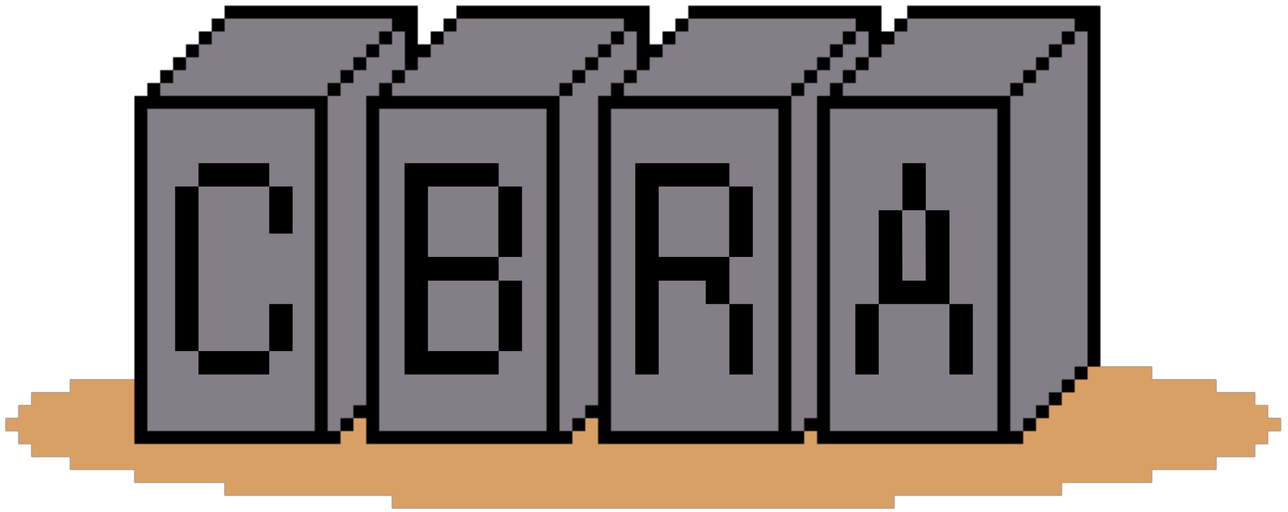
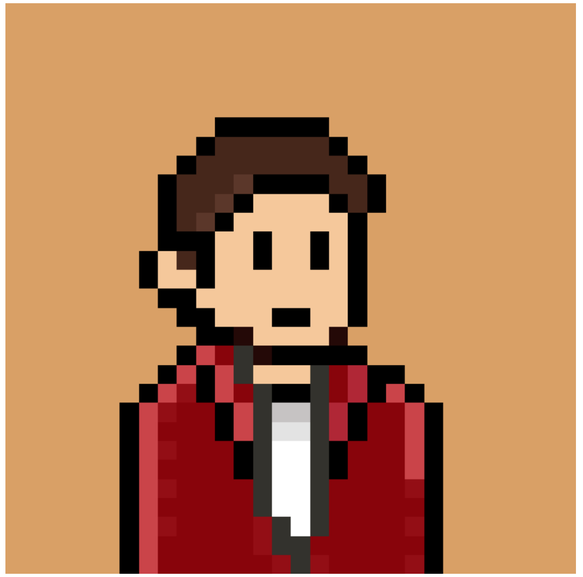
模块化的Rails，微服务以外的另一种选择

@Madao

github.com/Madao-3



自我介绍



@Madao

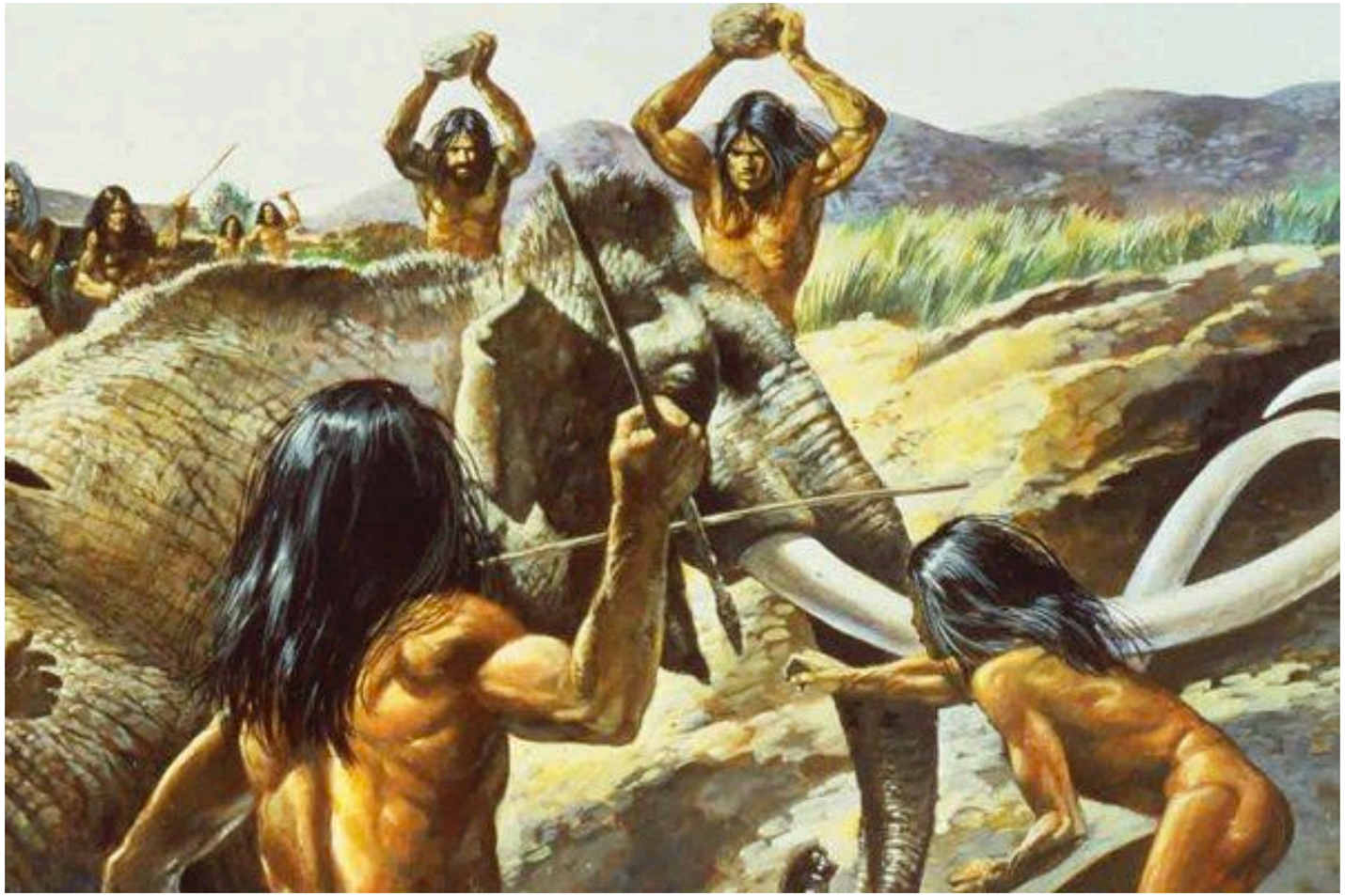
github.com/Madao-3

野生全栈工程师

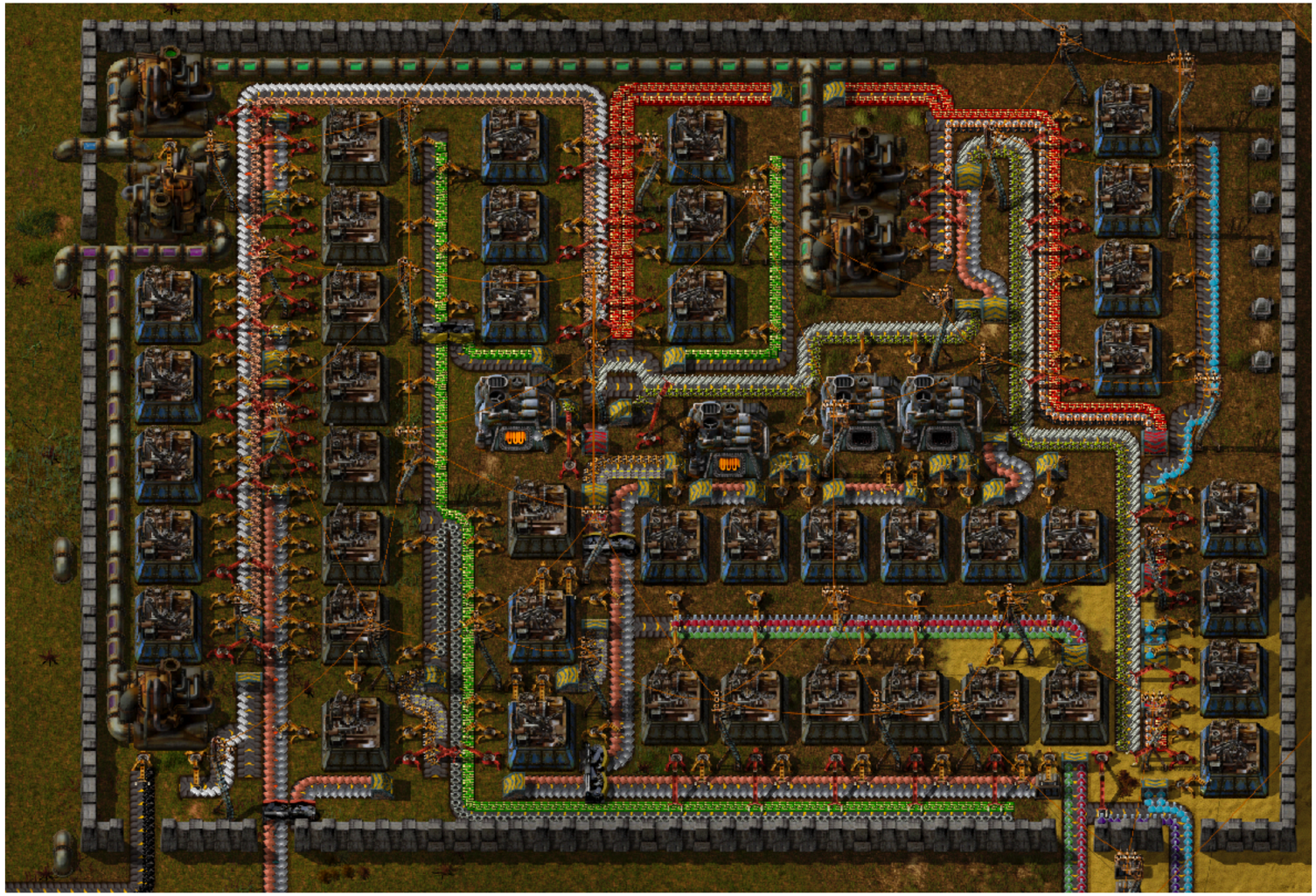
为什么不选择微服务?

单体应用优先策略

最早的时候我们是这样

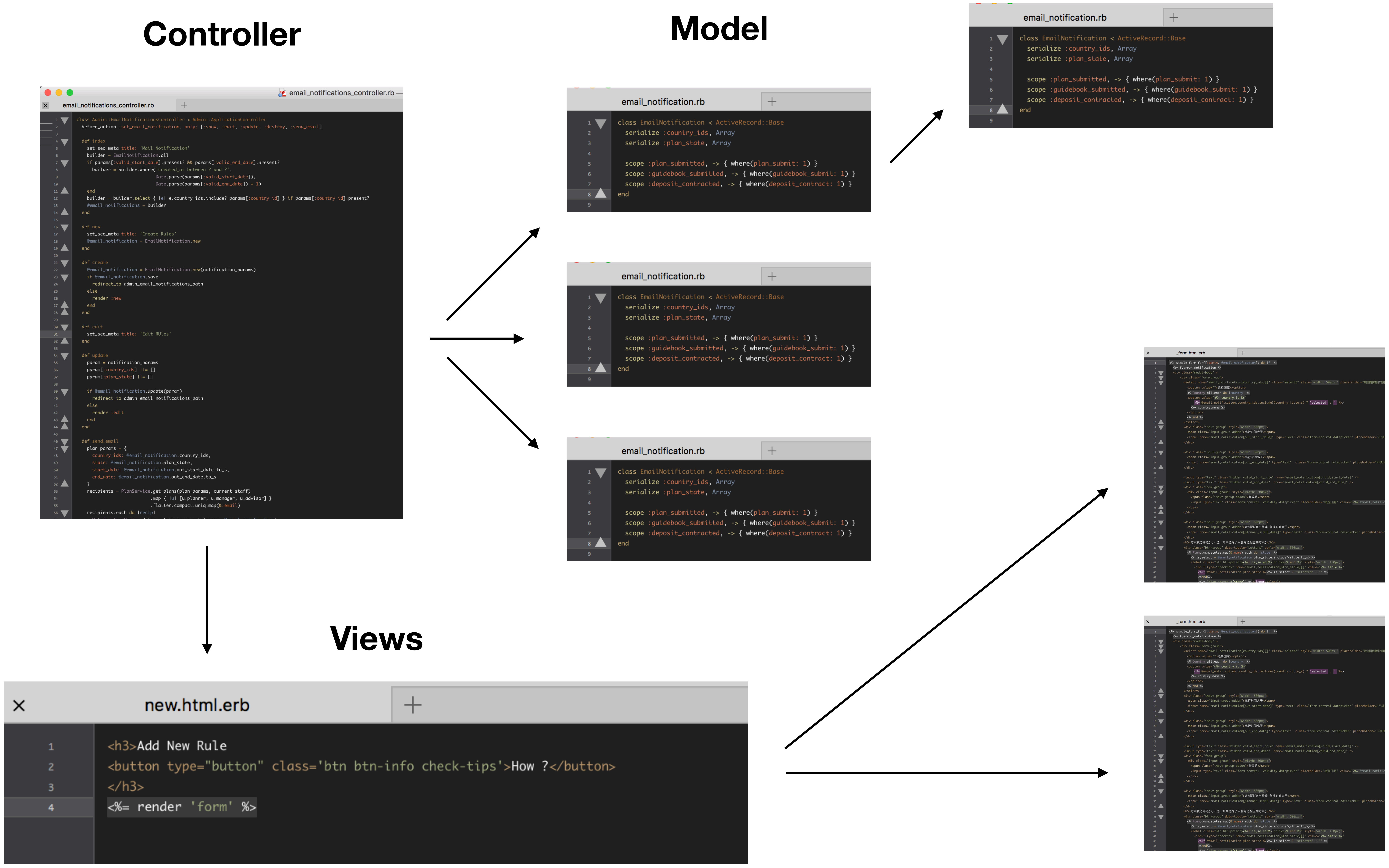


我们以为项目演进项之后项目会变这样

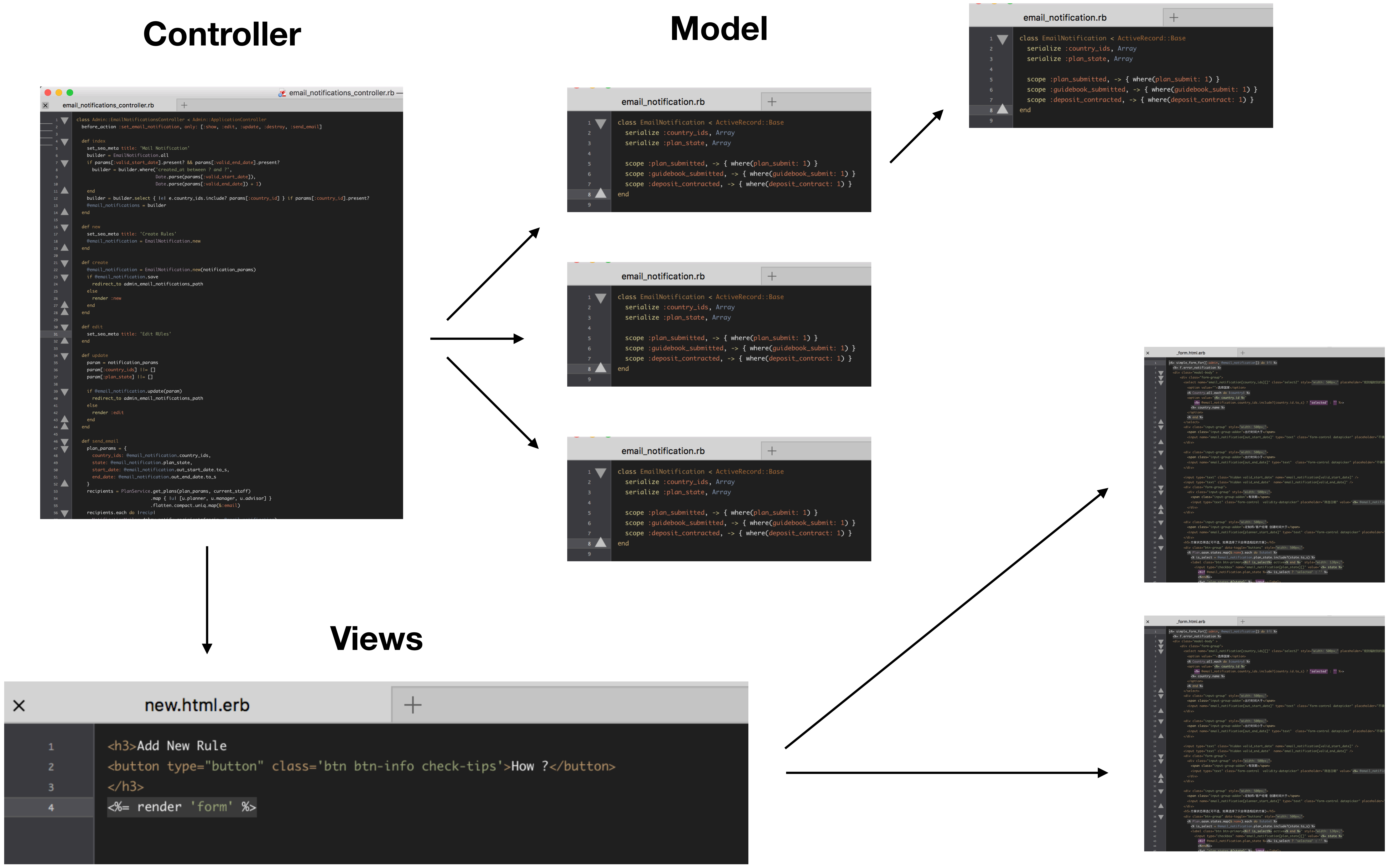


但其实一般是这样

让我们来假设一个场景


Controller
email_notifications_controller.rb
Model
email_notification.rb
class EmailNotification < ActiveRecord::Base
serialize :country_ids, Array
serialize :plan_state, Array
scope :plan_submitted, -> { where(plan_submit: 1) }
scope :guidebook_submitted, -> { where(guidebook_submit: 1) }
scope :deposit_contracted, -> { where(deposit_contract: 1) }
end
Views
new.html.erb
<h3>Add New Rule
<button type="button" class='btn btn-info check-tips'>How ?</button>
</h3>
<%= render 'form' %>
_form.html.erb

回忆一下常规单体开发的历程




Controller
Model
email_notifications_controller.rb
email_notification.rb
new.html.erb
BOOM!

需要解决的问题

1.降低单体应用的复杂度
2.避免过慢的CI
3.便于生成依赖关系图
4.减少实现过程中所需的上下文内容
5.快速交付

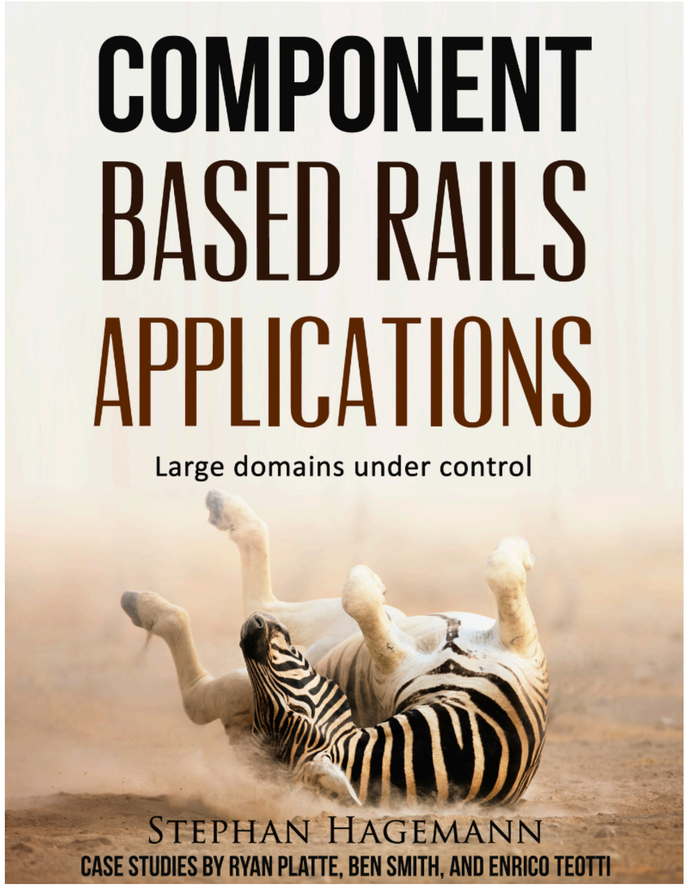
那么我们有什么办法呢?

首先要保证能够持续高效地交付，这时候你会发现继续维持原有的单体架构是不可行的。你需要拆分它，**降低它的复杂度，并且确定拆分的边界**。

那还不是微服务嘛

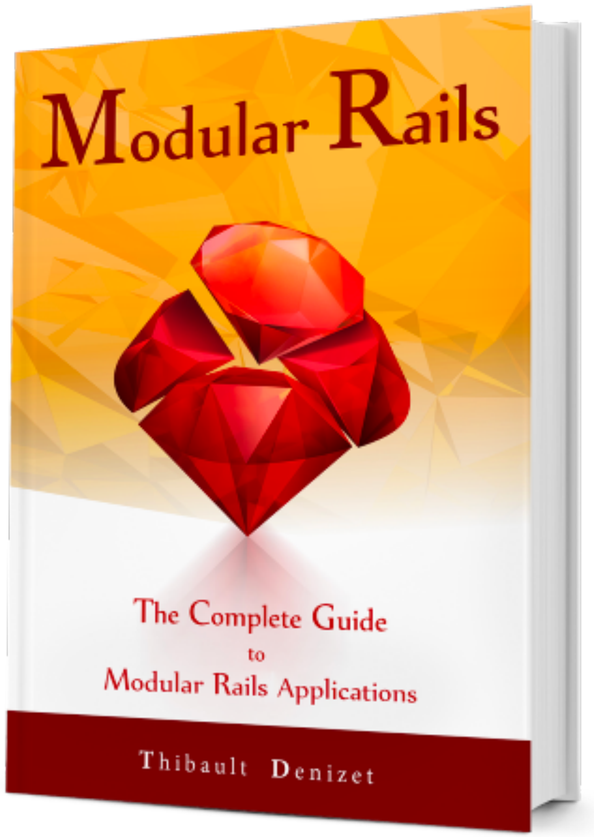
如果我们可以把每一个拆分出来的子 Rails 应用当做Gem 呢?

Component-Based Rails Applications



旧版本：http://shageman.github.io/cbra.info/

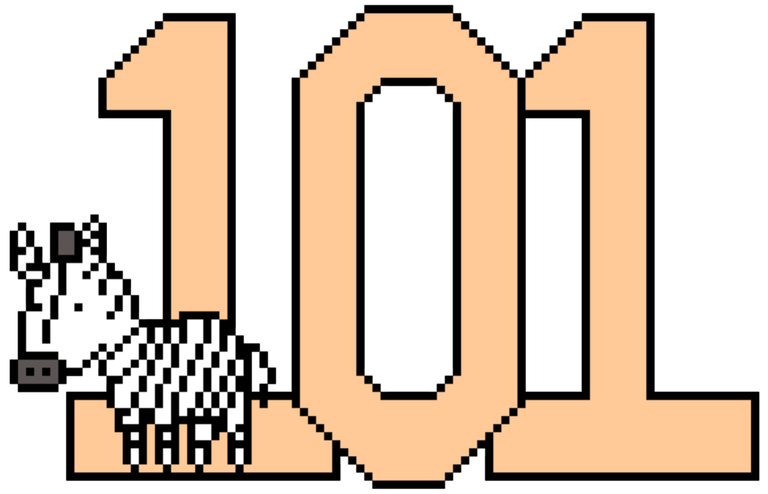
@shageman



http://modular-rails.samurails.com/

@T_Dnzt

让我们实践一下



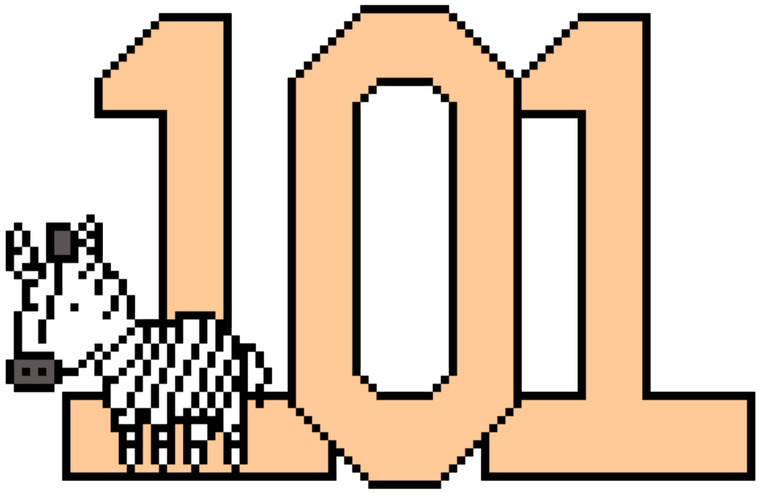
让我们开始我们的第一步：

```
rails new cbra
```

然后

```
cd cbra
```

```
rm -rf ./app
```



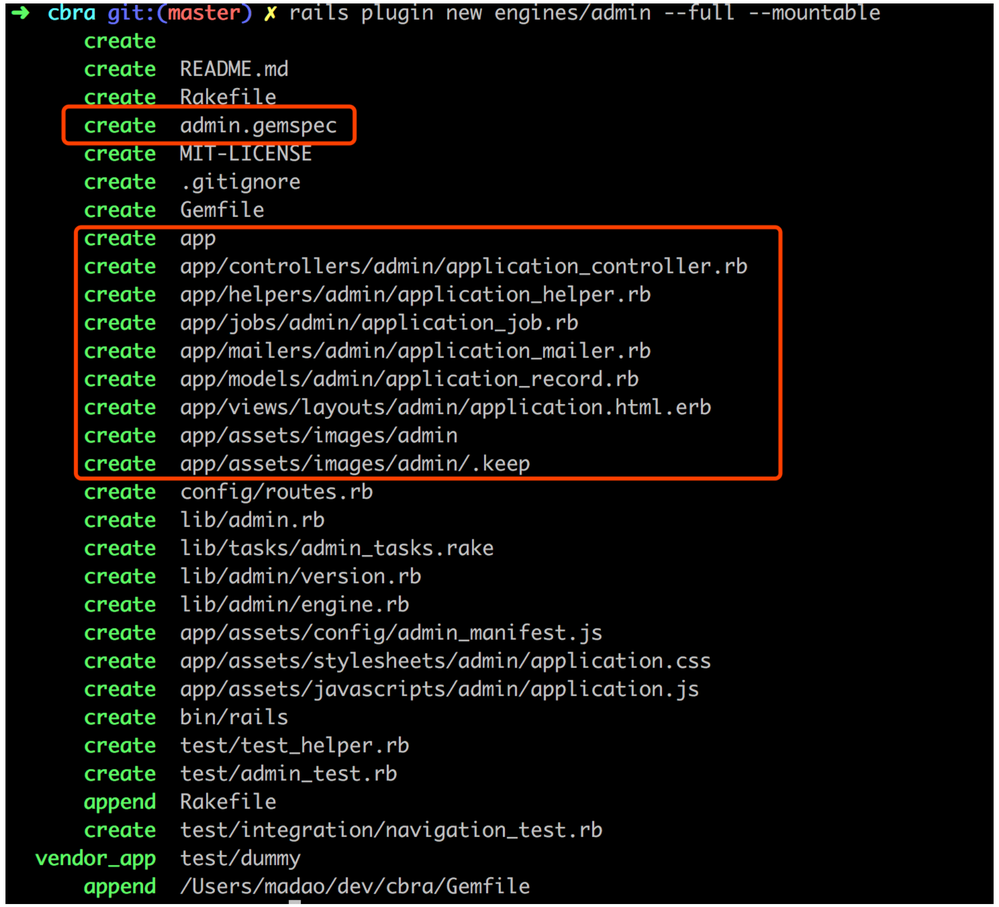
收工！

先别急着鼓掌

还有下文

假设我们要实现后台（admin）模块

```
rails plugin new engines/admin --mountable
```



Gemfile

```
64  gem 'admin', path: 'engines/admin'
```

然后执行 bundle install

```
➜  cbra git:(master) ✗ bundle install
You have one or more invalid gemspecs that need to be fixed.
The gemspec at /Users/madao/dev/cbra/engines/admin/admin.gemspec is not valid. Please fix this gemspec.
The validation error was '"FIXME" or "TODO" is not a description'
```

填充下必要的内容

admin.gemspec — admin

```ruby
$:.push File.expand_path("lib", __dir__)

# Maintain your gem's version:
require "admin/version"

# Describe your gem and declare its dependencies:
Gem::Specification.new do |s|
  s.name        = "admin"
  s.version     = Admin::VERSION
  s.authors     = ["Madao"]
  s.email       = ["madao@madao.me"]
  s.homepage    = "admin.yourdomain.com"
  s.summary     = "This is a Admin Module."
  s.description = "This is a Admin Module For A CBRA Project."
  s.license     = "MIT"

  s.files = Dir["{app,config,db,lib}/**/*", "MIT-LICENSE", "Rakefile",
"README.md"]

  s.add_dependency "rails", "~> 5.2.1"

  s.add_development_dependency "sqlite3"
end
```

首先

```
cd engines/admin
```

```
rails g controller welcome
```

```
➜  admin git:(master) ✗ rails g controller welcome
      create  app/controllers/admin/welcome_controller.rb
      invoke  erb
      create    app/views/admin/welcome
      invoke  test_unit
      create    test/controllers/admin/welcome_controller_test.rb
      invoke  helper
      create    app/helpers/admin/welcome_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    js
      create      app/assets/javascripts/admin/welcome.js
      invoke    css
      create      app/assets/stylesheets/admin/welcome.css
```

创建好Welcome#index 这个action 所需的内容

然后编辑路由

```
1 Rails.application.routes.draw do
2   mount Admin::Engine => "/admin"
3 end
4
```

然后打开你的 /admin 这个路径，Tada

为什么是Engines 目录下的?

我们用组件是否为一个Rails 实体来决定它所应该在的组件目录：

/engines 目录

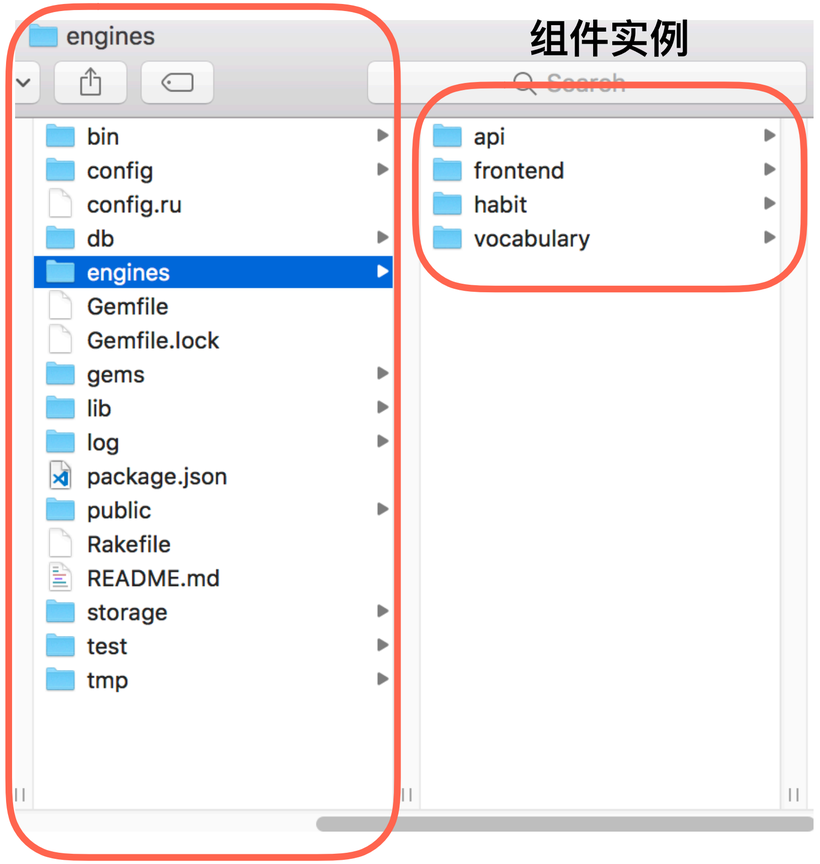
需要作为一个Rails 实体服务存在的组件。
或者可以直接理解为，这是一个可直接拆分为微服务的。

/gems 目录

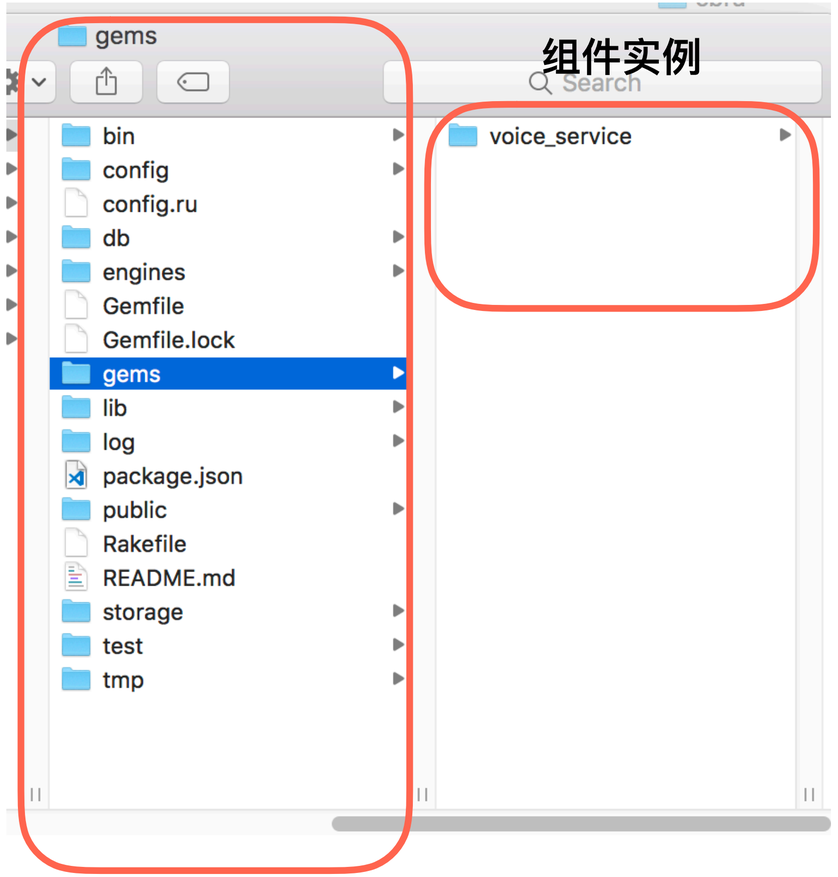
「无状态」，不依赖Rails 的，或者仅负责数据的流动，或者一些特定的计算。

所以项目会长这样

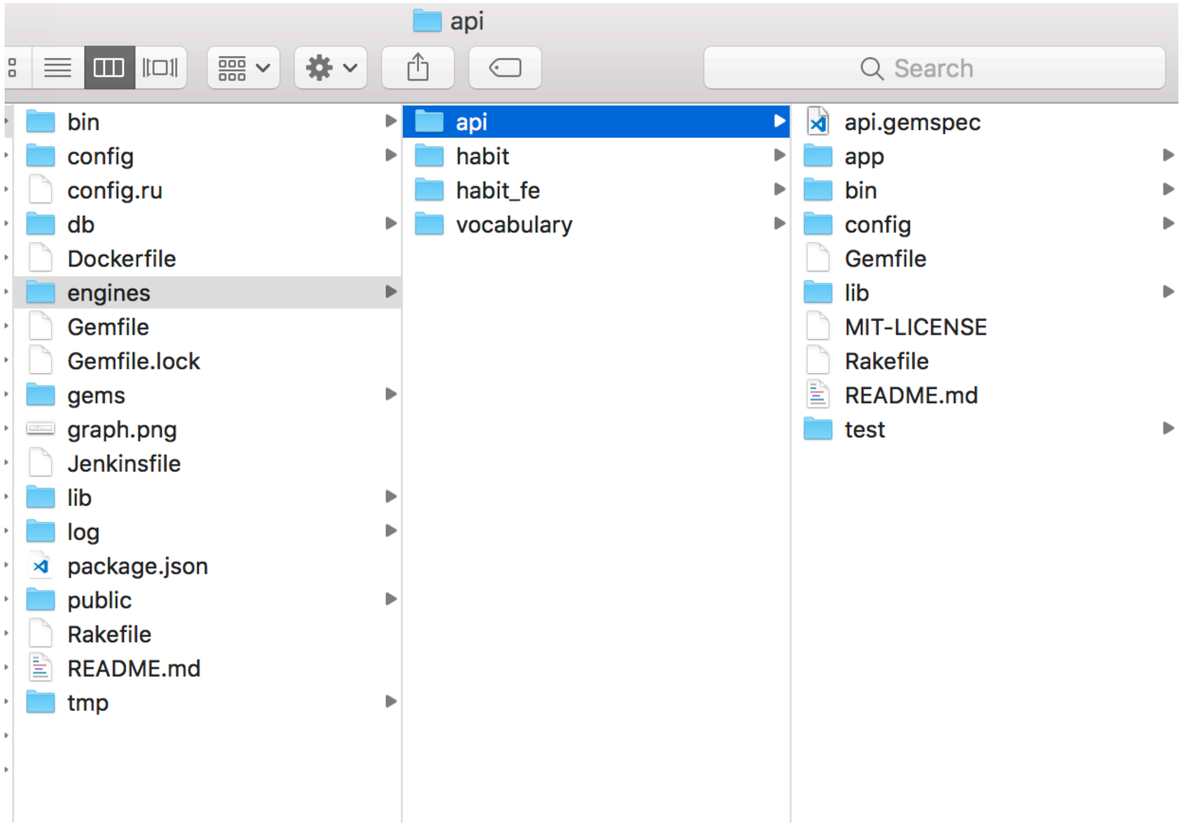
根目录



根目录



所以项目会长这样

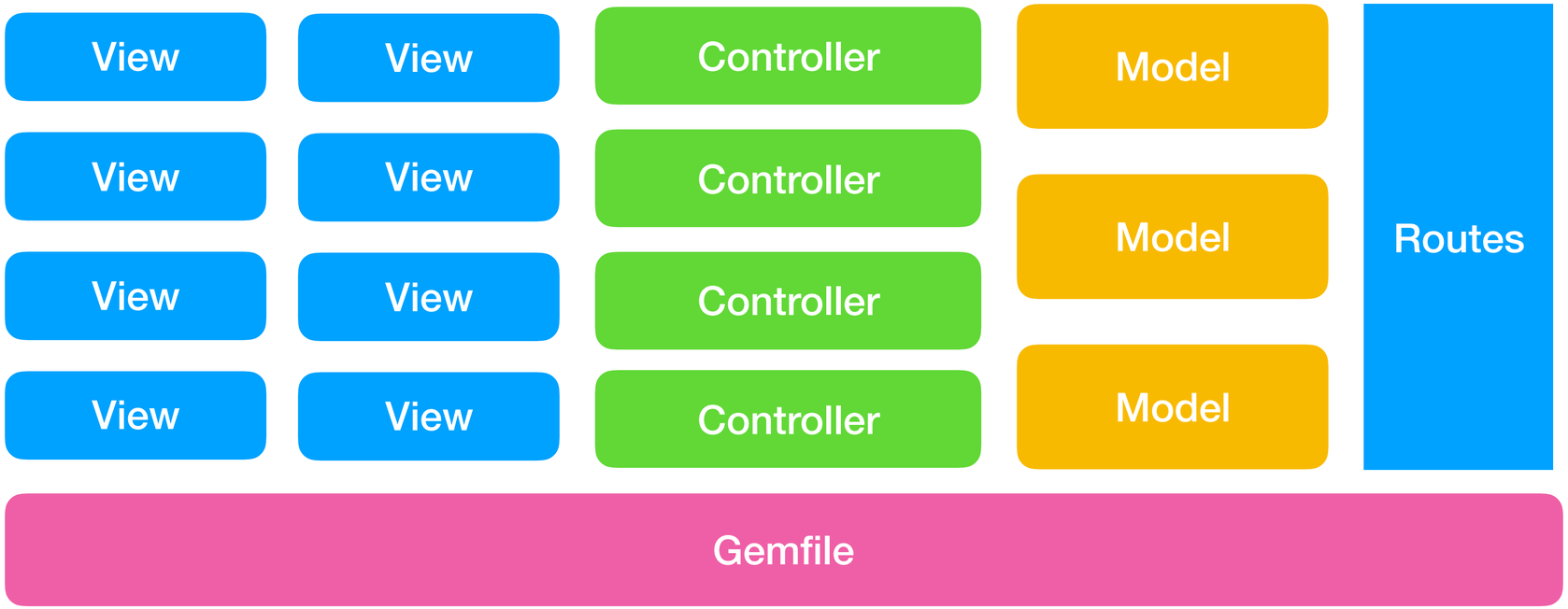

api
Search
bin
config
config.ru
db
Dockerfile
engines
Gemfile
Gemfile.lock
gems
graph.png
Jenkinsfile
lib
log
package.json
public
Rakefile
README.md
tmp
api
habit
habit_fe
vocabulary
api.gemspec
app
bin
config
Gemfile
lib
MIT-LICENSE
Rakefile
README.md
test

前置内容完毕，我们来尝试思考如何拆分已有的巨大单体

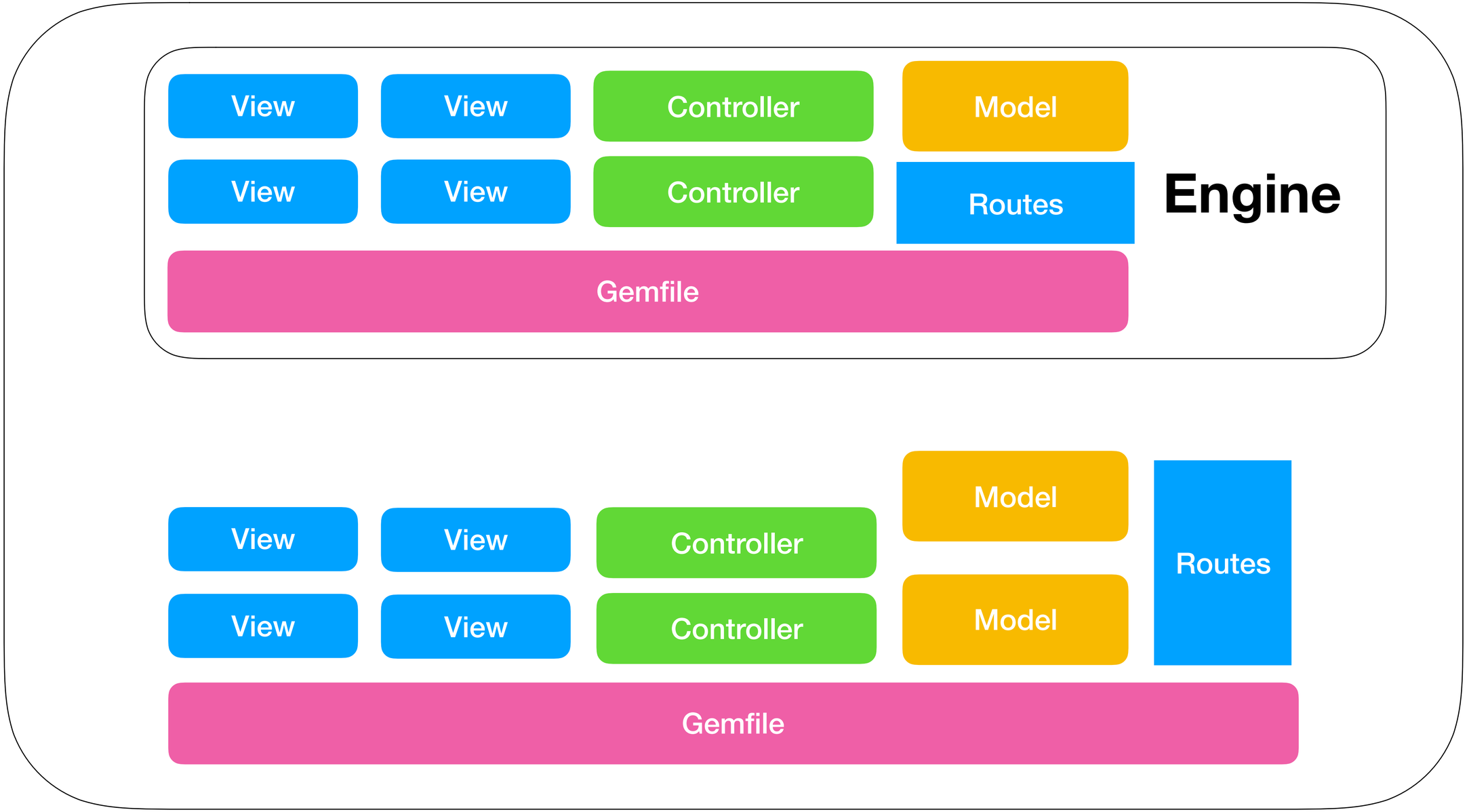
符合以下规则，我们拆成新的模块

1. 项目中明显内聚的领域，拆分成模块。
2. 粒度较大的服务，尝试拆分。
3. 用业务域将业务逻辑分解为多个有界上下文，每个上下文是一个模块

单体应用

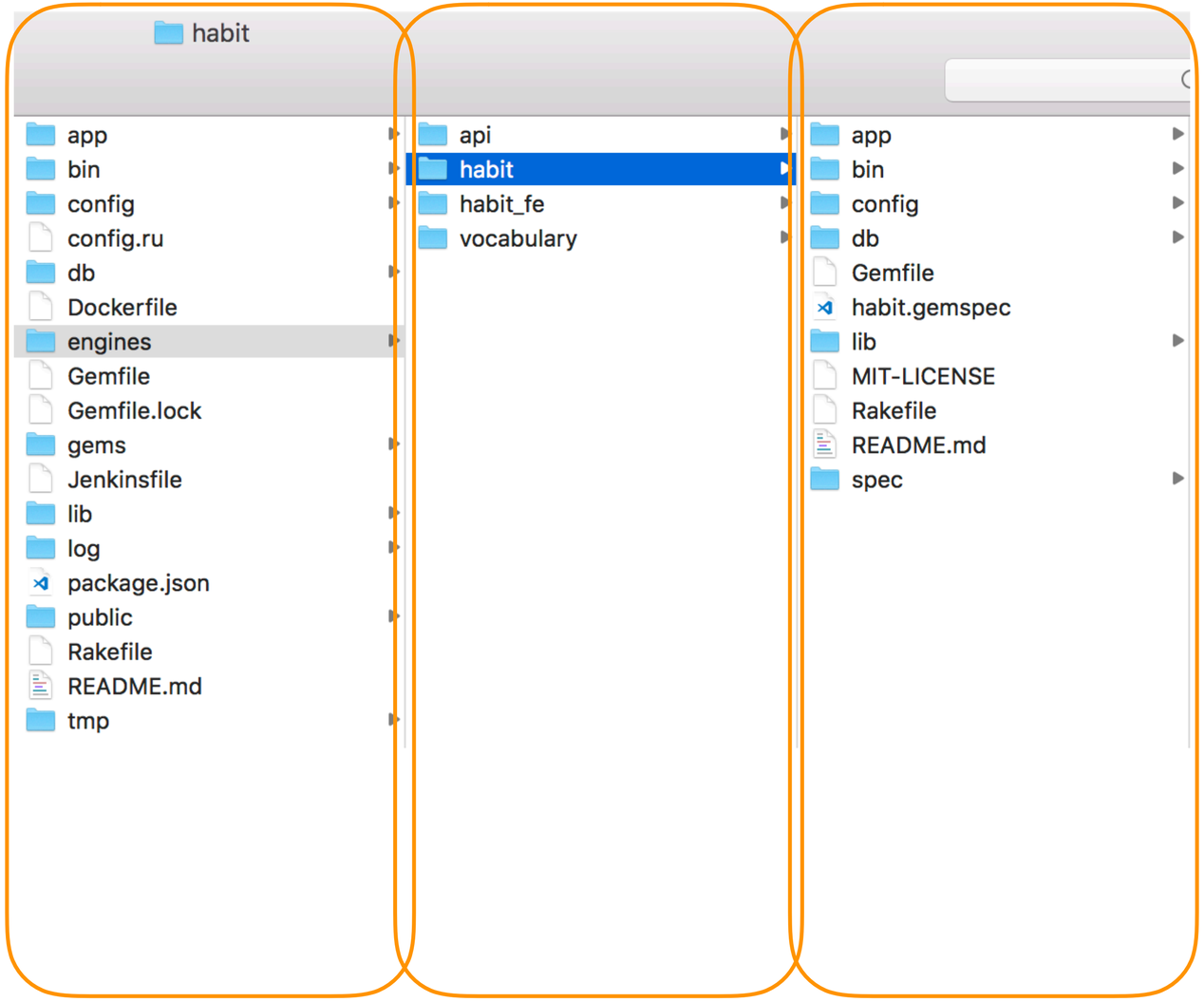

View
View
Controller
Model
View
View
Controller
Model
Routes
View
View
Controller
View
View
Controller
Model
Gemfile

单体应用

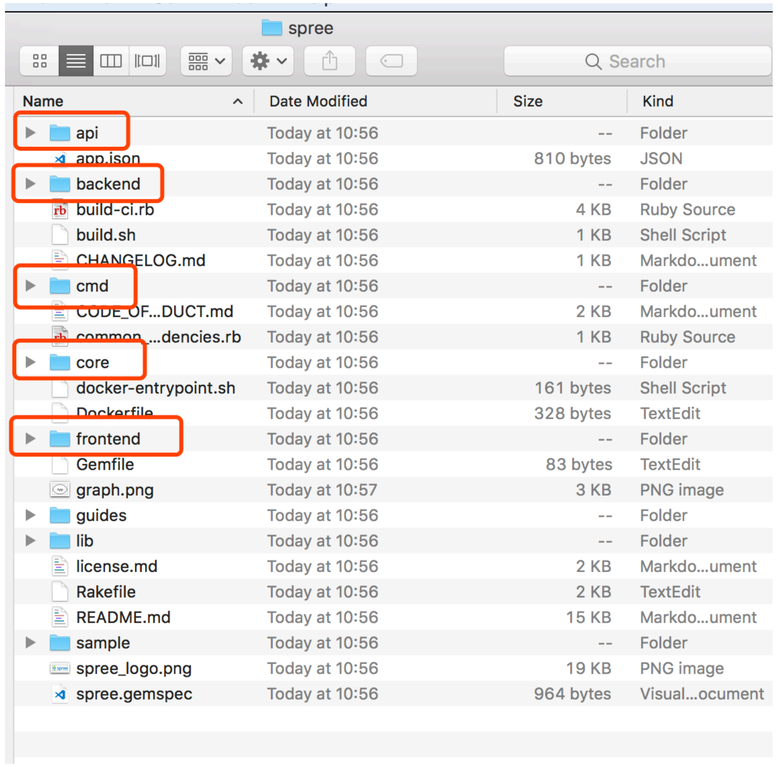

View
View
Controller
Model
View
View
Controller
Routes
Engine
Gemfile
View
View
Controller
Model
Routes
View
View
Controller
Model
Gemfile

Engines

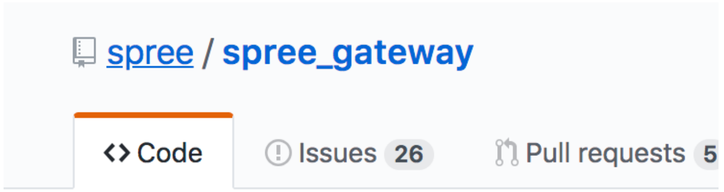
主项目 | Engines | Rails Engine



用 Spree 举例




```
s.add_dependency 'spree_core', s.version
s.add_dependency 'spree_api', s.version
s.add_dependency 'spree_backend', s.version
s.add_dependency 'spree_frontend', s.version
s.add_dependency 'spree_sample', s.version
s.add_dependency 'spree_cmd', s.version
```

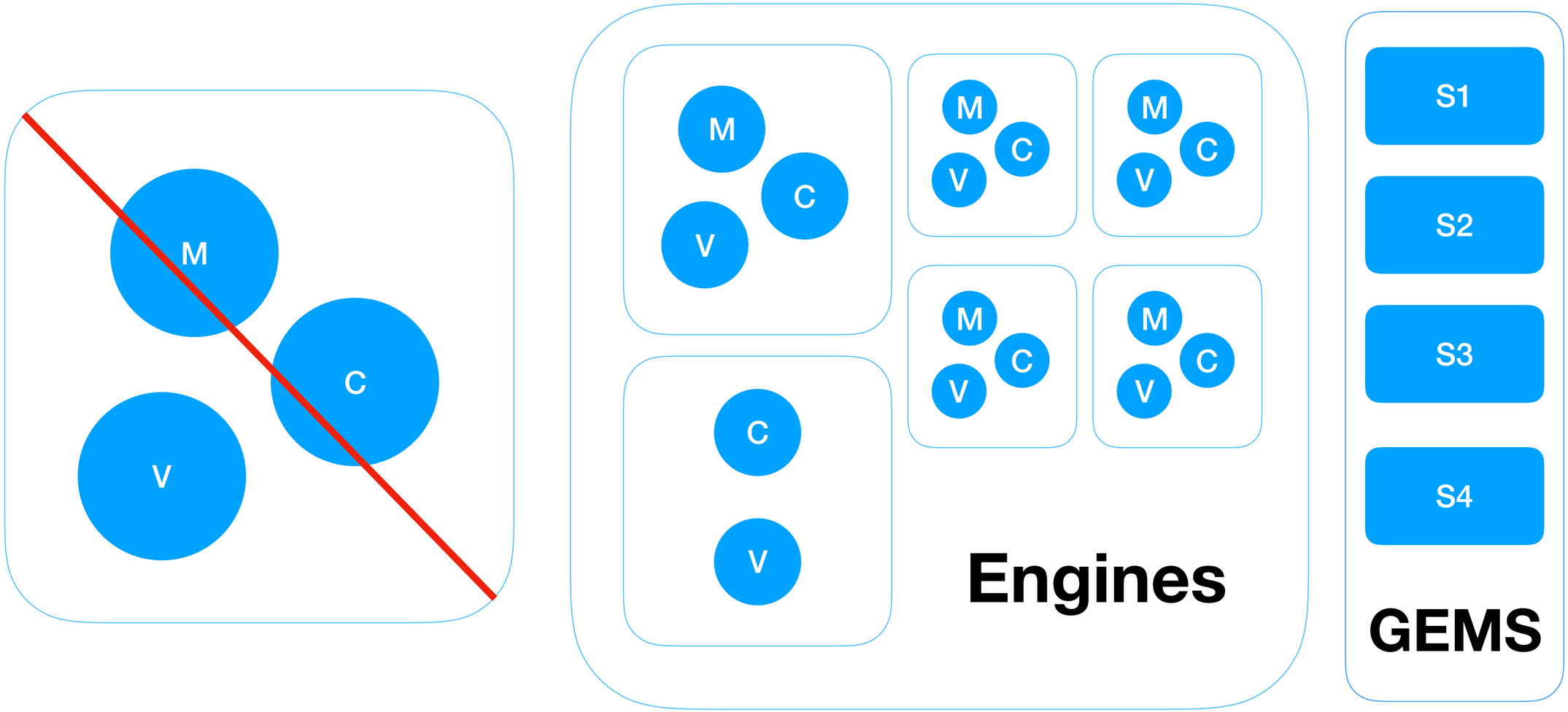

spree / spree_gateway
Code
Issues 26
Pull requests 5


spree-contrib / spree_address_book
Code
Issues 12
Pull requests 8


spree-contrib / spree_social
Code
Issues 18
Pull requests


spree-contrib / spree_wishlist
Code
Issues 7
Pull requests 12


spree-contrib / spree_sitemap
Code
Issues 1
Pull requests 1


M
C
V
M
C
V
C
V
M
C
V
M
C
V
M
C
V
Engines
S1
S2
S3
S4
GEMS

系统过于庞大是否会带来持续集成的负担?

事实上只要保证代码分割是正确的，
那就只需要跑对应component 下的CI 即可。

真的有这样的做的公司/项目吗?

ROOT

https://www.joinroot.com/

65,000多行Ruby / Rails应用程序代码
135,000多行测试代码
37 个 Engine
22 个Gem
30+ 工程师

https://powerhrg.com

491910 行代码

超过30个 Component

https://github.com/spree/spree

优点

更好的代码组织

模块内的代码需要的
上下文信息更少

随时可迁移成微服务

可以选择性的跑集成内容

劣势

同步/学习成本

测试复杂度增加

数据库迁移(Migration)依旧会很慢

考虑是否模块化也需要思考成本

部署效率低下的问题仍未消失

适合的场景

1. 体量较大臃肿不堪的单体应用
2. CI 时间过长的单体应用
3. 频繁出现合并冲突的单体项目
4. 符合合并规则的微服务

不建议的场景

1. 初期项目
2. 过小的团队
3. 已经有良好的代码组织的项目

如果你的项目连模块化架构都无法达成，
为什么你会觉得微服务能解决你的困难呢?

软件工程没有银弹

FAQ