

06 | Ruby on Rails: 如何分析一个软件的接口?

2020-06-05 郑晔

软件设计之美

[进入课程 >](#)



讲述: 郑晔

时长 15:59 大小 14.65M



你好! 我是郑晔。

在上一讲中, 我以 Spring 的 DI 容器为例, 给你讲解了如何理解一个项目的模型。在模型之后, 下一步就应该是接口了。

在任何一个项目中, 接口的数量都不是一个小数目。仅仅一个普通的程序库, 里面的接口少则几十个, 多则成百上千。难道我们理解接口, 就是要一个一个地读这些接口吗?

显然, 你不太可能把所有的接口细节都记住。我写 Java 程序差不多 20 年了, 但很多里的类我都不了解。甚至有时候, 还没有等我去了解这个类, 它就过时了。



那么, 如何才能从纷繁复杂的接口中, 披荆斩棘而出呢? 我给你个方法: **找主线, 看风格。**

找主线的意思是，你需要找到一条功能主线，建立起对这个项目结构性的认知，而不是一上来就把精力放在每一个接口的细节上。你对细节部分的了解会随着你对项目的深入而逐渐增加。而有了主线后，你就有了着力点，就可以不断深入了。

但是，我们要学习的不只是这些接口的用法，要想从项目的接口设计上学到更多，这就需要你关注它所引导的**风格**，换句话说，就是它希望你怎样使用它，或是怎样在上面继续开发。

从一个项目的接口风格中，我们不难看出设计者的品位。我们常把编程视为一种艺术，而在接口的设计上就能窥见一二。这些内容是我们在学习软件设计时，应该慢慢品味的。

为什么要看风格，还有一个很重要的原因，就是你要维护项目的一致性，必须有一个统一的风格。有不少项目，里面会共存多种不同风格的接口，就是每个人都在设计自己习惯的接口，那势必会造成混乱。

这一讲，我们就来一起来学习怎样看接口，我选择的项目是 Ruby on Rails，因为它的接口设计风格是带给我最多震撼的，无论是编程接口的优雅，还是开发过程接口的顺畅。不过，正如我在 [第 4 讲](#) 所说，看设计要先看模型。所以，我们还是先快速地了解一下 Ruby on Rails 的模型。

Ruby on Rails 模型

如果你是一个比较年轻的程序员，Ruby on Rails 这个名字可能会让你有一些陌生。但是在十多年前，它初出茅庐之际，可是给行业带来了极大的冲击。只是后来时运不济，编程模型发生了大的改变，使它失去了行业领导者的地位。这个故事还是要从你最熟悉的 Web 开发说起。

自从互联网兴起，人们对于 Web 开发的探索就从未停止过。

最早期的 Web 开发只是静态页面的开发，那时候，你只要熟悉 HTML，就可以说自己懂 Web 开发了。后来，人们不再满足于静态页面，开始尝试编写有动态效果的页面。

一方面，浏览器开始支持 JavaScript，让页面本身有了动态效果；另一方面，有人开始制作后台服务，在页面之间切换的时候，也可以有动态的效果。那个时候出现了像 CGI (Common Gateway Interface) 这样的编程规范。

当 Java 世界里出现了 Servlet、JSP 这些规范，Web 开发也逐渐由小打小闹变成了企业开发的主力，越来越多的公司开始正视 Web 开发。因为这些规范很沉重，一些号称要简化 Web 开发的框架开始出现，比如：Struts、Webwork 以及 Spring MVC 等等。

这些框架的出现，让 Web 开发摆脱了 Servlet 的初级阶段，使 MVC 模式成为了 Web 开发的主流。但即便如此，那个时候的 Java Web 开发依然是沉重的，比如写一个 Web 应用，光是配置文件就足以把人逼疯。

Ruby on Rails 正是在这样的背景下横空出世的。为了叙述方便，后面我就把 Ruby on Rails 简称 Rails 了。

从模型上讲，Rails 是标准的**基于 MVC 模型进行开发的 Web 框架**。在这一点上，它没有什么特殊的，它给行业带来巨大冲击的是它的接口设计。

Rails 一个重要的设计理念就是**约定优于配置**，无需配置，按照缺省的风格就可以完成基本的功能，这样的理念贯穿在 Rails 各个接口的设计中。

接下来，我们就来看 Rails 的接口。

前面我提到过理解接口应该先找主线，**找到项目主线的一个方法就是从起步走文档开始，因为它会把项目最基本的用法展现给你，你可以轻松地找到主线。**

Rails 的起步走文档做得就非常好，主线可以说是一目了然。它用了个 Web 项目帮你介绍了 Rails 开发的基本过程，通过这个过程，你就对 Rails 有了初步的印象。

有了主线之后，我们就要开始从中了解接口的风格。Rails 给我们提供的三种接口，分别是：

Web 应用对外暴露的接口：REST API；

程序员写程序时用到的接口：API；

程序员在开发过程中用到的接口：命令行。

接下来，我们就一个个地深入其中，了解它们的风格，以及它们给行业带来的不同思考。

REST 接口

先说应用对外暴露的接口：REST API。REST 如今已经成为很多人耳熟能详的名词，它把 Web 的各种信息当作资源。既然是资源，它就可以对这些 Web 信息做各种操作，这些操作对应着 HTTP 的各种动词（GET、POST、PUT、DELETE 等）。


REST 当年的问世是 Roy Fielding 博士为了纠正大家对 HTTP 的误用。REST 刚出来的时候，开发者普遍觉得这是一个好的想法，但怎么落地呢？没有几个人想得清楚。

Rails 恰逢其时地出现了。Rails 对 REST 的使用方式做了一个约定。只要你遵循 Rails 的惯用写法，写出来的结果基本上就是符合 REST 结构的，也就是说，Rails 把 REST 这个模型用一种更实用的方式落地了。

 复制代码

```
1 Rails.application.routes.draw do
2   ...
3   resources :articles
4   ...
5 end
```

在用 Rails 写程序的时候，你只要添加一个 resource 进去，它就会替你规划好这个资源应该如何去写、怎么设计 URL、用哪些 HTTP 动词，以及它们对应到哪些方法。

 复制代码

```
1 $ bin/rails routes
2      Prefix Verb   URI Pattern                      Controller#Action
3  articles GET    /articles(.:format)             articles#index
4                POST   /articles(.:format)             articles#create
5  new_article GET    /articles/new(.:format)          articles#new
6  edit_article GET    /articles/:id/edit(.:format)     articles#edit
7    article GET    /articles/:id(.:format)          articles#show
8                PATCH  /articles/:id(.:format)          articles#update
9                PUT    /articles/:id(.:format)          articles#update
10               DELETE /articles/:id(.:format)          articles#destroy
11    root GET    /                                welcome#index
```


看了 Rails 给你的这个映射关系后，你就知道自己该怎么写代码了。这就是一种约定，不需要你费心思考，因为这是人家总结出来的行业中的最佳实践。只要按照这个规范写，你写的

就是一个符合 REST 规范的代码，这就是 Rails 引导的外部接口风格。

API 接口

我们再来看 API 接口。当年我接触 Rails 时，最让我感到震惊的是它的数据库查询方式，与传统开发的风格截然不同，就这么简单的一句：


```
1 Article.find_by_title("foo")
```

 复制代码

要知道，那个时候用 Java 写程序，即便是想做一个最简单的查询，写的代码也是相当多的。我们不仅要创建一个对象，还要写对应的 SQL 语句，还要把查询出来的结果，按照一定的规则组装起来。

而 Rails 用一句轻描淡写 `find_by` 就解决了所有的问题，而且，这个 `find_by_title` 方法还不是我实现的，Rails 会替你自动实现。当我们需要有更多的查询条件时，只要一个一个附加上去就可以了。

```
1 Article.find_by_title_and_author("foo", "bar")
```

 复制代码

同样的事，如果放到 Java 里去做，还需要把前面说的事再做一遍，差别只是查询语句不一样。

虽然我说的是当年的场景，但时至今日，在这些简单问题上，很多使用 Java 的团队所付出的工作量并不比当年少。

从功能的角度说，这样的查询在功能上是完全一样的，但显然 Rails 程序员和 Java 程序员的工作量是天差地别的。这其中的差异就是不同的编程接口所造成的。

所以你看，一个好的接口设计会节省很多工作量，会减少犯错的几率。因为它会在背后帮你实现那些细节。


而设计不好的接口，则会把其中的细节暴露出来，让使用者参与其中。写程序库和写应用虽然都是写代码，但二者的要求确实相差极大。把细节暴露给所有人，显然是一个增加犯错几率的事情。

Rails 的 API 接口给行业带来的另一个影响是，它让人们开始关注 API 的表达性。比如，每篇文章可以有多个评论，用 Rails 的方式写出来是这样的：

 复制代码

```
1 class Article < ApplicationRecord
2   has_many :comments
3   ...
4 end
```

而如果用传统 Java 风格，你写出来的代码，可能是这个样子的：

 复制代码

```
1 class Article {
2   private List<Comment> comments;
3   ...
4 }
```

很明显，“有多个”这种表示关系的语义用 `has_many` 表示更为直白，如果用 `List`，你是无法辨别它是一个属性，还是一个关系的。


Rails 里面类似的代码有很多，包括我们前面提到的 `find_by`。所以，如果你去读 Rails 写成的应用，会觉得代码的可读性要好得多。

由于 Rails 的蓬勃发展，人们也开始注意到好接口的重要性。Java 后期的一些开源项目也开始向 Rails 学习。比如，使用 Spring Data JPA 的项目后，我们也可以写出类似 Rails 的代码。声明一对多的关系，可以这样写：

 复制代码


```
1 class Article {
2   @OneToMany
3   private List<Comment> comments;
4   ...
5 }
```


而查询要定义一个接口，代码可以这样写：

 复制代码

```
1 interface ArticleRepository extends JpaRepository<Article, Long> {  
2     Article findByTitle(String title);  
3     Article findByTitleAndAuthor(String title, String author);  
4 }
```

当你需要使用的时候，只要在服务里调用对应的接口即可。

 复制代码

```
1 class ArticleService {  
2     private ArticleRepository repository;  
3     ...  
4     public Article findByTitle(final String title) {  
5         return repository.findByTitile(title);  
6     }  
7 }
```

显然，Java 无法像 Rails 那样不声明方法就去调用，因为这是由 Ruby 的动态语言特性支持的，而 Java 这种编译型语言是做不到的。不过比起从前自己写 SQL、做对象映射，已经减少了很多的工作量。

顺便说一下，Spring Data JPA 之所以能够只声明接口，一个重要的原因就是它利用了 Spring 提供的基础设施，也就是上一讲提到的依赖注入。它帮你动态生成了一个类，不用你自己手工编写。


简单，表达性好，这就是 Rails API 的风格。

命令行接口

作为程序员，我们都知道自动化的重要性，但 Rails 在“把命令行的接口和整个工程配套得浑然一体”这个方面做到了极致。Rails 的自动化不仅会帮你做一些事情，更重要的是，它还把当前软件工程方面的最佳实践融合进去，这就是 Rails 的命令行风格。

如果要创建一个新项目，你会怎么做呢？使用 Rails，这就是一个命令：

```
1 $ rails new article-app
```

 复制代码

这个命令执行的结果生成的不仅仅是源码，还有一些鼓励你去做的最佳实践，比如：

它选择了 Rake 作为自动化管理的工具，生成了对应的 Rakefile；

它选择了 RubyGem 作为包管理的工具，生成了对应的 Gemfile；

为防止在不同的人在机器上执行命令的时间不同，导致对应的软件包有变动，生成了对应的 Gemfile.lock，锁定了软件包的版本；

把对数据库的改动变成了代码；

.....

而这仅仅是一个刚刚生成的工程，我们一行代码都没有写，它却已经可以运行了。


```
1 $ bin/rails server
```

 复制代码

这就启动了一个服务器，访问 <http://localhost:3000/> 这个 URL，你就可以访问到一个页面。

如果你打算开始编写代码，你也可以让它帮你生成代码骨架。执行下面的命令，它会帮你生成一个 controller 类，生成对应的页面，甚至包括了对应的测试，这同样是一个鼓励测试的最佳实践。

```
1 $ bin/rails generate controller Welcome index
```

 复制代码

在 Rails 蓬勃发展的那个时代，人们努力探索着 Web 开发中各种优秀的做法，而在这个方面走在最前沿的就是 Rails。所以，那个时候，我们经常会关注 Rails 的版本更新，看看又

有哪些好的做法被融入其中。

Rails 中那些优秀的实践逐步地被各种语言的框架学习着。语言编写者们在设计各种语言框架时，也都逐步借鉴了 Rails 中的那些优秀实践。比如，今天做 Java 开发，我们也会用到数据库迁移的工具，比如 Flyway。

当然，另一个方面，即便到了今天，大部分项目的自动化整合程度也远远达不到 Rails 的高度，可能各方面的工具都有，但是如此浑然一体的开发体验，依然是 Rails 做得最好。

最后，你可能会问，Rails 这么优秀，为什么今天却落伍了呢？

在 Web 开发领域，Rails 可谓成也 MVC，败也 MVC。MVC 是那个时代 Web 开发的主流，页面主要在服务端进行渲染。然而，后来风云突变，拜 JavaScript 虚拟机 V8 所赐，JavaScript 能力越来越强，Node.js 兴起，人们重新认识了 JavaScript。它从边缘站到了舞台中心，各种组件层出不穷，前端页面的表现力大幅度增强。

Web 开发的模式由原来的 MVC，逐渐变成了前端提供页面，后端提供接口的方式。Java 的一些框架和服务也逐步发展了起来，Spring 系列也越来越强大，重新夺回了 Web 后端开发的关注。

总结时刻

今天，我们学习如何了解设计的第二部分：看接口。看接口的一个方法是**找主线，看风格**。先找到一条功能主线，对项目建立起结构性的了解。有了主线之后，再沿着主线把相关接口梳理出来。

查看接口，关键要看接口的风格，也就是项目作者引导人们怎样使用接口。在一个项目里，统一接口风格也是很重要的一个方面，所以，熟悉现有的接口风格，保持统一也是非常重要的。

我还介绍了一个曾经火爆的 Web 开发框架：Ruby on Rails。借着它的起步走文档，我给你介绍了它的一些接口，包括：

Web 应用对外暴露的接口：REST API；

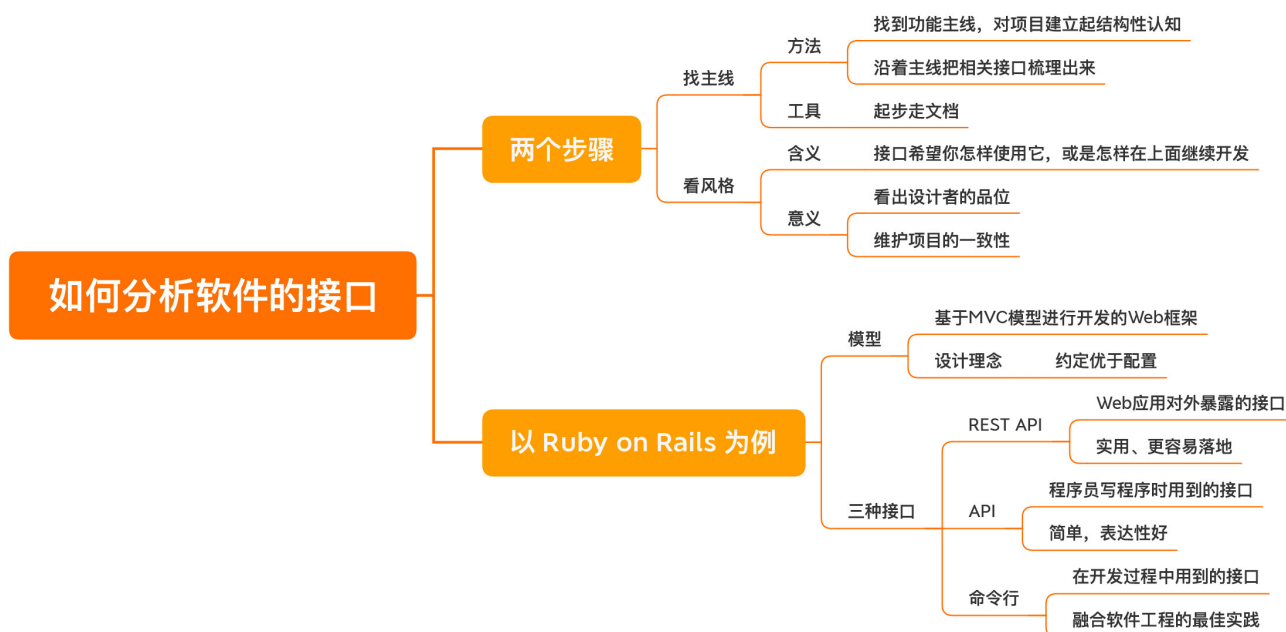
程序员写程序时用到的接口：API；

程序员在开发过程中用到的接口：命令行。

从 Rails 的接口设计中，我们可以看到，一个好的接口设计，无论是最佳实践的引入，抑或是 API 设计风格的引导，都可以帮助我们建立起良好的开发习惯。

当我们理解了模型和接口，接下来就该看实现了，这就是我们下一讲要讲的内容。

如果今天的内容你只能记住一件事，那请记住：**理解一个项目的接口，先找主线，再看风格。**



思考题

最后，我想请你来分享一下，你在哪个项目的设计中学到了一些好的开发习惯呢？欢迎在留言区分享你的经历。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

更多福利推荐

充值限时膨胀

充 ¥500 得 ¥580

下单即赠精选爆款商品

戳此充值

充 ¥500 得 ¥580

充 ¥1000 得 ¥1200

充 ¥2000 得 ¥2500



© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | Spring DI容器：如何分析一个软件的模型？

下一篇 07 | Kafka：如何分析一个软件的实现？

精选留言 (10)

写留言



Jxin

2020-06-05

1.最早自学的就是ruby，要不是因为找不到工作，可能就做不成javaer了。论快速搭建一个web项目，至今依旧是ruby on rails。一个多小时从无到有搭建一个博客系统的时候，信心爆棚。

2.本篇，明天得再看看。get不到点。只能理解风格应该是说设计偏好。至于主线，从ru...
展开



6



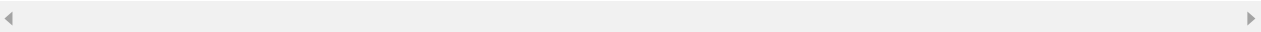
捞鱼的搬砖奇

2020-06-05

在 Spring 的源码中 接口 -> 抽象类->实现类，如 BeanDefinition -> AbstractBeanDefinition -> RootBeanDefinition。这样的设计风格。顶层接口规定定义，抽象类提供部分实现。用户需要扩展，可以选择从抽象类进行扩展，或从接口扩展。

展开 ▾

作者回复: 上一讲是 Spring, 这一讲是 Ruby on Rails, 这个评论让我有一种走错片场的感觉。:)



👍 4

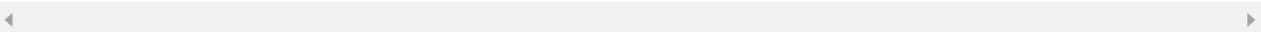


Kăfkă²⁰²⁰

2020-06-05

怀念RoR, 起初用过, 后来因为性能, 我们都替换成Java了

作者回复: 曾经沧海



👍 2



escray

2020-06-05

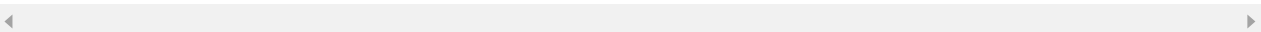
“优雅的编程接口, 顺畅的开发过程”, 虽然我也非常的喜欢 Ruby on Rails, 却没有办法像老师这样精辟的总结提升。

给我的感觉, RoR 在设计的时候非常的体贴程序员, 并且采用了一大堆优秀的设计范式。

...

展开 ▾

作者回复: 很快就轮到讲程序设计语言了, 简言之, 多学点。



👍 2



阳仔

2020-06-05

理解软件中的接口设计, 要抓住主线, 可以从文档开始入手, 了解软件设计者的风格品味, 看看作者希望我们是如何使用这些接口的。

我没用过Ruby, 但是通过分析之后, 其实它的接口设计中, 整合了许多极佳的工程实践, 提高编码效率, 解放生产力, 这些思想在软件设计的时候是可以学习和参考的

展开 ▾



👍 2



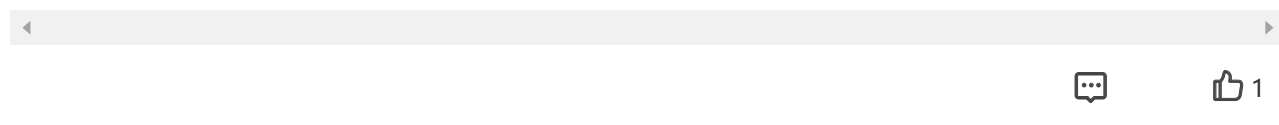
王智

2020-06-06

Ruby on Rails这个设计在当时感觉很超前,也不知道现在的SpringBoot是不是借鉴了,感觉从SpringBoot上能看倒Ruby on Rails的身影, SpringBoot的约定大于配置,还SpringBoot把命简洁的页面,勾选之后就可以创建一个简单的SpringBoot项目等等,Ruby on Rails的设计在现在看来可能确实没什么,但在当时感觉这个设计就太超前了.

展开 ∨

作者回复: 你说得很对, Rails超前是全方位的, 今天看来, 很多东西都影响了全行业。



qinsi

2020-06-05

最早接触BDD是cucumber, 当时觉得简直是magic。整个RoR框架也到处给人magic的感觉。但是软件开发不应该有magic, 否则容易失控。RoR式微部分原因可能也是Ruby过于灵活了, 印象中RoR有些严重的安全漏洞就是由此引发。此外Ruby的执行效率也逐渐落后于时代了。近年来有Crystal语言借由llvm复兴Ruby, 加入了很多现代语言的特性, 希望能有好结果。

展开 ∨



六一王

2020-06-08

接触的第一个框架就是 rails, 用起来确实很爽, 今天看到此篇文章, 让我对它更有敬意, 而不会因为现在不就行, 或者性能不够好, 就看不起它, 没有东西是绝对的好, 或者绝对不好, 今天全是对这个道理有更深层的理解了。

我之后学习了JavaScript, 转了前端, 学习 Vue 框架, 它使用虚拟dom将组件高度抽象化, 使页面组件可以多端运行, 比如node端, 如此可以前后端同构, 解决单页面应用的...

展开 ∨



Hank_Yan

2020-06-07

找主线, 看风格。找主线看文章, 看风格看接口。从上到下, 从整体到局部。不过这也是正常读源码的一个步骤。

展开 ∨



再来二两杜康酒

2020-06-07

好的设计要多从使用者角度考虑, 是否有助于释放程序员精力, 是否易用, 是否有良好的

可读性和可扩展性。自己先用，并在开始时就设定好边界，即使先只在一点上有所突破也比全面开花哪哪不灵要好。

