguagens de que o Matz estava se afastando.

Bloco Um objeto que guarda um trecho de código. Dá para entregar esse objeto a um método. O método decide quando chamar o trecho, e com qual valor. A ideia veio de Smalltalk. No Ruby ela está desde o começo: percorrer uma lista, filtrar, transformar, tudo isso é um método que recebe um bloco.

A sintaxe foi desenhada para ser lida com pouca pontuação. Método e variável em minúsculas, com sublinhado. Classe com inicial maiúscula. Um método que responde sim ou não pode terminar em `?`. Nada disso é regra do computador para existir. É regra do texto para outra pessoa conseguir ler em voz alta. O Matz tratou a leitura como parte do desenho, não como enfeite posterior.

A primeira versão pública foi a 0.95, em **21 de dezembro de 1995**, anunciada em grupos de notícia japoneses. O Ruby 1.0 saiu no **Natal de 1996**, 25 de dezembro. O Natal virou, com os anos, a data em que uma versão grande aparece. Não foi um calendário de fundação escrito de uma vez. Foi um hábito que a 1.0 fixou e as versões posteriores repetiram.

1.2 A saída do Japão

Durante os primeiros anos a conversa pública era em japonês. A lista de correio em inglês, a ruby-talk, abriu caminho no fim da década de 1990. O livro que tirou a linguagem desse círculo foi *Programming Ruby*, de Dave Thomas e Andy Hunt, publicado no fim de 2000 pela Addison-Wesley. A capa tinha uma picareta. O livro ficou conhecido como Pickaxe, e durante um tempo foi o manual de quem aprendia Ruby fora do Japão.

Em **4 de agosto de 2003** saiu o Ruby 1.8.0. Essa linha, sobretudo o 1.8.7 no fim dela, foi a que ficou instalada em servidor por anos. É a língua em que o primeiro Rails cresceu.

Biblioteca de terceiro, nessa época, era arquivo copiado para dentro do projeto. Em 2004 um grupo — Chad Fowler, Rich Kilmer, Jim Weirich e outros — publicou o RubyGems. Uma **gem** é uma biblioteca empaco-

tada, com nome e versão. O comando `gem` instala essa versão. O Ruby passou a ter um registro. O arquivo do projeto ainda não dizia, sozinho, o conjunto fechado do que devia ser instalado. Isso veio depois, com outro comando, quando o conflito entre versões ficou cotidiano.

1.3 Basecamp, Merb, e duas máquinas virtuais

Em 2003 e 2004 David Heinemeier Hansson, o DHH, trabalhava no Basecamp, o produto da 37signals. Ele extraiu do Basecamp um framework e publicou o Rails. A primeira versão pública circulou em **julho de 2004**. O Rails 1.0 saiu em **13 de dezembro de 2005**.

O desenho que ele publicou tinha decisões com nome. *Convenção sobre configuração*: o nome da classe indica o nome da tabela, e o arquivo não repete essa ligação se a convenção bastar. A classe `Contract` procura a tabela `contracts`. *Active Record* é o nome de um padrão do livro de Martin Fowler, *Patterns of Enterprise Application Architecture*, de 2002: um objeto que representa uma linha e sabe gravá-la. O Rails implementou esse padrão em cima da convenção de nomes. Quem abre `contract.rb` na Nortea está olhando essa decisão de 2004, mais vinte e três colunas que vieram depois.

Rails Framework escrito em Ruby, extraído do Basecamp e publicado em 2004. Já traz a estrutura de um sistema web: o pedido entra por uma rota, passa por um controller, fala com um model, e a resposta volta. A Nortea roda um Rails desde 2016. O framework não é a linguagem. É um programa Ruby que espera os arquivos da Nortea em pastas combinadas.

Em 2006 Ezra Zygmuntowicz publicou o Merb, outro framework Ruby, mais explícito e mais modular, usado por quem esbarrava no miolo do Rails da época. Em **dezembro de 2008** o DHH anunciou que o Merb seria incorporado ao Rails 3, e que os dois times passariam a trabalhar no mesmo projeto. O Rails 3.0 saiu em **agosto de 2010**. O miolo foi reescrito: o roteador, e a fronteira entre o framework e a biblioteca que fala com o banco. A convenção de nomes de 2004 permaneceu. O Merb, como produto separado, parou. O que ele disputava — um miolo em peças — entrou no Rails.

No mesmo mês de agosto de 2010 saiu o Bundler 1.0, de Yehuda Katz, Carl Lerche e outros. O problema que ele fechava era concreto: duas gems pediam versões diferentes de uma terceira, e o RubyGems ativava uma delas tarde demais, com o programa já rodando. O `Gemfile` lista o que o projeto pede. O `Gemfile.lock` grava o que foi resolvido. O

comando `bundle` instala esse conjunto. O `nortea` de 2016 tem os dois arquivos. São dessa safra.

Enquanto o framework se reescrevia, a máquina virtual também. Koichi Sasada escreveu o YARV, *Yet Another Ruby VM*, um novo executor para o Ruby. O Ruby 1.9.0, no **Natal de 2007**, passou a usar o YARV. Nessa linha a string ganhou codificação: o texto carrega a informação de qual conjunto de caracteres ele é. Código que tratava string como sequência crua de bytes quebrou. O 1.9.0 não foi, de saída, a versão que o servidor adotava. A linha 1.9 só ficou estável para uso geral anos depois. Muita gente permaneceu no 1.8.7, e as duas versões conviveram, cada uma com gems que só existiam para ela.

YARV A máquina virtual escrita por Koichi Sasada. A partir do Ruby 1.9 ela é o executor da implementação de referência, a escrita em C, no lugar do interpretador antigo. Máquina virtual, aqui, é o programa que lê o Ruby e o executa. Trocar a máquina virtual não troca o nome da linguagem. Troca o que acontece por baixo do arquivo, e às vezes o que o arquivo precisa declarar — como a codificação do texto, no 1.9.

O reencontro foi o Ruby 2.0.0, em **24 de fevereiro de 2013**, vinte anos depois do dia em que o Matz começou. A codificação padrão do arquivo-fonte passou a ser UTF-8. Entrou também o argumento com nome: ao chamar um método, dá para escrever o nome do parâmetro junto do valor, em vez de depender só da posição. A linha 1.8 chegava ao fim da manutenção estendida no ano seguinte. Quem ainda rodava 1.8.7 em 2013 não estava numa escolha de gosto: estava num servidor que ninguém tinha marcado data para tocar.

1.4 O calendário de Natal

Do 2.1 em diante, a versão maior de fim de ano saiu no Natal com regularidade: 25 de dezembro, ou a véspera quando o dia caía mal. Cada linha passou a ter duas fases publicadas. Na **manutenção normal**, entram correção de defeito e correção de segurança. Na **manutenção de segurança**, só a segunda. O fim costuma cair em 31 de março, cerca de três anos depois do Natal de lançamento. O calendário é do time do Ruby, não de um fornecedor que almoça com o Rômulo.

Manutenção As duas fases de uma linha do Ruby. Normal: defeito e segurança. Segurança: só segurança, até a data de fim publicada. “Está suportado” sem dizer a fase não diz se um defeito comum ainda ganha correção. No servidor da Nortea, em março de 2026, essa diferença tem data.

O Ruby 3.0 saiu no **Natal de 2020**. O Matz tinha anunciado, anos antes, a meta de um Ruby 3 três vezes mais rápido que o 2.0 nessa geração. O 3.0 trouxe o Ractor, um mecanismo experimental de paralelismo, e o RBS, uma linguagem à parte para descrever a forma das classes — não um sistema de tipos dentro do Ruby. A linguagem continuou dinâmica: o tipo mora no valor, e a descrição em RBS é outro arquivo, lido por outras ferramentas. A meta de velocidade não se cumpriu no 3.0 pelo interpretador sozinho.

Quem avançou a velocidade foi um compilador escrito fora do núcleo e depois incorporado. O YJIT, feito na Shopify — Maxime Chevalier-Boisvert, Alan Wu e o time de lá —, entrou no Ruby 3.1, Natal de 2021, como experimental. Ele traduz, durante a execução, o trecho que mais roda para código de máquina. Nas versões seguintes deixou de ser experiência e passou a ser opção estável. É um compilador com autor, empresa e Natal. Não é uma propriedade vaga da linguagem.

O Ruby 3.3 saiu no **Natal de 2023**. É a linha do servidor da Nortea. A manutenção normal dela vai até **1º de abril de 2026**. A de segurança, até **31 de março de 2027**. No dia 3 de junho, data da Serra Azul, um 3.3 ainda recebe correção de segurança e já não recebe correção de defeito comum. São duas datas diferentes. O Caio disse as duas na quinta.

O Ruby 3.4 saiu no Natal de 2024. Nessa linha o Prism, o analisador sintático escrito sob a condução de Kevin Newton, virou o padrão: o programa que lê o texto do arquivo antes de executar. O Ruby 4.0 saiu no **Natal de 2025**. O núcleo — objeto, método, bloco — é o de 1993. A linha nova é outra janela de manutenção, não outra linguagem.



FIGURA 1.1 Do projeto pessoal de 1993 à linha que o servidor da Nortea ainda roda.

RESUMO

- 24 de fevereiro de 1993: Matz começa o Ruby. Tudo é objeto. Bloco é código que se entrega a um método. A 0.95 sai em 21 de dezembro de 1995. A 1.0, no Natal de 1996.
- O Pickaxe, no fim de 2000, leva o Ruby ao inglês. O RubyGems, em 2004, empacota biblioteca com versão. O Ruby 1.8, em agosto de 2003, é a linha em que o Rails nasce.
- Julho de 2004: Rails, extraído do Basecamp. Dezembro de 2008: o Merb entra no projeto. Agosto de 2010: Rails 3 e Bundler 1.0.
- Natal de 2007: YARV vira o executor, no Ruby 1.9. 24 de fevereiro de 2013: Ruby 2.0, UTF-8 por padrão.
- Natal de 2023: Ruby 3.3. Na Nortea, manutenção normal até 1º de abril de 2026 e segurança até 31 de março de 2027. O Ruby 4.0 saiu no Natal de 2025.

- 1** O Rômulo repete o fornecedor: “Ruby é uma linguagem de script, como Perl”. Responda em três linhas, com a data em que o Matz começou e com duas decisões de desenho que ele declarou — uma vinda de Smalltalk, outra que separa o Ruby de um script de texto solto.
- 2** O fornecedor do almoço ofereceu “tirar o Rails e pôr um framework modular”. O que aconteceu, em dezembro de 2008 e em agosto de 2010, com o framework modular que já existia em Ruby?
- 3** O Rômulo abre um ticket: “atualizar o Ruby do servidor esta semana, para a versão mais nova”. A semana é a de 9 de março de 2026. A mais nova, nessa data, é o Ruby 4.0, de 25 de dezembro de 2025. Você aceitaria o texto?

CAPÍTULO

2

O que vamos construir

Nove canteiros, uma plataforma em dois lugares ao mesmo tempo, e uma página que o adendo já prometeu.

AO FIM DESTE CAPÍTULO

Dizer o que o dia 3 de junho cobra do módulo de contratos, separar a pasta de produção da pasta em que você vai escrever, e terminar com o Ruby 3.4 instalado e conferido.

ENQUANTO ISSO, NA EMPRESA A PT-118

A Marta pôs o adendo na mesa na segunda, 9 de março.

— A Serra Azul quer ver, na página, qual equipamento está em qual contrato. O Rômulo já assinou. 3 de junho.

O Renato leu a cláusula sublinhada.

— Uma página. A lista da manhã. O que a Helena já confere.

— O sistema diz uma coisa — disse a Marta. — A planilha diz outra. Ontem a plataforma PT-118 estava nos dois, em canteiros diferentes.

O Caio olhou o número do patrimônio.

— Doze metros. Serra Azul, Contagem. E também Serra Azul, Betim. O contrato não pode estar nos dois.

— Então a gente faz um sistema novo, do lado — disse o Rômulo, da porta.

O Renato devolveu o papel.

— Não. A página nova tem de bater com a planilha até a planilha perder. Outro sistema do lado são duas telas para a Helena conferir, e ela já conferiu duas ontem.

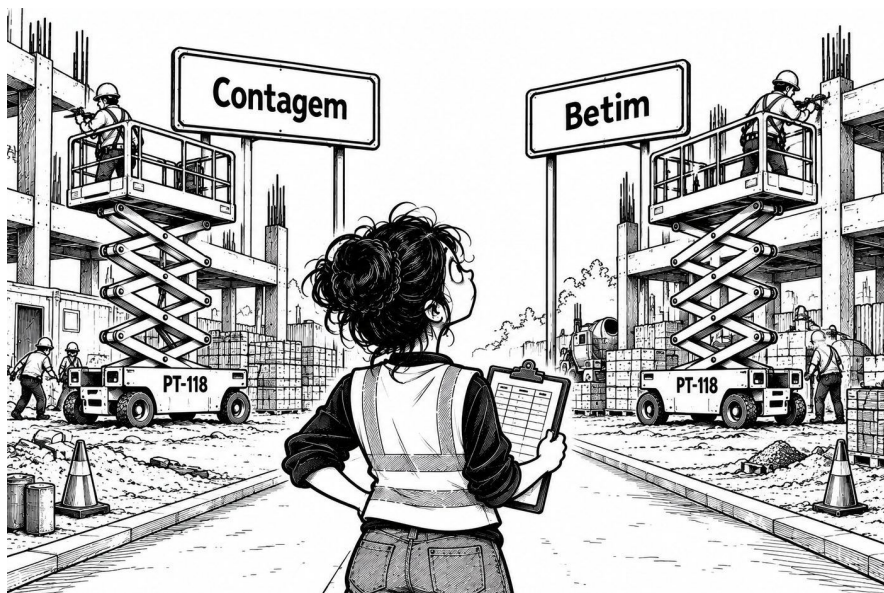


FIGURA 2.1 A PT-118 estava em Contagem. E em Betim.

O módulo se chama Nortea Hub na conversa da sala. No repositório ele não é outro projeto. É o pedaço de contratos do nortea que já está

no ar, o Rails de 2016, reescrito até a manhã conseguir confiar nele. A planilha `patio_SEMANAL.xlsx` continua sendo a lista pela qual o caminhão sai, enquanto a tela discordar.

2.1 O que o dia 3 de junho confere

A Helena, e depois a Serra Azul, precisam de três respostas sobre cada máquina contratada.

Qual equipamento. Qual contrato. Qual período.

Se o contrato foi cancelado, a máquina não pode continuar parecendo ocupada, e também não pode parecer livre se outro contrato ainda a segura. A multa de R\$ 1.800 por dia nasce exatamente nesse desacordo: o canteiro espera a plataforma, o pátio acha que ela está livre, ou o contrário.

A página que o adendo promete é essa consulta. Não é um pátio desenhado de novo. Não é um segundo banco. É a tabela `contracts` dizendo uma coisa só, a mesma que a planilha diria se alguém tivesse atualizado as duas.

A pasta em que você vai escrever se chama `patio`. Ela não é o `nortea`. O sistema de produção continua no servidor, com o Ruby 3.3. O que cresce na sua máquina, nos primeiros arquivos, é Ruby sem Rails: o bastante para ler o que aquele sistema já faz, quando o arquivo dele entrar na mesa.

2.2 Instalando

A linha que você vai instalar é o Ruby 3.4, publicado no Natal de 2024, em manutenção normal enquanto o 3.3 do servidor já caminha para a fase só de segurança. O Ruby 4.0, Natal de 2025, também fala a sintaxe dos exemplos daqui. Se o instalador oferecer os dois, fique no 3.4: é a linha em que estas saídas foram pensadas.

No Windows, baixe o **Ruby+Devkit 3.4** em <<https://rubyinstaller.org>>. O Devkit é o compilador que algumas gems precisam para instalar; sem ele, a gem que tem código em C falha com uma mensagem que não fala de contrato. Na instalação, deixe marcado o Ruby no PATH. PATH é a lista de pastas em que o terminal procura um comando.

No macOS e no Linux, o `ruby` que já vem no sistema pode ser uma linha antiga, às vezes a 2.6, que não recebe correção. Rode `ruby -v` antes

de confiar nele. Se não for 3.4, instale o 3.4 pela página <<https://www.ruby-lang.org/en/downloads/>> e use esse, não o do sistema.

Feche o terminal se ele já estava aberto, abra de novo, e confira:

```
$ ruby -v
ruby 3.4.0 (2024-12-25) [x64-mingw-ucrt]
```

O trecho entre parênteses e o nome da plataforma mudam conforme o mês e o sistema. A resposta precisa começar com `ruby 3.4`. Se o comando não existir, o terminal não viu a instalação: feche, abra, tente de novo antes de instalar por cima.

▲ O TERMINAL QUE JÁ ESTAVA ABERTO

Um terminal guarda o PATH do momento em que foi aberto. Instalar o Ruby não atualiza essa cópia. O sintoma é `ruby` desconhecido numa janela e reconhecido na janela ao lado.

↳ ERRO COMUM

`ruby` pode existir duas vezes na mesma máquina. O servidor da Nortea responde 3.3. A sua, se você seguiu o instalador, responde 3.4. Os dois se chamam Ruby.

No Windows, `where.exe ruby` lista os caminhos. No macOS e no Linux, `which ruby` mostra qual arquivo o terminal vai chamar. Descubra o seu agora. “Eu instalei” com o terminal chamando o Ruby do sistema é uma manhã inteira num erro que não está no arquivo.

► NA PRÁTICA

Rode `ruby -v`. Se a resposta começar com `ruby 3.4`, crie a pasta e entre nela:

```
$ mkdir patio
$ cd patio
```

Não rode `bundle install`. Não copie o Gemfile da Nortea. A pasta `patio` começa vazia: ainda não existe um programa.

RESUMO

- O Hub é o módulo de contratos do Rails que já está no ar, não um segundo sistema ao lado da planilha.
- O dia 3 de junho confere equipamento, contrato e período. Cancelado não pode continuar ocupando, nem liberar o que outro contrato segura.
- A sua máquina fica no Ruby 3.4. O servidor está no 3.3.

- `ruby -v` confirma a linha. A pasta `patio` começa vazia.

■ O PROJETO AGORA

Você tem o Ruby 3.4 instalado, a pasta `patio` criada, e as três respostas que a Helena vai conferir em 3 de junho. Ainda não há programa. É daqui que o arquivo começa.

- 1 A Helena liga às 6h10: “a PT-118 saiu para Contagem. O canteiro de Betim está pedindo a mesma”. Escreva, em duas linhas, o que ela precisa que o sistema responda, e o que ela não precisa saber para receber a resposta.
- 2 O Rômulo insiste num sistema novo, em outro servidor, “para não mexer no legado”. A planilha e a tela já discordam sobre a PT-118. O que a Helena passaria a conferir na manhã seguinte, se o sistema novo nascesse do lado?
- 3 Sem olhar de novo o calendário: em 9 de março de 2026, o Ruby 3.3 do servidor ainda recebe que tipo de correção, e até quando? E o que muda em 1º de abril?

CAPÍTULO

3

O primeiro programa

Dois Rubys na mesma manhã, e só um deles abre o pátio às 6h.

AO FIM DESTES CAPÍTULOS

Escrever, salvar e rodar um programa Ruby; interpolar um nome dentro do texto; ler o nil que o irb mostra; e reconhecer as três falhas da primeira hora.

**Capítulo disponível na
versão completa**

PARTE 2

A linguagem que o contrato já fala

Objeto, método, decisão, símbolo, lista e bloco. O pedaço do Ruby que o arquivo de produção já usa sem avisar.

NESTA PARTE

- 4 Objetos e tipos
- 5 Métodos
- 6 Condicionais
- 7 Strings e símbolos
- 8 Arrays e hashes
- 9 Blocos e enumerables

CAPÍTULO

4

Objetos e tipos

Três diárias de R\$ 480 somadas pela planilha davam R\$ 480.480.480. O boleto não saiu, e ninguém achou estranho a tempo.

AO FIM DESTES CAPÍTULO

Perguntar a qualquer valor que tipo ele é, separar número de texto antes de somar, e guardar dinheiro da locação em centavos, inteiros, em vez de confiar em ponto flutuante.

**Capítulo disponível na
versão completa**

CAPÍTULO

5

Métodos

*Um método de três linhas que ninguém chama.
A busca no repositório discordava.*

AO FIM DESTE CAPÍTULO

Definir um método, entender o valor que ele devolve sem return, ler o ? no fim do nome, passar argumentos por posição e por nome, e saber que um método sem chamada no arquivo pode estar sendo chamado de outro lugar.

**Capítulo disponível na
versão completa**

CAPÍTULO

6

Condicionais

O contrato estava cancelado desde segunda. O relatório dizia que a plataforma continuava ocupada, e estava certo segundo o próprio código.

AO FIM DESTE CAPÍTULO

Escrever decisões com `if`, `elsif`, `unless` e `case`; saber o que o Ruby considera falso; e reconhecer dois `ifs` independentes onde deveria haver uma escolha só.

**Capítulo disponível na
versão completa**

CAPÍTULO

7

Strings e símbolos

O status era active dos dois lados. Um deles tinha dois-pontos na frente, e a comparação dava falso.

AO FIM DESTES CAPÍTULO

Trabalhar com texto sem surpresa, saber quando uma string pode ser alterada no lugar, entender o que é um símbolo, e nunca mais comparar `:active` com `"active"` esperando verdadeiro.

**Capítulo disponível na
versão completa**

CAPÍTULO

8

Arrays e hashes

A planilha chama de Contrato o que o banco chama de code. O mesmo contrato, dois hashes, nenhuma chave em comum.

AO FIM DESTES CAPÍTULOS

Guardar a lista da manhã num array, guardar um contrato num hash, ler uma chave que pode não existir sem receber nil em silêncio, e levar o contrato da planilha e o do banco para a mesma forma.

**Capítulo disponível na
versão completa**

CAPÍTULO

9

Blocos e enumerables

A lista da Helena tinha quatro plataformas. O select devolveu três. A quarta tinha status nil desde uma importação de 2019.

AO FIM DESTES CAPÍTULOS

Entregar um trecho de código a um método, percorrer, transformar e filtrar a lista da manhã com each, map, select e find, e saber o que um filtro faz com o valor que ninguém previu.

ENQUANTO ISSO, NA EMPRESA A quarta plataforma

A lista de divergências saiu na terça, 17 de março, às 7h. A Helena respondeu às 7h20, por mensagem para a Marta, que encaminhou sem comentário:

*Serra Azul Contagem tem QUATRO plataformas. A lista diz três. Falta a PT-133.
Ela está lá, eu vi o caminhão sair.*

A Livia rodou de novo. Três. Procurou a PT-133 no banco de homologação.

```
{:code=>"CT-1987", :equipment=>"PT-133", :site=>"contagem",  
:status=>nil}
```

— status nulo — disse ela.

O Sérgio olhou por cima do ombro dela.

— Importação de 2019. Os contratos que vieram da planilha antiga entraram sem status. A tela mostra como ativo porque a view testa se é cancelado.

— E o meu filtro testa se é ativo.

— Então para a tela ela existe, e para o seu filtro não.



FIGURA 9.1 A lista dizia três. O caminhão levou quatro.

9.1 O bloco

O bloco é a ideia que o Matz trouxe de Smalltalk em 1993: um trecho de código entregue a um método, que decide quando chamá-lo. A sintaxe é um par de chaves — ou `do` e `end` — depois da chamada:

```
3.times { puts "carregando" }

3.times do
  puts "carregando"
end
```

As duas formas fazem o mesmo. `times` é um método do inteiro. Ele chama o bloco três vezes. A convenção da casa: chaves para bloco de uma linha, `do ... end` para bloco de várias.

O bloco pode receber valores de quem o chama. Os nomes ficam entre barras, logo no começo:

```
3.times { |i| puts "volta #{i}" }
```

```
volta 0
volta 1
volta 2
```

`|i|` é o parâmetro do bloco. O `times` passa 0, depois 1, depois 2.

Bloco Um trecho de código entre `{ }` ou `do ... end`, entregue a um método na chamada. O método decide quando executá-lo e com quais valores. Os parâmetros vêm entre `| |`. Um método recebe no máximo um bloco, e o bloco não é um argumento entre os parênteses: vem depois deles.

9.2 `yield`: o lado de quem recebe

Um método seu também pode receber bloco. `yield` chama o bloco que veio:

BLOCO.RB

ruby

```
1 def para_cada_turno
2   yield "6h"
3   yield "13h"
4 end
5
6 para_cada_turno { |hora| puts "carga das #{hora}" }
```

```
$ ruby bloco.rb
carga das 6h
carga das 13h
```

O método não sabe o que o bloco faz. Sabe quando chamá-lo e com o quê. É a divisão de trabalho do desenho de 1993: percorrer é do método; o que fazer com cada item é de quem chama.

9.3 `each`: percorrer

Todo array tem `each`, que chama o bloco uma vez para cada item:

MANHA.RB

ruby

```
1  # frozen_string_literal: true
2
3  contratos = [
4    { code: "CT-2041", equipment: "PT-118", status: "active" },
5    { code: "CT-2033", equipment: "PT-121", status: "cancelled" },
6    { code: "CT-2050", equipment: "PT-140", status: "reserved" },
7    { code: "CT-1987", equipment: "PT-133", status: nil },
8  ]
9
10 contratos.each do |c|
11   puts "#{c[:equipment]}: #{c[:status]}"
12 end
```

```
$ ruby manha.rb
PT-118: active
PT-121: cancelled
PT-140: reserved
PT-133:
```

O `nil` interpolado vira texto vazio. A linha da PT-133 sai sem status, e sem erro.

O hash também tem `each`, que passa chave e valor:

```
{ code: "CT-2041", site: "contagem" }.each do |chave, valor|
  puts "#{chave} = #{valor}"
end
```

9.4 `map`, `select`, `reject`, `find`

`each` percorre e devolve a própria lista: serve para efeito, como escrever na tela. Para **produzir** alguma coisa a partir da lista, há métodos que usam o valor que o bloco devolve.

`map` **transforma**. Devolve uma lista nova, com o valor do bloco para cada item:

```
irb> contratos.map { |c| c[:equipment] }
=> ["PT-118", "PT-121", "PT-140", "PT-133"]
```

select filtra. Devolve os itens para os quais o bloco é verdadeiro:

```
irb> contratos.select { |c| c[:status] == "active" }
=> [{code: "CT-2041", equipment: "PT-118", status: "active"}]
```

reject filtra ao contrário. Devolve os itens para os quais o bloco é falso.

find acha o primeiro. Devolve o primeiro item para o qual o bloco é verdadeiro, ou `nil`:

```
irb> contratos.find { |c| c[:equipment] == "PT-140" }
=> {code: "CT-2050", equipment: "PT-140", status: "reserved"}
```

Nenhum desses métodos altera `contratos`. Todos devolvem algo novo.

AS PARTES DE UMA CHAMADA COM BLOCO

```
ocupam = contratos.select { |c| c[:status] == "active" }
```

1 2 3 4

- 1 `contratos.select` é a chamada. O método percorre a lista.
- 2 `{ |c| ... }` é o bloco. `c` recebe cada contrato, um por vez.
- 3 O valor da última expressão do bloco é a resposta. `select` guarda o item quando ela é verdadeira.
- 4 O resultado é uma lista nova. `contratos` não muda.

9.5 Atalhos: `&` e `it`

Quando o bloco só chama um método no item, há uma forma curta:

```
patrimonios = ["pt-118", "pt-121"]
patrimonios.map { |p| p.upcase }
patrimonios.map(&:upcase)
```

As duas linhas devolvem `["PT-118", "PT-121"]`. `&:upcase` é “chame `upcase` em cada um”. É a forma do `select(&:active?)` que o `grep` da Livia encontrou numa view.

O Ruby 3.4 trouxe outra: `it`, o nome do item quando o bloco tem um parâmetro só e ninguém o declarou:

```
contratos.map { it[:equipment] }
```

O servidor da Norteia está no 3.3, que não tem `it`. Código que vai para o `norteia USA |c|`.

9.6 Somar, contar, ordenar, agrupar

```
1 diarias = [48_000, 18_990, 32_000]
2
3 diarias.sum # => 98990
4 contratos.count { |c| c[:status] == "active" } # => 1
5 contratos.sort_by { |c| c[:equipment] }
6 contratos.group_by { |c| c[:status] }
```

`sum` soma. `count` com bloco conta os verdadeiros. `sort_by` ordena pelo valor que o bloco devolve. `group_by` devolve um hash: a chave é o valor do bloco, o valor é a lista dos itens que deram aquela chave.

```
irb> contratos.group_by { |c| c[:status] }.keys
=> ["active", "cancelled", "reserved", nil]
```

`nil` aparece como grupo. O `group_by` não esconde o que ninguém previu.

9.7 O filtro e o `nil`

A lista de divergências da Lívia filtrava assim:

```
ocupam = contratos.select do |c|
  ["active", "reserved"].include?(c[:status])
end
```

Três itens. A PT-133, com `status: nil`, não está na lista de status, o bloco devolve falso, e o `select` a descarta.

A view do sistema de 2016 testava o contrário:

```
visiveis = contratos.reject { |c| c[:status] == "cancelled" }
```

Quatro itens. O `nil` não é igual a `"cancelled"`, então fica.

As duas linhas parecem dizer o mesmo — “os que ocupam” — e divergem exatamente no valor que nenhuma delas previu. Filtrar pelo que entra (“active ou reserved”) descarta o desconhecido. Filtrar pelo que sai (“não cancelado”) mantém.

● O ESSENCIAL

Um filtro é uma decisão sobre o valor que você não previu. `select` pelo que é permitido descarta o desconhecido; `reject` pelo que é proibido o deixa passar. Escolha sabendo qual dos dois o pátio aceita — e procure o desconhecido antes de escolher.

A correção não é trocar um filtro pelo outro. É descobrir o desconhecido e decidir sobre ele:

```
sem_status = contratos.select { |c| c[:status].nil? }
puts "#{sem_status.size} contrato(s) sem status"
```

```
1 contrato(s) sem status
```

Na homologação, o número real foi 212. Todos da importação de 2019. A Marta levou a lista à Helena, que confirmou quais ainda estavam no pátio. Enquanto ninguém corrige o banco, a lista da manhã os mostra à parte, com o aviso.

■ O PROJETO AGORA

Fim da Parte 2. Na pasta `patio` há Ruby sem Rails: diária em centavos, um predicado `ativo?`, o relatório de ocupação com uma condição só, o contrato da planilha e o do banco na mesma forma, e a lista da manhã filtrada sem perder a PT-133. É o pedaço da linguagem que o `nortea` usa em todo arquivo.

RESUMO

- Bloco é código entre `{ }` ou `do ... end`, entregue a um método. `yield` o chama de dentro do método.
- `each` percorre, para efeito. `map` transforma, `select` e `reject` filtram, `find` acha o primeiro. Nenhum altera a lista.
- `&:metodo` chama o método em cada item. `it` é do 3.4, e o servidor está no 3.3.
- `sum`, `count`, `sort_by` e `group_by` resolvem a maioria das contas da manhã.
- Filtrar pelo permitido descarta o desconhecido; filtrar pelo proibido o mantém. Procure o `nil` antes de escolher.

- 1 Com a lista `contratos` da manhã, escreva uma linha para cada pedido:
 1. Os patrimônios, em maiúsculas.
 2. Os contratos cancelados.
 3. O contrato da PT-118.
 4. Quantos contratos não têm status.
- 2 Escreva um método `por_canteiro(contratos)` que devolve um hash com o canteiro como chave e a lista de patrimônios como valor. Use `group_by` e `map`.

- 3** O Sérgio propõe resolver a PT-133 trocando o `select` da Livia por
... `reject { |c| c[:status] == "cancelled" }`, “igual à tela”. Diga o que isso
resolve, o que passa a entrar errado, e o que você faria.

PARTE 3

Classes, módulos e falhas

O contrato vira um objeto com nome. O módulo mistura comportamento. A exceção diz o que falhou, em vez de seguir calada.

NESTA PARTE

- 10 Classes
- 11 Módulos
- 12 Exceções
- 13 Gems e Bundler

CAPÍTULO

10

Classes

O hash aceitava qualquer chave, inclusive status. Durante dois dias, o contrato CT-2058 esteve ativo e sem status ao mesmo tempo.

AO FIM DESTE CAPÍTULO

Transformar o hash do contrato numa classe com nome: inicialize, variável de instância, attr_reader, métodos que perguntam ao próprio objeto, e um construtor de classe que lê a linha da planilha.

**Capítulo disponível na
versão completa**

CAPÍTULO

11

Módulos

O mesmo módulo, incluído em dois models. Um deles redefiniu o total e esqueceu o super. A taxa de entrega sumiu da reserva.

AO FIM DESTES CAPÍTULO

Compartilhar comportamento entre classes com um módulo, entender em que ordem o Ruby procura um método, usar `super` para estender em vez de substituir, e usar módulo como sobrenome para agrupar nomes.

**Capítulo disponível na
versão completa**

CAPÍTULO

12

Exceções

A lista da manhã saiu vazia. Não havia erro nenhum no log. Havia um rescue que devolvia uma lista vazia quando não achava o arquivo.

AO FIM DESTES CAPÍTULOS

Ler uma falha pelo tipo, capturar só o que se sabe tratar, criar a exceção da Norteia com o dado que quem captura precisa, e usar ensure para o que tem de acontecer com ou sem falha.

**Capítulo disponível na
versão completa**

CAPÍTULO

13

Gems e Bundler

Um bundle update sem nome de gem mexeu em quarenta e uma linhas do Gemfile.lock. A página de contratos parou de abrir, e só na máquina da Lívia.

AO FIM DESTES CAPÍTULOS

Usar uma biblioteca de terceiros, ler o Gemfile e o Gemfile.lock, entender o que o ~> promete, distinguir bundle install de bundle update, e rodar o nortea na sua máquina sem trocar o Ruby do servidor.

**Capítulo disponível na
versão completa**

PARTE 4

Entrando no Rails

*O framework que já está em produção desde 2016.
Modelo, tabela, associação, validação e o callback
duplicado.*

NESTA PARTE

- 14 O que é o Rails
- 15 Modelos
- 16 Migrations
- 17 Associações
- 18 Validações
- 19 Callbacks

CAPÍTULO

14

O que é o Rails

Nenhum arquivo do nortea diz que o endereço / contracts chama o ContractsController. O Rails decide pelo nome.

AO FIM DESTES CAPÍTULOS

Abrir o nortea sabendo onde cada coisa mora, subir o servidor local, seguir um pedido do endereço à página, e reconhecer o que o framework decide sozinho pelo nome dos arquivos.

ENQUANTO ISSO, NA EMPRESA Onde está escrito

Na segunda, 30 de março, a Livia subiu o `nortea` na máquina dela pela primeira vez e abriu `localhost:3000/contracts`. A lista de contratos de homologiação apareceu.

Ela procurou, no projeto inteiro, onde estava escrito que aquele endereço abria aquela lista.

— Tem uma linha nas rotas — disse ela. — `resources :contracts`. Mas não diz qual arquivo.

— Diz — disse o Caio. — Pelo nome.

— E a tabela? O `contract.rb` não fala `contracts` em lugar nenhum.

— Também pelo nome.

O Renato, que passava com a caneca, parou atrás deles.

— É a parte que mais ajuda e a que mais engana. Funciona sem você escrever. E, quando alguém muda um nome, para de funcionar sem ninguém ter mexido.

14.1 Framework

O Rails é um programa Ruby que espera os arquivos da Nortea em pastas combinadas. Até aqui, na `patio`, quem decidia tudo era você: o arquivo que roda, a ordem, o que chama o quê. No Rails, o framework decide a forma do programa e chama o seu código nos pontos combinados.

Um script da `patio` começa na primeira linha e termina na última. O `nortea` não tem primeira linha sua. Ele sobe, espera um pedido, e o framework escolhe qual pedaço do código da Nortea atende.

14.2 As pastas

```

nortea/
├── Gemfile, Gemfile.lock  as gems, travadas
├── .ruby-version  3.3.6
├── app/
│   ├── models/  contract.rb, equipment.rb, reservation.rb
│   ├── controllers/  contracts_controller.rb
│   ├── views/contracts/  as páginas
│   ├── jobs/
│   └── config/
│       ├── routes.rb  quais endereços existem
│       ├── database.yml  em qual banco cada ambiente grava
│       └── environments/  development, test, production
├── db/
│   ├── schema.rb  o desenho das tabelas
│   └── migrate/  o histórico das mudanças nelas
├── spec/  os testes
└── bin/rails  o comando do framework

```

A pasta `app` é o código da Nortea. A pasta `config` é o que diz ao Rails como juntar esse código. Quase todo o trabalho acontece em `app/models`, `app/controllers`, `app/views` e `config/routes.rb`.

A versão está no `Gemfile.lock`:

```

$ grep " rails (" Gemfile.lock
rails (7.2.2)

```

O `nortea` nasceu num Rails 4.2, em março de 2016. Passou pelo 5, pelo 6, e chegou ao 7.2 no fim de 2024, numa atualização que o Sérgio fez em três meses. As pastas são as mesmas desde 2016. O que mudou foi o miolo.

14.3 Subir o servidor

```

$ bin/rails server
=> Booting Puma
=> Rails 7.2.2 application starting in development
=> Run 'bin/rails server --help' for more startup options
* Listening on http://127.0.0.1:3000

```

`bin/rails` é o comando do framework, na versão do `lock` — o equivalente a `bundle exec rails`. `server` sobe um servidor web local. Puma é o nome do servidor; o Rails o usa por padrão.

`development` é o **ambiente**. O mesmo código roda em três: `development` na sua máquina, `test` quando os testes rodam, `production` no servidor. O `database.yml` aponta cada um para um banco diferente. A lista que a

Lívia viu era do banco de desenvolvimento dela, preenchido com uma cópia anonimizada da homologação.

▲ O BANCO DA SUA MÁQUINA

O `database.yml` de desenvolvimento aponta para um banco local. Se alguém um dia apontá-lo para o de produção “só para testar uma coisa”, o `bin/rails` da sua máquina passa a gravar no pátio. O arquivo com a senha de produção não fica no repositório por isso.

14.4 Um pedido, do endereço à página

O navegador pede `GET /contracts`. O caminho:

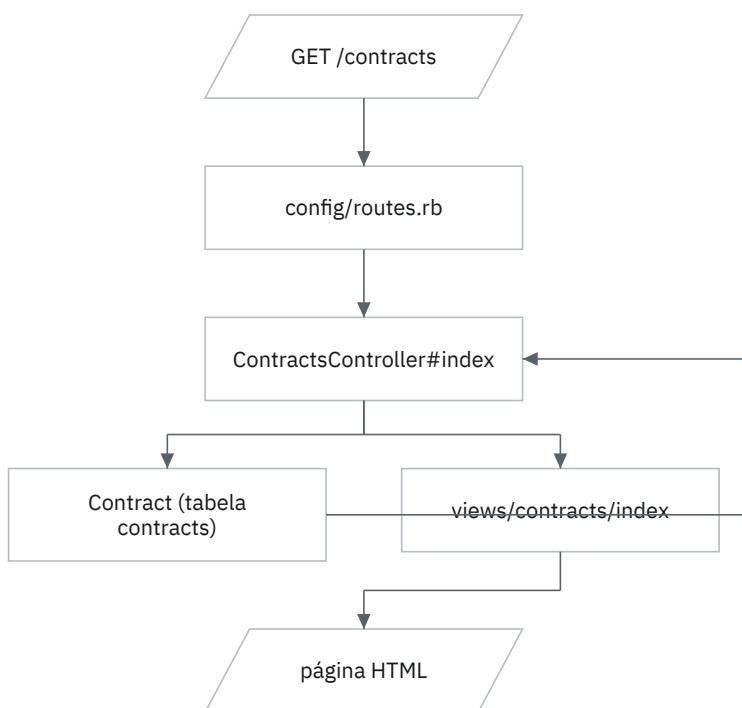


FIGURA 14.1 O pedido entra por uma rota, passa por um controller, fala com um model, e a resposta volta por uma view.

A **rota** decide qual controller e qual método atendem aquele verbo e aquele caminho.

O **controller** recebe o pedido. O método `index` pede os contratos ao model.

O **model** fala com a tabela. `Contract.all` vira uma consulta ao banco.

A **view** monta o HTML com os contratos que o controller entregou. Por enquanto, basta saber em que ordem elas aparecem.

14.5 O que o nome decide

Nenhum arquivo do `nortea` diz, por extenso, a maior parte dessas ligações. O Rails as deduz dos nomes:

NOME	O RAILS CONCLUI
<code>resources :contracts</code>	endereços <code>/contracts</code> , <code>/contracts/:id</code> , e os outros
<code>/contracts</code>	classe <code>ContractsController</code> , arquivo <code>contracts_controller.rb</code>
método <code>index</code>	view <code>app/views/contracts/index.html.erb</code>
classe <code>Contract</code>	tabela <code>contracts</code>
coluna <code>equipment_id</code>	liga ao model <code>Equipment</code>

TABELA 14.1 O singular é o model; o plural é a tabela e o controller. Os arquivos têm o nome em `snake_case` as classes, em `CamelCase`.

É a *convenção sobre configuração* de 2004. O arquivo não repete a ligação quando a convenção basta. Para quem conhece a convenção, o projeto é previsível: dado um endereço, você sabe qual arquivo abrir. Para quem não conhece, é invisível: a ligação não está escrita em lugar nenhum para ser procurada.

● O ESSENCIAL

No Rails, **renomear é mudar comportamento**. Renomear `contracts_controller.rb` para `contratos_controller.rb` não é faxina: é a rota apontando para uma classe que não existe mais. Antes de mudar um nome no `nortea`, pergunte que outro nome o Rails deduz a partir dele.

A convenção fala inglês. `Contract` vira `contracts` porque o Rails sabe pluralizar inglês. É por isso que o `nortea`, feito por brasileiros, tem todos os nomes de código em inglês e todos os textos da tela em português. Com um nome que parece exceção e não é: a tabela de equipamentos.

14.6 Ver as rotas

As ligações que o Rails deduziu podem ser listadas:

```

1  $ bin/rails routes -c contracts
2      Prefix Verb      URI Pattern      Controller#Action
3      contracts GET      /contracts(.:format)  contracts#index
4              POST      /contracts(.:format)  contracts#create
5  new_contract GET      /contracts/new(.:format)  contracts#new
6      contract GET      /contracts/:id(.:format)  contracts#show

```

A lista tem mais linhas; estas são quatro delas. `contracts#index` quer dizer “controller `contracts`, método `index`”. É a documentação que nunca desatualiza, porque é gerada do próprio projeto.

RESUMO

- O Rails decide a forma do programa e chama o código da Nortea nos pontos combinados.
- `app` é o código; `config` diz como juntá-lo; `db/schema.rb` descreve as tabelas.
- `bin/rails server` sobe o servidor local no ambiente `development`, com banco próprio.
- Um pedido passa por rota, controller, model e view, nessa ordem.
- Os nomes decidem as ligações: singular é model, plural é tabela e controller. Renomear é mudar comportamento.

- 1 Para o endereço `GET /equipment/412`, diga, pela convenção, qual controller, qual método e qual view o Rails procuraria.
- 2 A Livia quer criar uma página nova, “ocupação por canteiro”, em `.. /occupancy`. Liste os três arquivos que ela vai precisar tocar ou criar, na ordem em que o pedido os atravessa.
- 3 O Sérgio sugere, “para ficar mais claro para a Helena”, renomear o model `Contract` para `Contrato`. Liste o que deixaria de funcionar pela convenção, e diga onde a clareza para a Helena de fato mora.

CAPÍTULO

15

Modelos

Vinte e três colunas, três começos, e cada relatório da Nortea lendo um deles.

AO FIM DESTES CAPÍTULO

Ler um model do Rails sabendo o que ele herda, entender por que ele não declara as colunas, consultar e ler registros no console, e descobrir o que cada uma das três colunas de começo quer dizer antes de mexer nelas.

**Capítulo disponível na
versão completa**

CAPÍTULO

16

Migrations

O banco de produção tinha uma coluna que o schema.rb não tinha. Alguém a criou à mão em 2023, direto no servidor, numa sexta.

AO FIM DESTES CAPÍTULOS

Mudar uma tabela com histórico em vez de mão: escrever uma migration, rodá-la e desfazê-la, entender o que o schema.rb é e o que ele não é, e preencher os 212 contratos sem status sem apagar nenhuma das três datas.

**Capítulo disponível na
versão completa**

CAPÍTULO

17

Associações

*O contrato da PT-121 foi cancelado na segunda.
Na quarta, a plataforma ainda não aparecia
entre as disponíveis.*

AO FIM DESTE CAPÍTULO

Ligar contrato e equipamento com *belongsto* e *hasmany*, entender que a associação devolve a coleção inteira, dar nome ao filtro de ativo com um escopo, e montar a lista de disponíveis sem esquecer o cancelamento.

**Capítulo disponível na
versão completa**

CAPÍTULO

18

Validações

*O formulário gravava contrato sem responsável.
O motorista chegava ao canteiro sem ninguém
para assinar a entrega, e a máquina voltava no
mesmo caminho.*

AO FIM DESTES CAPÍTULOS

Recusar no model o contrato que o pátio não aceitaria: presença, período coerente, sem duas reservas da mesma máquina ao mesmo tempo; ler os erros que a validação produz; e saber quais caminhos passam por cima dela.

**Capítulo disponível na
versão completa**

CAPÍTULO

19

Callbacks

Um contrato criado gerava duas reservas da mesma máquina. Um callback era de 2016, o outro de 2021, e os dois tinham razão sozinhos.

AO FIM DESTES CAPÍTULOS

Entender o que um callback é e em que ordem o Rails os chama, reproduzir a reserva duplicada com um script, corrigi-la observando a mudança certa, e saber o que não deve morar num callback.

**Capítulo disponível na
versão completa**

PARTE 5

Do pedido à tela

A rota, o controller de trinta e sete commits, a view e o console que a Marta achou que não alterava nada.

NESTA PARTE

- 20 Rotas
- 21 Controllers
- 22 Views
- 23 Console

CAPÍTULO

20

Rotas

O botão dizia Cancelar contrato. O pedido que ele mandava era POST /contracts — o mesmo endereço de criar um contrato.

AO FIM DESTES CAPÍTULO

Ler o pedido HTTP como verbo e caminho, entender o que resources declara, dar ao cancelamento uma rota com nome em vez de um parâmetro escondido, e usar os nomes de rota em vez de montar endereço à mão.

**Capítulo disponível na
versão completa**

CAPÍTULO

21

Controllers

Trinta e sete commits num método de vinte e oito linhas. Cada um corrigia um caso, e nenhum tirava a correção anterior.

AO FIM DESTES CAPÍTULOS

Ler um controller sabendo o que é dele e o que já devia estar no model: params e strong parameters, o if do save, render e redirect, e o begin que some quando a falha vira validação.

**Capítulo disponível na
versão completa**

CAPÍTULO

22

Views

Na mesma linha da tela, a plataforma aparecia sem destaque de ocupada e com o rótulo Em uso. A view fazia a mesma pergunta de dois jeitos.

AO FIM DESTES CAPÍTULO

Montar a página que a Helena vê com ERB: imprimir e executar, percorrer uma lista, quebrar em partial, mostrar os erros do formulário, e fazer a view perguntar ao model em vez de refazer a conta.

**Capítulo disponível na
versão completa**

CAPÍTULO

23

Console

Contract.last não era o contrato que a Livia queria. O update seguinte gravou nele mesmo assim, e o console respondeu true.

AO FIM DESTES CAPÍTULOS

Usar o console do Rails sabendo, linha por linha, o que só lê e o que grava; ler o que cada expressão devolve; trabalhar em sandbox; e conferir no log o UPDATE que não parecia gravação.

**Capítulo disponível na
versão completa**

PARTE 6

Provar, integrar, publicar

O teste que criava um contrato impossível, o trabalho que não cabe no pedido, a API da Serra Azul e a manhã em que o pátio confere.

NESTA PARTE

- 24 Testes
- 25 Jobs
- 26 APIs
- 27 Autenticação
- 28 Deploy
- 29 Projeto final

CAPÍTULO

24

Testes

O teste passava desde 2016. A factory fabricava um contrato sem equipamento, sem cliente e sem data — um contrato que o pátio devolveria no portão.

AO FIM DESTES CAPÍTULOS

Ler e escrever testes com RSpec, entender o que uma factory fabrica, ver o teste ficar vermelho quando o contrato é impossível e verde quando a factory passa a exigir o que o pátio exige, e transformar a reprodução da reserva dupla num teste que roda sozinho.

ENQUANTO ISSO, NA EMPRESA O contrato impossível

Na segunda, 4 de maio, a Lívia rodou os testes do `nortea` pela primeira vez inteiros.

```
$ bundle exec rspec
.....

Finished in 0.41 seconds
9 examples, 0 failures
```

— Nove — disse ela. — Para um sistema de dez anos.

O Diego abriu a factory:

SPEC/FACTORIES/CONTRACTS.RB

ruby

```
FactoryBot.define do
  factory :contract do
    code { "CT-0001" }
  end
end
```

— Isso fabrica o quê? — perguntou ele.

— Um contrato com código — disse o Caio. — Mais nada.

— Sem equipamento?

— Sem equipamento, sem cliente, sem data. O teste do `active?` monta um desses, põe status e data de fim, e pergunta.

O Diego leu o teste e depois olhou para a Lívia.

— Então os nove passam porque nenhum deles tenta gravar.

— Nenhum.

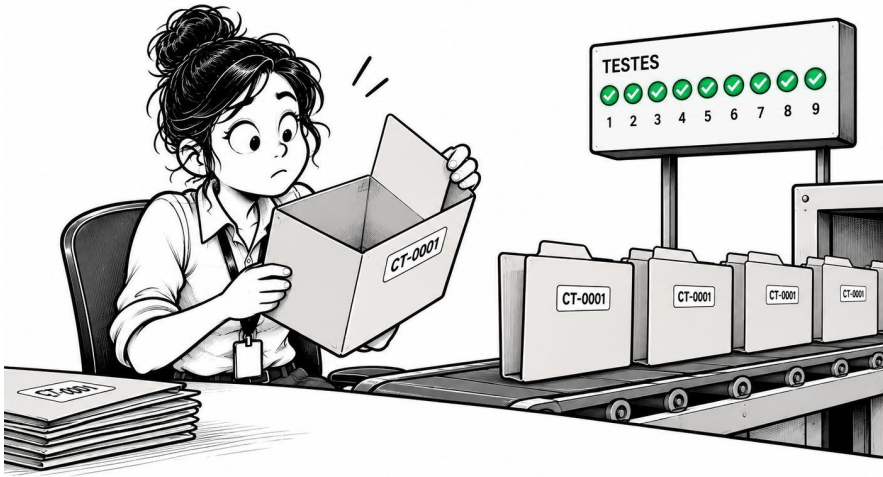


FIGURA 24.1 Nove testes verdes. Nenhum contrato que o pátio aceitaria.

24.1 O que um teste é

Um teste é um programa que roda o seu código e confere o resultado. Não prova que o código está certo em todos os casos. Prova que **um comportamento combinado continua acontecendo** — hoje, e depois da próxima alteração de qualquer pessoa.

A reprodução da reserva dupla já era quase isso: um script que criava um contrato e contava as reservas. Faltava ele conferir sozinho e rodar junto com os outros, sem alguém olhar a saída.

O `nortea` usa `RSpec`. Os testes ficam em `spec/`, e os arquivos terminam em `_spec.rb`:

SPEC/MODELS/CONTRACT_SPEC.RB

ruby

```

1  require "rails_helper"
2
3  RSpec.describe Contract do
4    describe "#occupying?" do
5      it "é verdadeiro para contrato ativo dentro do prazo" do
6        contrato = Contract.new(status: "active",
7                                end_date: Date.new(2026, 6, 30))
8
9        expect(contrato.occupying?(Date.new(2026, 5, 4))).to
be(true)
10      end
11    end
12  end

```

`describe` agrupa. `it` é um exemplo — um comportamento, com uma frase que diz qual. `expect(...).to be(true)` é a conferência: se o valor não for `true`, o teste falha e diz o que veio.

```

$ bundle exec rspec spec/models/contract_spec.rb
.

Finished in 0.08 seconds
1 example, 0 failures

```

O ponto é um exemplo que passou. Um `F` seria um que falhou.

Teste Um programa que executa um pedaço do sistema e confere o resultado. `describe` diz o que está sendo testado; `it` diz o comportamento; `expect` confere. Um teste verde diz que o comportamento descrito na frase aconteceu. Não diz nada sobre o que a frase não descreve.

24.2 Factory

Montar um contrato em cada teste, com os campos à mão, fica longo. A factory é uma receita: o teste pede um contrato, e ela o fabrica com valores padrão.

```

contrato = build(:contract, status: "active")
contrato = create(:contract, status: "active")

```

`build` monta o objeto na memória, sem gravar. `create` monta e grava no banco de teste — e passa pelas validações.

A factory de 2016 tinha um campo. Com `build`, ela funciona, porque nada confere o contrato: o teste só pergunta ao objeto. Os nove testes usavam `build`. Nenhum sabia que o contrato fabricado não seria aceito pelo próprio sistema.

24.3 O vermelho

O Diego escreveu o primeiro teste que precisava gravar: o dos disponíveis, o defeito da PT-121.

SPEC/MODELS/EQUIPMENT_SPEC.RB

ruby

```
1  require "rails_helper"
2
3  RSpec.describe Equipment do
4    describe ".available" do
5      it "inclui a máquina cujo único contrato foi cancelado" do
6        pt121 = create(:equipment, patrimony: "PT-121")
7        create(:contract, equipment: pt121, status: "cancelled")
8
9        expect(Equipment.available).to include(pt121)
10     end
11   end
12 end
```

```
1  $ bundle exec rspec spec/models/equipment_spec.rb
2  F
3
4  Failures:
5
6    1) Equipment.available inclui a máquina cujo único contrato foi
7       cancelado
8       Failure/Error: create(:contract, equipment: pt121,
9                          status: "cancelled")
10
11       ActiveRecord::RecordInvalid:
12         A validação falhou: Cliente é obrigatório, Responsável no
13         canteiro não pode ficar em branco, Início da cobrança não
14         pode ficar em branco, Devolução prevista não pode ficar em
15         branco
```

Vermelho, e pelo motivo certo. O teste não falhou na regra de disponibilidade. Falhou ao fabricar o contrato: a factory produz um contrato que o model recusa, e o `create` usa `save!`, que levanta.

A mensagem é a lista do que o pátio exige: cliente, responsável, início, devolução. Tudo que a factory de 2016 não sabia.

24.4 A factory honesta

SPEC/FACTORIES/CONTRACTS.RB

ruby

```
1  FactoryBot.define do
2    factory :contract do
3      sequence(:code) { |n| format("CT-%04d", 5000 + n) }
4      customer
5      equipment
```

```

6      site { "contagem" }
7      responsible { "Rogério Tavares" }
8      status { "active" }
9      start_date { Date.new(2026, 5, 4) }
10     end_date { Date.new(2026, 6, 30) }
11     daily_rate_cents { 48_000 }
12
13     trait :cancelled do
14       status { "cancelled" }
15       cancelled_at { Time.zone.local(2026, 5, 4, 9, 0) }
16     end
17
18     trait :overdue do
19       start_date { Date.new(2026, 3, 2) }
20       end_date { Date.new(2026, 4, 30) }
21       returned_at { nil }
22     end
23   end
24 end

```

`sequence(:code)` gera um código diferente a cada contrato fabricado — CT-5001, CT-5002 —, porque o código é único e dois contratos no mesmo teste recusariam um ao outro. `customer` e `equipment`, sem valor, dizem à `factory` para fabricar também o cliente e o equipamento, com as `factories` deles.

`trait` é uma variação com nome: `create(:contract, :cancelled)` é um contrato cancelado, com a data do cancelamento. O teste diz o que importa para ele e herda o resto.

As datas são fixas. Uma `factory` com `Date.today` fabrica contratos que vencem em dias diferentes conforme o dia em que o teste roda, e o teste que passava em maio falha em julho sem ninguém ter mexido.

```
create(:contract, :cancelled, equipment: pt121)
```

```

$ bundle exec rspec spec/models/equipment_spec.rb
.
1 example, 0 failures

```

Verde. Agora o verde diz uma coisa: um contrato **que o pátio aceitaria**, cancelado, não prende a máquina.

• O ESSENCIAL

Uma factory é uma afirmação sobre o que é um registro válido. A de 2016 afirmava que um contrato era um código. Todo teste construído sobre ela testava um objeto que não existe no pátio. A factory precisa passar pelas mesmas validações que o formulário — e o jeito de conferir é criar com `create` pelo menos uma vez.

Os nove testes antigos continuaram verdes com a factory nova. O que mudou não foi o resultado deles. Foi o que eles significam.

24.5 A reserva dupla, de novo

O script das seis reservas vira um teste:

SPEC/MODELS/CONTRACT_SPEC.RB

ruby

```
1 describe "reserva" do
2   it "cria uma reserva ao criar contrato ativo" do
3     expect { create(:contract, status: "active") }
4       .to change(Reservation, :count).by(1)
5   end
6
7   it "não cria reserva ao corrigir o responsável" do
8     contrato = create(:contract, status: "active")
9
10    expect { contrato.update!(responsible: "Juliana Prates") }
11      .not_to change(Reservation, :count)
12  end
13
14  it "cria a reserva quando a reserva vira ativa" do
15    contrato = create(:contract, status: "reserved")
16
17    expect { contrato.update!(status: "active") }
18      .to change(Reservation, :count).by(1)
19  end
20 end
```

`expect { ... }.to change(Reservation, :count).by(1)` roda o bloco e confere que a contagem de reservas subiu exatamente um. É a frase do script — “criar um contrato ativo cria uma reserva” — escrita para o computador conferir.

O segundo exemplo é o que o Diego mais queria: corrigir o nome do responsável **não** cria reserva. Se alguém, daqui a um ano, trocar o callback de volta para um `after_save` sem condição, este exemplo fica vermelho antes de chegar à homologação.

ERRO COMUM

Cada exemplo começa com o banco de teste limpo, e o que ele grava é desfeito no fim. Um teste não pode depender de outro ter rodado antes. Se depender, ele passa rodando a suíte inteira e falha rodando sozinho — ou o contrário, dependendo da ordem, que o RSpec embaralha de propósito.

RESUMO

- Teste confere um comportamento combinado. `describe`, `it` e `expect`.
- `build` monta sem gravar; `create` grava e passa pelas validações.
- Uma `factory` que só funciona com `build` fabrica registros que o sistema recusa. O vermelho do `create` mostra o que falta.
- `sequence` para campos únicos; `trait` para variações com nome; datas fixas.
- `change(...).by(1)` transforma “cria uma reserva” numa conferência. O exemplo do que **não** deve acontecer é o que protege a correção.

- 1 Escreva o exemplo que confere que um contrato fabricado com o `trait :overdue` está ocupando e atrasado em 4 de maio.
- 2 Escreva o teste da validação de sobreposição: duas reservas da mesma máquina, com períodos que se cruzam, e a mensagem com o código do primeiro contrato.
- 3 O Diego propõe um exemplo para cada coluna das vinte e três, “para cobertura”. O Renato pergunta o que a Helena ganha com isso. Responda os dois, e diga quais cinco exemplos você escreveria primeiro para o dia 3 de junho.

CAPÍTULO

25

Jobs

Salvar um contrato levava oito segundos. Sete e meio eram o e-mail para a Helena, que ela lia no dia seguinte.

AO FIM DESTES CAPÍTULOS

Tirar do pedido o trabalho que não precisa acontecer antes da resposta: escrever um job, agendá-lo depois que o banco confirmar, saber o que o worker é, e fazer o job aguentar rodar duas vezes.

**Capítulo disponível na
versão completa**

CAPÍTULO

26

APIs

A API devolvia status active. O sistema da Serra Azul comparava com ativo. Nenhum contrato da Nortea aparecia no painel deles.

AO FIM DESTES CAPÍTULO

Escrever o contrato da API numa tabela curta antes do código, responder JSON com os nomes e os valores combinados, recusar com 422 o pedido que não faz sentido, e manter a API separada das telas.

**Capítulo disponível na
versão completa**

CAPÍTULO

27

Autenticação

A credencial da Serra Azul era de leitura. O Diego mandou um POST com ela, e o nordea criou o contrato.

AO FIM DESTES CAPÍTULOS

Saber quem está do outro lado do pedido: a Helena entra com usuário e senha guardada do jeito certo; a Serra Azul consulta com uma credencial que só lê os próprios contratos; e o teste prova o 403 do que ela não pode fazer.

**Capítulo disponível na
versão completa**

CAPÍTULO

28

Deploy

Terça, 2 de junho, 17h40. O Sérgio queria só subir o ajuste. O pátio carregava a Serra Azul às seis da manhã seguinte.

AO FIM DESTES CAPÍTULOS

Levar o código da sua máquina ao servidor sabendo o que precisa ser igual entre os dois, em que ordem as coisas acontecem, como voltar atrás — e em que horário não se sobe nada.

**Capítulo disponível na
versão completa**

CAPÍTULO

29

Projeto final

Quarta, 3 de junho, 6h. Nove canteiros, a planilha numa mão, o tablet na outra. A Helena conferiu linha por linha.

AO FIM DESTE CAPÍTULO

Juntar o que o livro construiu num módulo que cobre a manhã de 3 de junho — a lista dos nove canteiros, o cancelamento que libera, a factory honesta, a consulta da Serra Azul — e fazer, sozinho, o próximo pedido que o pátio vai trazer.

**Capítulo disponível na
versão completa**

Respostas

Fevereiro de 1993

- 1 Ele começou em 24 de fevereiro de 1993. A entrevista de 2003 diz o critério: mais poderoso que Perl, mais orientado a objetos que Python.

De Smalltalk veio o bloco: código guardado num objeto e entregue a um método. A outra decisão é que não há valor solto ao lado dos objetos. O número e o `nil` também são objetos, e um método é mensagem enviada a um deles.

- 2 Esse framework era o Merb, de Ezra Zygmontowicz, de 2006. Em dezembro de 2008 o DHH anunciou que o Merb seria incorporado ao Rails 3 e que os times trabalhariam juntos.

Em agosto de 2010 saiu o Rails 3. O roteador e a fronteira com a biblioteca de banco foram reescritos. A convenção de nomes de 2004 — a classe `Contract` e a tabela `contracts` — ficou. O Merb, como produto separado, não seguiu.

- 3 Não com esse texto. O servidor está no 3.3. A manutenção normal acaba em 1º de abril de 2026; a de segurança segue até 31 de março de 2027. No dia 3 de junho o 3.3 ainda recebe segurança.

“A mais nova” aponta o 4.0 sem dizer se o `Gemfile` da Norteia já foi resolvido nessa linha. Trocar é nomear a linha — 3.4 ou 4.0 —, olhar o lock e escolher um horário em que o pátio das 6h não dependa do resultado. O ticket não faz nenhuma das três coisas.

O que vamos construir

- 1 Ela precisa saber qual contrato segura a PT-118 agora, e em qual canteiro. Se houver dois, os dois números. Se não houver nenhum, “livre”.

Ela não precisa saber se a resposta saiu de `start_date` ou de `started_at`, nem qual arquivo Rails montou a frase. A conferência dela é a plataforma no caminhão.

- 2 Três listas: a planilha, a tela de 2016 e a tela nova. O caminhão continua saindo pela planilha enquanto qualquer uma discordar.

O adendo pede uma consulta que bata com o pátio. Um terceiro lugar aumenta o número de lugares em que a PT-118 pode estar escrita, sem apagar os dois que já discordam.

- 3 Até 1º de abril de 2026 ele ainda está na manutenção normal: defeito comum e segurança. A partir dessa data, até 31 de março de 2027, só segurança.

No dia 3 de junho, portanto, um defeito que não seja de segurança não ganha correção do time do Ruby nessa linha. A Serra Azul não espera por isso. A data dela é a do adendo, não a do calendário do Ruby.

Blocos e enumerables

1

```
contratos.map { |c| c[:equipment].upcase }
contratos.select { |c| c[:status] == "cancelled" }
contratos.find { |c| c[:equipment] == "PT-118" }
contratos.count { |c| c[:status].nil? }
```

A terceira devolve o hash, ou `nil` se a PT-118 não estiver na lista.

2

```
def por_canteiro(contratos)
  contratos
    .group_by { |c| c[:site] }
    .transform_values { |lista| lista.map { |c|
      c[:equipment] } }
end
```

`group_by` agrupa os hashes por canteiro. `transform_values` troca cada valor — a lista de hashes — pelo que o bloco devolve: a lista de patrimônios. As chaves não mudam.

Um contrato com `site: nil` vira um grupo com chave `nil`. É o mesmo aviso da PT-133: o desconhecido aparece em vez de sumir.

3

Resolve a PT-133: o `nil` deixa de ser descartado.

O que passa a entrar errado é qualquer outro status desconhecido. Um `"closed"`, um `"suspended"`, um `"Active"` com maiúscula vindo de outra importação — todos contam como ocupando, porque não são `"cancelled"`. O defeito muda de lado: em vez de a lista esconder plataforma, ela passa a prender plataforma livre.

O que faria: manter o filtro pelo permitido, e tratar o desconhecido à parte, visível — a contagem de contratos sem status, e a lista deles para a Helena conferir. O desconhecido não é decidido por um filtro. É decidido por quem sabe onde a plataforma está.

O que é o Rails

- 1 Controller `EquipmentController`, no arquivo `app/controllers/equipment_controller.rb`. Método `show`, porque o caminho tem um identificador. View `app/views/equipment/show.html.erb`.

O controller não é `EquipmentsController`: em inglês, `equipment` não tem plural, e o Rails sabe.

- 2 `config/routes.rb`, para o endereço existir e apontar para um controller.

`app/controllers/occupancy_controller.rb`, com a classe `OccupancyController` e um método `index` que busca os dados.

`app/views/occupancy/index.html.erb`, com a página.

O model não precisa ser criado: os dados vêm de `Contract` e `Equipment`, que já existem.

- 3 A classe `Contrato` passaria a procurar a tabela `contratos`, que não existe. As associações que apontam para `:contract` e a coluna `contract_id` em outras tabelas deixariam de achar o model. As rotas `resources :contracts` continuariam chamando `ContractsController`, que buscaria `Contract` e não acharia. As views e os testes que chamam `Contract` quebrariam.

Cada uma dessas ligações pode ser reescrita à mão, com configuração. É o trabalho de vários dias para trocar um nome que a Helena nunca vê.

A clareza para a Helena mora no texto da tela: o título da página, o rótulo da coluna, a mensagem de erro. Esses já estão em português, e é lá que “Contrato” aparece para ela.

Testes

- 1


```

1  it "vencido e não devolvido continua ocupando" do
2    contrato = build(:contract, :overdue)
3    hoje = Date.new(2026, 5, 4)
4
5    expect(contrato.occupying?(hoje)).to be(true)
6    expect(contrato.overdue?(hoje)).to be(true)
7  end
      
```

`build` basta: o teste pergunta ao objeto e não precisa do banco. A factory honesta garante que esse mesmo contrato seria aceito se fosse gravado.

2

..

```

1  it "recusa a mesma máquina em períodos que se cruzam" do
2    pt118 = create(:equipment, patrimony: "PT-118")
3    primeiro = create(:contract, equipment: pt118,
4                      start_date: Date.new(2026, 5, 4),
5                      end_date: Date.new(2026, 6, 30))
6
7    segundo = build(:contract, equipment: pt118,
8                   start_date: Date.new(2026, 6, 1),
9                   end_date: Date.new(2026, 7, 15))
10
11    expect(segundo).not_to be_valid
12    expect(segundo.errors[:equipment].first)
13      .to include(primeiro.code)
14  end

```

`be_valid` chama `valid?`. O primeiro contrato é criado — precisa estar no banco para a validação encontrá-lo. O segundo só é montado: o que se confere é que ele seria recusado.

3

...

Um exemplo por coluna confere que o Rails lê coluna — o que o Rails já testa. A cobertura sobe e nenhum comportamento que custa multa fica protegido. A Helena não ganha nada.

Os cinco primeiros, na ordem da multa:

1. Contrato cancelado libera a máquina nos disponíveis.
2. Vencido e não devolvido continua ocupando.
3. A mesma máquina em dois períodos que se cruzam é recusada.
4. Criar contrato ativo cria uma reserva; corrigir o responsável não.
5. Contrato sem responsável é recusado.

Cada um corresponde a uma ligação da Helena ou a um print do Diego deste livro. É esse o critério: um exemplo por defeito que já aconteceu, antes de um exemplo por linha de código.

Glossário

Bloco Um objeto que guarda um trecho de código. Dá para entregar esse objeto a um método. O método decide quando chamar o trecho, e com qual valor. A ideia veio de Smalltalk. No Ruby ela está desde o começo: percorrer uma lista, filtrar, transformar, tudo isso é um método que recebe um bloco.

Bloco Um trecho de código entre `{ }` ou `do ... end`, entregue a um método na chamada. O método decide quando executá-lo e com quais valores. Os parâmetros vêm entre `| |`. Um método recebe no máximo um bloco, e o bloco não é um argumento entre os parênteses: vem depois deles.

Manutenção As duas fases de uma linha do Ruby. Normal: defeito e segurança. Segurança: só segurança, até a data de fim publicada. “Está suportado” sem dizer a fase não diz se um defeito comum ainda ganha correção. No servidor da Nortea, em março de 2026, essa diferença tem data.

Rails Framework escrito em Ruby, extraído do Basecamp e publicado em 2004. Já traz a estrutura de um sistema web: o pedido entra por uma rota, passa por um controller, fala com um model, e a resposta volta. A Nortea roda um Rails desde 2016. O framework não é a linguagem. É um programa Ruby que espera os arquivos da Nortea em pastas combinadas.

Teste Um programa que executa um pedaço do sistema e confere o resultado. `describe` diz o que está sendo testado; `it` diz o comportamento; `expect` confere. Um teste verde diz que o comportamento descrito na frase aconteceu. Não diz nada sobre o que a frase não descreve.

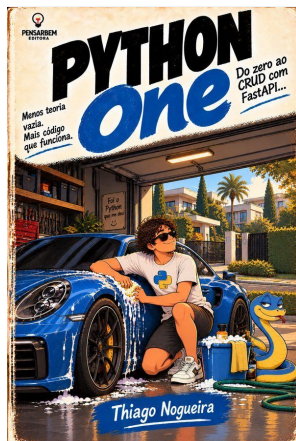
YARV A máquina virtual escrita por Koichi Sasada. A partir do Ruby 1.9 ela é o executor da implementação de referência, a escrita em C, no lugar do interpretador antigo. Máquina virtual, aqui, é o programa que lê o Ruby e o executa. Trocar a máquina virtual não troca o nome da linguagem. Troca o que acontece por baixo do arquivo, e às vezes o que o arquivo precisa declarar — como a codificação do texto, no 1.9.

Conheça as demais obras dessa coleção



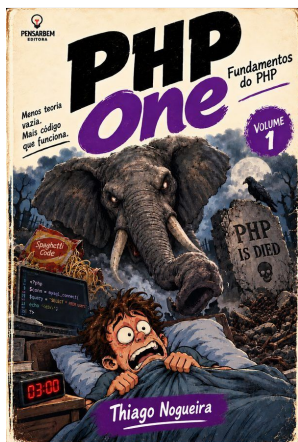
Java One

Do primeiro println à API REST em produção



Python One

Do primeiro print à API REST em produção



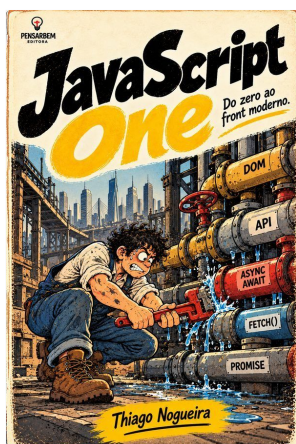
PHP One

Volume 1 · Fundamentos do PHP



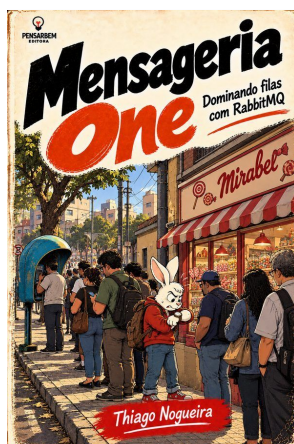
PHP One

Volume 2 · Acelerando com Laravel



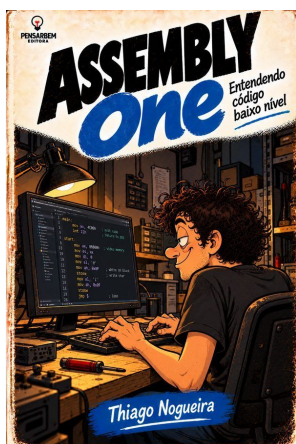
JavaScript One

Do primeiro console.log ao painel em produção



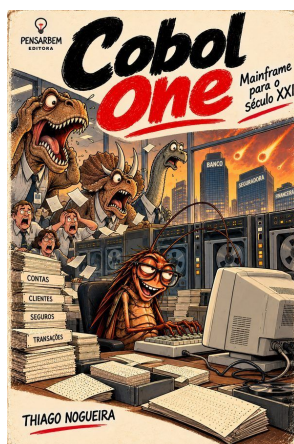
Mensageria One

RabbitMQ do primeiro publish à Black Friday sem perder pedido



Assembly One

Do zero ao RISC-V



COBOL One

Mainframe para o Século XXI

Visite a editora Pensar Bem →

pensarbem.com.br