shaojunda-blog

css-sprites

Image Replacement
add descriptive text to image-replaced elements, user text-indent hides the placeholder text:
Seven's bed

```
.logo {
  background: url(logo.png);
  display: block;
  height: 100px;
  width: 200px;
  text-indent: -9999px;
}
```

sprites

```
.logo {
  background: url(logo.png);
  display: block;
  height: 100px;
  width: 200px;
  text-indent: -9999px;
}

.logo:hover, .logo:focus {
  background-position: 0 -100px;
}
```

实现只加载一张图片，通过坐标来显示不同的图片。

css-imageuse

图片处理

```
<ul>
  <li class="crop">
    <img src="black.jpg" alt="black pic"/>
  </li>
  <li class="crop">
    <img src="red.jpg" alt="red pic"/>
  </li>
  <li class="crop">
    <img src="blue.jpg" alt="blue pic"/>
  </li>
</ul>
```

让图片不会被压缩，还可以统一图片的高度

```
.crop {
   height: 300px;
   width: 400px;
   overflow: hidden;
}

.crop img {
   height: 300px;
   width: auto;
}
```

更好的办法

- 将图片调整到高度小于图片本身的高度，宽度等于图片的宽度。
- 在服务端进行 resize。
- 上传图片是对图片进行剪切。

css-safety

1. 为 layout 中每个部分分别设置间距。
2. margin collapse，margin 会彼此抵消，取最大值。当进行了以下设置的元素不会发生 collapsing margin：
   . padding or border。
   . relative or absolute positioning。
   . A float left or right。
3. 使用 reset.css 与 normalize.css。

css-position

position 属性

1. static，为元素的默认值，没有特殊的定位，按照页面的正常 flow 显示。
2. relative，设置了这个属性之后该元素的位置就会相对于它之前的位置进行偏移。
3. fixed，相对于可是区域进行偏移。
4. absolute，相对于最近的已经定位了的元素进行偏移，如果附近没有已经定位的元素，那就相对于 body 进行偏移。

z-index

对于已定位的元素可以使用 z-index 属性来调整堆叠的顺序，值大的在上面，值相等的话后面的元素会覆盖前面的元素。

css-grooming

css 代码整理

1. DRY(Don't Repeat Yourself)，核心思想是不断向上抽象，将重复定义的属性定义在上一层元素中或定义在统一的 class 中，如果在定义了统一的 class 之后仍有定制化的需求就定一个 sub_class。

```
<input type='submit' class='button submit'/>
<a herf='next_page.html' class='button'>Next</a>
```

1. Style shorthand

```
.ski_poles {
   margin: 15px 10px 0 20px;
}
```

` ## Display Types
.div {
display: none / block / inline / inline-block
}

1. block，宽度与装它的容器相等，前后有短行，可以设置所有的盒模型属性。块级元素有 : , <p>, <ul>, <ol>, <li>, <h1>…<h6>。
2. inline，一般写在块级元素中，占用的空间与其内容所占用空间相等，不会在前后产生空行。
3. inline-block，与 inline 元素的 flow 相同，但是它有自己的宽高。

Centering

1. 真对块级元素
   . 定义一个小于父级元素的宽度。
   . 设置 margin: 0 auto;。
2. 真对 inline 和 inline-block

设置 text-align: center。

全栈之旅，一个逗号

8 周的全栈训练，转眼间已经进行了 7 周，从不懂 Rails 到有了成功上线的 作品；从做项目不知道如何下手到带领团队稳步推进；从不知道如何为网站带来增长到理解了 Growth Hack 的基本规则。这些改变都在过去的七周相继发生。

技术成长

技术上的成长主要得益于 放下你的無效學習方式 和 領悟「學習的黃金通道」 这两篇文章以及“拼图理论”，看过 放下你的無效學習方式 之后让我放下了心中的傲慢，决定按照 Xdite 所传授的方式进行学习，这是我这几个月技术进步的地基。領悟「學習的黃金通道」 让我意识到了想要学好一项技能的几个要点：

1. 模仿，找到可以模仿的“教材”，先跟着教材做 三遍，一开始不懂没关系，先知道“套路”；
2. 刻意练习，不断打磨旧“套路”，并学习新套路；
3. 成就感，争取在学习的过程中不断产生成就感，成就感可谓是进步的润滑剂，它可以助人不断向前。回想起大学时期没有通过日语 2 级考试，很大程度上是因为那段学习过程很枯燥，不会为自己设置小奖赏，也没能持续产生成就感。而全栈班的学习，有奖赏机制，比如有一套积分规则，通过抢先提问、完成某个功能、分享好工具或撰写好文章等方式来获得一定的积分，这些机制可以让我们在不断向前的同时获得相应的奖赏，然后逐渐形成正反馈。团队中的每日效果展示、每周的 Demo 演示都会让我们产生不同程度的成就感，这些都十分有利于个人的成长。

项目管理

项目管理能力上的成长主要得益于使用了 Xdite 所教授的四种工具：

1. 功能拆解，通过 User Story 完成功能拆解与优先级划分。
2. 每日晨会，通过每日晨会先了解每位组员的任务进展情况与遇到的问题，并分配下一阶段的任务。
3. Deadline，为每项任务设置 deadline，人为压缩时间对按时完成任务有极大的帮助。
4. 日常展示，每天都要做出新的可以展示的效果。这会潜移默化地推动每位成员自主向前进发。
5. 及时反馈，项目中遇到十分棘手的问题，不要自己闷头想太久，如果不能在一个小时内找到解决方案就要告知我，我会协助解决。

6. 分享总结，每周五下午开分享会，组员间彼此交流本周遇到的“大坑”与学到的好概念、好工具。这样做不仅有助于增进组员间的感情还会加速彼此的成长。

遇见“战友”

通过后三周的团队项目，我有幸结实了四位新战友，我们一起创建并上线了一个为内容创作者打造的 资金众筹平台， 他们分别是自信、高调的 free 哥，开朗、强悍的 rebecca，踏实、稳定的 liber 以及冷静、博学的李教授。项目能够顺利推进除了使用正确的方法之外更重要的因素是团队成员的状态是否积极与价值观是否一致。在团队配合的过程中每个人都要以项目为重，放弃部分自身的“利益”才能确保项目能够顺利推进。恰好我的四位战友都是非常积极主动的人，在需要美化前端页面的时候 free 哥和 rebecca 挺身而出，制作出了十分精美的前端页面，使我们网站的美感得到了大幅提升。在网站初期，踏实稳定的 liber 经过不断的修正很好地完成了网站的核心功能。在完善产品阶段，李教授更是为团队解决了一项疑难杂症，提升了网站的统一性与易用性。他们四位都十分优秀，积极主动解决困难，对项目十分负责。可以说，他们是项目能够顺利推进更为重要的原因。

总结

学习一项技能首先要放下心中的傲慢，否则将会如盲人摸象般不得其法，它更像是一场“拼图游戏”，拼拼图最佳的方式是先从边缘开始拼，逐步拼出图片的全貌。类似的，学习全栈也应该先了解其框架，然后再按照拼图的方式不断补齐欠缺的知识，Xdite 通过这次训练在最短的时间里为我们搭建好了基础的框架、传授了“拼拼图”的方法、并示范性地帮助我们拼好了部分拼图，接下来的修行了就要靠个人的努力啦。

Xdite 的全栈班更像是一个成长加速系统，在一开始就逐步将我们放置于真正的开发环境之中，通过不同的课程完成不同的目标，将我们引向不同的境界。这个过程中她不断帮助我们消除疑虑、降低我们的挫折感，激励我们不断向前的同时，也会及时纠正我们的错误。她仿佛在我们身旁安装了“监控器”，可以实时掌握我们的动态，做出相应的调整，根据不同的项目有针对性地传授不同的技能。这些都是我在回顾两个月的学习生活时才发现的，平日里你会觉得她的存在感似乎不是很高（除了上课的时间）但其实她是系统背后的管理员，不断对来访“用户”的行为进行分析并有针对性地做出反馈，遇到一位能够将你置身于一套完整成长系统的老师真是一件幸事，这个过程可谓是润物细无声，潜移默化地影响着我。

回过头看，全栈训练让我的“操作系统”进行了一次全新的升级。全栈并不是终点，它是一个思维模式，一个新的起点。

细碎知识点

细碎知识点

后台获得当前的 csrfToken 'csrfToken' => form_authenticity_token

前台获得当前的 csrfToken var authenticityToken = $('meta[name=csrf-token]').attr("content");

session 中的 csrfToken token = session[:_csrf_token]

修改项目的时候默认使用的是 patch 方法，使用$("input[name='_method']").val("post"); 可以改为 post

使用 seed rake db:seed

编译资源文件 assets:precompile

使用 heroku 命令行 heroku run bash

验证 boolean 类型的字段的值是否存在不能使用 validates :price, presence: true 要使用 validates_inclusion_of :need_add, in: [true, false]

使用 bullet 检测 N + 1 query 的步骤，安装 gem gem "bullet"，在 development.rb 中 配置 bullet

```
config.after_initialize do
  Bullet.enable = true
```

```
  Bullet.alert = true
  Bullet.bullet_logger = true
  Bullet.console = true
  # Bullet.growl = true

  # Bullet.xmpp = { :account  => 'bullets_account@jabber.org',

                    # :password => 'bullets_password_for_jabber',

                    # :receiver => 'your_account@jabber.org',

                    # :show_online_status => true }

  # Bullet.rails_logger = true

  # Bullet.honeybadger = true

  # Bullet.bugsnag = true

  # Bullet.airbrake = true

  # Bullet.rollbar = true

  Bullet.add_footer = true
  # Bullet.stacktrace_includes = [ 'your_gem', 'your_middleware' ]

  # Bullet.stacktrace_excludes = [ 'their_gem', 'their_middleware' ]

  # Bullet.slack = { webhook_url: 'http://some.slack.url', channel: '#default', username: 'notifier' }

end
```

然后逐个页面打开即可，使用以上配置当遇到 N + 1 query 时页面会有弹窗。

[weekly-07]

本周回顾

这周项目进入了“收尾”阶段，在本周我们主要进行了以下更新：

1. 优化页面 js 文件加载方式（放在页尾加载）
2. 去掉程序中的 N + 1 Query。
3. 重新设计 landing page。
4. 重新设计 how-it-works。
5. 基础 SEO 优化（动态添加页面 title 和 description）。
6. 使用 cancan 实现权限控制。
7. 修复 bug...

昨天大家走的都很晚，在尽力做最后的优化，李教授 12 点才回家，有一帮这样的战友，真是幸福。

今天我们意外的收到了 xdite 老师的 🎁，表示对我们这段时间付出的肯定。这种大家没有为了得奖，都是为了项目在努力，最后得奖的这种惊喜真是振奋人心。

使用 seo_helper

使用 seo_helper gem 来动态改变页面 title 和 description

临近项目上线，我们开始进行基础的 SEO 优化，每个页面手动改 title 和 description 会很烦，xdite 推荐了 seo_helper 来实现动态修改页面 title 和 description 很好用，但是在使用的时候遇到了一个坑。

先说一下使用步骤：

1. 安装 seo_helper gem。
2. 创建 config/initializers/seo_helper.rb 文件，进行配置：

```
SeoHelper.configure do |config|
  config.skip_blank = false
  config.site_name = "火箭项目"
  config.default_page_description = "下一个火箭项目的助跑平台"
  # config.default_page_image = "#{Setting.domain}#{Setting.default_og_url}"

end
```

然后在 action 中分别使用 set_page_title title 和 set_page_description 来配置页面 title 和 description

最后在 layout 中添加

```
<%= render_page_title_tag %>
<%= render_page_description_meta_tag %>
```

按理说这么做应该可以正常使用了，但每当程序执行到 set_page_title 时都提示找不到这个方法，这是什么问题呢？百思不得其解，后来我比较一下 seo_helper 的源代码发现我所安装的这个版本，不支持 rails 5。我去... 解决办法是，将 Gemfile.lock 文件中的 seo_helper (1.0.2) 改为 seo_helper (1.0.3)。重启服务器，一切又正常了。

提升网站访问速度

使用 bullet 检测 N + 1 query

使用 bullet 检测 N + 1 query 的步骤：

1. 安装 gem gem "bullet"。
2. 在 development.rb 中 配置 bullet

```
config.after_initialize do
  Bullet.enable = true
  Bullet.alert = true
  Bullet.bullet_logger = true
  Bullet.console = true
  # Bullet.growl = true

  # Bullet.xmpp = { :account  => 'bullets_account@jabber.org',

                    # :password => 'bullets_password_for_jabber',
```

```
                    # :receiver => 'your_account@jabber.org',

                    # :show_online_status => true }

  # Bullet.rails_logger = true

  # Bullet.honeybadger = true

  # Bullet.bugsnag = true

  # Bullet.airbrake = true

  # Bullet.rollbar = true

  Bullet.add_footer = true
  # Bullet.stacktrace_includes = [ 'your_gem', 'your_middleware' ]

  # Bullet.stacktrace_excludes = [ 'their_gem', 'their_middleware' ]

  # Bullet.slack = { webhook_url: 'http://some.slack.url', channel: '#default', username: 'notifier' }

end
```

然后逐个页面打开即可，使用以上配置当遇到 N + 1 query 时页面会有弹窗，随后逐条调整即可，这样会减少网站访问数据库的次数。

“提高”页面加载速度

使用 <%= yield :orders_javascript %> 配合

```
<% content_for :orders_javascript do %>
  <script>
    ...
  </script>

<% end %>
```

来实现最后加载 js 文件。

使用阿里云邮件服务

阿里云邮件服务配置

1. 在 heroku 上添加域名，执行 heroku domains:add www.xxxx.xx。
2. 在阿里云上配置发信域名

   (截图：https://goo.gl/photos/GoCvwxS4KcoAmEHM6)
3. 域名验证

   (截图：https://goo.gl/photos/LgRUEhzrcCkNVctw8)

4. 在 production.rb 中配置邮件发送服务参数。
5. 在 ApplicationMailer 中配置 from 邮箱地址 。
6. 传递参数到 heroku figaro heroku:set -e production

项目中需要配置的信息如下：

```
Rails.application.routes.default_url_options[:host] = 'talent-rocket.herokuapp.com'
config.action_mailer.delivery_method = :smtp
config.action_mailer.smtp_settings = {
  address:                ENV["EMAIL_ADDRESS"],
  port:                   ENV["EMAIL_PORT"],
  user_name:               ENV["EMAIL_USER_NAME"],
  password:                ENV["EMAIL_PASSWORD"],
  openssl_verify_mode: 'none',
  enable_starttls_auto: false   }

config.action_mailer.default_options = {
  reply_to: "shaojunda@XXX.com"
}
```

[发布项目到 Heroku...]

发布项目到 Heroku

昨晚将项目发布到了 Heroku 上，打开首页以为万事大吉，就去睡了，结果今早小组成员使用时发现了很多问题，一些之前好用的功能完全不能用了，后来发现是 js 加载出了问题，总结如下：

1. 之前在页面通过
2. scss 中的使用的图片文件也都无法加载（404）。

解决 javascript 文件 无法加载的问题

我之前是在页面上直接通过 <script> 标签引用的 js 文件，没有将需要引用的代码写进 application.js（没通过 require 引用）造成发布到 heroku 上之后无法加载。原因是在开发环境下 heroku 去 public/assets/ 下寻找需要引用的文件，但是我没用 require 所以我需要的 js 文件没有被放进 public/assets 里。加入 require 之后问题修复。

但是我的 js 文件不需要全局使用，所以我创建了一个新的 manifest 文件，并在其中引用特定的 js 文件。然后在 assets.rb 中添加配置信息 Rails.application.config.assets.precompile += %w(account/project.js)

解决 css 中图片无法加载问题

1. 在需要使用图片的地方使用 <%= asset_path("bg2.jpg") %> 的形式加载。
2. 在 production.rb 中配置 config.assets.css_compressor = :sass（不确定是否是必须）

通过个问题让我意识到在 rails 中开发环境和生成环境相差很大，而且都是通过配置文件来进行不同的配置，虽然暂时解决来问题，但坑还很深，需要继续挖。

[js 写页面的纠结]

用 jquery 写页面

这周我个人重要的任务就是优化项目发布流程，将项目与回报的发布整合在一起，这样可以让用户有一个比较流畅的体验。为了实现这个功能我想到了使用 jquery 来实时生成页面元素，

由此我把自己带入来一个大坑。使用 js 来生成页面会有两个主要的问题：

1. 逻辑会乱掉。
2. 不易修改页面元素的样式。

具体纠结的程度要视情况而定。
真对逻辑混乱我的解决办法

1. 首先想好要实现哪些功能。
2. 将功能写成独立的方法。

对于样式问题，目前还没有很好的解决方案，修改起来还是会很麻烦。

weekly06

本周回顾
这周项目进入了“装潢”阶段，在本周我们主要进行了以下更新：

1. 添加用户指南，来介绍我们的运作原理，以及常见问题。
2. 添加关于我们，来介绍网站本身，增强可信度。
3. 添加在线客服，用来实时回复客户疑问。
4. 整合用户后台的操作按钮，将按钮控制在 3 个以内。
5. 使用 helper 与 partial，优化 view 页面。
6. 重新设计 landing page。
7. 重新设计项目展示页。
8. 重新设计项目列表页。
9. 优化发布流程，将发布项目与发布回报整合在一起。
10. 统一网站关键词语的使用。

这周团队成员相继显露出疲态，这也是常态，经过调整状态会逐渐恢复过来。

关于熬夜

这周遇到的最大坑就是连续熬了两天夜，好处是让功能成功上线，坏处是导致早上根本起不来，而且眼睛一直不是很舒服。这件事给我的经验是，其实每个人每天的有效工作时间是基本固定的，用在了晚上，白天是要睡回来的。

基础功能身后的用户体验

这周我们完成了 talent 项目的基础功能，然后我们找来种子用户来尝试在我们网站上发布项目，用户的反馈是基本完全不知道网站该如何使用，这让我们意识到我们的网站遇到了新的问题，即用户体验的问题。
通过分析我们了解到我们存在以下不足：

1. 缺少 onboarding。
2. 缺少 landing page。
3. 使用流程不流畅。
4. 缺少描述文案。
5. 角色、按钮等文案不规范。

接下来我们要将重点集中放在改善用户体验上，准备改善的条目有：

1. 添加 landing page。
2. 重构新建项目和方案的流程，增加用户使用指引。
3. 体验竞品的使用流程，来统一网站名词、方案的使用。

weekly-05

本周回顾
这周我们完成了网站基础功能的搭建，新增功能有：

1. 全新的项目展示页的布局。
2. 新增项目审核。
3. 新增用户手机验证。

4. admin 后台新增资金流水 log。
5. admin 后台拨款功能。
6. 普通用户可以支持特定项目。
7. 优化项目描述的输入域高度。
8. 优化项目发布流程。
9. 项目显示页可以播放视频。
10. 优化了部分使用体验。

目前存在的主要不足是：

1. 却少 onboarding。
2. 网站没有统一的风格。
3. 支付流程需要优化。

full-stack-note-day15

今日回顾

1. git branch |grep 'branchName' |xargs git branch -D 批量删除本地分支
2. 删除所有 DS_Store
3. rails s -b xxx.xxx.xxx 绑定 IP 地址 实现通过 ip 访问项目。
4. 处理异常的方式

```
begin
    @plan = self.plans.create!(title: "自定义金额", description: "不谢，就是想支持你。", price: 1, plan_goal: 999_999, plan_type: 0)
rescue => e
    logger.error e.message
    logger.error e.backtrace.join("\n")
end
```

1. git reflog show --date=iso <branch name>

今日达成

1. 修复了投资后项目人数不变的 bug 。
2. 修改注册流程，在用户中心增加手机认证。
3. 实现搜索项目功能。
4. 实现自定义资助金额功能。
5. 修改 description 域的显示高度。
6. 更新 seed 至最新状态。
7. 修复 geetest key。
8. 实现 twilio 发送手机验证码。

今日情绪

平静

今日领悟

每天工作的时间其实基本是恒定的，晚上熬夜了，早上就没什么精神。

今日结语

继续打磨产品。

今日不足

没产生大对突破。

学习的另一种方式

团队合作

有人会认为凡事只有亲力亲为才学得会，但似乎并不是这样。

这一点是在上周的团队合作项目中体会到的。每个人的兴趣点不同，看问题的角度不同，分享的知识不同，使得我可以接触到之前没掌握的知识。

由于每个人负责的模块不同，虽然我不能亲自实现某个模块的某个功能，但代码在团队内是公开的，我们可以互相学习其他队员写的好的地方， xdite 老师的代码批改也会让我们学习到某些编码方式的最佳实践。所以团队合作也是一种“学习方式”。

客观看待批评

有人批评是一件很幸福的事，有时批评中会隐含价值点，要调整心态客观看待批评，如果产品真的存在缺陷就要用于承认并迅速改正，这个过程本身也是学习。

[weekly-04]

本周回顾

在周一我学习了 landing page 的制作规则：一般分为 4 个部分，

1. 一句话形容自己（给用户的价值）。
2. 使用此服务的好处。
3. 运作原理。
4. 使用者见证或媒体报导。
5. CALL TO ACTION。

其中前三个部分的内容分别来自于问卷调查中的：

1. 当你用来、做了、学会了 XXX，完成了什么事后，你最想达成什么目标？
2. 当你学会、买到、知道 XXX 后，你最想要做什么事？
3. 当你在做 XXX 时，你最讨厌的事情是什么？

随后我开始做“人才火箭”计划，经过重新分组我认识了来自其他小组的 3 位新成员，他们都各具特点。free 强迫症患者，写 code 排版十分用心，乐于分享。 艳增兄，一名合格的老司机，善于发现各路消息。Liber 虚心好学，做事踏实。

rebecca 一位走在证实自己具有超强逻辑能力的前产品汪，善于捕获需求。大家都善于合作，以推进项目为首要目标，因此上周项目进展的非常顺利，完成了所有 must have 要做的事情。

主要实现了：

1. 用户登录注册。
2. 简单的用户中心，可以修改昵称。
3. 创建项目（普通用户、admin）。
4. 创建计划。
5. 支持项目（下单并支付）。
6. 用户管理（升降级）。
7. 项目管理（前、后台）。
8. 计划管理（前、后台）。
9. 订单管理（钱、后台）。
10. 优化页面布局。

现在的状态还不具备上线的能力，下周会增加支付系统以及对上述功能的进一步打磨。

[很重要]

不要在项目上线前一晚加功能

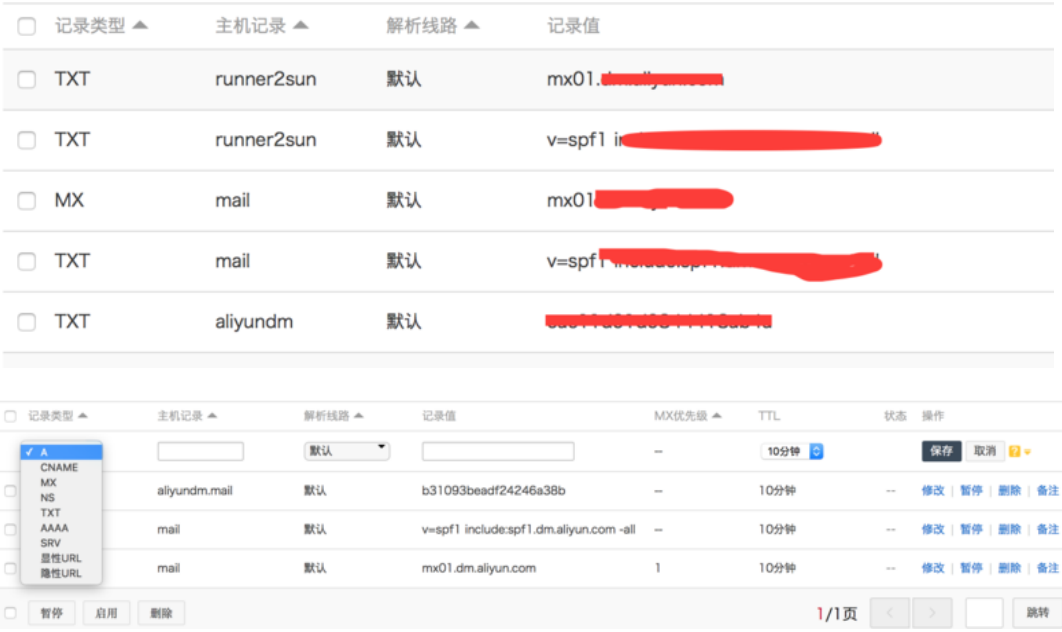
不要在项目上线前一晚加功能
不要在项目上线前一晚加功能
重要的事情说三遍！
full-stack-note-day14
今日回顾
1. 今天学习通过 unbounce 制作 landingpage。
2. landingpage 分为四部分，第一部分是一句话描述；第二部分是运作原理；第三部分是三大好处；第四部分是使用者见证。
3. 进行了新项目的分组，做人才火箭计划。

今日达成
1. 名人购的 landingpage
2. 人才火箭的 must have、should have、could have、nice to have 的讨论。
3. 项目基础框架的搭建。

Read on
weekly-03
本周回顾
这周主要任务是实践微型商城，通过这个项目我增强了对 rails 基本架构对了解，在实践的过程中我使用 Tower 来进行项目管理，让项目一直有条不紊地向前进展，看着需求的逐步实现，技术细节的逐步上线，任务数量的不断增多内心会有些许成就感。通过这个项目也让我了解到了一些 gem 的使用：
1. bootstrap-sass 使用 bootstrap。
2. devise 实现用户登陆系统。
3. simple_form 简化表单。
4. annotate 在 Model 中显示出表结构。
5. carrierwave 实现文件上传。
6. file_validators 对上传文件进行验证。
7. mini_magick 实现图片压缩。
8. font-awesome-rails 提供一些常用对图标。
9. letter_opener 本地预览邮件。
10. figaro 配置信息管理。
11. aasm 实现有限状态机。
12. fog 对云服务提供支持。

本周遇到的问题就是还没想出一个好的产品创意。
GrowthHack？
这周接触到的最棒的概念莫过于 Growth Hack 了，随着 Growth Hack 的火爆，Growth Hacker 在互联网圈也备受追捧，但什么是 Growth Hack 呢？我非常赞同 xdite 老师给的一种定义：
把产品做好
有人会觉得这很搞笑，但参与过一次创业之后，我特别认同这个定义。因为，如果产品不好，其他的招数都只是花拳绣腿，即便能带来短期的流量“暴涨”，也只会如昙花一现般不可持续，真正重要的还是要开发出好的产品。按照 《Inspired》 中的说法，好的产品就是用户喜爱的产品，即：
有价值、可用的、可行的产品。
我个人认为，只有对用户喜爱的产品做 Growth Hack 才更有意思。至于 Growth Hack 的具体操作我会单独再写一篇文章。
full-stack-note-day13

这周遇到的最大的坑是使用邮件发送服务，起初想使用 mailgun 来当作邮件发送服务器，注册好帐号发现在 Heroku 上使用 mailgun 需要绑定新信用卡，帮好信用卡之后 mailgun 的帐号又变为不可用状态，然后和客服联络解决，终于帐号恢复正常，这时又发现，Heroku 没有 DNS 服务器，而 mailgun 需要做 DNS 验证，然后打算用阿里云的 DNS 服务器，但总是配置不成功，后来开通了阿里云的邮件推送服务，但又遇到了大坑，就是 DNS 验证，由于之前没配置过，不知道各个字段该填什么，不过现在知道了。

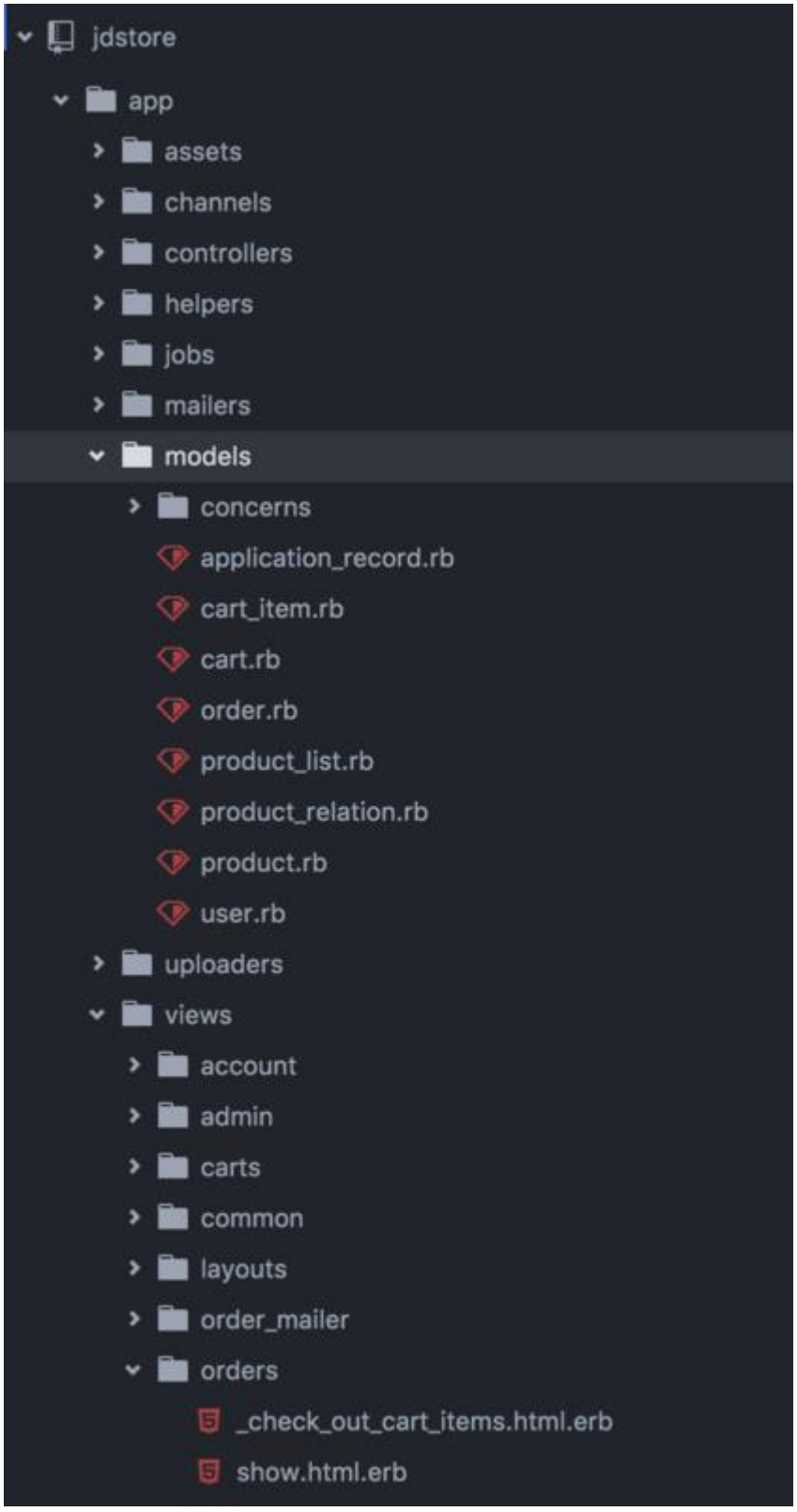


记录类型按照要求选取即可，这里用到了两种类型，TXT 与 MX。主机记录这里填二级域名（第一条记录填给定值），记录值填写给定值即可。

[full-stack-note-day12]

今日回顾

1. heroku console 可以开启 heroku 上项目的后台。
2. 使用 AWS 的用户策略来降低信息泄露风险。
3. Atom 的 fileicons package 可以在不同文件类型前显示出相应的图标，很炫酷。



[Read on ActiveRecordAssociations]

为什么使用 association

在 rails 中 association 表示两个 Active Record 间的关联关系。使用 association 可以简化一些常见操

作。

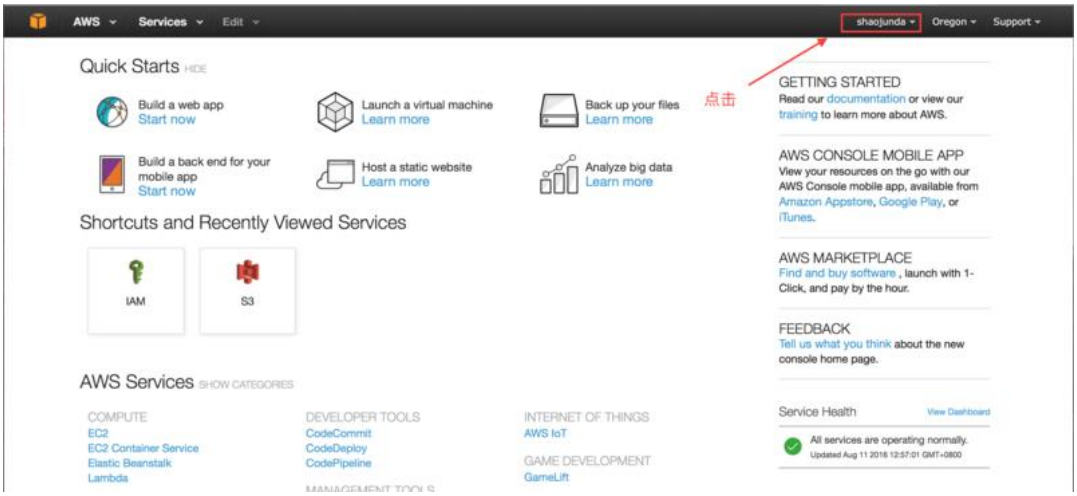
association 的 6 种类型

1. belongs_to
2. has_one
3. has_many
4. has_many :through
5. has_one :through
6. has_and_belongs_to_many

Read on

关于 AWS 密钥

想必大家都担心自己密钥泄露导致自己的信用卡被盗刷，有一个方法可以消除你这方面的疑虑，就是使用 AWS 的用户管理策略，步骤如下：



Read on

full-stack-note-day11

今日回顾

1. 使用 javascript:void(0) 设置为点击后不发送请求到后台。
2. 使用 mailgun 做邮件服务器。
3. 发布项目到 heroku。
4. 使用 figaro 管理配置信息。
5. 同步配置信息到 heroku。
6. heroku restart -a jdstore 重启 heroku 项目。
7. heroku create --a jdstore 创建特定名称的项目。
8. heroku config 查看配置参数。
9. 了解 delete_all 与 destroy_all 的区别。
10. rails 通过在不同环境下使用不同的配置文件来分离不同的动作。
11. 销售漏斗模型（AARRR）。
12. Growth Hack 的前提是要测量数据。
13. Growth Hack 的核心精神：增强信心、降低疑虑。
14. 着陆页的重要性。

今日达成

1. 一篇使用 figaro 管理配置信息。

2. 一篇 delete_all vs destroy_all

今日情绪

昨晚睡的比较晚，早上没什么精神，情绪不高，中午睡一觉之后，半血复活。

今日领悟

Growth Hack 并不是万灵丹，运营、增长的前提是产品本身要有价值。

今日结语

如何发现痛点、价值点？

今日不足

今天学习了 Growth Hack 的一些基本概念与工具，但理解的还不是很不够。

今日 BUG

push 项目到 heroku 报错 NameError: uninitialized constant evn 原因是没有上传 aws 的参数到 heroku。上传之后还是报错，后来发现是 ragion name 写错了。

ActiveRecord 的 delete_all VS destroy_all

ActiveRecord 中的 delete(id_or_array) 方法

使用 SQL DELETE 语句删除给定的一个或一组 id 的记录。相应的 ActiveRecord 对象不会实例化，所以也不会执行回调函数。

ActiveRecord 中的 destroy(id_or_array) 方法

根据给定的一个或一组 id 来销毁对象，在这个过程中，会首先生实例化相应的 ActiveRecord，所以该对象的所有回调函数和过滤器都会在对象被删除之前得到执行。

ActiveRecord 中的 delete_all(conditions = nil)

Read on

使用 figaro 管理配置信息

为什么要用 figaro 管理配置信息

当我们使用 githut 管理我们代码的时候，一些诸如 AccessKey 等敏感信息是不能够让其他人获得的，但代码中又要提供获取特定配置信息的方式，使用 figaro 管理配置信息非常简单，只需要安装好 gem 再写好配置文件即可。

figaro 管理配置信息的步骤

1. 安装 figaro gem。
    1. 在 Gemfile 中添加 gem 'figaro'。
    2. bundle install。
    3. figaro install。
2. 复制一份 application.yml 作为示例文件。 cp config/application.yml config/application.yml.example。
3. 修改配置文件加入配置信息
4. production:
5. AWS_ACCESS_KEY_ID: "xxxxxxxxxx"
6. AWS_SECRET_ACCESS_KEY: "xxxxxxxxxx"
7. AWS_BUCKET_NAME: "xxxxxxxxxx"
8. REGION: "xxxxxxxxxx"

Read on

full-stack-note-day10

今日回顾

1. 使用 mailer 发送邮件。

2. 使用 AASM 进行订单状态管理。
3. current_cart.cart_items.find(params[:id]) 确保只从当前用户的购物车里拿数据。
4. 不同事物的 CRUD 应该放在不同的 Controller 中。比如 Cart 和 cartItem。

今日达成

1. 假支付，并记录支付方式。
2. 特定操作之后给用户通知邮件，用户进行取消订单等操作后邮件通知管理员。
3. 取消订单。
4. 发货。
5. 申请取消订单。
6. 申请退货。

今日情绪

看到其他队伍的分数都在猛涨，略微着急，但大体还算平静。

今日领悟

选择相信就要坚信到底。

今日结语

七夕，大家一起开飞机。

今日不足

积累的欠缺知识点越来越多了。

今日 BUG

小组同学遇到的典型 bug 是在有据数据的情况下，新增 token 栏位导致旧数据的 token 为 nil，引发报错，清空掉 order 里的数据后恢复正常。

full-star-note-day09

今日回顾

1. nil?、empty? 和 blank? 的区别
   - nil? 检查当前变量是否引用了一个对象。
   - empty? 检查变量是否是 "" 或 []。
   - blank? 相当于 nil? or empty?
2. helper 与 model 的区别
   helper 用来执行装饰性操作，比如时间日期的显示、一些字符或者标签。

   model 用来处理业务逻辑。

Read on

weekly-02

本周回顾

Active Record 相关知识点。

使用 carrierwave 配合 mini_magick 进行图片的上传与压缩。

bootstrap 的 grid system 的基本用法。

controller 中 render 与 redirect_to 的区别。

render partial 在 view 中的使用。

Layout 的结构。

使用 Tower 进行项目管理。

做项目时要关注的点以及优先顺序。

xdite 管理项目的方式。

本周我们通过小组间的通力配合，成功实现了总分上的逆袭。由此更加印证了，团队中每位成员只要最好自己最擅长的事情，团队的合力就会很强。

拼写大坑

这周遇到的最大的坑就是在创建 controller 的时候将 users 打成了 uses，然后我在 routes.rb 中是按照 users 来配置的。导致我在 view 里写访问路径的时候各种出错。

我以为手动修改文件名就可以了，后来发现不行...正确的解决方案是先通过 rails d controller 卸载，然后重新创建。

拼写错误的背后是不够细心且太急躁，慢下来，细心点儿，这类错误是完全可以避免的。

关于拖延你应该知道的一点提示

我有一个叫小黄的朋友，中学时期，他总是在假期的最后一周才开始写作业。大学时期，他会在考试前一天才开始复习。工作后，他会在截稿日前几天才开始动笔。他觉得小时候是因为贪玩，才最后一刻才完成作业。长大后，认为自己是拖延症患者。他很幸运，总能在最后一刻把事情做完，但这也让他产生了一种错觉即，“特定”的事情只需要在最后时刻开始做，而且只有在最后一刻才能做好。

有一次和他聊天，聊到为什么非要在快到截稿日了才开始写作，听到这个问题他略显激动地说：“你要是也写东西就知道了，因为截稿日前几天写东西简直是如有神助，之前大脑会一片空白，什么也写不出来。”。可是为什么在接近最后时刻才开始动手会“如有神助”？

昨天 xdite 老师在分享她参加 Facebook Hackathon 时通过人为“压缩”时间来帮助她完成产品的开发并最终赢得全球第一名时说到：“资源的稀缺会逼迫人作出选择，选择做那些对现阶段来说 最重要 的事情。”。这让我联想到了之前困惑着我的那个问题：“为什么小黄在最后时刻才动手做事会如有神助？”。我想答案就是时间的稀缺迫使他进行权衡并调用了各种“资源”以完成那些 最重要 的事情。

Read on

full-stack-note-day08

今日回顾

做项目时要关注的点以及优先顺序:

1. 确定要做的产品最终的样子。
2. 盘点资源。
3. 确定风险。
4. 做 must have。
5. 做 should have。

学习了 xdite 管理项目的方式：

1. 先定义“赢”。
2. 再算手头上的资源。(time, people...)，例如：这个案子工期是三个月。保留最后完整一个月的时间做上线验收。
3. 剩下 8 周时间，会被分为 2 3 3 周。
4. 头两周做基础建设与抓出风险的时间，找出最有风险的部分，实现基础建设、顺流程、梳理优先权。
5. 接下来的三周，做主要功能以及工程的骨干。
6. 再下来的三周，做次要功能以及顺流程。

Read on

full-stack-note-day07

今日回顾

1. 使用人物地图和用户故事对小型商城进行了分析。
2. 初步尝试了通过 tower 进行项目管理。
3. 使用 carrierwave 上传图片，使用 minimagick 压缩图片。

1. 在 Gemfile 中加入 gem 'carrierwave' 和 gem 'mini_magick'。
2. 执行 bundle install。
3. 执行 rails g uploader image。
4. 在 model 中挂上 uploader。
5. class Product < ApplicationRecord
6. mount_uploader :image, ImageUploader
7. end
8. 设置不同的图片压缩尺寸
9. process resize_to_fit: [800, 800]
10.
11. version :thumb do
12. process resize_to_fit: [200, 200]
13. end
14.
15. version :medium do
16. process resize_to_fit: [400, 400]
17. end
18. 在 view 中通过 @product.image.url 、 @product.image.thumb 和 @product.image.medium 来分别引用原尺寸图片、缩略图以及中等尺寸图片。

Read on
full-stack-note-day06

今日回顾

1. rails 里的！表示执行可能会产生中断，可以用来做调试 User.create! 如果创建失败出会报错。！其余的用法👉 关于-Ruby-方法名后的问号和感叹号
2. @jobs.to_a 可以将对象的值打印出来。
3. 用户注册的连接 new_user_registration_path。
4. 用户登录的连接 new_user_session_path。
5. 用户注销的连接 destroy_user_session_path。
6. bootstrap 的 grid system。
7. render 后使用 return 终止代码的执行。
8. def show
9. @book = Book.find(params[:id])
10. if @book.special?
11. render action: "special_show" and return
12. end
13. render action: "regular_show"
14. end
15. 返回上一个页面：
    - redirect_back(fallback_location: root_path)。
    - redirect_to :back。
16. 设置 redirect_to 的 HTTP 状态码：redirect_to photos_path, status: 301。
17. redirect_to 与 render 在 Controller 中的区别
18. def index

19. @books = Book.all
20. end
21.
22. def show
23. @book = Book.find_by(id: params[:id])
24. if @book.nil?
25. render action: "index"
26. end
27. end

render 不会执行目标 action 中的代码，只会把 view 展示在我们的面前，所以当 @book 为 nil 的时候，index.html.erb 所需要的 @books 的值是空的，页面就会是没数据的。使用 redirect_to 可以修复这个问题：

```
def index
  @books = Book.all
end

def show
  @book = Book.find_by(id: params[:id])
  if @book.nil?
    redirect_to action: :index
  end
end
```

redirect_to 发起一个访问 index 的新的请求，这样 index 中的代码就会被执行，@books 的值就不会为空了（如果存在数据的话）。但是这样做会有一个弊端，就是会造成浏览器多产生一次请求，可以使用下列代码进行优化：

```
def index
  @books = Book.all
end

def show
  @book = Book.find_by(id: params[:id])
  if @book.nil?
    @books = Book.all
    flash.now[:alert] = "Your book was not found"
    render "index"
  end
end
```

28. 使用 head 构建只返回报头的响应：head :bad_request。
29. Layout 的结构，在 layout 或 view 中可以使用以下三种工具组合在一起完成一次响应。
    - Asset tags
    - yield and content_for
    - Partials
30. Asset tags helper，用来在页面插入一些标签（不会检查打算引用的文件是否存在）

- auto_discovery_link_tag 链接到 Feed。
- javascript_include_tag 在页面添加引用 javascritp 的连接。<%= javascript_include_tag "main" %> 会被解析为：<script src='/assets/main.js'></script>，<%= javascript_include_tag "main", "columns" %>会分别插入 man.js 和 columns.js。默认会在 app/assets/javascripts 目录下寻找 js 文件。
- stylesheet_link_tag 在页面添加引用 css 的连接，默认会在 app/assets/javascripts 目录下寻找 css 文件。
- image_tag 会在页面生成 <img /> 标签，默认会在 public/images 下寻找图片文件。
- video_tag 会在页面生成 <video /> 标签，用来连接视频。
- audio_tag 会在页面生成

31. yield 定义一个用来插入其他 view 的内容的区域，分为匿名 yield 和命名 yield。

匿名 yield：

```
<html>
  <head>
  </head>
  <body>
  <%= yield %>
  </body>
</html>
```

命名 yield:

application.html.erb

```
<html>
  <head>
  <%= yield :head %>
  </head>
  <body>
  <%= yield %>
  </body>
</html>
```

head.html.erb

```
<% content_for :head do %>
  <title>A simple page</title>
<% end %>
<p>Hello, Rails!</p>
```

整合结果为：

```
<html>
  <head>
  <title>A simple page</title>
  </head>
  <body>
  <p>Hello, Rails!</p>
  </body>
</html>
```

今日达成

1. 为 jdstore 挂上 bootstrap、devise、simple_form。
2. 为 jdstore 添加响应式 navbar。
3. 为 jdstore 添加停留在底部的 footer。

今日情绪

平静。

今日领悟

关注 自我 的成长，关注 自身 的价值，不要太在意他人的成就或者在做什么。

要和过去的自己比较，争取每天进步一点点。每天进步一点点的方法就是不做让自己后退的事儿，同样的错误不能错三次。

今日结语

一点一滴积累，关注自我价值。

今日不足

今日 BUG

1. fork 下来之后无法启动，原因是缺少 database.yml 文件。
2. 在给 jdstore 挂 bootstrap 的时候，样式始终是错的，后来发现我复制了一份 appplication.css 为 appplication.scss 看是原来的 appplication.css 没有删除，系统应该是先调用了 appplication.css，将其删除后一切正常。
3. 响应式 navbar 点击菜单不展开下来列表，发现是没有引用 collapse.js。

full-stack-note-day05

今日回顾

学习了 migration 的相关操作，rake db:rollback，rake db:reset 和 rake db:drop。

render 在 view 中和 controller 中的用法。

使用 Helper 的几种用法：

1. 封装文本格式化的方法。
2. def render_job_description(job)
3. simple_format(job.description)
4. end
5. 封装 view 中的逻辑代码。
6. def render_job_status(job)
7. if job.is_hidden
8. content_tag(:span, "", style: "color:red", class: "fa fa-lock fa-1")
9. else
10. content_tag(:span, "", style: "color:green", class: "fa fa-globe fa-1")
11. end
12. end
13. 配合 partial 封装复杂的逻辑。
14. def render_job_state(job)
15. if a
16. render :partial => "a"
17. else
18. render :partial => "b"

```
    end
  end
```

今日达成

1. PartialRender 一篇。
2. RenderUseInController 一篇。
3. Scope 一篇。

今日情绪

在 xdite 分享自己的幸运公式时，情绪达到顶峰，其余时间很平静。

今日领悟

幸运的公式：

1. 看说明书，可躲过 80% 的坑。
2. 选择一个比你所做的决定 高一级的决定 ，选择一条 比较远的路 。

今日结语

错过的就不是你的。ps：没吃上猪蹄。

今日不足

没有产生突破感，继续努力。

今日 BUG

```
<%= render layout: @jobs.to_a do |job| %>
  Title: <%= job.title %>
<% end %>
Missing partial jobs/_[#<Job
```

Scope

scope 的作用

将常用的查询定义成可以通过 association object 或 model 来调用的方法。

定义一个简单的 scope

```
class Job < ApplicationRecord
  scope :show_public, -> { where(:is_hidden => false) }
  scope :recent, -> { order(created_at: :desc) }
end
```

或者写成

```
class Job < ApplicationRecord
  def self.show_public
    where(:is_hidden => false)
  end
end
```

两种方式是等价的。

定义链式 scope

```
class Job < ApplicationRecord
  scope :show_public, -> { where(:is_hidden => false) }
  scope :show_public_and_recent, -> {show_public.order(created_at: :desc)}
end
```

传递参数到 scope 中

定义：

1. scope 到形式

```
class Job < ApplicationRecord
    scope :created_before, ->(time) { where("created_at < ?", time) }
end
```

1. 类方法到形式

```
class Job < ApplicationRecord
    def self.created_before(time)
        where("created_at < ?", time)
    end
end
```

调用：

```
Job.created_before(Time.zone.now)
```

推荐使用类方法的形式定义带参数的 scope。

带有条件判断的 scope

```
class Job < ApplicationRecord
  scope :created_before, ->(time) { where("created_at < ?", time) if time.present? }
end
```

或

```
class Job < ApplicationRecord
  def self.created_before(time)
    where("created_at < ?", time) if time.present?
  end
end
```

Read on RenderUseInController

渲染同一个 controller 下 action 的 view

1. 格式：render "edit" or render :edit。
2. 例子：
3. <!-- jobs_controller.rb -->
4. def update
5. @job = Job.find(params[:id])
6. if @job.update(job_params)
7. redirect_to jobs_path, notice: "Job Update Success."
8. else
9. render :edit
10. end
11. end

Read on PartialRenderer

命名规范

作为局部视图（partial） 的文件的文件名必须以下划线开头。

访问方式

默认情况下 rails 会在引用局部视图（partial）的文件所在的目录下寻找该文件所引用的局部视图

（partial）文件。
例如：在 Job#index 的模版 jobs/index.html.erb 中使用 <%= render partial: "_job_item.html.erb" %> 就会将 jobs/_job_item.html.erb 文件加载到 index.html.erb 文件中。
也可以使用其他 controller 下的 partial，格式为 <%= render partial: "xxxcontroller/_xxx.html.erb" %> xxxcontroller 为某个 controller 下的 view 所在的文件夹，比如要在 admin/jobs_controller 下的 view 里引用 jobs_controller 下面的 partial 的话，就可以这样写 <%= render partial: "jobs/_xxx.html.erb" %> 反之如果想在 jobs_controller 下的 view 里引用 admin / jobs_controller 下面 partial 的话，就可以这样写 <%= render partial: "admin/jobs/_xxx.html.erb" %>。
Read on
weekly-01
本周回顾
本周我们从什么是程序思维开始，逐步了解了 任务地图和用户故事，通过这两个工具可以将问题进行逐步分解。学习了 git 的基本操作，并进行了 fork 与 pull request 的练习。通过练习 job-listing 项目，进一步巩固了 CRUD 的熟悉度，也学会了一些新的技巧，比如：
Read on
BigSurprise
这周我接触到了四个特别棒的概念，在反复思考这四个概念并尽力践行之后，它们正在慢慢改变我的行为以及思考方式。
注意力才是最宝贵的资源
什么才是最宝贵的？有的人认为是钱，但钱一定不是最宝贵的，因为钱是可再生的资源。有的人认为是时间，因为时间是不可再生的资源，是有一定的道理，但是我们对时间束手无策呀，因为时间不会受任何人的控制，只会自顾自地一直向前流淌...
那什么才是最宝贵的呢？笑来老师告诉我们，我们的 注意力 才是最宝贵的，因为首先注意力伴随着时间，所以从这个角度来看 注意力 也是不可再生的，在某一段时间里我们把注意力投向了哪里，那这段时间过后我们便再也无法改变在那段时间里注意力的投向了。从这个角度来看，我现在之所以很菜，就是因为我把宝贵的 注意力 浪费在了太多与我无关的人或事上面。换言之，想要在某方面变得厉害，那就把 注意力 多投向那个方面吧，随着不断地积累，一定会大不一样。知道这个概念之后，我给自己的大脑安装了一个简单的“插件”，当我发现我准备或已经开始投入注意力在某个方面的时候，我会问“这跟我有关系吗？”，如果答案是没有，我会放弃投入自己的注意或将自己的注意力拉回来。
Read on
BigBang
初级坑
回顾这一周，踩过一些坑。归纳起来主要有以下几种类型：

1. 拼写以及语法错误
    - 在 Controller 中把 before_action 写成 defore_action 😂。
    - => 写成 = >。
    - 应该写 <% %> 的地方写成了 <%= %>。
2. 样式错误

比如按钮没有加 class 属性，结果就是它长的根本就不像个按钮。
Read on
full-stack-note-day04

今日回顾

1. 定制 gem 方式一

   去 github 上 fork 想要修改的 gem 的 repo，然后将 Gemfile 修改为 gem 'nokogiri', :git => 'https://github.com/shaounda/xxxx.git'
2. 定制 gem 方式二

   执行 gem unpack xxx xxx 为 想要定制的 gem 的名字。

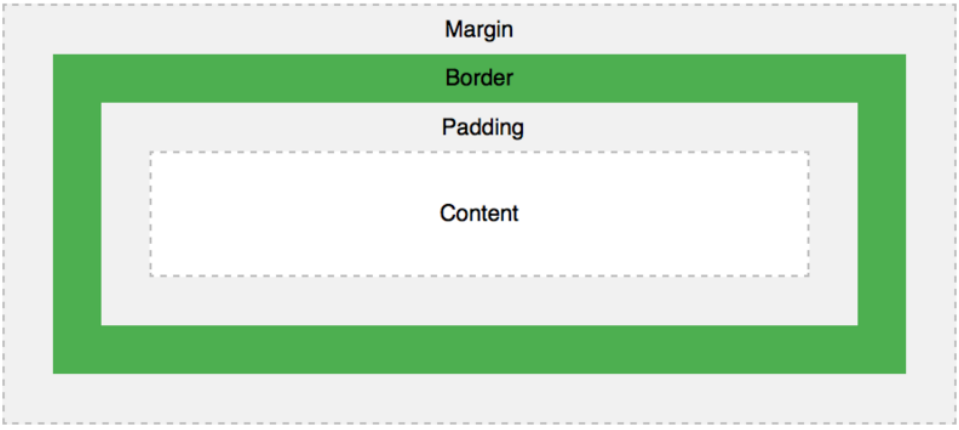
   修改 Gemfile 为 gem 'nokogiri', :path => './vendor/nokogiri'
3. Ruby 方法名后的问号和惊叹号的含义
4. CSS 的 box model。

今日达成

Read on

CSS Tips

什么是 box model



所有的 HTML 元素都可以被看做是盒子，在 CSS 中当我们讨论设计和布局的时候会使用 box model，每个 HTML 元素的外围都包裹着一个盒子，这个盒子由 margins 、borders、paddings 以及 content 组成。

- Content - 文本和图片会出现在这个部分。
- Padding - Content 周围的透明区域。
- Borker - 围绕 Padding 和 Content。
- Margin - border 外面的透明区域。
  当我们设置 width 和 height 的时候，只是指定了 content 的宽和高，在计算一个元素框的实际尺寸时除了元素本身的宽和高以外还应该算上 padding、margin 以及 border。(在 IE 8 以及 IE 8 之前版本的 IE 浏览器中会将 width 和 height 解析为算上 marging、 padding 以及 border 的最终值。)

margin 与 padding 的区别

1. margin 表示外边距，是围绕在元素边框之外的空白区域，而 padding 表示内边距，是元素与元素边框之间的部分，见 box model。
2. margin 的值可用为负数，而 padding 的值不可以为负数。

Read on

HTML(5) 编码风格总结

Html 元素部分

1. 在文档的第一行进行文档类型声明。使用<!DOCTYPE html> 或 <!doctype html>。
2. 文档中可以不包含 <head> 元素，必须包含<html>、<body> 和 <title> 元素。
3. 元素最好都是正确嵌套。
    - 错误示范：This text is bold and italic
    - 正确示范：This text is bold and italic
4. 元素最好都是闭合的，即使是空标签。例如：
    - 错误示范： This is a paragraph
    - 正确示范： This is a paragraph
5. 元素最好都使用小写字母。
    - 错误示范：<B></B>
    - 正确示范：
6. 元素属性的名字最好使用小写字母。
    - 错误示范：<table WIDTH="100%">
    - 正确示范：<table width="100%">
7. 元素属性的值最好使用双引号扩起来。
    - 错误示范：<table width=100%>
    - 正确示范：<table width="100%">

Read on

full-stack-note-day03

今日回顾

1. 公用的方法提取到 ApplicationController 中。
2. Model 中的验证方法是按照先后顺序执行的。
   validates :wage_upper_bound, presence: true, numericality: {greater_than: 0} 即使输错也提示不许为空。 validates :wage_upper_bound, numericality: {greater_than: 0}, presence: true 即使不输入也提示必须是数字。
3. 使用 在 validates 中使用 :message 自定义错误信息。

Read on

full-stack-note-day02

今日回顾

1. 修改后不需要重启的部位。
    - app/controllers
    - app/views
    - app/models
    - app/assets/stylesheets
    - app/assets/javascripts
    - db/migrate (需要执行 rake db:migrate)
2. rake routes 查看所有路径。
3. Gemfiel 所有套件存放的地方，修改后需要 执行 bundle install 并重启服务器。
4. 不希望 git 保管的文件要在 .gitignore 中进行声明。
5. rails c 带有 rails 环境，而 irb 不带有 rails 环境。
6. <%= @post.title %> 是打印，而 <% @post.title %> 是执行。

Read on
HTMLTips
div 与 span 的区别
1. div 是块级元素，而 span 是行内元素。
2. div 主要用作 HTML 元素的容器或用来布局， 而 span 主要用作文本容器。

class 与 id 的区别
1. class 用来指向样式列表中的类，而 id 用来给 HTML 元素设置唯一的编号。
2. 同一个 HTML 元素可以有多个 class，而同一个 HTML 元素只能有一个 id。
3. 在同一个 html 文件中不同的 HTML 元素可以有相同的 class，而 id 必须是唯一的。

Read on
full-stack-note-day01
程序思维
程序思维就是用电脑解决实际问题的思维。
步骤：
1. 把大问题拆解成可用解决但小问题。
2. 在实操的过程中找到资源。

Read on
基础开发环境配置
安装 Xcode
在 App Store 下载 Xcode
安装 Command Line Tools
1. 执行 xcode-select --install 进行安装。
2. 执行 xcode-select -p 若有输出则为安装成功。

安装 Homebrew
ruby -e"$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
Read on
rails101-3 summary
Bootstrap 是什么？
Bootstrap 是一个有名的 CSS 框架，官网地址为：http://getbootstrap.com/。
scss 是什么？
以编程的方式生成 CSS 文件。
@import 是什么？
application.scss 中的 @import "bootstrap" 表示引入 bootstrap 模块。
application.html.erb 是什么？
所有页面的默认组成部分。
Read on
rails101-2 ch07
生成某 namespace 下的 controller
rails g controller account/groups
实现 Post 的 edit、update 以及 destroy 方法（初级版）
class PostsController < ApplicationController

```
  before_action :authenticate_user!, :only => [:new, :create, :edit, :update, :destroy]

  def new
    @group = Group.find(params[:group_id])
    @post = Post.new
  end

  def create
    @group = Group.find(params[:group_id])
    @post = Post.new(post_params)
    @post.group = @group
    @post.user = current_user

    if @post.save
      redirect_to group_path(@group)
    else
      render :new
    end
  end

  def edit
    @group = Group.find(params[:group_id])
    @post = Post.find(params[:id])
  end

  def update
    @post = Post.find(params[:id])
    @post.update(post_params)
    redirect_to account_posts_path
  end

  def destroy
    @post = Post.find(params[:id])
    @post.destroy
    redirect_to account_posts_path
  end

  private

  def post_params
    params.require(:post).permit(:content)
  end
end
```

初级版的问题是任何人都可以对某一篇 Post 进行修改，缺少权限控制，还存在代码冗余的情况。
实现 Post 的 edit、update 以及 destroy 方法（升级版）

```
class PostsController < ApplicationController
  before_action :authenticate_user!, :only => [:new, :create, :edit, :update, :destroy]
  defore_action :find_post_and_check_permission, :only => [:edit, :update, :destroy]

  def new
    @group = Group.find(params[:group_id])
    @post = Post.new
  end

  def create
    @group = Group.find(params[:group_id])
    @post = Post.new(post_params)
    @post.group = @group
    @post.user = current_user

    if @post.save
      redirect_to group_path(@group)
    else
      render :new
    end
  end

  def edit
    @group = Group.find(params[:group_id])
  end

  def update
    if @post.update(post_params)
      redirect_to account_posts_path, notice: "Update Success"
    else
      render :edit
    end
  end

  def destroy
    @post.destroy
    redirect_to account_posts_path
  end

  private
```

```
  def find_post_and_check_permission
    @post = Post.find(params[:id])
    if current_user.id != @post.user_id
      redirect_to account_posts_path, alert: "You have no permission.(only the post's writer can do it.)"
    end
  end

  def post_params
    params.require(:post).permit(:content)
  end
end
```

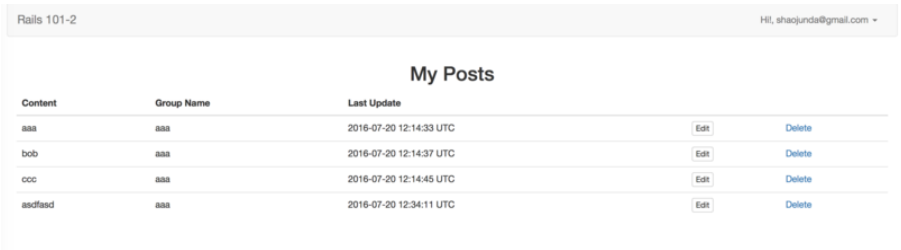
升级版加入了 find_post_and_check_permission 方法用来消除冗余代码并进行权限验证（只有 Post 的 writer 能够修改或者删除该 Post ），并将其加入到 before_action 中。
信息提示的两种方式

1. one
   redirect_to account_posts_path, alert: "You have no permission.(only the post's writer can do it.)"
2. two
3. flash[:alert] = "You have no permission.(only the post's writer can do it.)"

4. redirect_to account_posts_path

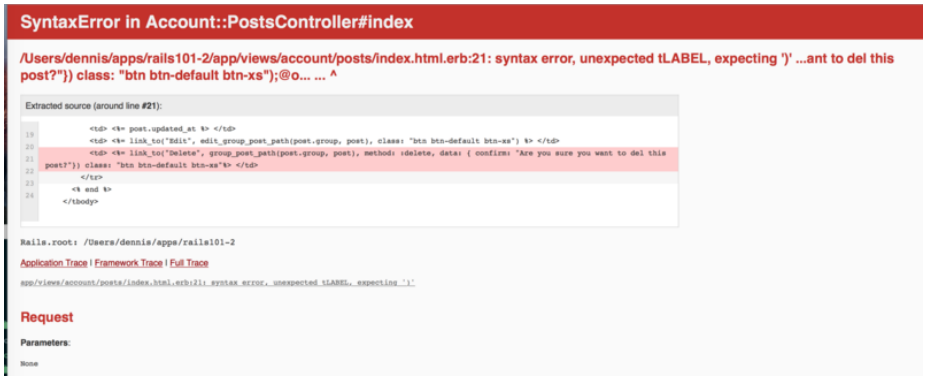
错误处理

1. My Posts 页面的 delete 按钮没有样式。



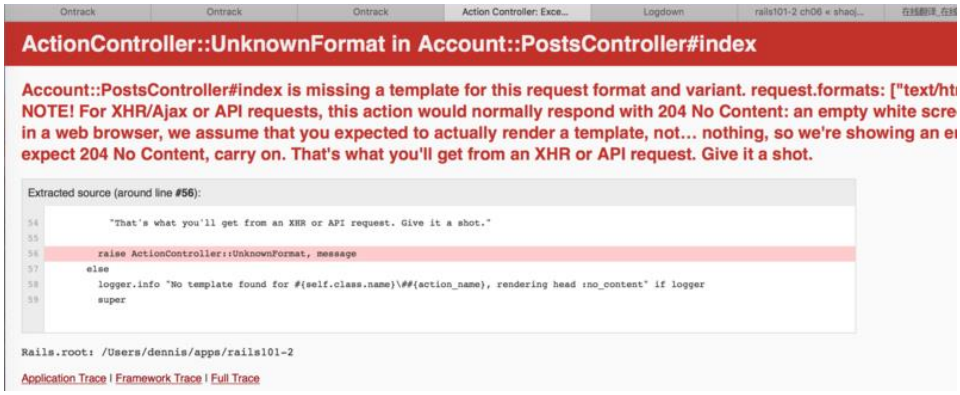
发现是 delete 按钮忘记加 class 属性了。

1. 加上 class 属性后报错。



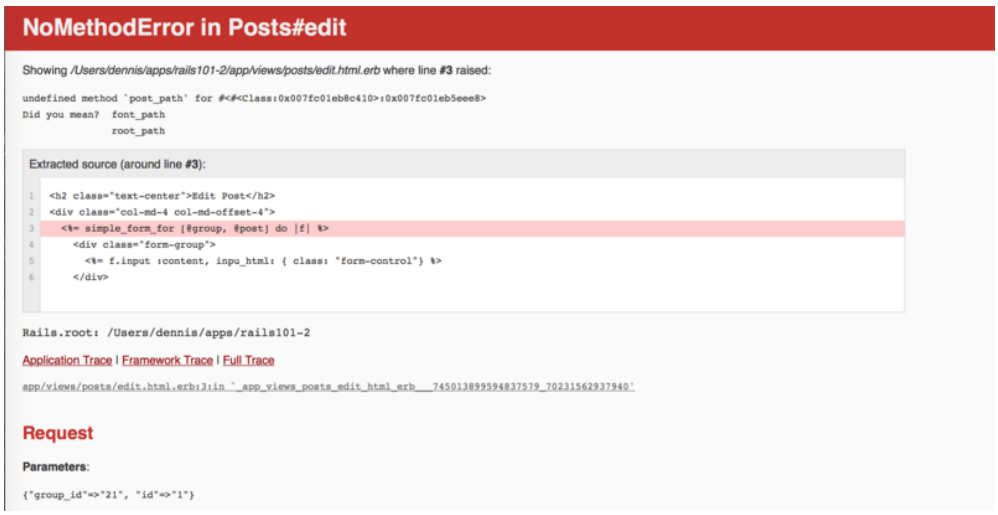
发现是 class 属性前忘记加 逗号 了。

1. 访问 http://localhost:3000/account/posts 报错。



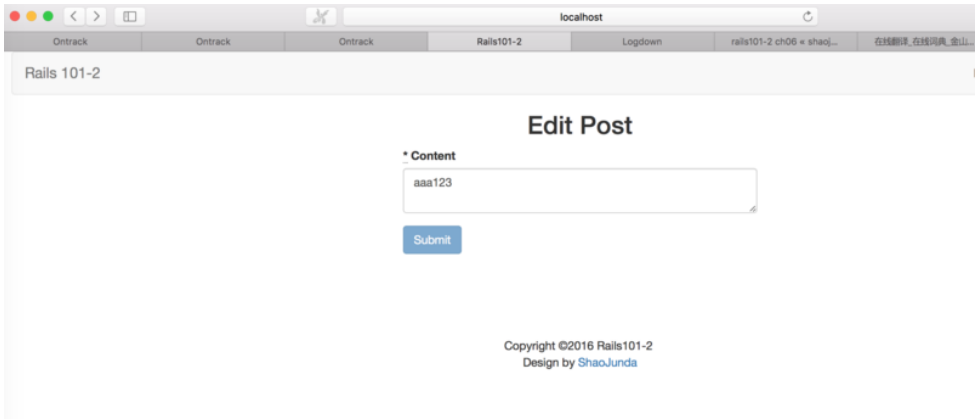
发现是还没有创建 index.html.erb。

1. 访问 http://localhost:3000/groups/21/posts/1/edit 报错。（在做修改 post）



发现是后台没有获得 @group。

1. 修改 Post 之后页面不动了，Submit 按钮变为了不可点击状态。



发现是 update 方法中忘记指定跳转路径了。redirect_to account_posts_path 了。

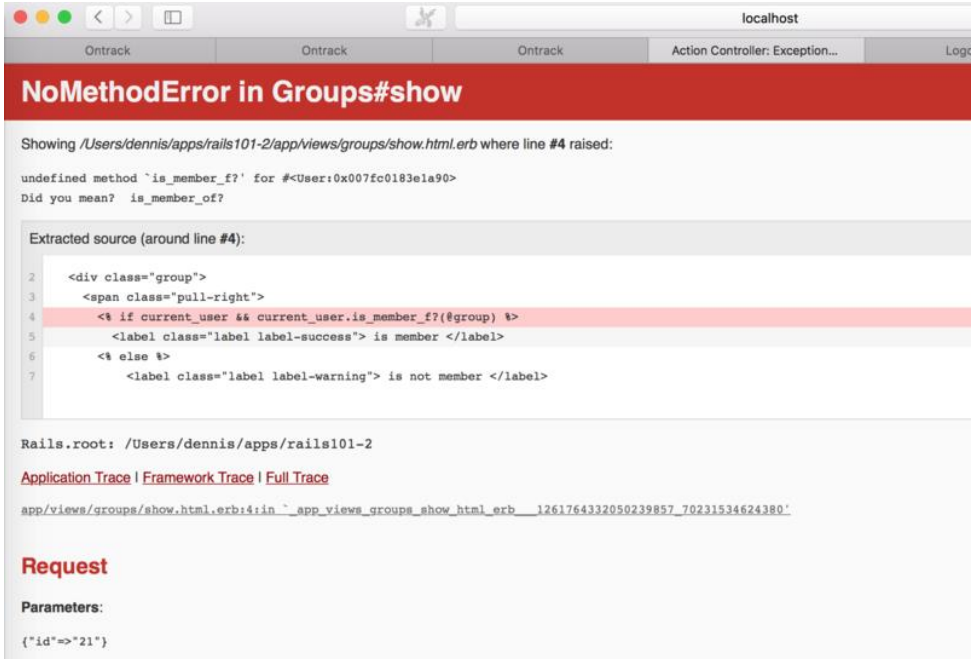
1. 执行修改 Post 的操作时报错。

发现是将 before_action 写成了 defore_action。

[rails101-2 ch06]

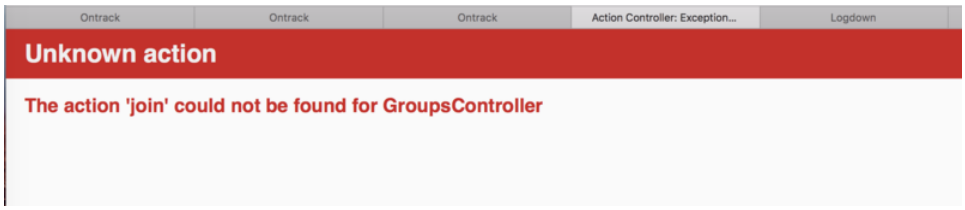
错误处理

1. 访 问 http://localhost:3000/groups/21 报 错 。



发现是 将 current_user.is_member_of? 写成了 current_user.is_member_f?

1. 申 请 加 入 某 个 讨 论 组 时 报 错 。 URL ： http://localhost:3000/groups/21/join



发现是将 join 方法 写在 private 后面了。事实证明写在 private 后面的方法外部无法访问。

[rails101-2 ch05]

使用 scope 将 SQL 语句变得易于理解

1. 在相应的 Model 文件中 加入 scope :recent, -> { order("created_at DESC")}
2. 在 controller 中使用 XXX.all.recent

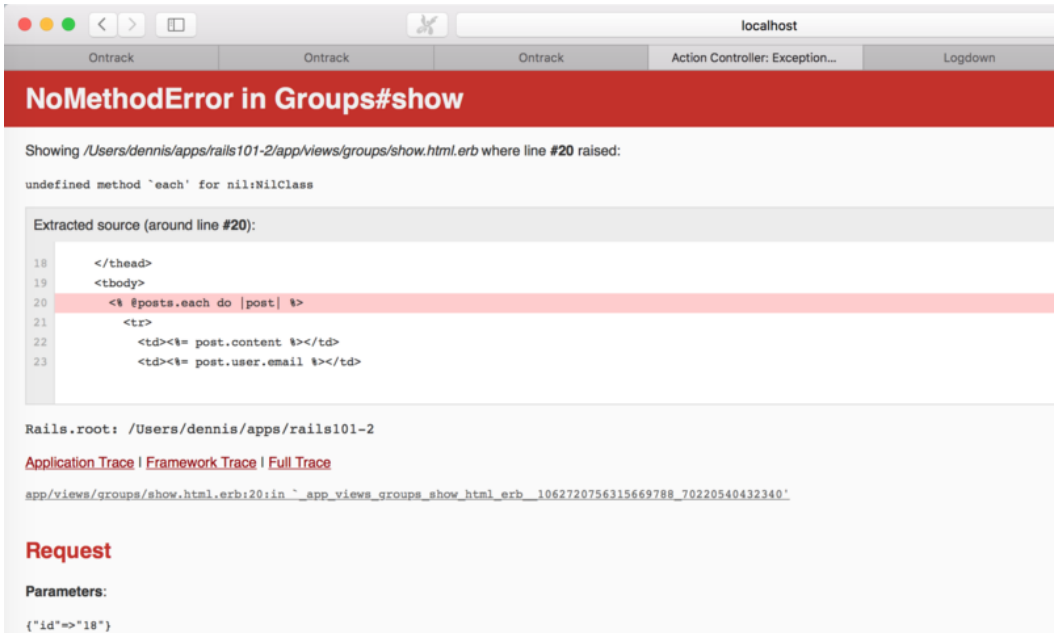
使用 will_paginate 进行分页

1. 在 Gemfile 中加入 gem "will_paginate"。
2. 执行 bundle install。

3. 重启服务器 rails s。
4. 修改 controller 代码，将 @posts = @group.posts.recent 改为 @posts = @group.posts.recent.paginate(:page => params[:page], per_page => 5)。其中 per_page 为每页显示的个数。
5. 在需要显示分页的页面中加入 <%= will_paginate @posts %>

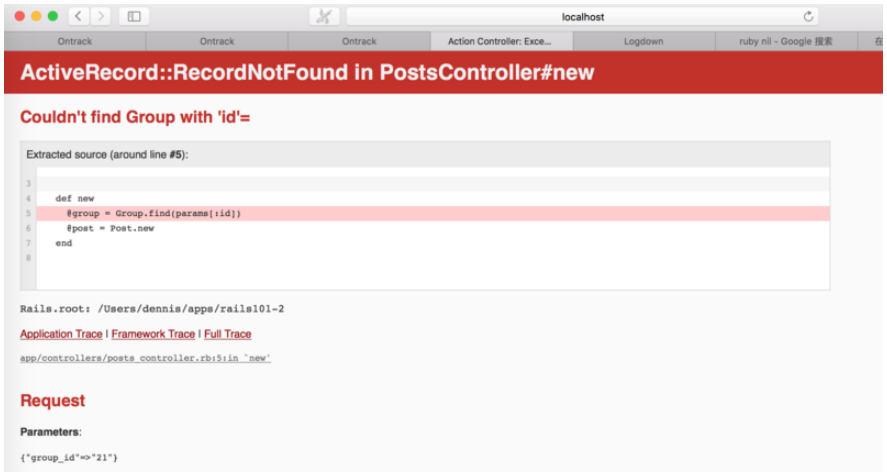
错误处理

1. 当一个讨论组下没有任何文章但时候，访问讨论组会报错。http://localhost:3000/groups/18



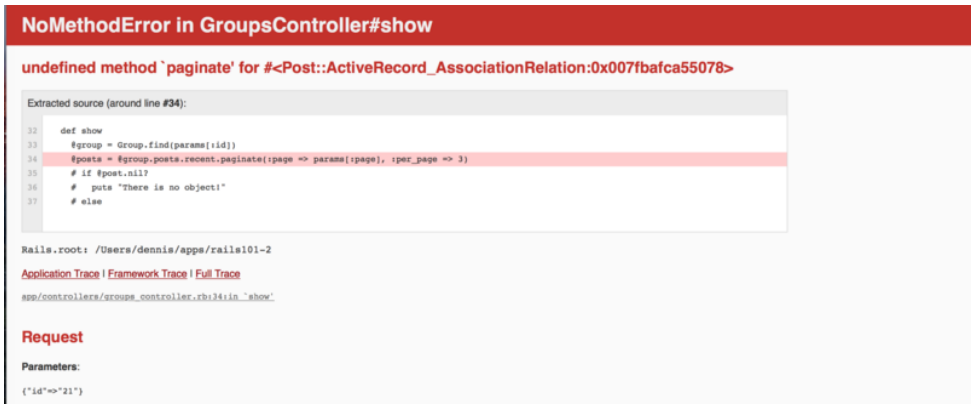
发现是在 controller 中把 @posts 写成了 @post

1. 新建文章的时候报错。URL：http://localhost:3000/groups/21/posts/new



发现是把 controller 中 new 方法的 @group = Group.find(params[:group_id]) 写成了@group = Group.find(params[:id])

1. 加入分页功能后访问 http://localhost:3000/groups/21 报错



发现是加入 will_paginate 之后没有重启服务器。

[rails101-2 ch04]

使用 devise 实现用户登陆系统

1. 在 Gemfile 中添加 gem devise。
2. 执行 bundle install。
3. 执行 rails g devise:install。

这一步生成了 2 个文件：

```
~/apps/rails101-2 ch04± rails g devise:install
Running via Spring preloader in process 76595
      create  config/initializers/devise.rb
      create  config/locales/devise.en.yml
```

4. 执行 rails g devise user。
   - 这一步生成了 5 个文件：

```
~/apps/rails101-2 ch04± rails g devise user
Running via Spring preloader in process 76636
      invoke  active_record
      create    db/migrate/20160720083128_devise_create_users.rb
      create    app/models/user.rb
      invoke    test_unit
      create      test/models/user_test.rb
      create      test/fixtures/users.yml
      insert    app/models/user.rb
       route  devise_for :users
```

   - 在 config/routes.rb 文件中加入 devise_for :users
5. 执行 rake db:migrate。

```
~/apps/rails101-2 ch04± rake db:migrate
== 20160720083128 DeviseCreateUsers: migrating ================================
-- create_table(:users)
   -> 0.0078s
-- add_index(:users, :email, {:unique=>true})
   -> 0.0008s
-- add_index(:users, :reset_password_token, {:unique=>true})
   -> 0.0006s
== 20160720083128 DeviseCreateUsers: migrated (0.0094s) =======================
```

这一步创建了 users 表。

6. 重启 server。

设置操作前的登陆限制

在 controller 中加入

before_action :authenticate_user! , only: [:new, :create, :edit, :update, :destory]
可以限制 only:[xxx] 中的操作必须登陆后才能执行。
修改表结构

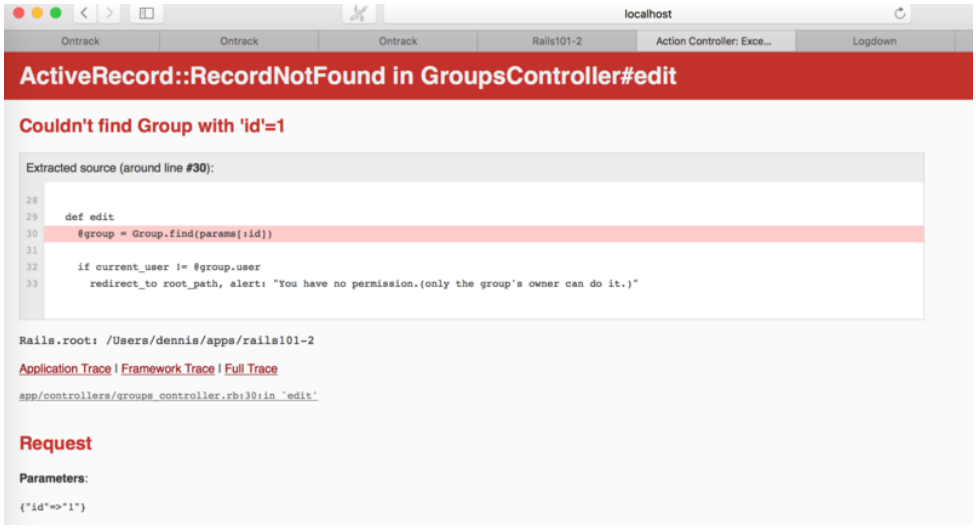
1. 生成 migration 文件。
   rails g migration add_user_id_to_group
2. 修改新生成的 migration 文件。
   新增：
3. def change
4. add_column :groups, :user_id, :integer
5. end
6. 执行修改操作
   rails db:migrate

tips

- db/schema.rb 文件中有所有表的创建信息。
- db/migarte 文件夹中逐个存放了表的创建信息或表结构的修改信息

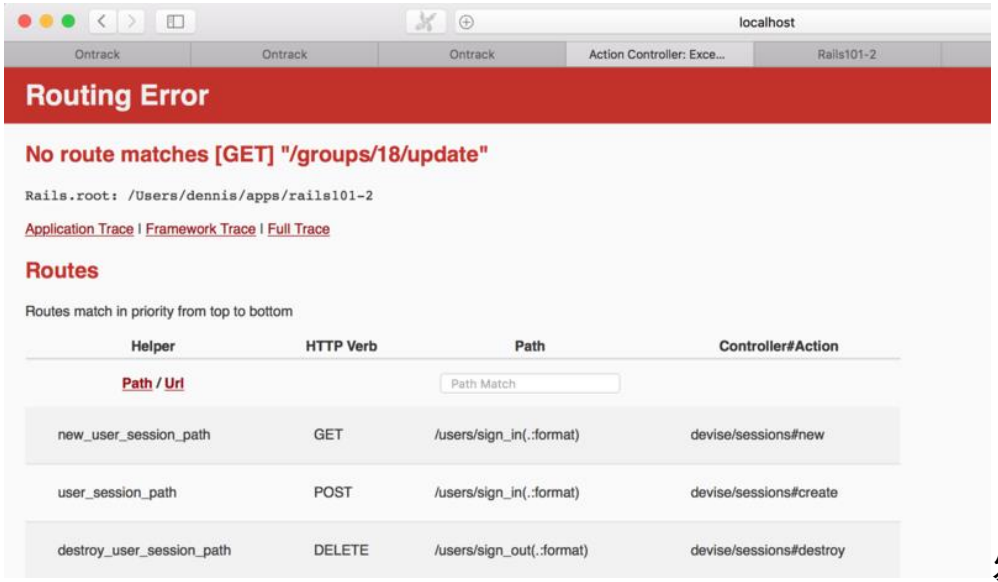
错误处理

1. 访 问 http://localhost:3000/groups/1/edit 报 错 。



发现是数据库中没有 id 为 1 的 group 记录。

1. 访 问 http://localhost:3000/groups/18/update/ 报 错



发现是没有匹配的 route。

[rails101-2 ch03]

创建数据

1. Group.new 只是创建了一个空的 Group Model，用于保存表单中的数据。

```
Started GET "/groups/new" for ::1 at 2016-07-20 13:51:25 +0800
Processing by GroupsController#new as HTML
The string has #<Group:0x007fcbda2ac198> characters
  Rendering groups/new.html.erb within layouts/application
  Rendered groups/new.html.erb within layouts/application (1.2ms)
  Rendered common/_navbar.html.erb (0.4ms)
  Rendered common/_flashes.html.erb (0.3ms)
  Rendered common/_footer.html.erb (0.2ms)
Completed 200 OK in 36ms (Views: 29.3ms | ActiveRecord: 0.5ms)
```

1. Group.new(group_params) 根据表单中的数据创建一个新的 Model。
2. @group.save 将 Model 插入到数据库中。

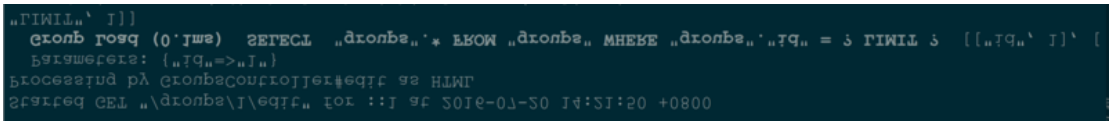
```
Started POST "/groups" for ::1 at 2016-07-20 13:51:28 +0800
Processing by GroupsController#create as HTML
  Parameters: {"utf8"=>"✓", "authenticity_token"=>"MOlt17RP/x/OY19MYkKtEnrcRq3h2TdLZf167OddyX4A8rxSUs
6DJIe0NeMiAhbUbshUKUp3Dd3xSfLrMvaXSw==", "group"=>{"title"=>"2312", "description"=>"1231"}, "commit"=
>"Submit"}
   (0.1ms)  begin transaction
  SQL (0.3ms)  INSERT INTO "groups" ("title", "description", "created_at", "updated_at") VALUES (?, ?
, ?, ?)  [["title", "2312"], ["description", "1231"], ["created_at", 2016-07-20 05:51:28 UTC], ["upda
ted_at", 2016-07-20 05:51:28 UTC]]
   (2.1ms)  commit transaction
The string has #<ActionController::Parameters:0x007fcbd6e5a4f8> characters
Redirected to http://localhost:3000/groups
Completed 302 Found in 6ms (ActiveRecord: 2.6ms)
```

1. if @group.save 可用来判断是否成功保存数据。

查找数据

Group.find(params[:id])

会根据 id 执行 select 语句，查找出相应记录。



修改数据

@group = Group.find(params[:id])

@group.update(group_params)

先根据 id 找到一条记录，然后根据页面传递到后台的参数修改这条记录。

设置表格中某个输入域为必填项

将 group.rb 修改为

```
class Group < ApplicationRecord
    validates :title, presence: true
end
```

使用 partial 较少重复代码

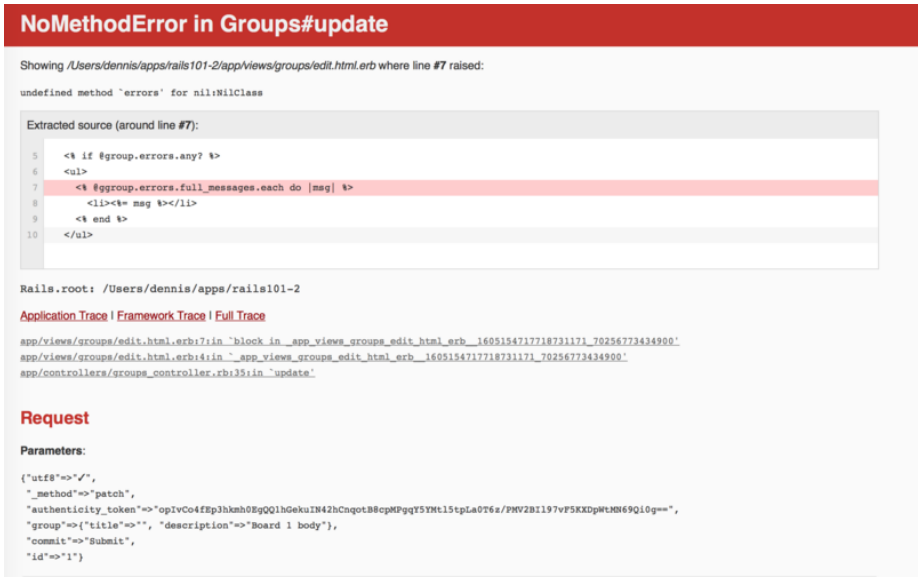
1. 将页面中的重复代码提取到新建的 partial 页面中。
2. 在具有重复代码的页面中应用 partial 页面。 <%= render "form" %>。
3. 应用规则：具有重复代码的页面会去其所在目录下寻找 _xxxx.html.erb 。

tip

app/ 下的文件都是动态载入的，修改后不需要重启服务器，其余的文件夹的文件修改后需要重启服务器。

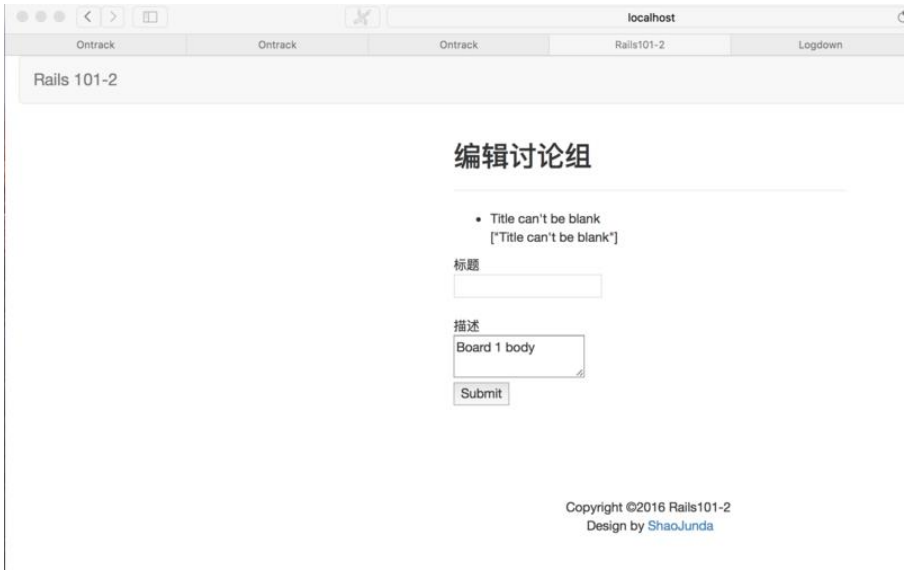
错误处理

1. 在讨论组列表页点击删除按钮并没有弹出“是否要删除”的对话框。 发现是将 confirm: "确定要删除这个讨论组吗？" 写成了 comfirm: "确定要删除这个讨论组吗？" 一字之差...
2. 删除后也没有显示删除成功的提示信息。 发现是将 alert: "Group deleted"写成了 alter: "Group deleted" 再一次...
3. 添加 title 验证后的修改页面报错。



发现是将 @group 写成了 @ggroup

1. 使用了 partial 的页面错误信息出现重复。



发现是将 <% @group.errors.full_messages.each do |msg| %> 写成了 <%= @group.errors.full_messages.each do |msg| %> 多打了一个 = 。

[rails101-2 ch02]

MVC 的通用实现步骤

1. 设计 Model 并创建出相应的表结构
    1. rails g model group title:string description:text

这条命令共生成了 4 个文件：

    2. db/migrate/20160719194213_create_groups.rb

3. app/models/group.rb
4. test/models/group_test.rb
5. test/fixtures/groups.yml
6. rake db:migrate
   这条命令在数据库中创建了 groups 表。

2. 创建 controller
   rails g controller groups
3. 创建 Views
   创建 app/views/groups/index.html.erb
4. 配置访问路径
   修改 config/routes.rb 文件

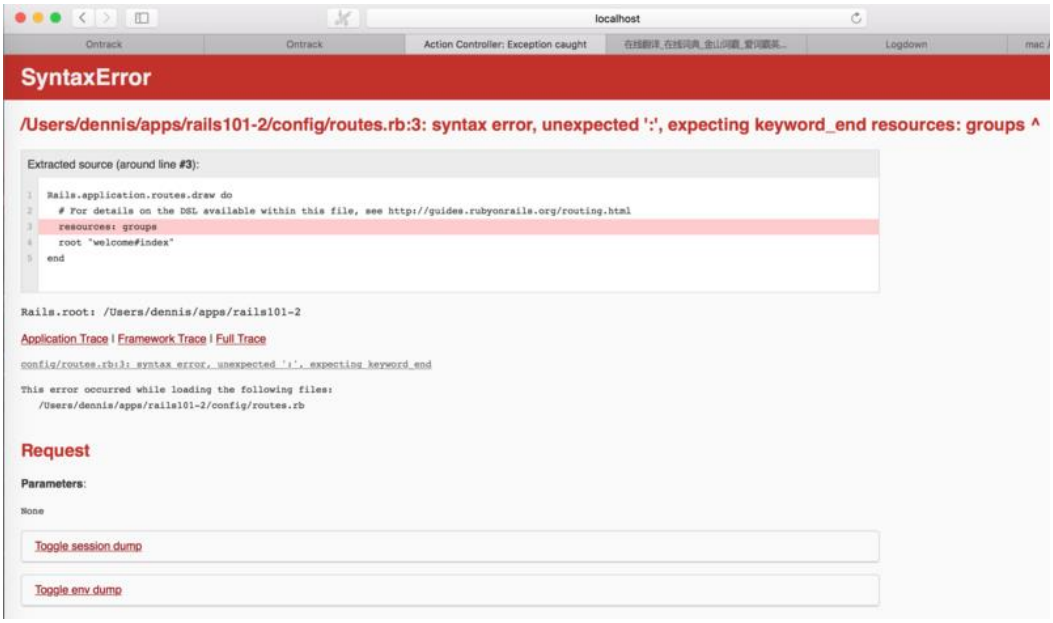
向数据库中插入数据

1. rails console
2. Group.create(title: "Board 1", description: "Board 1 body")

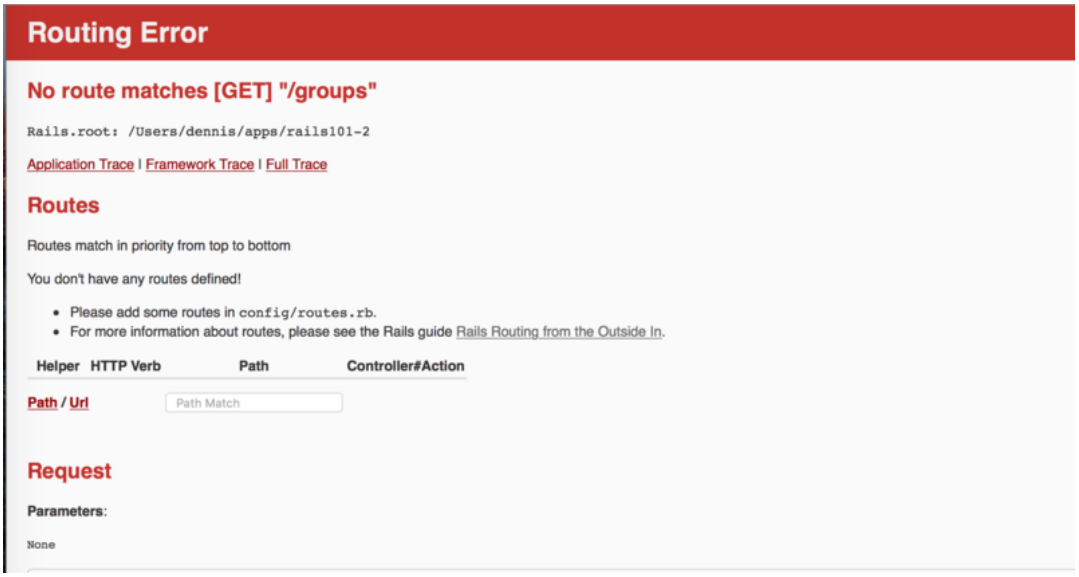
该命令会开启一个事务，并向 groups 表中插入一条数据。

错误处理

1. 配置完 config/routes.rb 之后访问 http://localhost:3000/ 报错

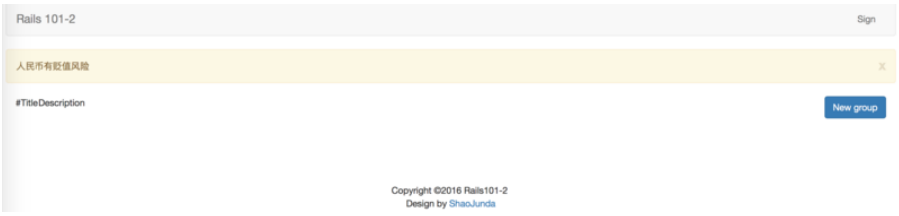


2. 配置完 config/routes.rb 之后访问 http://localhost:3000/groups/ 报错



发现是将 resources :groups 写成了 resources: groups

1. 列 表 页 # Title Description 紧 贴 在 了 一 起 。



发现是 <table> 标签缺少了 class="table table-hover"

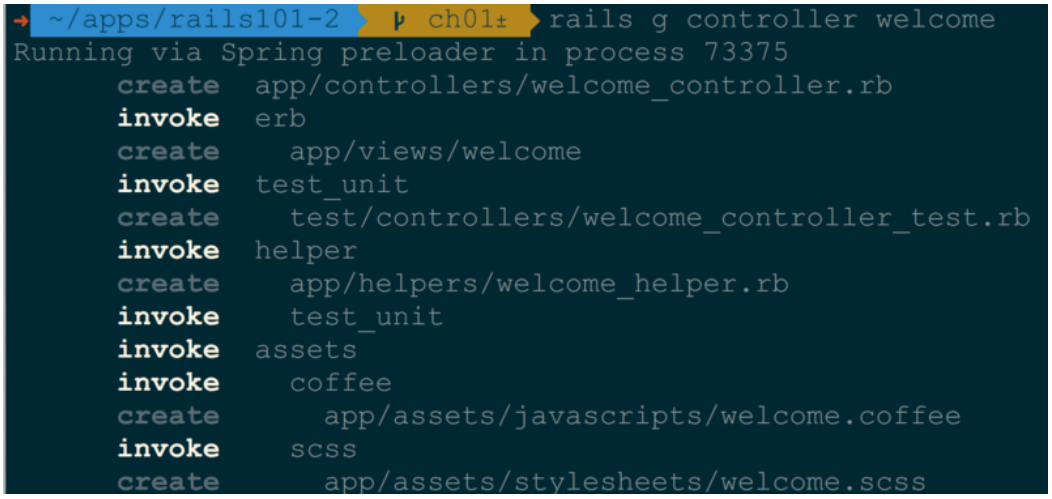
[rails101-2 ch01]

在项目中使用 bootstrap-sass

1. 在 Gemfile 中加入 gem 'bootstrap-sass'
2. bundle install
3. 在 Bootstrap 的 css 文件中加入 ```@import "bootstrap-sprockets"; @import "bootstrap";
4. ### 使用 Bootstrap 的 HTML 样式
5. mkdir app/views/common 新建公用文件夹。
6. touch app/views/common/_navbar.html.erb touch app/views/common/_footer.html.erb 分别创建导航栏文件和页尾文件。
7. 分别在 _navbar.html.erb 和 _footer.html.erb 中加入相应的 HTML 代码。
8. 在全域 HTML 样式 application.html.erb 中引用 _navbar.html.erb 和_footer.html.erb

生成 welcome controller

rails g controller welcome



这条命令一共生成了 5 个文件 1 个文件夹：

1. app/controllers/welcome_controller.rb
2. test/controllers/welcome_controller_test.rb
3. app/helpers/welcome_herlper.rb
4. app/assets/javascripts/welcome.coffee
5. app/assets/stylesheets/welcome.scss
6. app/views/welcome

新增欢迎界面

7. touch app/views/welcome/index.html.erb
8. 在欢迎界面中加入欢迎信息。

将欢迎界面设置为首页

在 config/routes.rb 文件中加入 root welcome#index

启动服务器查看效果

rails s

制作美观的提示信息

9. 在 app/assets/javascripts/application.js 中 加入 //= require bootstrap/alert。
10. 制作信息提示页面 app/views/common/_flashes.html.erb。
11. 制作信息提示 helper app/helpers/flashes_helper.rb 。
12. 在全域 HTML 样式 application.html.erb 中引用信息提示页面。

信息提示 helper 的不同样式

绿色提示样式：flash[:notice] = "早安！你好！"

红色警告样式：flash[:alert] = "早安！你好！"

黄色提醒样式：flash[:waring] = "早安！你好！"

错误处理

13. 界面显示

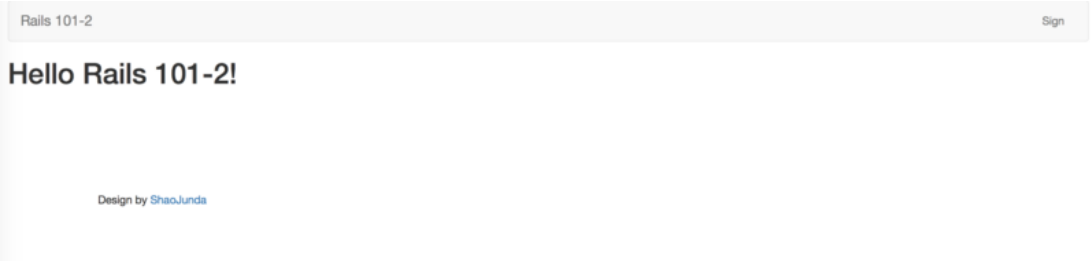
Unknown action

The action 'index' could not be found for WelcomeController

发现是没有在 app/views/welcome/ 目录下创建 index.html.erb 造成的。

9. 首页没有样式
   发现是 app/assets/stylesheets/application.scss 文件中忘记加入
   ```
   @import "bootstrap-sprockets";
   @import "bootstrap";
   ```

10. 页尾没有居中显示



发现是 <p> 标签缺少了 text-center 样式

---

git 撤销修改

1. 还没有执行 git add filename 使用 git checkout -- filename 可撤销新的修改与版本库保持一致。
2. 已经执行过 git add filename
    1. 使用 git reset HEAD file 可将文件变为没加入暂存区的状态。
    2. 使用 git checkout -- filename 丢弃修改。
3. 已经执行过 git add filename 和 git commit filename
    1. 使用 git log --pretty=oneline 查找到版本号。
    2. git reset --hard commit_id。

查看命令历史

git reflog

合并提交

1. git rebase -i HEAD~~。
2. 将要合并的记录前的 pick 改为 squash。
3. 编辑提交信息。

修改提交

1. git rebase -i HEAD~~。
2. 将要合并的记录前的 pick 改为 e。
3. 编辑要修改的文件。
4. 执行 git add filename
5. 执行 git commit --amend。
6. 修改提交信息。
7. 执行 git rebase --continue。

查看分支合并图

git log --graph

创建并切换到新分支

git checkout -b <branchname> 相当于：

1. git branch <branchname>
2. git checkout <branchname>

删除分支

git branch -d <branchname>

git 的配置文件

git 的配置信息存放在 ~/.gitconfig 文件中

git 别名配置

git config --global alias.st status

git config --global alias.a add

git config --global alias.ci commit

git config --global alias.co checkout

rails101-1 day03

使用 SimpleForm 简化表单样式

1. 安装 simple_form 在 Gemfile 中新增 gem "simple_form"
   执行 bundle install
2. 安装 simple_form for bootstrap rails generate simple_form:install --bootstrap
3. 修改 _form.html.erb

使用 devise 实现会员登录系统

1. 在 Gemfile 中新增 gem “devise”
2. bundle install
3. 产生会员系统的必须文件
   - rails g devise:install
   - rails g devise user
   - rake db:migrate
4. 重启 server rails s
5. 在 controller 中加入 before_action :authenticate_user! , only: [:new]

为 group 表添加一个列

1. rails g migration add_user_id_to_group
2. 在新生成的文件中加入 add_column :groups, :user_id, :integer
3. rake db:migrate

可以将每个动作之前都要进行的操作加入到 before_action 中、

before_action :find_group_and_check_permission, only: [:edit, :update, :destory]

增加分页功能

1. 在 Gemfile 中新增 gem “devise”
2. bundle install
3. 重启 server rails s
4. 在 controller 中加入分页代码 @posts = @group.posts.recent.paginate(:page => params[:page], :per_page => 5)
5. 在页面加入分页代码。<%= will_paginate @posts %>

推送所有分支到 github

git push --all origin

将最新分支作为 master 上传到 heroku

git push heroku ch08:master

rails101-1 day02

新建 Model 并生成表结构

1. rails g model group columnName:type[... columnName:type] eg: rails g model group title:string description:text
2. rake db:migrate

打开 rails 控制台并插入数据

1. rails c 打开控制台
2. Group.create(title: "Board 1", description: "Board 1 body") 插入数据

共用表单

1. 创建共用结构 _form.html.erb
2. 在页面引用共用结构 <%= render "form" %>

rails101-1 day01

查看 ruby 版本

ruby -v

安装 Rails（版本 5.0.0)

gem install rails -v 5.0.0

新建 rails 项目

rails new rails101

使用 git 做版本控制

git init

git add .

git commit -m "Initial Commit"

创建并切换到 ch01 分支

git checkout -b ch01

创建文件

touch _navbar.html.erb

新建 controller

rails g controller welcome

指定项目首页

将 config/routes.rb 文件内容修改为

```
Rails.application.routes.draw do
    root 'welcome#index'
end
```

在 terminal 中跳转到行首

ctrl + a

在 terminal 中跳转到行尾

ctrl + e