

lizenwei

[spring stop 命令](#)

当你运行 rails g 创建 controller 和 model 出现卡住的时候，你可以试着 ctrl + c 终止，然后运行 spring stop，接着再去创建 controller 或者 model

简单来说 spring 的作用是对 rails development 环境进行预加载和加速。

[rubyconf 2016 整理](#)

turbolinks 可用于网页加速，让网站成为单页应用（SPA）

- turbolinks 两大坑

①Turbolinks Event

解决：将事件委托到 document

使用 UUS 和 SJR

②Turbolinks Caching

让 JavaScript 操作幂

- turbolinks mobile 提供了移动端开发的一种选择

<https://github.com/ruby-china/ruby-china-android>

<https://github.com/ruby-china/ruby-china-ios>

- 重构约束

routes 中没有动词

action 转化为 resource 来表达

最多只用 7 个预设方法

- 自动升级 gem

deppbot.com 自动升级 gem 的 gem

[fullstack-course 9/12](#)

今天学到了什么

①jQuery

②简历的修改

今天的坑

对 jQuery 的学习近乎崩溃，陷入了新手学技能想去理解的坑，也有部分原因是因为英文实在太不好了。在同学和老师的提醒下，发现了这个问题。

明天继续学 jQuery 的 part-2 部分，方法改进：

①首先，看教学视频（打开英文字幕），看不懂听不懂没有关系，把 pattern 和 pattern 比较，只要大致知道做了某个动作会产生某个效果就好。

②直接做练习，照着答案打。**不需要理解！不需要理解！不需要理解！**甚至都不用去看那个练习是要让你做什么。先把这些东西练成肌肉记忆。

③去实际项目中应用

简历优化方法

①弱化培训营的说法，可以改成“跟培训班的朋友经营一个项目”，甚至在面试时可以谈谈这个项目的规划和想法。

②拿掉“学习”的字眼，大部分公司是不需要学生的。要改用“开创”、“探索”等字眼，公司是需要能做事、能成事的人，所以要让人感觉到你是一个可以为公司打江山的战士，而不是学生，从这种角度去书写。

③作品的排放顺序：最好的作品要放在最前面（所有的信息都一样，最重要的放最前面）。

④不要写太多的转职经历，当你应聘的是工程师岗位，招聘者其实不会太关心你以前在旧行业实际做了多少事情。而要多强调的是你 rails 开发的相关内容。

⑤关于技术博客：列出最好的三篇文章，附上标题和链接。

⑥要强调自己做的作品，而不是学到的东西。

⑦小细节类：

- 如果要写类似参与了“xx、xx、xx 三个项目”，最好写参与了从小到大的三个项目，各是什么，这样会给人一种在不断进步、成长的感觉。
- 不要写对什么东西“感兴趣”，要写“狂热”。
- 面试时可以聊 landingpage，把 4p 背下来。

和老师聊天的干货感悟

①关于什么时候该去理解知识？

当你觉得需要理解时就是不理解的时候，当你觉得不需要理解时其实就已经理解了。（很绕，但是对的，懂的人自然懂）

②关于好奇心与肌肉记忆

一般人会说好奇心是很重要的，人一定要保持好奇心。但其实在学习一项新东西的时候，抑制住自己的好奇心是最重要的。把东西变成肌肉记忆，变成自己的直觉，这才是最重要的。其实，编程、内容制造这些东西没没什么神秘的，归根到底都是手艺。

之所以要将其变成直觉，是因为只有当你把这些东西变成直觉以后，你的大脑才能腾出空间去思考更重要的事情。举两个例子：一个是足球比赛时，如果一个球员需要去思考怎样控球，怎样让球按自己想要的方式移动，那他就没有办法腾出脑子去真正分析场上局势，没有办法腾出脑子去寻找好机会，要做到这些，一定要他将控球这些技能变成直觉才行，要“人球合一”才可以。

第二个例子就是阅读英文文章。如果你一直在查英文单词的意思，那同样道理，你的脑子都用在了查单词上，你是根本没有办法理解整篇文章的内容的。

所以正像老师说的，你要步入下一个阶段，就一定要把这阶段的东西变成自己的肌肉记忆，内化成自己的直觉。

③求职时帮企业做 landing page 意义

求职时帮企业做 landing page 会让企业觉得你欣赏他们的公司，对他们的公司有深入研究了解，认同他们的价值观。企业在应聘程序员时，有时是把**对价值观的认同看的比技术水平更重要的**。所以，求职时帮企业做一个 landing page 是很有必要的。

④高效阅读英文文章的一个好方法

先通过 google 翻译把英文文章翻译成中文（虽然很多地方是不通顺的，但没有关系，那不重要），快速把译文浏览一遍，这样就可以大致把握文章要讲什么了。然后再去看英文原版，这样就不会陷入一些不懂的语言坑中，导致不理解文章在说什么。因为看文章的根本目的是获取信息与知识，所以语言细节真的不重要。

⑤奖励的套路（重赏+限时）

分析老师的悬赏写心得总结的套路：

还多人其实是很想写心得总结的，但是因为懒惰，所以很多人最后没有写。还有很多人写的心得总结是支离破碎的，质量不高。要解决这两个问题，办法就是采取重赏去 push 一把。这样，可以提高写作积极性和心得总结的质量。

限时也是一招，因为只有限时，才能在特定时间内充分调动积极性，如果时间很长的话，大家对奖赏的感知也会渐渐变弱。限时以三天左右为佳，有奖征文（心得）发布时间以学员课程临近结束且自信心爆棚的时机

为最佳。

最后，这些心得就是最好的营销内容了。所以细想，这是个十分节省成本的好套路。

[fullstack-course 第七周周记](#)

本周主要做了三大功能

①用 jQuery 实现点赞和收藏

因为自己技术实力不够，点赞基本是学霸帮忙做完的。收藏是理解原理后照着学霸的代码模仿着写出来的，但实际上自己没有掌握。这部分需要继续自学 code school 的 jQuery 的教程来补充自己的知识。

②评论功能

因为不好 jQuery 技术，所以评论功能做了部分妥协，编辑框没有直接放在页面底部，而是需要跳转一个新的页面才能查看。以后可以尝试用 jQuery 去再优化，实现异步加载。

③文章公开/隐藏功能

经验教训

本周因为个人原因（睡眠不足和身体不适）在周四和周五都不是很有精神，几乎没有做事情，导致效率低下。因此，保证身体健康和呵护住自己的精力必须放在所有其他事的优先级之上。否则，得不偿失。

学到的东西

学到的东西（包括前几周）已经用尽洪荒之力写在全栈班的心得里了。（[全栈班心得](#)）

[fullstack-course 第七周最大的坑/最棒的工具](#)

这周其实没有遇到特别的坑，但却让我发现了一个前几周掉进去的坑。

之前在做页面布局设计的时候，我们的做法一直是根据自己感觉做一点调试一点，这样效率极低，而且没有全局观。直到那天知道了设计哥的小组先画出设计稿，在根据设计稿进行布局的方法才恍然大悟，琢磨出做基础前端的简单流程。这也可以算是本周掌握的最棒的工具吧。

高效前端流程：

①先用 ps 或 sketch 画好效果图（如果要用 bootstrap，可以下载 bootstrap grid 的模板，这个很重要！！！！根据 grid 的布局格画好效果图，可以极大的简化前端的工作量）

②套用 bootstrap，照着效果图和 grid 的分布样式写前端

③进行效果微调

[凤凰涅槃，浴火重生——全栈班心得](#)

当得知老师让我们写一篇全栈班的心得感想时，我发现这是我一直想做而还没做的事，课程快结束了，是时候梳理一下自己这两个月来学到的东西了。首先映入我脑海的就是八个字“凤凰涅槃，浴火重生”。所以我把文章分成了三个部分——浴火篇、重生篇、涅槃篇。

浴火篇

来上这个课之前，我就做好了死磕到底的准备。开始上课才发现，这个课确实挺折磨人的，但磨人的地方更多不是在体力上的累，而是无知的痛苦和团队协作的不适应。

刚来上课时，虽然已经做过 4 遍课前作业，不过还是会发现自己什么都不懂。HTML 是干嘛的？CSS 又是什么鬼？ruby 和 rails 是什么关系？这些问题不断困扰着自己。不过还好我一直遵从老师的方法，放下自己的傲慢，先通过大量练习，让编程变成自己的肌肉记忆。而没有试图让自己去弄清楚为什么，因为一开始什么都不懂，如果试图通过查资料去弄懂知识，势必会让自己陷入用未知解释未知的怪圈。不过总的来说，前三周的自己还是会时不时陷入想要弄清原理的好奇中，而自己也在不断挣扎，克服这种种的错误的观念。后来第四周开始做项目后最大的挑战可以说是来自团队协作。因为大家都是有想法、有能力的人，所以对项目的看法难免会产生分歧，这也这是现实生活中会遇见且必须去解决的问题。我们一开始也会吵架，会对网

站的架构、事情的优先级、网站未来的发展方向不断提出自己的看法，不过后来这些问题在沟通后都能解决。如何沟通才是最重要的，我在遇到团队协作问题时也进行了反思（[项目协作的一些问题和解决办法](#)）。

重生篇

这是我想重点写的部分。

我觉得两个月来，就我自己而言，最明显的就是成长的纵深度和速度有了极大的飞跃。这里分知识和方法两部分来阐述。

知识

两个月，我发现自己真的完成了程序开发的入门。虽然我离全栈工程师这个目标还有一定的距离，不过我已经可以自己做后端的功能开发和前端的页面设计了。我一直记得笑来老师《斯坦福大学创业成长课》这本书中第一章第一节的标题“能做出完整的作品”。因为世界上大多数人不具备做出自己的作品的能力的，而通过 Ruby on Rails，可以用最短的时间做出最小可行性产品的功能。基础的 HTML 和 CSS 知识可以帮助我做出最小可行性产品的基础页面。两者结合，我觉得在产品开发的路上，我自由了，虽然离卓越还有距离，但至少我实现了自己最初来班级的初衷——“做出自己的产品”。

还有一个重生的领域是项目管理。两个月来，我真正见识了老师所说的硅谷最先进、高效的项目管理方式，并且真正把它运用到了自己的项目中。以前也经常在一些讲互联网的书籍里看到一些关于 MVP、迭代思维、Standup Meeting 的概念，但始终没有应用到实际中，所以也没有给自己的工作带来怎样的改变。而从开始自己做项目的第四周开始，xdite 老师带着我们每天用这种流程管理自己的项目，从每天的 Standup Meeting 确定前一天工作进度、明晰当天工作目标，到工作中运用项目管理工具和 github 实时管理自己的项目，再到每天工作结束后录制自己产品运行的 GIF 图来保证当天的产品有所迭代且保证产品的完整性，最后还有每周五的分享会。在一个多月的运用中，这一套流程已经印在我的脑海中，我也不断在对这些流程进行再思考，让这套先进的管理流程可以更好的应用到我未来的工作中。

还有一个不能不提的领域——Growthhack。作为一个曾经的广告人，Growthhack 这套方法革新了我对于营销的认知。由于 Growthhack 体系太庞大，以后的文章里再来慢慢总结，我这里想先说下 Growthhack 里对我影响最大的一个理论——LandingPage。xdite 老师在第四周刚开始的时候教我们 Landing Page 的理论，包括 Landing Page 的五结构和 4P(Picture/Promise/Prove/Push)理论，然后现场让我们用一个小时做了一个 Landing Page 页面。这是我用一个小时多一点的时间做出来的。



说实话，看到自己作品的时候我是震惊的。因为不管是站在消费者的角度还是曾经的广告人的角度来看这个页面，我都会觉得这个页面是极具吸引力的，如果产品做出来，是有很大的可能性通过这个页面把用户者吸引过来使用的。（[Landing Page 制作要点](#)）反思一下自己过去一年竟然也没学到这样的功力，惭愧的同时也不免感到兴奋，因为通过这个事情我明白了这个道理，学习的效果不止取决于你的学习时间长度，更取决于方向是否正确，是否有实战经验丰富且总结概括能力强的人真诚地为你讲明方法。Landing Page 给我的启发不止于做营销方面，还有对社交、求职、写作的许多启发，以后再写专门的文章来阐述。

方法

一开始的时候，老师就分享了一篇文章给我们（[混乱是学习的常态](#)）。这篇文章讲了一个我在两个月的学习中始终坚持的观念——学习其实不是线性增长，学习的真实状态是像拼图一样一块一块拼起来的。所以混乱才是学习的常态。就像拼图一样，如果一个人没有看过苹果是什么样，你告诉他苹果是什么样对他完成拼图是没有任何帮助的，最好的办法是让他克制住自己的好奇心，尽管一开始脑子是混乱的（这是常态），但最重要的是用尽各种办法把苹果拼出来。对技能的入门学习也应该是这样。

第二个观念是学习一定要放下自己的傲慢。（[放下你的无效学习方式](#)）三体里有句话：“弱小和无知不是生存的障碍，傲慢才是。”在学习中也是。所谓学习，要么就是整理总结自己过往的知识形成新的认知，要么就

是向更高一级的人学更高一级的方法，也就是传说中的“高手指路”。其实很多人也都经历过“高手指路”，但他们并没有得到很大的成长。究其原因，就是因为他们没有放下自己的傲慢，用自己的低一级的认知去揣测高一级的认知，还美其名曰：“批判性的接受” ([关于批判性接受](#))。最终他们学到的就只能是阉割版的技能，甚至最惨的是根本什么都没学到（因为他们“批判”完后可能什么都不做了）。所以，放下傲慢是打开学习之门的的第一步。

在放下傲慢之后，老师还一直向我们灌输一个方法，“学编程最好的方法就是不管怎么样，先至少练三遍，将之变成自己的肌肉记忆。”因为技术类的东西很多时候是熟能生巧，一开始学不会其实不是因为基础不够牢固之类的，而是“想的太多还动手太少”。所以编程其实没有想象的那么神秘，也没有想象的那么高深晦涩，最好的办法就是通过反复的练习，将其固化成自己的肌肉记忆，把编程练成直觉反应，自然“兵来将挡水来土掩”。

我这两个月其实就是在不断贯彻、践行这三个学习方法的过程。

涅槃篇

老师给我们的项目管理规范里面有一句话，我一直记着，就是“为自己的产品感到骄傲”。所以作为两个月学习的总结，一定不能少了我们努力做出的作品。



这是我们 4 个人的团队用一个月的时间从零开始打造的一个网站，名叫 GrowthHack 黑板报。 ([GrowthHack 黑板报](#)) 作为一个提供优质 GrowthHack 内容的网站，它拥有几乎所有内容型网站该有的功能：文章的排版、发表、管理，搜索，点赞，收藏，评论，用户反馈等。

在网站架构上，它在主页会筛选出最新的文章供用户学习。为了提高用户检索优质文章的效率，我们根据阅

读量的排行为用户开辟了一个热门文章的板块，可以方便用户直接看到最优质的文章。我们还提供了往期合辑，这样用户就可以像到图书馆查看周刊一样获取每期的内容了。未来，我们还会根据用户的需要新增论坛、招聘（招聘的后台功能已经写完，随时可以根据需要上线）、Growthhack 工具等功能。

网站当然还有跟多 bug，这是不可避免的，我们会不断根据优先级筛选出最重要的、对用户影响最大的一些 bug 来修改。不断根据根据用户反馈优化我们的网站。

笑来老师说过，七年就是一辈子。我觉得每次升级自己认知的学习都是开启自己新的一辈子。希望自己每次成长都可以让自己凤凰涅槃，浴火重生。

[fullstack-course 9/8](#)

在 controller 里面写 `@post = Post.find(params[:id])`，出现如下情况，报错为找不到 ID，log 显示如下

```
Parameters: {"post_id"=>"1"}
```

```
User Load (0.2ms)  SELECT  "users".* FROM "users" WHERE "users"."id" = ? ORDER BY "users"."id" ASC LIMIT ?  [["id", 1], ["LIMIT", 1]]
```

```
Post Load (0.2ms)  SELECT  "posts".* FROM "posts" WHERE "posts"."id" = ? LIMIT ?  [["id", nil], ["LIMIT", 1]]
```

从 `Parameters: {"post_id"=>"1"}` 可以看出实际需要传入的是 `post_id`，而不是 `id`。所以将 controller 稍作效果即可 debug，`@post = Post.find(params[:post_id])`。

[fullstack-course 9/7](#)

- 一个之前没有掌握的很初级的知识，关于 controller 和 view 的关系

app/controller/posts_controller.rb

```
def show
```

```
  @post = Post.find(params[:id])
```

```
  @issue = @post.issue
```

```
  @comments = @post.comments.order("created_at DESC")
```

```
  drop_breadcrumb(@issue.title, issue_path(@issue))
```

```
  drop_breadcrumb("正文")
```

```
end
```

app/views/posts/show.html.erb

```
<div>
```

```
  <h3>文章评论</h3>
```

```
  <div class="comment_inner">
```

```
    <div class="col-md-1 user_avatar">
```

```
      <% @comments.each do |comment| %>
```

```
        <%= image_tag comment.user.avatar.thumb, width: 30, height: 30, class: "img-circle" %>
```

```
    </div>
```

```
    <div class="col-md-11 reviews">
```

```
      <h3 class="reviews_name"><%= comment.user.name %></h3>
```

```
      <p class="reviews_time"><%= comment.created_at %></p>
```

```
      <p class="reviews_con"><%= comment.content %></p>
```

```
      <% end %>
```

```
    </div>
```

①只要在对应的 controller 里面已经定义的变量，view 里面都可以直接引用

② `<% @comments.each do |comment| %>` 这段写成 `<% @post.comments.each do |comment| %>` 也是可以读

出所有的评论的，但是这样则是通过 post 来读取，如果要限制 comment 的一些排序等，则会无效。
比如，在 app/controller/posts_controller.rb 里面写 @comments = @post.comments.order("created_at DESC"),
如果是 <% @comments.each do |comment| %> 能生效， <% @post.comments.each do |comment| %> 则排序不会生效。

- 刻意练习的三个特点
 1. 不好玩
 2. 用的时间少：专门培养天才的最好的音乐学校里，每天真正练琴的时间绝对不超过 2 个小时。没人能长时间坚持那样的强度，而没有强度的训练还不如不练
 3. 不追求快

- 一个新理念

还有很多东西自己还没有掌握，不要想着去让队友做什么，要 focus 在自己学什么。所以有觉得要优化的，值得优化的，不要管谁做的，尽管自己去把它优化，或者重新做出来也可以。多用学到的东西来增加自己的快感，多做那些自己稍微跳一跳就能碰到的东西。

- 有启发的两句话

①软件行业的加班文化是有其根源的，程序员花了大量时间来解决难题，而项目完成之后就把它一扔。下一个项目只是再次做这个循环而已。

②设计界有名的鸡汤句很应景：不会用工具时，工具是最重要的；会用工具时，工具是最不重要的。（程序界也一样）

- 老师荐物和推荐的音乐

程序员写代码三宝：mac 电脑、Aeron 椅子、Bose 耳机

写程序时最适合听的音乐：The Piano Guys

[fullstack-course 9/6](#)

昨天和今天学到的东西

- 怎么看 rails log

Started GET "/issues/6/posts/30" for ::1 at 2016-09-05 12:18:23 +0800

Processing by PostsController#show as HTML

Parameters: {"issue_id"=>"6", "id"=>"30"}

Post Load (0.3ms) SELECT "posts".* FROM "posts" WHERE "posts"."id" = ? LIMIT ? [["id", 30], ["LIMIT", 1]]

Issue Load (0.5ms) SELECT "issues".* FROM "issues" WHERE "issues"."id" = ? LIMIT ? [["id", 6], ["LIMIT", 1]]

Rendering posts/show.html.erb within layouts/application

SQL (1.0ms) UPDATE "posts" SET "pv" = COALESCE("pv", 0) + 1 WHERE "posts"."id" = ? [["id", 30]]

User Load (0.3ms) SELECT "users".* FROM "users" WHERE "users"."id" = ? ORDER BY "users"."id" ASC LIMIT ? [["id", 16], ["LIMIT", 1]]

Post Load (0.3ms) SELECT "posts".* FROM "posts" WHERE (id < 30) ORDER BY "posts"."id" DESC LIMIT ? [["LIMIT", 1]]

CACHE (0.0ms) SELECT "posts".* FROM "posts" WHERE (id < 30) ORDER BY "posts"."id" DESC LIMIT ? [["LIMIT", 1]]

CACHE (0.0ms) SELECT "posts".* FROM "posts" WHERE (id < 30) ORDER BY "posts"."id" DESC LIMIT ?


```
["LIMIT", 1]]
```

```
Post Load (0.3ms) SELECT "posts".* FROM "posts" WHERE (id > 30) ORDER BY "posts"."id" ASC LIMIT ?
```

```
["LIMIT", 1]]
```

```
CACHE (0.0ms) SELECT "posts".* FROM "posts" WHERE (id > 30) ORDER BY "posts"."id" ASC LIMIT ?
```

```
["LIMIT", 1]]
```

```
Rendered posts/show.html.erb within layouts/application (13.0ms)
```

```
Rendered welcome/_search_bar.html.erb (1.0ms)
```

```
Rendered common/_navbar.html.erb (6.0ms)
```

```
Rendered common/_banner.html.erb (0.5ms)
```

```
Rendered common/_flashes.html.erb (0.4ms)
```

```
Rendered common/_footer.html.erb (0.3ms)
```

```
Completed 200 OK in 153ms (Views: 144.1ms | ActiveRecord: 2.8ms)
```

有 SQL 那串代码，证明有写入数据库，可以根据这段来判断网站的某个动作写入了数据库什么东西，是否正确写入，也可以用于出现 bug 时 debug。

参考 w3chool 的 SQL 教程，可以知道 `SELECT "posts".* FROM "posts" WHERE (id < 30) ORDER BY "posts"."id" DESC LIMIT ?` `["LIMIT", 1]` 这种信息具体表示的信息是什么，可以知道是如何取到数据，如何写入数据等运作细节。

- 用 jQuery 写异步加载的点赞功能和收藏功能

思路：

①在 show 里用 jQuery 写代码，实现根据不同的回传信息显示不同的状态的功能

②在 controller 用 `message = {}` 的方式来重新写 controller 的内容，最后 render 到 json。示例：

```
def like
```

```
  message = {}
```

```
  unless current_user
```

```
    message[:status] = "e"
```

```
    render json: message
```

```
    return
```

```
  end
```

```
  @post = Post.find(params[:id])
```

```
  if !current_user.is_like?(@post)
```

```
    current_user.like!(@post)
```

```
    @post.support = @post.support + 1
```

```
    @post.save
```

```
    message[:status] = "y"
```

```
    message[:support] = @post.support
```

```
  else
```

```
    message[:status] = "n"
```

```
  end
```

```
  render json: message
```

```
end
```

- 如何更高效的来写前端（如何更好的把设计效果图和 css 排版结合）

高效前端流程：

①先用 ps 或 sketch 画好效果图 (如果要用 bootstrap, 可以下载 bootstrap grid 的模板, 这个很重要 !!!!! 根据 grid 的布局格画好效果图, 可以极大的简化前端的工作量)

②套用 bootstrap, 照着效果图和 grid 的分布样式写前端

③进行效果微调

- 非技术性知识

①写作的另一种方式与思维: 可以通过别人和自己聊天激发自己的一些思路, 把聊天内容整理出来, 就是一些干货了。两个重点, 一是提对问题, 二是要利用录音或录影设备记录下聊天, 转化为文字 (可以找别人代劳), 在进行整理。

②实验一个新思路: 晚餐少食少盐少油, 理论上这样可以让自己第二天的精力更加旺盛。

ps: 精力管理的思路

饮食 (过午少食)

睡眠

情绪

运动

不断学习新知识, 不断成长

③关于代码书写习惯的一个重要观念: 好的习惯是为了更好的和未来的自己协作。

④学到了很多超棒的求职和写简历的方法技巧。写简历的核心就是写一份以个人为对象的 landing page, 把自己当成一个产品。其实 landing page 和 onboarding 的思路可以迁移到生活中很多方面, 求职、社交、工作等等, 多想想这些方法还能用在哪些方面。

[fullstack-course 第六周最大的坑](#)

第六周的坑应该是自己的性格缺陷。

逃避关键问题, 转而关注细枝末节的事情, 用投入的时间 (努力的假象) 欺骗自己。导致效率下降, 项目无法很好的推进。

[fullstack-course 第六周最棒的工具](#)

chrome 和 firefox 的 inspect

在做前端排版时, inspect 超级重要。可以看到所有元素, 也可以在 inspect 里面直接修改看效果。

chrome 版的 inspect 在使用时可以把标尺打开, 更适合定位。而且 chrome 版的 inspect 有显示手机版网页的功能, 而且有大多数主流手机的屏幕尺寸, 在手机页面调试时很好用。

firefox 的 inspect 在页面上会自动画出参考线, 在页面排版时很好用。

[fullstack-course 第六周周记](#)

这周主要做了什么

这周主要对网站做上线前的最后一些细节的调试, 和上周一样, 主要还是一些 CSS 样式的调整。

这周我自己的问题是什么

这周发现了自己在做事过程中一个挺大的问题, 就是会逃避重要的事, 而挑一些无关痛痒的、较为轻松的事来进行。以做网站为例, 其实页面优化过程中最重要的是文章的页面呈现和主页的排版, 但自己一直没有想去碰这两块内容, 导致项目的推进效率其实一直是不高的。直到最后两三天, 才真正大家坐下来, 把需要改的点一个个抠出来, 一个一个, 去解决掉。如果自己早点做这个事的话, 项目的推进应该会更好。所以, 逃避问题其实是最大的浪费时间的黑洞, 分解问题, 各个击破才是正道。

这周还发现了自己一个很大的缺点。就是我自己很喜欢 (“擅长”) 对事情评价、提意见看法、告诉别人怎样怎样才是更好的, 却也只是停留在提出看法上, 自己也没有提出解决方案或者自己其实也不会。这种 “发号施令”, “指导别人” 其实只是自己欺骗自己, 自己满足自己虚荣心的手段, 因为自己没有思考, 没有动手去做, 所以能力其实没有增长, 更像是一种夸夸奇谈。

我觉得我应该采取这样一种方法来改正:

- ①向别人提看法时，衡量一下自己到底是为了解决问题还是为了满足自己的虚荣心
- ②尽量少提一些自己没做过的事，因为自己没实现过其实是没办法准确衡量难度的，所以这样的意见其实也是价值不高的
- ③遇到问题，动手解决，解决后再提建议。动手！动手！动手！！

ps：这周还学到了两个新认知

1、平台的核心作用应该是筛选、引导

现在做平台的企业很多，平台的核心作用应该是筛选、引导。平台一开始可能只是利用各种手段聚拢用户，但作为平台可以体现其价值的事情并不多，平台做的最多的是让用户在自己的平台上完成一些事情，服务好用户。一个真正优秀的、有自我价值的平台，一定是能帮助用户进行筛选信息、进而甚至引导用户。因为从用户的角度来看，来到一个平台，最希望的就是可以更方便的解决自己的问题。互联网可以通过各种信息解决用户的问题，但信息爆炸也是互联网的弊端。而平台起的作用就可以是帮助用户从海量信息中筛选出真正对自己有用的东西。

其实不管是电子商务平台、还是知识提供平台，本质都是一样的，筛选出用户需要的信息，帮助用户解决问题，在用户信赖平台后，进而开始引导用户行为。

2、认知升级新方向

新一代巨富都是全球公民，甚至可以说他们自己就是一个国家。他们教育全球化，生意全球化，身份认同也是全球化的，利益在全球。这也是我们认知升级的一个方向。

[fullstack-course 9/4](#)

jQuery 中 parent、parents 和 closest 的区别

- 1.parent()方法从指定类型的直接父节点开始查找，parent()返回一个节点。一级一级向上找。
- 2.parents()方法查找方式同 parent()方法类似，不同的一点在于，当它找到第一的父节点时并没有停止查找，而是继续查找，最后返回多个父节点。
- 3.closest()方法查找时从包含自身的节点找起，它同 parents()很类似，不同点就在于它只返回一个节点，使得代码量减小。可见，closest()方法在项目中的使用频率是比较大的。

[fullstack-course 8/31](#)

感悟：学写功能 code 的另一种思路

不止是自己写才能学习，看别人的代码去找茬，帮别人改代码，或者是看老师帮别组改代码也是可以学习。

width 和 margin 的排版思路

navbar margin 不能固定的时候，可以转变思路，固定 width，保持 margin 是 auto 的。

CSS 中规则@media 的用法

@media 可以让你根据不同的屏幕大小而使用不同的样式，这可以使得不需要 js 代码就能实现响应式布局。

CSS 写法规则

在 CSS 中，.btn-primary.search{}（中间无空格）和.btn-primary .search{}(中间有空格)的含义不一样，前者（无空格）表示无层级关系，对应的是 class="btn btn-default search"，后者（有空格）则是有层级关系，对应的是

```
<div class="btn-primary">
  <div class="search">
    </div>
  </div>
```

.btn-primary .search{}等价于.btn-primary > .search{}

图片插入链接的嵌套写法

```
<a href="% issue_post_path(hot_post.issue,hot_post)%"><%= image_tag hot_post.image.hot %></a>
```

Inline element & Block level element

HTML (超文本标记语言) 元素大多数都是行内 (内联) 元素 (inline element) 或块级元素 (block level element)。行内元素只占据它对应标签的边框所包含的空间。块级元素占据其父级元素 (容器) 的全部空间, 因而创造出一个"块"。默认情况下, 行内元素不会以新行开始, 而块级元素会新起一行。

常见的<h1>-<h6>,<p>,
,<hr>,,,,<div>都是块级元素, 占一整行。

常见的,<button>,,<a>是行内元素。

在 css 里面设置 display: inline-block 可以把会计元素转化为行内元素。

[fullstack-course 第五周/最棒的概念或工具](#)

atom 一些好用的设置和工具

Settings

Show Indent Guide (内建设定-程式碼對齊線)

Themes

seti-ui (檔案小圖示)

monokai-seti (程式碼配色)

Packages

atom-beautify 自動內容格式化(自動整理)

git-time-machine(图示化显示 git 前后对比, 超好用)

highlight-line (高亮選取該行程式碼)

highlight-selected (螢光筆功能)

webbox-color (css 碼底色會對應該顏色)

emmet (原名:Zen coding, 快速編寫元素程式碼 ex : ul>li*3 或 posa = position:absolute)

minimap (程式碼範圍總覽, 這個裝完才能再裝其他 highlight 外掛)

minimap-git-diff (修改過的程式碼, 如果沒有同步就會 highlight)

minimap-highlight-selected (跟 Aptana 的螢光筆功能一樣)

jshint (javascript 除錯功能)

[fullstack-course 第五周/ 最大的坑](#)

陷在手机版网页的排版里, 浪费了很长时间, 主要原因:

- ①没有对任务进行细分, 陷在未知的海洋里, 没有重点
- ②没有对思路进行整理, 茫然的在看各种文档, 其实不知道自己具体要做什么
- ③没有及时调整, 没有及时发现自己节奏被打乱并及时调整, 导致自己在一片茫然中不断受挫, 进而导致自己心态不好, 影响自己的精力, 导致整个人都很累。

[fullstack-course 第五周周记](#)

我梳理了一下我们这周的一些不足和应该采取的措施。

我们的不足

- ①项目没办法正常推进上线, 课程结束后有可能没办法拿出一个满意的作品
 - ②因为项目没有推进, 一直在一些细节里打转, 没办法学到新的东西, 可能实现不了学习更多技术的初衷
- 差距在哪里**

与别的队相比, 不管是功能还是界面, 暂时我们都是落后的, 所以当下重要的问题是如何实现弯道超车, 在三周后拿出令自己满意, 对得起自己的产品, 同时尽可能多的学到技术

原因是什么

个人原因

上周的项目进展十分不顺利, 有个人原因, 我上周生病了, 谢萍电脑挂掉了, 国峰出勤较晚, 组长有时会陷入对未来的畅想也会耽误一些时间。

团队配合

也有团队配合的因素，上周我们虽然制定了每天的任务，但明显四个人的执行力都不够。以我为例，周二开始做网页手机版，但因为对任务进行细分，没有对思路进行整理，没有方向，就在网上四处看别人的文章教程，导致自己陷在未知的海洋里，没有重点，不知道自己具体要做什么，致使自己在茫然中受挫，影响个人精力，且对项目推进的贡献几乎为 0。

进度脱节

除了执行力不够以外，我们每天在结束任务后没有正式的碰一下，过一下进度。这有两个不好的地方，对队友来说，因为没有这个会议，所以我们没办法知晓对方做了什么、什么进度，也没办法知道对方遇到了什么需要一起解决的困难。而对自己来说，因为不用向队友报告，所以没有压力在推动自己去推进项目进程。这是导致项目没办法很好的推进。

每天的 Standup Meeting 也是一个问题。我们经常带着各自不成熟的问题和想法在开会的时候直接讨论，这样有两个大问题。第一，这会浪费大量的时间。因为每个人都还没有对自己的想法进行总结整理就提出来，那大家势必会将时间浪费在理解这个“不成熟”的想法上，这是没有任何必要的。第二，每个人都有不自觉捍卫自己观点的本能，如果这个观点是经过自己思考、整理、沉淀的那还好，但如果这个观点本来就是自己随便一想的看法，那浪费时间在对这样的观点的辩论上岂不是显得过于幼稚。“真理越辩越明”总是建立在大家是针对自己笃信的观点的基础上的。所以，每天开会前把自己的想法整理好在开会实在是很有必要的。

优先级问题

我们还经常会遇到的一个问题就是关于优先级的事。学东西和推进项目哪个更重要？（虽说两个东西是并行不悖的，但总有时候会遇到一些情况是必须做出抉择的？）比如在做一个功能时，自己发现盲点实在太多，是憋一股气研究个一天把它做出来，还是用 10 分钟时间请教已经掌握的人，先把功能做出来。我觉得两种方式无非价值选择不同，无关对错。但对我们项目现阶段来说，我觉得还是应该把推进项目放在首位。因为我们距离这个课程结束仅剩 3 周，我们每个人都希望能拿出一个令自己满意、能上线、有用户、甚至以后能发展良好的作品，可以说这就是我们剩下 3 周的“指挥官任务”。基于此，我觉得如果我们在遇到类似的问题，应该把最快速度推进项目放在第一位。

半成品问题

当然还有一个问题，就是我们到上周其实是没有交出一个成品的，我们仅仅交出了一个“貌似能动的尸体”。我记得敏捷管理的书里说过，如果用户要一辆跑车，正确的做法应该是，先给用户一辆自行车，然后给他一辆摩托车，然后给一辆普通汽车，最后给了他一辆跑车，甚至后来给了他一架飞机。这个方法论我们没有很好的践行。我觉得最大的原因还是在于我们早会没有规划好，下课前没有碰进度，所以导致大家的项目进度近乎失控，更不用谈在作出一个完整的作品了。

我们该怎么办

1. 严格遵照老师的理念，我摘取了两段话，和我们目前的情况比较契合。

①“【规划】其实绝对不是开发一个网站的最重头项目。【施工】和【调整】才是。不要 overthinking，因为专案中最高的原则是 get things done, not over design。”

②“因为我深信：长期处在失火/救火的环境下，会快速减低一个团队的战力。

热血的投入通常会让人有假象，我投入的工时越高，成果会越好。事实上这是一个彻底的伪命题。而创业初期的不稳定，忙碌，失火，更让你会有只要我努力加班，一切就改善的错觉。肾上腺素最多只能让你撑三个月，接下来一切都会破滅的。作一个网站要到可以出场，大家比得是命长，而不是 Startup weekend 冠军。”

2. 严格执行老师的方法

①每天早上 9 点半开 startup meeting，开会前每个人把**昨天做了什么事、遇到了什么坑、今天要做什么事、需要和谁对接什么**用文字的形式写在 Tower 的文档上，确保自己的想法是**成熟、可表决的**。

②在 Tower 上开了一个 issue tracking system 的文档，每个人对网站的优化有什么建议，直接以文字或截图

的形式放在文档里，开会时确定是否实作。如果实作，将任务进行拆分。

③每天中午吃饭时，过一下早上做了什么、遇到了什么困难、有没有陷进什么泥潭里出不来，大家一起解决。

④17:00 统一制作做出成品，并由国峰进行 merge。如果手头有还没做完的新功能，则在这一轮的 merge 暂时放进去，留待晚上做完 merge。具体原则就是：当天产品的功能或界面的美化上不封顶，看个人意愿，但是必须保证 17:00 的产品是成品，而非“尸体”。

⑤merge 完成后开一次短会，过一下今天的工作量，明确一下项目的进展。

ps：每天两个确认的时间点，9:30 和 17:00，没有准时的请大家喝下午茶或吃好吃的，当然也是上不封顶 O(∩_∩)O 哈哈~

[fullstack-course 8/24](#)

今天没有记什么技术日记，就写下早上吃早餐时的想到的一点东西吧。

早餐的时候看了这篇文章

(http://mp.weixin.qq.com/s?__biz=MjM5NTE5NzUwMA==&mid=2650978049&idx=1&sn=6ec579d2d9a67ee5f7e282fbb544edcd&scene=1&srcid=0824ECmBcnj2FARQd9sSK8jj#rd)，有一个感悟。

有压力会有动力，但还会有其他负面的东西。而动力并不一定要压力才能产生，比如对某个观点的笃信也能产出动力。所以我们应该去寻找更健康、更优质，同时能产生动力的方式。

[fullstack-course 8/23](#)

感悟

反思昨天：

昨天一天都陷在手机版网页的排版里，有几个原因需要反思：

- ①没有对任务进行细分，陷在未知的海洋里，没有重点
- ②没有对思路进行整理，茫然的在看各种文档，其实不知道自己具体要做什么
- ③没有及时调整，没有及时发现自己节奏被打乱并及时调整，导致自己在一片茫然中不断受挫，进而导致自己心态不好，影响自己的精力，导致整个人都很累。

所以——

节奏感很重要。

事情要一步一步做（像做数学证明题一样），脑子不能乱，不要慌，保持自己的节奏。

关于聚焦：

除了之前的日记里提到的聚焦的思考，聚焦的还有另一个重点是放弃。放弃一些现阶段不需要做的功能，放弃脑袋里想到就想马上去做的冲动，把想做的东西都先记下来，控制住自己的注意力，全力聚焦。

Q&A

用手机访问 localhost:3000

- ①找到自己的 ip（设置 - Network - Advanced - TCP/IP，查看 IPv4 Address 即本地 IP）
- ②rails s -b XXX.XXX.XX.XX(本地 IP)
- ③在电脑或手机的浏览器输入 XXX.XXX.XX.XX:3000 即可访问

制作 markdown 思路

- ①用 gem 'redcarpet' 做出 markdown to html，挂在需要显示最终效果的页面上
- ②用 "simplemde"（markdown 编辑器）做编辑器，挂在编辑内容的页面上

[fullstack-course 8/22](#)

Q&A

如何调用 API？（以 increment_counter 为例）

阅读量是需要写入资料的，这种情况要在 model 里面定义

从 ActiveRecord::Base 可以看出是在 model 里面应用的（model 的 class 是 XXXRecord，controller 则是 XXXController）。

Example 里面可以看出怎么用。

具体实现代码

```
def visit
```

```
  Post.increment_counter(:pv, self.id)
```

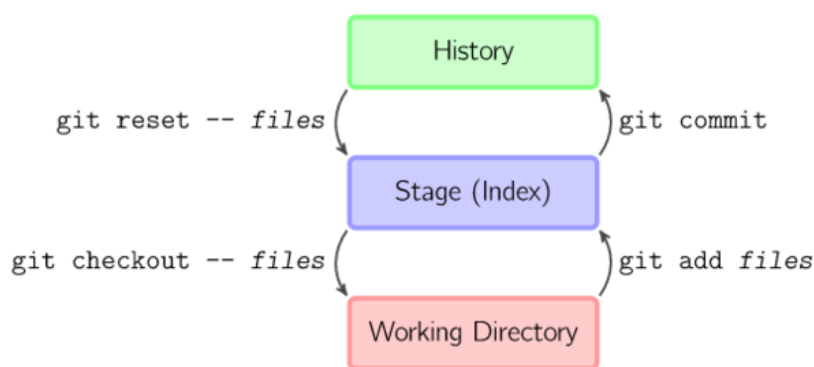
```
end
```

increment_counter 是调用 API, self.id 可以实现每篇 post 都对自身进行统计 (简单说, 可以实现统计所有 post 的功能)。

如何通过 git 命令丢弃本地修改 ?

git clean 可以删除一些没有 git add 的文件, git clean -df 删除文件和目录。

git checkout — files 可以把文件从 stage 复制到 working directory, 用来丢弃本地修改。git check .对所有文件执行命令。



手机屏幕的导航栏怎么做 ?

application.js 里面添加 `// = require bootstrap/collapse`, 可以做出手机屏幕导航栏的效果。

ps : getbootstrap.com 里有 bootstrap 能提供的所有的功能, css、js 等

[使用 ransake 写搜索功能——fullstack-course 第四周/最棒的概念或工具](#)

使用 ransake 写搜索功能

以 jobs 的搜索为例

step 1:

修改 views/jobs/index.html.erb, 加入如下代码

略

```
<%= render :partial => "jobs/search_bar" %>
```

略

新建一个 search_bar, touch app/views/jobs/_search_bar.html.erb, 加入如下代码

```
<div class="row">
```

```
  <div class="col-sm-9 col-lg-8 col-lg-offset-2">
```

```
    <%= form_tag "#", :method => :get, :class => "jobs-search-form" do %>
```

```
      <div class="input-group">
```

```
        <input type="text" class="form-control" name="q" value="<%= params[:q] %>" placeholder="搜索工作关键字...">
```

```
      <span class="input-group-btn">
```

```
        <button type="submit" class="btn btn-default">
```

```

        <span class="glyphicon glyphicon-search"></span>
      </button>
    </span>
  </div>
<% end %>
</div>

<div class="col-sm-3 col-lg-2">
  <%= link_to '发布一份工作', "#", :remote => true, :class => "btn btn-primary btn-block" %>
</div>
</div>

```

step2 :

安装 gem 'ransack'、gem "will_paginate"和 gem 'seo_helper','~> 1.0' (具体 gem 文件可 google), 执行 bundle install

step3 :

修改 jobs_controller.rb, 加入如下代码

```

def search
  if @query_string.present?
    search_result = Job.ransack(@search_criteria).result(:distinct => true)
    @jobs = search_result.paginate(:page => params[:page], :per_page => 20 )
  end
end

```

protected

```

def validate_search_key
  @query_string = params[:q].gsub(/\|'|V|\/|"/, "") if params[:q].present?
  @search_criteria = search_criteria(@query_string)
end

```

```

def search_criteria(query_string)
  { :title_cont => query_string }
end

```

step4 :

修改 routes.rb, 加入 search

```

resources :jobs do
  collection do
    get :search
  end
end

```

step5:

新增搜索页面，touch app/views/jobs/search.html.erb

```
<%= render :partial => "jobs/search_bar" %>
```

```
<br>
```

```
<div class="search-result">
```

```
  <table class="table table-bordered">
```

```
    <% @jobs.each do |job| %>
```

```
      <%= link_to(job.title,job_path(job)) %>
```

```
      <hr>
```

```
    <% end %>
```

```
</table>
```

```
</div>
```

step6:

修改 jobs/_search_bar.html.erb 里的路径

略

```
<%= form_tag search_jobs_path , :method => :get, :class => "jobs-search-form" do %>
```

略

step7 :

制作搜索的 highlight 功能。

在 app/helpers/jobs_helper.rb 里添加如下代码

```
def render_highlight_content(job,query_string)
  excerpt_cont = excerpt(job.title, query_string, radius: 500)
  highlight(excerpt_cont, query_string)
end
```

app/views/jobs/search.html.erb 里修改代码

```
<%= link_to(job.title,job_path(job)) %>
```

改为

```
<%= link_to(render_highlight_content(job,@query_string),job_path(job)) %>
```

[Tweet](#)

[fullstack-course 第四周/最大的坑](#)

在项目进展过程中，如果遇到命名错误最好直接删掉重建，不要去修改。

因为修改的东西又多又杂，还容易出错，会花费更多的时间。

[fullstack-course 第四周周记](#)

项目管理

这周 xdite 老师开始转变身份，从老师变成了我们的项目经理。通过一周的观察，我学到了一些项目管理的方法。

- 第一天到第四天：

项目开展流程：

早上开小会，时间在 5-10 分钟（讲每个人昨天完成了什么，昨天遇到了什么困难，今天想做什么）。

开始各自工作，不断向前冲刺。

下课前把所有人的工作 merge 到 develop，并且录 gif 呈现工作成果。

背后的项目管理思想：

“早上开小会，时间在 5-10 分钟。”

压缩开会时间，要求每个人想好并且整理好自己的想法在参加会议。开会的目的不是讨论不成熟的观点，而是把各自相对成型的想法拿出来大家表决，或者把各自的想法进行整合。

这样做的好处是可以避免大家在会议过程中临场抛观点，抛出的观点可能是杂乱甚至未经思考的零散的东西。这些零散、未经思考整理的想法本来就不应该抛出来，但如果不仅抛了出来，甚至大家还针对这些想法去扯皮、讨论的话，则会议的效率会大大降低，容易出现一开就是几个小时的无效率会议。

团队表决，分配任务，开始行动。这是开小会的目的。

“开始各自工作，不断向前冲刺。”

我觉得这种将任务不断细分，然后针对每个任务专注、快速完成的方法就是敏捷管理的重要思路。

这个阶段我觉得有两个核心：“聚焦”和“死线”。“聚焦”和“死线”能帮助我们集中精力专注在 must have 的任务上，遇到新想到的任务可以在 Tower 上先记下来。只开需要的工作页面（chrome 只开需要的网页，atom 只开相关的页面），保证自己可以全力聚焦在当下需要完成的工作上，在借助“死线”带来的压力，让自己不至于分心。这样，可以让自己最高效的完成工作。

“下课前把所有人的工作 merge 到 develop，并且录 gif 呈现工作成果。”

每天的 gif 录制可以逼迫我们在做项目时每天都能给出 100% 的作品，虽然作品可能很 low，但一定是 100% 可以给用户使用的成品，不至于交出一堆无法使用的“尸体”。这个方法也会倒逼我们在提想法的时候，先聚焦在 must have，有额外的时间在做一些附加功能。不至于想到一堆看似完美的想法，在落地时却做不出实际可用的产品。

在项目管理，有一个比喻很生动。如果用户要一辆跑车，你想先造一个完美的轮子，再造一个完美的引擎，造成所以完美的部件拼凑完交给用户。如果是这种想法，那这辆跑车永远造不出来。正确的做法应该是，先给用户一辆自行车，然后给他一辆摩托车，然后给一辆普通汽车，最后给了他一辆跑车，甚至后来给了他一架飞机。

所以，每天的工作都把做出一个 100% 的产品放在第一位。正如笑来老师也说过的：“成长的起点就是做出一个完整的作品。”

- 第五天：

项目开展流程：

开小组会，总结当周做了哪些事情，有哪里做的不足，有哪些方向需要调整。做过总结后，在讨论下周要做什么，有哪些功能，哪些技术细节，最后要给出一个怎样的成品。

开小组分享会，讨论本周最棒的概念/工具，遇到的最大的坑。（也可以各自写在博客里）

如果有一些想自己玩看看的新功能，也可以在周五玩一玩。因为在前四天为了保证“主干”的顺利搭建，可能没有时间做一些额外的功能。

背后的项目管理思想：

经过四天每天高速的“奔跑”，每个人都会疲惫。这时候可以慢下来，讨论一下本周的进展、得失，调整方向，为下周做计划。

- 总结：

这种 4+1（每周 4 天“狂跑”，1 天总结）的工作方式，工作时长可能不长，但效率确实惊人。因为高度的聚焦和专注，可以保证项目顺利开展。有张有驰的时间安排，能让人在快疲惫的时候慢下来，休息的同时调整方向，为下周新一轮的“冲刺”做充分的准备。

感悟

之前看笑来老师写过“做到虽然不能完全读懂，但确实能够完整读完”，这也是一种很强的能力。今天在做 css

的练习，突然发现写代码其实也是这样。

一个月前，很多 css 的语言我完全不知道什么意思（现在也很多不知道什么意思），但我还是写了很多 css 代码，完成了几个完整的编程作品。我做到了“虽然不能完全读懂，但确实能用上”，然后后来我在去做这些 css 的教程才真正懂了很多代码的含义。

除了“不懂先用，且做出完整作品”这个方法论外，我还发现学习顺序真的很重要。我们可以做一个简单的数学计算。如果一开始什么都不懂，我就去研究 css 语言的含义的话，我可能要花掉 20 个小时才能懂的一点点东西（数字乱说的，为了表达意思），然后懂了以后，我还是要花 10 个小时去用，我才知道遇到特定情况怎么选用具体代码。这一共花费 30 个小时。

如果调换一下顺序，我一开始不管懂不懂，先抑制住自己的好奇心，先练习 10 个小时，大概知道什么情况该用什么代码，再去读一些 css 教程的话，因为有了前面使用的基础，我可能读个 5 个小时就基本能掌握个大概了。这一个花费 15 个小时。高下立判啊！

[Tweet](#)

[fullstack-course 8/18](#)

感悟

和优秀的人在一起与锚定效应

和优秀的人在一起耳濡目染的意义在于你锚定的标准会跟着变高，所以你自然也会变得更加类似。

当优秀的人不仅是书里或网络里触摸不到的人物时，你会觉的大家都一样是普通人，不会再觉得他们遥不可及，自然你也会去想自己要怎么变得和他们“类似”，那么你的 level 就上去了。回过头看以前和自己一个 level 的人，更会觉的自己进步很多了。

编码学习

早期不重规范，能写出来实现功能即可，重点是成就感

一定阶段后，必须开始重视规范

很多人的做法却是一开始就重视规范，然后各种碰壁，失去信心，然后放弃。所以在不同的阶段做对应正确的事是最重要的。

命名错误如何处理

直接删掉重建，不要去修改。因为修改的东西又多又杂，还容易出错，会花费更多的时间。

Q&A

如何更改 trix 编辑框的高度？

```
<%= simple_form_for [@issue,@post] do |f| %>
```

略

```
<p><strong>文章内容
```

```
<%= f.trix_editor :content, :input_html => { :cols => 5, :rows => 100 }%></strong></p>
```

略

```
<style >
```

```
trix-editor.formatted_content {
  min-height: 400px !important;
  width: 1000px !important;
}
```

```
</style>
```

用 css 修改语言修改。但是修改后可以在浏览器里发现并没有成功。用 inspect 检查发现是 trix 不允许直接

这样修改。这时可以用 `!important` 强制修改。

ps：可以用 `inspect` 不断检查是否可以用

[fullstack-course 8/17](#)

Q&A

gem annotate 安装

gem install annotate,然后 annotate

Issues 中包含 Post, controller 和 view 的写法

- controller-post

```
def edit
```

```
  @issue = Issue.find(params[:issue_id])
```

```
  @post = Post.find(params[:id])
```

```
end
```

edit 如果用 `@post = Post.find(params[:id])`，则点击编辑时会出现之前在 new 里新建后或上一次编辑后的表单。一般来说 edit 用的都是这种方法。

```
def edit
```

```
  @issue = Issue.find(params[:issue_id])
```

```
  @post = Post.new
```

```
end
```

edit 如果用 `@post = Post.new`，则点击编辑时是全空的表单。

- view-post

```
<td><%= link_to("编辑",edit_issue_post_path(post.issue, post), class:"btn btn-xs btn-primary") %>
```

(`post.issue,post`)指的是引用 `issue_id` 和 `post_id`。这里的 `post.issue` 等价于 `post.issue_id`，两种写法都可以。

seed 更新后需要执行的命令

```
rake db:reset
```

```
rake db:migrate
```

感悟

自己做专案最重要的是有条不紊开展工作，不要乱不要慌，每次只做现在正在做的，想到其他功能或发现其他欠缺的功能先记下来，回头再去做，这样才不会乱掉。

atom 也只开正在做的标签，其他关掉。

要让自己的脑子保持专注，不要注意力分散。因为再每个当下，注意力恒定，所以，呵护住自己的注意力才能最大化提升效率。

[Tweet](#)

[fullstack-course 8/16](#)

Q&A

heroku 更改网站地址后如何在本地同步修改？

方法：在 `iterm` 输入命令 `atom .git/config`，在 `atom` 里面找到旧网址，改为新网址。

push 到 heroku 的命令

```
git push heroku job-listing:master
```

如果不加当前分支，默认就是推本地的 `master`。而如果本地的 `job-listing` 改了，而 `master` 没改，则可能会报错。

mailer 里的格式问题

```
def notify_order_cancelled(order)
```

```
  @order = order
```

```
  @product_lists = @order.product_lists
```

```
  mail(to: "123@123.123", subject: "[JDStore] 有一个客户取消了订单! #{order.token}")
```

```
end
```

对邮箱的直接调用一定要加双引号 (""), 如果是 to: @user.email 这种情况则不用。

如何在 iterm 里通过命令直接回到过去的版本?

①git log 查看版本号 (ps: 推出输入 Q)

②git reset --hard 版本号 即可回到 commit 的某个版本 (ps: hard 后面带空格)

关于栏位犯的一个错误

description 是栏位名称, 栏位是 text。

如何使用 trix 编辑器?

使用方法: <%= f.trix_editor :title %>

详细参考: <https://github.com/maclover7/trix>

注意: f.input :body, as: :trix_editor 必须是使用 formtastic 这个 gem 的情况下使用, simple_form 不可以用这个方法。

如何通过正则表达式限制邮箱的正确格式?

```
validates :contact_email, format: { with: /\A[\w+\-]+\@[a-z\d\-\-]+\.[a-z\d\-\-]+\.[a-z]+\z/i, message: "请输入正确的邮箱格式!" }
```

[Tweet](#)

[fullstack-course 8/15](#)

今天学习了 landing page 的实际操作, 用了一个多小时做出了自己的第一个 landing page。而且效果还不错。

总结一下 landing page 制作的几个要点:

①采用 4P 原则撰写: 描绘、承诺、证明、敦促

②撰写的核心顺序: why——how——what

③撰写时注意的要点:

快速写出来, 一开始越 low 越好, 一定要让别人一眼就能看懂, 避免过分华丽的辞藻

不要写一些负面的东西, 要从正面切入

有没有产品不重要, 重要的是描绘一种让用户沉浸其中的场景 (产品是产品, 广告是广告)

[Tweet](#)

[fullstack-course 第三周/最棒的概念或工具](#)

本周学到的最好的概念是 growthhack。

着重在怎么用技术手段和技术思想来促进营销增长, 是一个很好的概念。growthhack 的核心是做好产品, 增加用户购买信心, 打消用户购买疑虑。基于核心思想有很多方法手段来辅助达成目的。最重要的是先对自己企业进行“监视”, 知道自己的具体现实状况。

[Tweet](#)

[fullstack-course 第三周/最大的坑](#)

- 关于.gitignore 隐藏的文件如何不显示和一个世纪大坑 如何不显示 :可以在 tree-view 里面设置 Hide VCS Ignored Files 让加在.gitignore 里的隐藏文件不显示 世纪大坑 :tree-view 里面的 Hide VCS Files 不勾选的话,在默认的 Atom Dark Themes 里面隐藏文件显示的是灰色字,而在 Spacegray Themes 里面却和其他公开文件的显示没有差别,是看不出来的 !! 一个让人欲哭无泪的坑。

[Tweet](#)

[delete_all 与 destroy_all 的区别与用法](#)

delete_all : 直接执行删除, 不查询数据, 不调用:destroy 方法。优点 : 过程快, 占用资源少。缺点 : 如果出现问题, 维护代价高。

destroy_all : 查出所有数据, 一条条删除, 调用:destroy 方法。优点 : 能终止和解决大多数相关问题。缺点 : 占用资源多。

[Tweet](#)

[fullstack-course 第三周周记](#)

JDStore 的功能

商城基本结构

- 建立前台和后台 : 实作 admin 后台 CRUD (额外 : carrierwave 上传图片)
- 会员系统 : gem "devise"
- 权限管理 : authenticated_user!/admin_required
- console 添加 admin
- 上传图片 : gem "carrier wave"/gem "mini_magick"/image_uploader/view

购物车

- 将产品加入购物车 : 实作 **【加入购物车】按钮 add_to_cart/ 实作 current_cart.add_product_to_cart(@product)**
- 购物车明细
 - 显示购物车明细 : 点击购物车按钮, 显示购物明细页面
 - 计算总价 : <% sum= sum+XXX> (因为逻辑写在 view 里太丑, 所以 Refactor 到 helper : sum += XXX) (因为算总价是 model 而不是 helper 的任务, 所以 Refactor 到 model:def total_price)
 - 库存为 0 的货物不能购买 (view 和 controller 里不能加入购物车)
 - 清空购物车
 - 删除购物车的某一样商品
 - 更改购物车内购买的数量
 - 购物车新增的数量不能超过库存数量
- 建立订单
 - 点击确认结账按钮, 显示订单页面 (包括购物明细和订单咨询——订购人和收货人信息)
 - 点击提交订单, 表单提交至 order
- 建立订单明细
 - 新建 model:product_list 来存储当时购买咨询
 - controller: def create 里建立订单明细的缓存
 - 订单明细页面
- 订单网址改为秘密 (self.token = SecureRandom.uuid)
- 使用支付宝结账

- 订单分为已付款和未付款（migration 添加 is_paid 栏位）
 - 记录订单付款方式（migration 添加 payment_method）
 - 实作点击后可以提示付款成功并记录状态的功能
- 已付款的订单，不可以再重复付款
- 寄送订单通知信
 - rails g 产生 mailer
 - mailer 里新增寄信功能（def notify_order_placed）
 - view 里设定订单内容通知信
 - 安装 gem "letter_open" 预览信件（触发方式：rails c : OrderMailer.notify_order_placed(Order.last).deliver!
 -
- 建立 account/orders 可以看到自己过去所有订单
- 部署 JDStore 到 Heroku
 - 部署 JDStore 到 Heroku
 - 在 Heroku 上也可以上传图片
 - 使用 figaro 管理密码 gem "figaro"（使用 AWS，切记数据安全，不要泄露 S3 金钥！）
- 订单状态管理（有限状态机）
 - 安装 gem "aasm"
 - 加栏位（add_aasm_state_to_order）
 - 设计架构
 - 建立 admin/orders 可以看到系统内所有订单
 - admin 的 order 列表能显示订单状态
 - 后台管理员可以【取消订单】、【出货】
 - 使用者可以【申请取消订单】
 - 后台管理员【出货】后，系统应该寄出通知信
 - 使用者【申请取消订单】后，管理员应该要收到【申请通知信】
 - 完善订单管理系统细节

理念

什么是好代码？

优雅、简洁、可维护性强这些都不重要，最重要的是能赚钱。能赚钱的代码才是好代码。

思考路径？

方向性的思考才有意义，对细节的思考意义不大，有时甚至浪费时间。

以 code 为例，思考如何做一个功能，如何利用代码的逻辑解决问题是有意义的。在初期阶段，细究每个代码的意思，陷入对细节的纠结是没意义的。前者最适合（不是“好坏对错”）的方法是理性思考，后者最适合的方法是背记下来。

所以，当不知道是去问“how”还是“why”的时候，先想想当前问题是方向问题还是细节问题。

网站优化

网站的优化不应该想到一个功能去改一个功能，这样容易顾此失彼，没有全局观。

应该先系统检查网站，把觉得该优化的地方列出来，然后一起改代码。改代码过程中遇到新的思路也先不要做，应该先写下来，第二轮优化再一起改。

每个阶段都要用对应的思维做事，而且各个阶段应该独立，本阶段绝不想下个阶段的事。（详见：《原则》）检查网站时应该不去想代码难度，用产品经理的眼光检查功能。code 的时候不要去想优化，全力把该修改的功能写出来。

Q&A

heroku 如何直接通过 rails console 加 admin ?

console

heroku run rails c

User.all

u = User.find(2) *找到 id=2 的 user

u.is_admin = true

u.save

rails c 报错时有时也需要重开 (如发送邮件这个操作)

为什么有的 method 要放在 ApplicationController 里面 ?

因为放在 ApplicationController 里面的 method 其他的 controller 都能用。

[Tweet](#)

[fullstack-course 8/11](#)

技术知识

- 在 heroku 报错如何查看错误 ?
可以在 iterm 里面用命令“heroku log”查看错误
- 如何用“作弊”的手法新建 admin ?
可以先在 migration 改栏位 (新建一个 migration, change_column, 记住不是 add_column), 将 is_admin 的 default 改为 true。这样新注册的用户就都是 admin 了。
之后再改栏位, 将 default 改回来。方法同上。
ps : 因为是部署到 heroku, 所以执行完操作后都要记得 push 到 heroku, 当设计资料库变更时, 还要运行 heroku run rake db:migrate
- controller 里面加不加@的问题
加@是要在 view 里引用的情况, 如果是只在 controller 里面直接调用, 不用加@
- 订单管理的 ! 和 ?
有限状态机里面的动作 (如取消) 都需要加!, 因为都是改变自身的状态。
如果是返回是 true 或者 false 的后面一定要加 ?
注意 :

①有限状态机的“取消”只能取消一次, 不能重复取消。

②有限状态机是有存在逻辑的, from...to..., 这个需要注意一下

- 发送邮件的一个问题
在 rails c 中, 发送文件的命令是这样的 OrderMailer.notify_order_cancelled(Order.last).deliver!, 而在 controller 里面要改一下。
- def cancel
- @order = Order.find(params[:id])
- OrderMailer.notify_order_cancelled(@order).deliver!
- @order = @order.cancell_order!
- redirect_to :back
- end

Order.last 改成了@order。理由 : 在 rails c 中, Order.last 是去取最近的一个 order, 而在 controller 中, 因为已经找到 order 并且赋值到变数@order 了, 所以直接引用@order 就可以了。

- 发送邮件的另一个问题
- `def cancel`
- `@order = Order.find(params[:id])`
- `OrderMailer.notify_order_cancelled_by_admin(@order).deliver!`
- `@order = @order.cancell_order!`
- `redirect_to :back`
- `end`

`OrderMailer.notify_order_cancelled_by_admin(@order).deliver!` 这行代码不能放在 `@order = @order.cancell_order!`后面，因为`@order = @order.cancell_order!`有一个取消订单的动作，所以不合逻辑。发出邮件这个指令应该直接跟在找到 order 并赋值到`@order` 这行命令下面。

- 加标签进行有限状态机管理订单的方法
最重要的是 `orders_helper.rb` 里面的这段代码。有两种方法可以使用（其实用 hash 也可以）。

第一种:

- `def render_order_state(order)`
- `case order.aasm_state`
- `when "order_placed"`
- `content_tag("span", "已下单", class:"label label-default")`
- `when "paid"`
- `content_tag("span", "已支付", class:"label label-default")`
- `when "shipping"`
- `content_tag("span", "发货中", class:"label label-default")`
- `when "shipped"`
- `content_tag("span", "已发货", class:"label label-default")`
- `when "order_cancelled"`
- `content_tag("span", "已取消", class:"label label-default")`
- `when "good_returned"`
- `content_tag("span", "已退货", class:"label label-default")`
- `end`
- `end`

第二种：

```
def render_order_state_if(order)
  if order.paid?
    content_tag("span", "已支付", class:"label label-default")
  elsif order.order_placed?
    content_tag("span", "已下单", class:"label label-default")
  elsif order.shipping?
    content_tag("span", "发货中", class:"label label-default")
  elsif order.shipped?
    content_tag("span", "已发货", class:"label label-default")
  elsif order.order_cancelled?
    content_tag("span", "已取消", class:"label label-default")
  elsif order.good_returned?
```

```
content_tag("span", "已退货", class:"label label-default")
end
end
```

[Tweet](#)

[fullstack-course 8/10](#)

感悟

- 追求做出完美的东西其实是一件不可能的事情。你觉得完美的东西其实可能就是高你 N 个 level 的作者眼中不完美的产品。所以，重点不是做出现阶段你觉得完美的东西（当然这本身也是不可能的），而是去向更好的层次攀登，品味高了，质量自然就高了。
- 看了得到上面万维钢的一篇文章《失败不是成功之母，成功是成功之母》里面讲到两个观点觉得不错。
 - ①“我的失败不是我的成功之母，但是别人的失败则可能是我的成功之母”（要注意，这里对失败的定义指的是大失败，会给人造成较大打击的失败，如项目失败、公司倒闭等。而不是训练过程中遇到的强反馈的小错误。）给我们的启示是，我们可以去看别人的失败，去体会那些错误，然后促成自己成长。最简单的方式就是去看同学写 code 遇到的错误，然后试着用自己的思路去解决那个问题。
 - ②“好的失败’应该满足三个条件：第一是及时。一旦不对马上就有人给你指出来。第二是对事不对人。你错了，下次改正过来就是，没有必要上升到‘你这个人行不行’的层面。第三是错误的代价很小”看到这三个条件，我的第一反应就是，卧槽，这不就是 code 当中遇到的 bug 的特点么？！反馈及时、对事不对人、错误代价极小。真是“好的失败”。
- 做事要快，心态要慢。

growthhack

今天老师讲了 growthhack 的知识，着重在怎么用技术手段和技术思想来促进营销增长，是一个很好的概念。growthhack 的核心是做好产品，增加用户购买信心，打消用户购买疑虑。基于核心思想有很多方法手段来辅助达成目的。最重要的是先对自己企业进行“监视”，知道自己的具体现实状况。在表现上最重要的则是掌握 landing page 的结构。具体怎么作品 landing page 老师下周一会教，期待自己也能一个小时写出一个 landing page。

ps:growthhack 更详细的知识可以参考老师的书籍和今天同学录制的视频。

技术知识

- 在建立“我的订单”时学到的两个知识：
 - ①双层循环关系
- <% @order.each do |order| %>
- <% order.product_lists.each do |product_list| %>
-
-
- <% end %>
- <% end %>

这行代码的意思就是在捞出 order 数据的循环里在嵌套一个捞出 product 数据的循环。

之所以可以这么做，是因为 order 这个 model 有和 product_list 这个 model 建立联系（即 order has_many :product_lists/product_list belongs_to order），所以可以在循环里嵌套。

用 `order.product_lists` (复数) 是因为表单名称是复数。

这样做的好处就是我可以 在一个 `views` 里面同时捞出 `product_list` 和 `order` 的所有数据为我所用。关键点就是两个 `model` 必须建立联系。

② 循环的另一种写法

`index.html.erb` 里面有一段这样的代码：

```
<%= render :partial => "orderlist", :collection => @orders, :as => :order %>
```

`:collection => @orders, :as => :order` 表示读取 `controller` 传来的 `@orders` 的集合，并在其中取到每个 `order` 的数据。其实就等于循环的意思。

所以这里循环了之后，在 `_orderlist.html.erb` 就不用再套 `<% @orders.each do |order| %>` 这个循环了，否则显示的内容会重复（总之就是会错）。

二者选其一，当然，以第一种方法较为简便。

- 制作支付宝支付（或微信支付）“假支付”的思路：
 - ① 添加一个 `is_paid` 的栏位，`default` 为 `false`。即默认状态为未支付。
 - ② 在 `controller` 里面实作一个动作（`def` 的名称与 `routes` 里 `post` 的 `method` 名称对应），当点击支付时，`is_paid` 赋值（`="`）为 `true`。意思是点击按钮，则 `is_paid` 为 `true`。
 - ③ 在 `view` 里面对应设置当 `is_paid` 的值为 `true` 时（这里要用 `=="`），则标签显示“已付款”，否则（即“`false`”时）显示“未付款”。因为默认情况是 `false`，所以订单刚建立为点击按钮就会显示“未付款”。
 - ④ 同样，在 `view` 里面设置如果 `is_paid` 的值是 `false`（`=="`），则显示付款的按钮，否则（即“`true`”时），不显示按钮。`view` 里的具体代码如下：
- 略
- ```

```
- ```
  <% if @order.is_paid == true %>
```
- ```
 <label class="label label-success">已付款</label>
```
- ```
  <% else %>
```
- ```
 <label class="label label-warning">未付款</label>
```
- ```
  <% end %>
```
- ```

```
- 略
- 
- 
- ```
<% if @order.is_paid == false %>
```
-
- ```
<div class="group pull-right">
```
- ```
  <%= link_to(" 支 付 宝 支 付 ",pay_with_alipay_order_path(@order),:method
```
- ```
 => :post,class:"btn btn-danger btn-lg ") %>
```
- ```
  <%= link_to("微信支付",pay_with_wechat_order_path(@order),:method => :post ,class:"btn
```
- ```
 btn-danger btn-lg ") %>
```
- ```
</div>
```

<% end %>

- html 里面的用 class="", 如果是引用, 比如引用 bootstrap 里面的按钮或者 link_to 里面, 则要用 class:""
- rails c 删除全部资料的命令: XXX.delete_all 例如: Group.delete_all
- 小计的写法: <%= product_list.price * product_list.quantity %> rails 里面可以直接用*
- 关于.gitignore 隐藏的文件如何不显示和一个世纪大坑
如何不显示: 可以在 tree-view 里面设置 Hide VCS Ignored Files 让加在.gitignore 里的隐藏文件不显示
世纪大坑: tree-view 里面的 Hide VCS Files 不勾选的话, 在默认的 Atom Dark Themes 里面隐藏文件显示的是灰色字, 而在 Spacegray Themes 里面却和其他公开文件的显示没有差别, 是看不出来的!! 一个让人欲哭无泪的坑。

[Tweet](#)

[fullstack-course 8/9](#)

今天学的东西

今天主要学了如何做订单系统, 如何生成订单和提交订单, 制作订单页面

今天的困扰

今天遇到的困扰就是感觉自己遇到了瓶颈, 被老师说没有安全感, 哈哈~~

好吧, 不过遇到问题还是要解决问题, 借此梳理一下。

我之所以觉得自己遇到瓶颈, 其实本质上还是因为觉得自己应该懂一些原理性的东西才会知道怎么做。当然原理是需要了解的, 但是这种以前遗留的观念是错的。

应对这样的想法, 我想了两个办法帮自己破除困扰, 让自己不再质疑。

①想想二周前的自己, 再看看现在的自己, 不去看不会的东西, 只看自己还算挺大的进步。可以发现, 其实在 coding 这个“拼图”上, 我已经拼了很多块了。之所以觉得自己还不够熟练, 不够了解, 是因为拼的还不够多, 并不是自己不够好, 只是时候未到。掌握是早晚的事。

②去看了笑来老师的那篇 [《<人人都是工程师>前言》](#), 给自己灌了一碗浓浓的鸡汤。

“重复才能练就技艺。”

“我们的大脑就是这样的, 有强大的能力把我们常用的物体、技能“吸收”进来, 就好像那些东西是我们身体的一部分一样, 进而随心所欲地控制。厨艺大师手中的刀子, MBA 明星手中的篮球, 都是这样的, 虽然看起来是“身外之物”, 实际上, 对他们的大脑来说, 那些东西都是他们身体的一部分, 是他们可以随心所欲地控制的 —— 当然, 外人看来那实在是太神奇了!”

“一切看起来复杂的技艺, 其实都并不难, 很多人最终学不会, 其实只是练不成, 就是说, 他们并不是不理解那道理、那原理; 可理解本身并无太大用处, 因为真正需要做的是通过大量的重复与实践, 把那道理、那原理转化为大脑皮层表面的沟回…… 缺少了刻意训练的环节, 学什么都是白搭。”

“你就可以开始边学习, 边实践, 在学习中实践, 在实践中学习更多, 虽然有时掉进陷阱, 有时误入歧途, 但, 请你放心, 肯定不会死人的。是谓不断进步, 是谓 ‘路漫漫其修远兮, 吾将上下而求索’。”

牢记《卖油翁》里的那句金句“无他——但手熟尔!”

Q&A

- self.products.include?(product)
这是在 cart.rb 的代码。

含义：从 cart 自身读取 products，判断 Cart 里的 product 是否包含了 product。括号里面的 product 是从 controller 里面传进来的。

如果要做一个不包含的意思，则应该是 `!self.products.include?(product)`。

- migration 不能直接修改，必须新增。那怎么让其他文件能精确读到栏位呢？

```
def change
  add_column :orders, :user_id, :integer
end
```

add_column 的结构是这样的：`add_column(table_name, column_name, type, options = {})`。

所以在第一格 table_name 里面已经确认了是在哪个表单里添加栏位，跟命名没有关系。命名只是为了更好的表达意思。

- 关于变数
ruby on rails 的原则是尽量“说人话”。所以变数尽量不要用缩写，因为缩写会让人不知道是什么意思。
例如：不要用 @ci，要用 @cart_item
- 记得把图片(文件夹地址)放在 .gitignore 里面，防止上传到 github

[Tweet](#)

[fullstack-course 8/8](#)

情绪

今天自己做了五六个功能，解题成功的快感超爽。这算是这两周来最兴奋的一天了。

感悟

这次课程的实质，付费购买的应该是头脑的改变，认知的升级。前两周一直有一种想法，想把老师教的东西都“记”下来，“录”下来，恨不得把老师以前的视频也要来收藏起来。其实这是认知的一种落后，以为付费购买的一定要是“实体产品”，被一种虚幻的占有的感觉控制，过分执着于“占有”这件事。这也算是工业化时代的一种思维局限。

反思一下，不管是付费课程，买书学习，或是和牛人学习，其实这些关乎成长的事最终改变的应该是人的认知，只要认知升级，就是成长了。否则，取得再多的“视频”、“笔记”、“书籍”也只是一种自我安慰的虚妄。

总之，要以“个人”为参照点，拥不拥有不重要，认知升级才重要。

如何践行：以后学习要把关注焦点发在自己改变成长了什么，而不是自己记下了什么可以炫耀或者是自我安慰的东西。

如何和高段位程序员交流学习：

- ①参加 hackthon
- ②写 blog，让别人看到你的价值
- ③参加一些比自己 level 略高的 conference

学习资源

rubyweekly

ruby inside

confreaks.com(比较难)

railscasts.com

gorails.com

ruby5

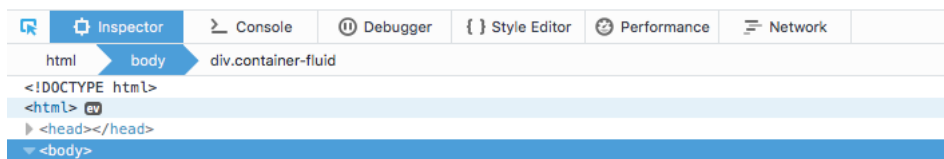
greent ruby

Q&A

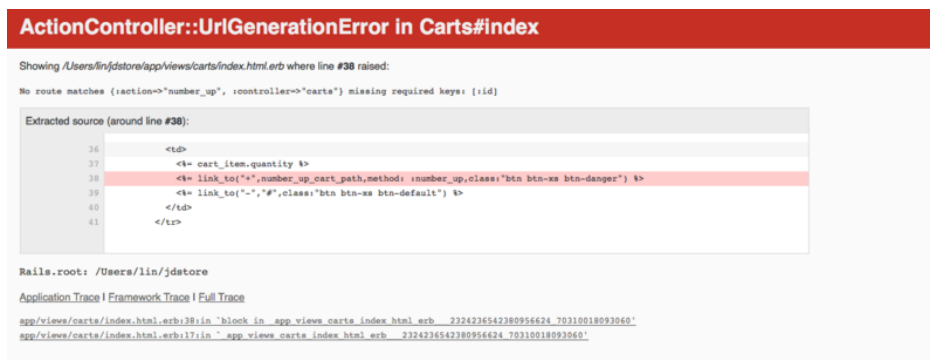
- views 里什么时候引用@product, 什么时候引用 product ?
@product 是调用 controller 里面的数据, 之所以用 product, 一般是在 views 里面有用循环去取到 product 的值。根据这个思路就可以判断用@product 还是 product。

```
<% @products.each do |product| %>
  <div class="col-md-3 col-lg-2 col-xs-2">
    <div class="img-thumbnail">
      <%= image_tag product.image.thumb %>
    </div>
    <div class="text-left" style="margin-top: 10px">
      <%= link_to(product.title, product_path(product)) %>
    </div>
  </div>
<% end %>
```

- 更方便的使用 inspect
调用 inspect 功能, 点击 inspect 面板左上角的鼠标按钮。可以实现在页面上 pick 一个元素, inspect 里对应显示的功能。



- @ci.quantity += 1 的意思是@ci.quantity =quantity+1
- 缺少 ID 的错误 (missing required keys[:id])



这是缺少[:id]的错误

ERROR:

```
<td>
  <%= cart_item.quantity %>
  <%= link_to("+",number_up_cart_path,method: :number_up,class:"btn
  <%= link_to("-", "#",class:"btn btn-xs btn-default") %>
</td>
```

OK:

```

<td>
  <%= cart_item.quantity %>
  <%= link_to("+",number_up_cart_path(cart_item),method: :number_up
  <%= link_to("-",#"",class:"btn btn-xs btn-default") %>

```

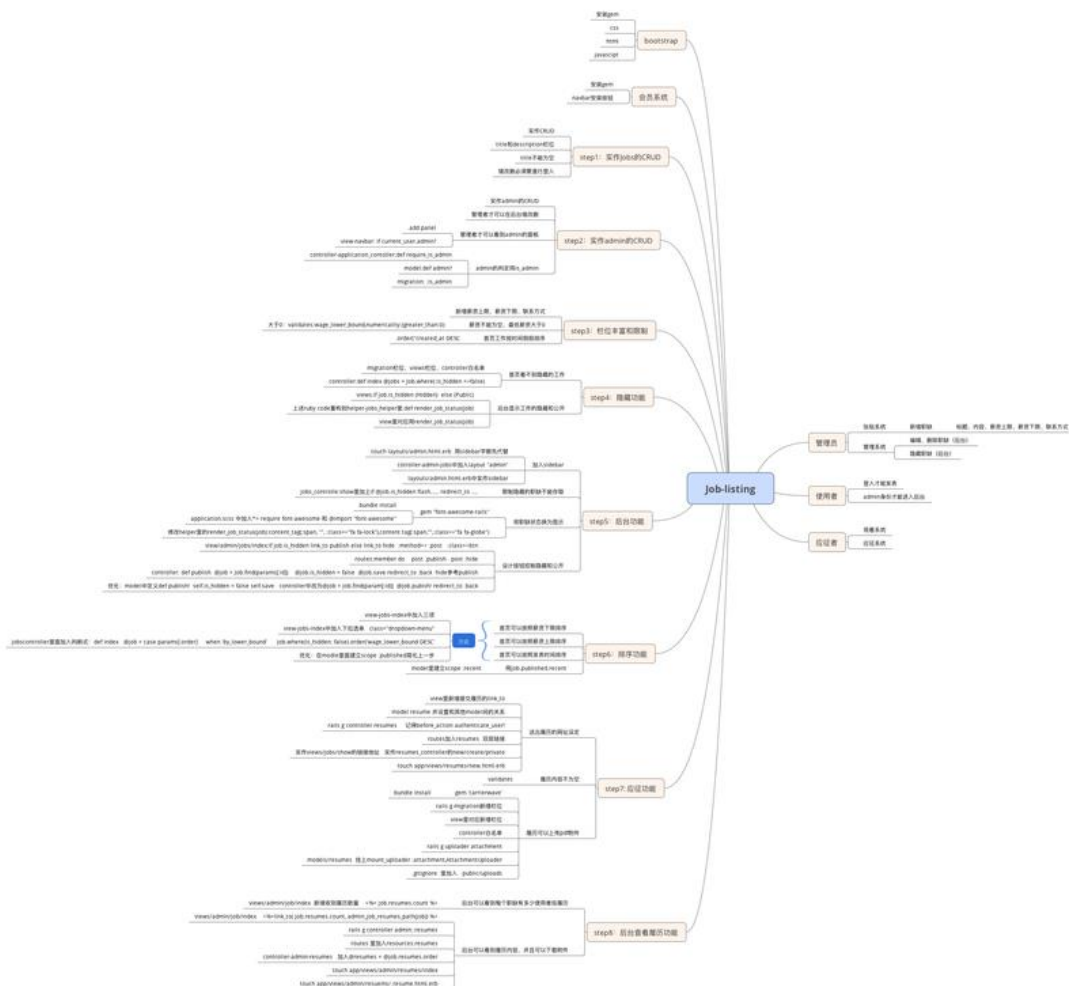
- 做一些功能的思路 ①先做出 view, 然后在 model 中寻找栏位, 用这样栏位来对应 controller 的调用命令。②model 和 controller 不能直接关联(即 controller 里的@job 在 model 里不能直接调用), 要通过别的办法(如 self.xxx)来实现 ③model 中 cart_item.rb,有调用这个 model, 要关联这个 model, 用的名称应该是 CartItem, 表示调用这个表单。例如: @id = CartItem.find(params[:product_id]) ④购物车商品数量增加功能, contrller 里 def 后记得加上 redirect_to(可以是某路径, 也可以是:back)。没有加的话要刷新才能看到结果。其他类似的功能也有参考这个思路。

[Tweet](#)

fullstack-course 第二周周记

本周周记, 不写鸡汤, 不写感悟, 只写技术细节。

对 job-listing 的 code 流程进行梳理



其他常用流程总结：

- 加栏位步骤：
 - rails g model 或 rails g migration(直接加或手动加)
 - controller 设置白名单

- ③view 设置相应页面
 - 几个常用 gem 的安装方法
 - ①"bootstrap-sass"
mv .css .scss
app/assets/stylesheets/application.scss:
@import "bootstrap-sprockets";
@import "bootstrap";
app/assets/javascripts/application.js:
//= require bootstrap/alert
 - ②"simple_form"
rails generate simple_form:install --bootstrap
 - ③"devise"
rails g devise:install
rails g devise user
rake db:migrate
 - ④"will_paginate"
bundle install 即可
 - ⑤"carrierwave"
新增栏位, 如 (image)
rails g uploader 'name'(实际不用加引号, 如 : image)
 - ⑥"font-awesome-rails"
app/assets/stylesheets/application.scss :
*= require font-awesome
@import "font-awesome";
 - 实做 CRUD 步骤
 - ①rails g model
 - ②rails g controller
 - ③routes 设置路径
 - ④controller 内容填充
 - ⑤view 内容填充
 - 重开 rails 的几种情况
 - ①装完 gem 后要重开
 - ②资料库有变更时要重开 (rails g model 或 rails g migration)
- 总结 : app 文件夹外的资料有修改都要重开

ps:

一个常见错误 :

上传后信息（字符串或图片等）不显示，也不报错，一般是 controller 的白名单没有加进去。

[Tweet](#)

[fullstack-course 8/5](#)

完成的事

1. jdstore 的收藏夹功能
2. jdstore 的一些功能优化
3. 重看 rails for zombies

新学会的技能

1. 快速新建栏位的方法
rails g migration Create<TableName>Table [columnName:type] [columnName:type]
举例，创建一个名为 Products 的表格，包含 title 和 price 这 2 列，类型分别是 string 和 float。
跑指令 rails g migration CreateProducts title:string price:float，生成的数据库文件。
2. 简化 CRUD
show/edit/update/destroy 中的 @job = Job.find(params[:id]) 可以单独提出来
3. def show
4. @job = Job.find(params[:id])
5. end

```
class JobsController < ApplicationController
  before_filter :authenticate_user!, only: [:new, :create, :edit, :destroy]
  before_filter :get_job, only: [:show, :edit, :update, :destroy]
  def get_job
    @job = Job.find(params[:id])
  end
end
```

6. 没有上传图片时读取固定图片的方法
在 uploader 里的 ruby 文件有定制了一些功能，默认是注解掉的，可以根据需要选用。其中就包括没有上传图片时读取固定图片的方法。如下图：



```
image_uploader_uploader.rb — /Users/lin/jdstore
image_uploader_uploader.rb  index.html.erb  no_pic.jpg
33 #
34 #  "/images/fallback/" + [version_name, "default.png"].compact.join('_')
35 # end
36
37 def default_url
38   # For Rails 3.1+ asset pipeline compatibility:
39   # ActionController::Base.helpers.asset_path("fallback/" + [version_name,
40
41     "/images/fallback/" + [thumb, "no_pic.jpg"].compact.join('_')
42 end
43
```

只要在 app/public/images/fallback 里面放入需要读取的图片即可。

[Tweet](#)

[fullstack-course 第二周/最棒的概念或工具](#)

用注解的方法 debug

与 HTML 用 <!-- --> 注解不同，ruby 语言的注解标准格式应该是 <%# %>，例如

```

    <%# @products.each do |product| %>
    <tr>
    <td>

    <%= link_to(product.title, admin_product_path(product)) %>

```

这个方法可以用来 debug，遇到不确定的时候，可以把那段注解掉，而不用删掉，然后就可以检查是不是该段出错了。这种方法可以提高效率。

总结：

html 语言的注解：<!-- -->

XXX.html.erb 文件的注解：<%# %>、<%= %>

XXX.rb 文件的注解：# Table name: installs

bootstrap 全局 css 样式

可以利用栅格系统进行页面布局。

因为 code 和视窗节目不一样，不能用视觉化的方式进行布局。所以要用到<div>这种区块的方式来布局。

要掌握这种区块布局的思想。用 code 的方式来呈现前端。

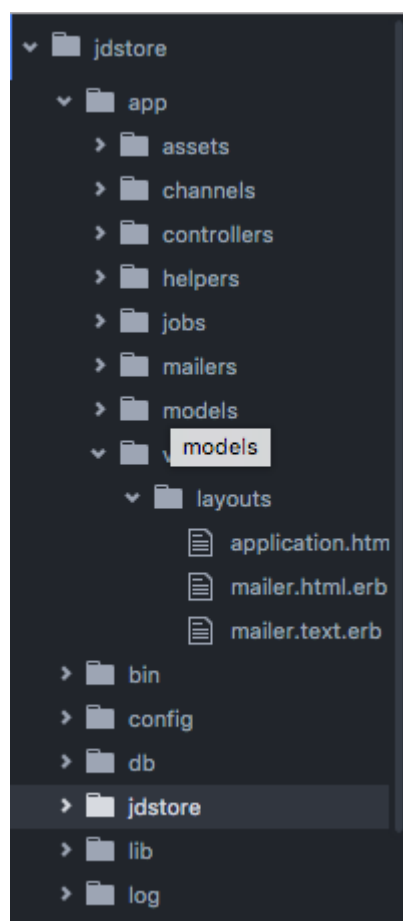
[Tweet](#)

[fullstack-course 第二周/最大的坑](#)

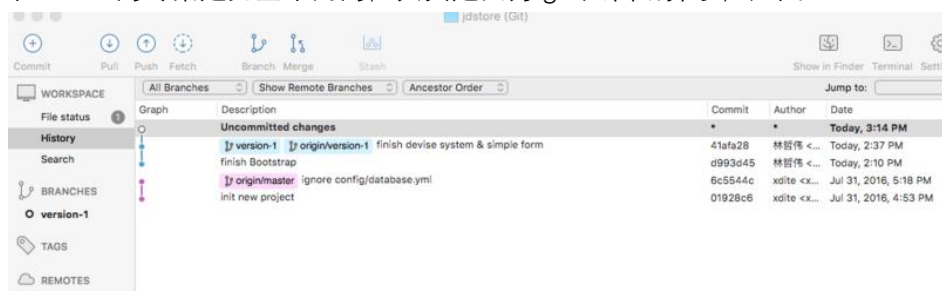
关于 git clone 的一个大坑

从 git clone 下来是 clone 包括文件夹在内的东西。要直接在根目录或者某个建立的空文件夹里 clone。

一个大错误就是先通过 rails new 建立了一个新专案，然后在新专案里 clone，就会出现 clone 下来的专案在建立的新专案里（专案套专案），如图：



这样在 rails new 建立的专案里进行编辑操作，最后 push 到 git 的时候就会失败，会显示 push 上去的专案和 clone 的专案是完全不同的，其实是因为 git 文档断掉了，如图：



所以，切记——

不能在一个新专案里进行 clone !!!

不能在一个新专案里进行 clone !!!

不能在一个新专案里进行 clone !!!

ps：clone 到一个新的空文件夹可以用这个命令 `git clone https://github.com/linzhewei/jdstore.git <新文件夹名称>`，这样，专案就会 clone 到新文件夹里。如果命令不带<新文件夹>，则会默认 clone 到 jdstore 的文件夹里。

[Tweet](#)

[fullstack-course 8/4](#)

今天完成的功能

完成了 story2、story3，初步完成了 clone 京东网页的任务。

今天学到的 code 方法

今天和昨天一样，开始独立进行开发。遇到了很多 bug，但自己已经有了初步 debug 的能力，所以绕过了很多坑，开发的也比较顺利。

今天卡壳最厉害的地方是如何对网页进行布局。因为 code 和视窗节目不一样，不能用视觉化的方式进行布局。所以要用到<div>这种区块的方式来布局。今天初步理解了如何用 css 和 bootstrap 自带的很多功能来布局网页，主要是 css 的一些 style 和 bootstrap 的栅格系统。虽然写出来的代码很乱，也不是很简洁美观，但毕竟可以实现自己需要的效果。

我觉得代码可以再优化，最重要的是掌握这种区块布局的思维。

项目管理和 hackthon 获胜秘诀

xdite 和我们分享了 hackthon 获胜的秘籍。参加这种比赛，首先目的要清晰，结果导向，目的就是为了“赢”。根据这个结果去研究自己要做什么事。

要点有三个。第一，也是最重要的是做出优秀的 presentation；第二，认清比赛风向，做出能让举办公司“刊登报纸”的作品；第三，代码能动，实际运行无 bug。其实归纳起来就是一点：用最短时间做出一个风向正确的 100%的产品（或功能），并且展示出来。

xdite 的项目管理能力实在强大。hackthon 比赛（8 小时时长），她只做一个功能，开发只用 5 小时（中间还有时间贱贱地去跟对手炫耀，乱其阵脚），做出 100%的作品，并用 1 小时准备 presentation。这种强大的时间管控能力和自控能力实在需要去学习。当然，熟练的 code 能力是这些的基础。

我能学到有两点，最重要的一点就是项目管理第一秘籍：把时间砍掉！假如有 3 个月时间做项目，就把时间砍成 2 个月，让时间的稀缺来逼迫自己集中注意力，激发自己对事件优先级的排序能力（参考《稀缺》）。同时留出充分的时间来处理各种不可控事件，比如 bug 或者一些乱七八糟的事情。

第二点就是：项目完成优先！确定功能后先把产品开发出来，先做 must have，做完后再做 should have。must have 的优先级不可动摇，这就是一个“指挥官命令”，一切事务以 must have 为主，雷打不动。因为 3 分的 100%的产品比 9 分的 60%的“产品”强，所谓 60%的产品其实就是一堆零件，没有卵用。

无聊编个段子：

A：“你代码好丑，看我的，多简洁优雅美观啊。”

B（冷漠脸--!）：“哦，但是你产品没做出来。”

A：“看我画面多美，你那是啥乡村非主流。”

B（冷漠脸--!）：“哦，但是你产品没做出来。”

A：“看我这创意多牛逼，分分钟改变世界，你那是啥鬼？”

B（冷漠脸--!）：“哦，但是你产品没做出来。”

A：“你能不能不说这个。”

B（冷漠脸--!）：“哦，我 demo 做完了。”

line 台湾区经理 Sting 分享

创业失败三个原因：

第一次，没经验。不过第一次就创业成功某种意义上是一种诅咒，容易让人迷失。

第二次，和队友价值观不和。

第三次，投资方干预过多。

经验：

在创业这件事上，“好死不如赖活着”绝对是错的，千万不要将就，风向错的或没意义的，就最好尽快结束掉，还能换取一些经验和能力。“早死早超生”才是创业阶段面对自己觉得不行的项目时正确的价值观。

扯蛋心得

①聂师傅用半年时间高考市第一名，eve 用三个月时间考上公务员，顺便撰文“如何用三个月考上公务员”。让我更加确信一切社会大众对时间的感知都是值得再反思的，让我更确信逃逸速度的存在（[关于逃逸速度](#)），确信自己可以用逃逸速度去做事，摒弃过去的众多错误认知。

②和三观相仿的人讨论就是爽，感觉聊天都能让人高潮。真的要多找牛逼的人聊天。常与同好争高下，不共傻瓜论长短！

ps：好吧，其实我自己写项目管理要砍掉时间，但写这篇文章还是超时了二十分钟。看来是因为自己写作能力还需锻炼，还有就是对自己能力的预估不准确。当然，还有就是对自己不够狠，没有舍得砍掉一些不是 must have 的东西。继续努力！

[Tweet](#)

[fullstack-course 8/3](#)

Objective

课程：简单的 User story 项目管理

完成：商店的两个部分

Reflective

今天情绪：平静

高峰期：晚上的时候效率很高

低点：白天的时候各种出错，各种莫名奇妙的错，快疯了

Interpretive

课程主要内容

- 1、 user story 的简单项目管理
- 2、 做商店的思路和方法

重要领悟

我发现自己对一些基础知识能提出的问题比较少，不是说都懂了，而是不知道去问出问题。我想了个解决办法，要开始实践一下。就是多去看 issues 上的东西，每题都试着去回答。因为老师之前的分享说过，其实问题来来去去就是那些。所以我觉得我既然还不是很擅长在这个领域提出问题，我就先去看别人的问题，去弄懂，这也能去完善自己的拼图。而且我觉得，我在回答别人的问题的过程中也会提出自己的一些问题，良性循环。^_^

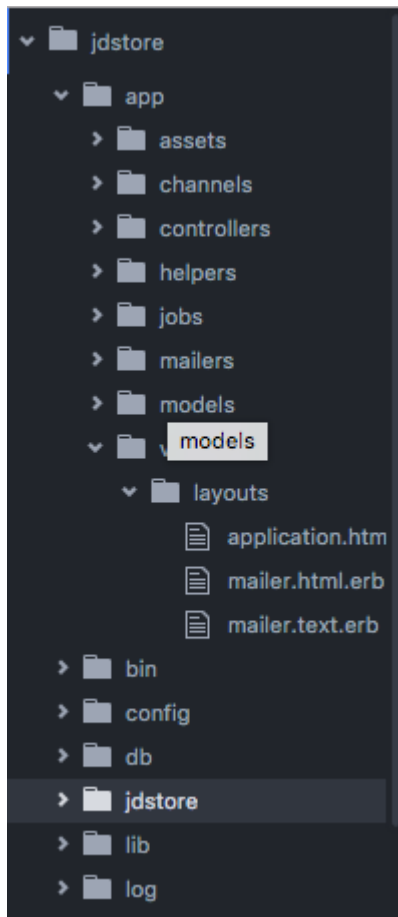
Decisional

形容今天的工作：开始正式没有解答自己做东西，虽然遇到很多错，但一遍一遍的敲代码，熟悉了很多。发现自己真的可以自己来做出网页了，哈哈哈！！

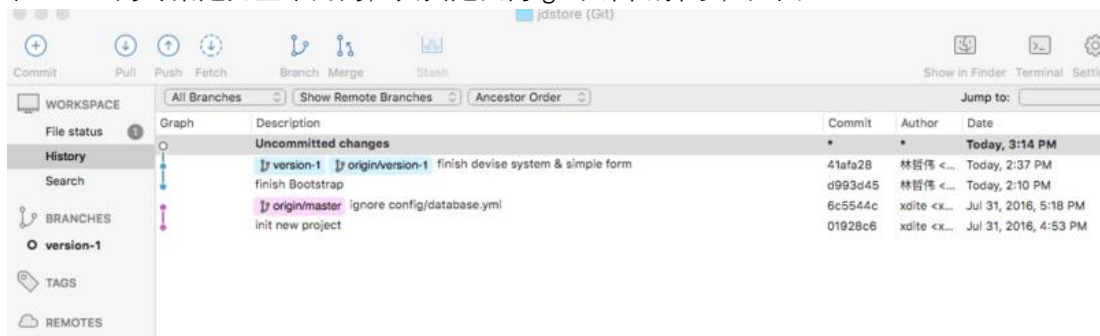
明天努力方向：继续向前努力

Q&A

- 从别人的 git 上 fork 一个专案，出现 git push 失败的情况？
可以先尝试 `git remote set-url [--push] <name> <newurl>`，例如 `git remote set-url --push origin https://github.com/linzhewei/jdstore.git` 如果成功，就可以开始 push。
如果出现 fatal: No such remote 'origin' 的提示，则要添加一下 origin 再进行 push，命令为 `git remote add origin <url>`，例如 `git remote add origin https://github.com/linzhewei/jdstore.git`
- 关于 git clone 的一个大坑
从 git clone 下来是 clone 包括文件夹在内的东西。要直接在根目录或者某个建立的空文件夹里 clone。一个大错误就是先通过 rails new 建立了一个新专案，然后在新专案里 clone，就会出现 clone 下来的专案在建立的新专案里（专案套专案），如图：



这样在 rails new 建立的专案里进行编辑操作，最后 push 到 git 的时候就会失败，会显示 push 上去的专案和 clone 的专案是完全不同的，其实是因为 git 文档断掉了，如图：



所以，切记——

不能在一个新专案里进行 clone !!!

不能在一个新专案里进行 clone !!!

不能在一个新专案里进行 clone !!!

ps：clone 到一个新的空文件夹可以用这个命令 `git clone https://github.com/linzhewei/jdstore.git <新文件夹名称>`，这样，专案就会 clone 到新文件夹里。如果命令不带<新文件夹>，则会默认 clone 到 jdstore 的文件夹里。

- 关于 ruby 的注解
与 HTML 用 `<!-- -->` 注解不同，ruby 语言的注解标准格式应该是 `<%# %>`，例如

```

<%=# @products.each do |product| %>
<tr>
<td>

<%=# link_to(product.title, admin_product_path(product)) %>

```

这个方法可以用来 debug，遇到不确定的时候，可以把那段注解掉，而不用删掉，然后就可以检查是不是该段出错了。这种方法可以提高效率。

总结：

html 语言的注解：<!-- -->

XXX.html.erb 文件的注解：<%=# %>、<%=# = %>

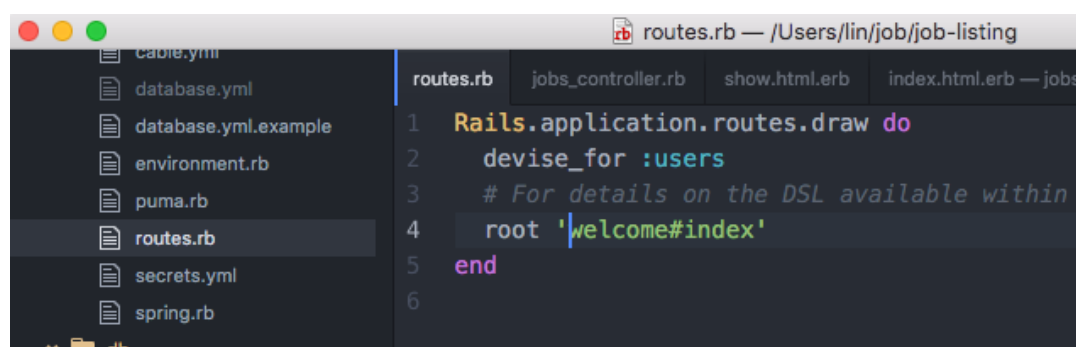
XXX.rb 文件的注解：# Table name: installs

[Tweet](#)

[fullstack-course 8/2](#)

实干

devise_for :users 是自动生成的，不能手动添加

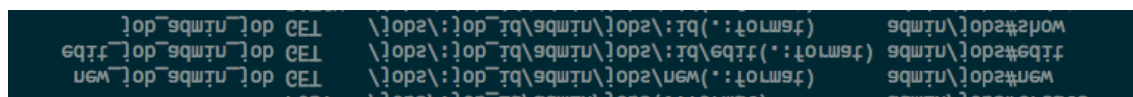


devise_for :users 是 devise 这个 gem 安装后自动生成的，不能手动添加。如果手动添加会占了这个位置，导致最后安装失败，总之就是会乱掉啦。

simple_form 这个 gem 安装后还要安装 simple_form for bootstrap 的设定

执行 rails generate simple_form:install --bootstrap，然后重开 rails s，这样 simple_form 的表单才能生效。

rails g controller admin::jobs 指的是在 controller 里新建一个 admin 的文件夹(当然，像直接建立一个 controller 一样，也会建立其他的一些文件，比如 helpers 文件夹里的对应文件等等)，文件夹里面包括 jobs 这个文件什么错误导致出现双层目录？



error：

```

routes.rb
1  Rails.application.routes.draw do
2    devise_for :users
3    # For details on the DSL available within
4    resources :jobs do
5
6      namespace :admin do
7        resources :jobs
8      end
9    end
10   root 'jobs#index'
11 end

```

ok :

```

routes.rb
1  Rails.application.routes.draw do
2    devise_for :users
3    # For details on the DSL available within
4    resources :jobs
5
6    namespace :admin do
7      resources :jobs
8    end
9
10   root 'jobs#index'
11 end
12

```

新建栏位两个方法

- ①在新建 model 时，直接定义栏位
- ②要在已建立的 model 里面建立栏位，则要通过 rails g migration 用数据库来添加 rails 中 ! 和 ? 的用法

! 放前面 !current_user 不等于的意思

! 放后面 STRING.downcase! 表示执行此 method 会变更自己的状态

? 放后面 admin? 表示执行此 method, 预期回传的是 TRUE/FALSE

rails 实作中遇到的一些小问题

- rails g model job title:string description:text 含义解读 (rails 产生 model, 名字是 job[单数], 两个栏位及它们的属性是什么)
- rails g controller jobs (名字是 job[复数])
- simple_format 意思 回传 text 的改变到 html 的一个 rules 例 :<%= simple_format(@job.description) %> 因为 description 是 text, 所以要会传这个资料要用到 simple_format

鸡汤

5 个成长的 tips :

换工作
帮助别人
扛起责任
公开分享
回答别人的问题

今天做了什么

今天开始自己根据 user story 自己写代码，尽量只看教程的 step 部分来做，当然还是有一些细节要偷瞄一下才知道怎么做。写完了三个部分左右，感觉不依赖老师的代码自己写，难度还是很大的。一个是很多东西想不到怎么写，还有代码写不全，再有就是很多细节上（标点啊，空格啊）之类的会混乱。我采取的办法就是先背下来。

在不断练习的同时，我开始会抽一些时间看同学的 blog 和 issues，补充自己的一些基础知识。

今天最大的坑

今天最大的坑是电脑不断在给我挖坑，出现各种老师一出现什么都不用做，问题就自动解决的情况。老师说我的电脑不爱我了，说我需要休息。好吧，那我今天就不写太多东西了。。。。。

[Tweet](#)

[scope 是什么？怎么用？](#)

什么是 rails 中的 scope?

scope 是用于包装常用 query 的方法。可以把 scope 理解成是一个懒人包，以后要用到某个功能可以直接在 scope 里面写好，然后引到 scope 就可以了。

举例：

```
class Topic < ActiveRecord::Base
  scope :recent, -> { order("created_at DESC") }
end
```

这段定义了 recent 这个 scope，以后只要下 recent 这个指令就等于 order("created_at DESC")，这样可以使程式码更简洁。

什么时候使用 scope？

- ①当有过于复杂的资料查询
- ②当有重复使用的资料查询

为什么我们要用 scope?

可以简化程序码，让程序码读起来更易懂，更像人话。

如何使用 scope?

- 没带参数的方式
- `class Post < ActiveRecord::Base`
- `scope :published, -> { where(published: true) }`
- `end`
- 带有参数的方式

- `class Post < ActiveRecord::Base`
- `scope :created_before, ->(time) { where("created_at < ?", time) }`
- `end`
- 串接一起用，顺序没有影响
- `class Event < ActiveRecord::Base`
- `scope :published, -> { where(published: true) }`
- `scope :created_before, ->(time) { where("created_at < ?", time) }`
- `end`

[Tweet](#)

[redirect_to 的两种用法](#)

用法一：`redirect_to post_path`(指向的是/post)

用法二：`redirect_to "http://www.google.com"`(指向的是特定外部网站)

[Tweet](#)

[view 内 render 的用法](#)

`<%= render "item" %>`

等价于

`<%= render :partial => "item" %>`

等价于

`<%= render :partial => "welcome/item" %>`

最后一种写法可以保证更精准的定位。

前两种会默认在所处文件夹，比如，代码写在 `app/views/admin/rusumes` 里面，则会去引用 `rusumes` 文件夹里的 `_item.html.erb`。如果想引用其他文件夹的则要参照第三种方法。

[Tweet](#)

[controller 内 render 的用法](#)

render :new 的用法

`render :new` 指渲染 `new` 的页面

render :text 的用法

`render :text => "ok"` 即印出文本"ok"

```
ok
```

render :layout 的用法

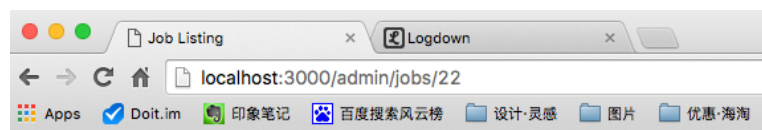
`render :layout => false` 指的是针对某个页面让其不套用 `layout` 的样式

`def show`

`@job = Job.find(params[:id])`


```
render :layout => false
end
```

效果如下：

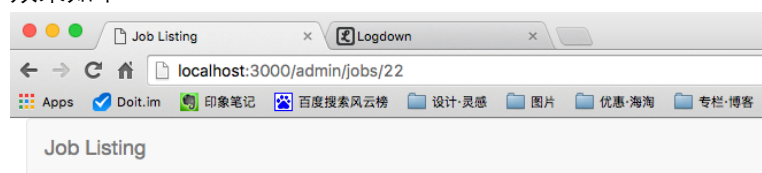


123

123

```
render :layout => "admin" 指的是针对某个页面让其套用 layout 中 admin 的样式
def show
  @job = Job.find(params[:id])
  render :layout => "admin"
end
```

效果如下：

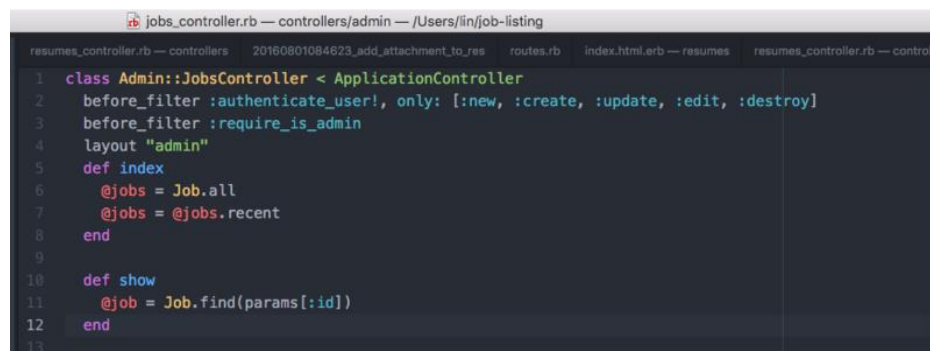


Jobs 123

123

Copyright ©2016 Rails101
Design by linzhewei

如果是想对全局的 layout 进行修改套用，可以如下图，在开头插入 layout "admin"的命令



[Tweet](#)

[fullstack-course 8/1](#)

Lizenwei

43 / 74

Objective

课程：过滤简历、上传简历、admin 后台简历管理

完成：stop6、7、8

Reflective

今天情绪：开心吃猪蹄

高峰期:吃猪蹄

低点：好像也没什么低点

Interpretive

课程主要内容

1、今天学到的很多东西整理在博客的文章里

2、老师的一些分享

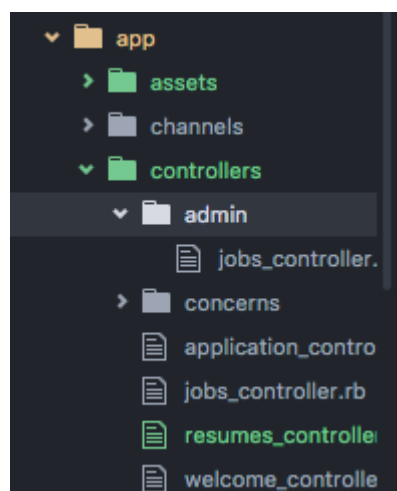
- 如何更幸运（减低不幸几率，增加幸运几率）
 - ①看说明书（前人犯过的坑都在那）
 - ②因为你不知道你不知道，为了跳出这个死循环，在做重大决策时走稍远的路，做别人建议更高级的决定（别人的 level 你不一定现在理解的了，但是这样照做才能提自己的 level）——所以路径应该是这样，首先确定这个“别人”是个 level 比你高的高手（这点很重要，在你要谈及的这个领域 level 比你低或者和你一样的人的建议不听也罢），接着听从他的意见，然后就可以抄家伙开干了。
- 感悟：如果你的老师在你要学习的领域 level 比你低，只能证明你是傻逼，跟一个 level 比你低的老师，那能怪谁。现在假如你不是个傻逼，你的老师 level 比你高，那他的所有方法、流程你都要照做，因为他的 level 看的到很多你看不到的东西，能很好的指导你。最可怕的是，你竟然妄图用你较低的 level 去揣测甚至修改 level 比你高的人的方法，还自以为是在聪明的学习，最后的后果只能是你只能在你自己的 level 里面打转，甚至学不成，受挫，各种不开心，何苦呢\ (∇) / 还有件更可怕的事，就是还人居然会拿出自己是在批判的接受这种说法来，尼玛！！你自己 level 都还没到那，你怎么去批判，批判地接受是高 level 的人对低 level 的态度好么？至少也要是一个水平线上的人才来说这种话，你都还领会不到别人的精华，谈个球球的批判啊！！
- 不要以为找个清闲的工作，然后用业余时间去自学可以（躲在深山老林练剑）有效，因为你每天可能利用业余时间学习两个小时，别人找那个行业，每天 8 个小时都在做自己喜欢的事，还是和大神一起工作，高下立判
- 不要怕被人笑，被人笑可以换取一个问题的解答，这是多么的值得啊
- 和牛人学习不用非得等到和他水平接近的时候才可以，水平相差 3 倍就可以了。如果你差 3 倍的时候不去跟，想等到 1.5 倍再去追随，你会发现等你 1.5 倍的时候，这些变态的牛人又跑了好远，所以不要等，想尽一切办法去接近牛人，他们能带你更快的成长

Q&A

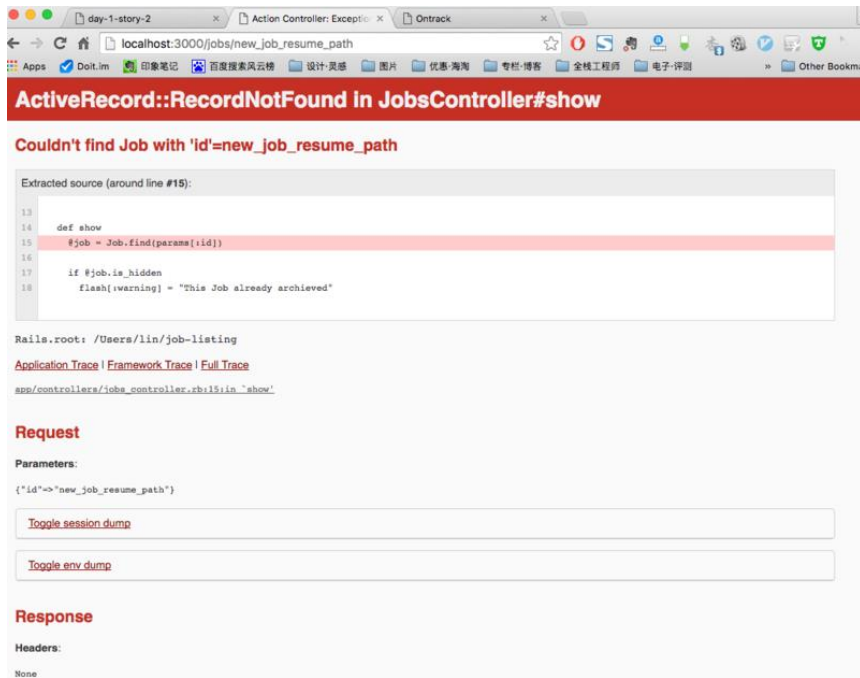
namespace 对应的是文件夹下的另一个文件，即 jobs 是包含在 admin 里面，为了和不包含在 admin 里面那个 jobs 区分开，所以这么写。而如果是两个独立的（不是包含关系）两个 controller 文件，则要直接用 resources，

也是通过 do 来表示双层关系。

```
resumes_controller.rb | routes.rb | jobs_controller.rb
1 ▾ Rails.application.routes.draw do
2   devise_for :users
3   # For details on the DSL available within this file, see http://guides.rubyonrails.org/routing.html
4 ▾ namespace :admin do
5 ▾   resources :jobs do
6 ▾     member do
7       post :publish
8       post :hide
9     end
10
11
12   end
13 end
14
15 ▾ resources :jobs do
16   resources :resumes
17 end
18
19 root 'jobs#index'
20 end
21
```



path 缺少 id



ERROR:

```
<div class="text-center">
  <%= link_to("提交履历", "new_job_resume_path", :style =>
</div>
```

OK:

```
<div class="text-center">
  <%= link_to("提交履历", new_job_resume_path(@job),
</div>
```

【①不能带引号②要有@job，因为要先读到是 job_id 才能确定具体链接地址】

套用 admin 的 layout

```
resumes_controller.rb — controllers/admin — /Users/lin/job-listing
resumes_controller.rb — controllers  20160801084623_add_attachment_to_res  routes.rb  resumes_controller.rb — cont
1 class Admin::ResumesController < ApplicationController
2   before_action :authenticate_user!
3   before_action :require_is_admin
4
5   layout 'admin'
6
7   def index
8     @job = Job.find(params[:job_id])
9     @resumes = @job.resumes.order('created_at DESC')
10  end
11 end
12
```

关于 rake 的命令

rake db:migrate(可以前进)

rake db:rollback (可以后退，但是尽量不要用，可以用叠加的方式来改变)

一开始发现字打错了，可以 rake db:rollback，修改，再 rake db:migrate，如果已经过很久，才发现打错字，

就用叠加来改

params[:xxx]的意思

当使用者点击链接时，会发出一个 http request，在 Rails 中，http request 会经过 ActionController 這個內建的机制來消化，並將可用的變數轉換為 params 這個變數，交由 controller 处理。

就是说 request 中可能包括很多东西，params[:xxx]会读取到:xxx 这个东西，转换为 parmas 变数，然后对其做处理。

关于 controller 中的 private

在 Rails 4 当中有 strong parameters 的机制，限定必须是我们设定可以在这个 action 当中进行设定的参数，才可以传入，详细解释可在延伸阅读中查询。为了将前端传回的变数储存到 model 当中，我们会在 controller 撰写一个 private method。就是说这个 private 为了能把我们的变数写进 model 而做的。

view 里面重复出现的代码，可以到对应的 modle 里面去新增一个 scope 进行定义，就可以简化代码了

用 before_action，不用 before_filter

google 技巧

可以指定某个网站进行站内搜索，例如：在搜索框输入 scope site:logdown.com 就可以在 logdown.com 里面检索 scope

[Tweet](#)

[fullstack-course 第一周周记](#)

知道全栈班，是我还在广州的时候，当时第一印象就是我想报名。因为当时觉得能自己做产品是挺酷的一件事，加上自己是做广告的，深深体会到为一个不优秀的产品做推广的那种无力感，经常会想如果能参与产品迭代，一定挺好玩的。当然，笑来老师也是我报这个班的原因，当时是以为可以和笑来老师一起开发产品，可以和学习能力这么强的人一起学习，是个千载难逢的机会，不料笑来老师并没有直接参与学习，伤害了我们感情，还一笑而过。不过 xdite 老师的课程设置、xdite 老师每天“不绝于脸”的笑容和我自己的进步速度让我都快忘了这个课和笑来老师有什么关系了。

回想一下参加全栈工程师班的课程，其实还算比较曲折。一开始一直再犹豫，一方面是价格昂贵，一方面是不知两个月时间到底能不能学成。所以报名又退出，退出又报名。不过我记得在报这个班时我爸和我说的一句话：“别人的建议只能是别人的建议，大事自己要握得牢（闽南话，“把持得住”的意思）”。所以在经过很多思考后我还是借了钱报了班。

在报名后我开始练习 xdite 老师的作业，而且严格听从 xdite 老师的建议，放下傲慢，遵从“学习的黄金通道”。我在课前一共练习了 4 遍中级教程，然后开始上课。一开始上课确实是迷迷糊糊的，因为有太多知识盲点，所以很多地方不知道老师在讲什么，不过还好有了前面 4 遍的练习，所以在不断练习中很多东西真的像老师说的一样，慢慢的就懂了。感觉自己慢慢在做着一个“拼图作品”，今天几块，明天几块，甚爽！

这一周我觉的我领悟了三个特别重要的东西。

1. 学习方法:做比懂更重要。

放下好奇心，不要管懂不懂，先上手做，多做几遍，做熟练了，在某个时间点，该懂的东西一点会懂的。遇到不懂的地方，可以去查 google，但是查了之后如果不懂，也不要纠结，果断向前继续做，反正以后还会再去看的，总能在某个阶段恍然大悟。现阶段最重要的是知道 how，而不是 why。以前的我可不这样，我会看各种乱七八糟的东西，在各种乱七八糟的方法或理论上纠结。我会看各种各样学习方法、时间管理等等教程类的文章，这种文章看完很容易给人一种假象，就是好像我看完我就已经达到了那个状态一样，其实真相是“听过很多道理，却依然过不好这一生”，反倒浪费了很多时间和精力。而 xdite 老师的这种方法正好解决了我的这个问题，让我对学习也产生了一个新的认知。正如笑来老师讲的：“学的好不好不重要，会用才重要。”学习的目的不是增进知识，而是增进行动，唯有行动才能改变一个人，其他的，扯得再多都没用。xdite 老师这套直接上战场，以战养战，向前突击，向后钻研的方法算是我领悟到的新理念最好的实践方法论。

2. 心态：期待错误。

面对错误的心态改变了。真正懂得并相信这句话：我们把该犯的错都犯过一遍，把该掉的坑都掉过，甚至还要去看别人犯的错，去弄懂，自然就学会了，自然效率就高了，也就成长了。

我们的教育一直再告诉我们要如何少走弯路，好像犯错是成功的对立面似的，天天讲“失败是成功之母”，但真正相信这句话的也没几个。想起我刚开始做练习时，每次看到跳出的红色错误页面，整个人情绪就特别低落，心想“尼玛，怎么又 TM 错了”。但是经过这么一周多的练习和 xdite 老师思想的启发，我现在看到错误页面已经很平静了，甚至有时候还会期待错误出现（好变态~~~），因为只要出现错误，就证明我成长的机会就来了，我可以藉由解决这个问题，让自己更加成长。所以也乐于和同学一起解决他们的问题。错误真的是促进一个人成长最好的方法，看一本指南，有时候真不如实打实的犯一次错误。以前有一个课程说过人要成长，就得“期待苦难”，现在是真的懂了。

3. “什么更重要”

笑来老师说人的一生中最重要的问题是“什么更重要”。(原文) 笑来老师来和我们做分享的时候，我发现他经常用这样的句式“...但我知道这不重要，...才是最重要的”。笑来老师果然是知道了一个道理后，就不断去践行。我在这一周的学习中也一直在践行这个方法，让自己的行动更加合理。

比如当我在编程遇到问题时，我经常抑制不住自己的好奇心，想要去弄去清楚。但以我现在的知识深度，想弄清楚很多问题无疑是一种“用未知解释未知”的行为，是没有意义的。这时候我就会告诉自己“知道这是什么不重要，知道怎么做更重要。所以先敲代码吧”再比如我以前经常会为了买个东西看很多测评，花很多时间去比较，但自从我那天知道“人最宝贵的不是钱，甚至也不是时间，而是注意力”时，我再购物总是用最短的时间完成，直接买自己能接受的最高价产品，因为我会告诉自己“只要确定这个东西有用，价格自己能接受就买。因为金钱不重要，商品的微小差异不重要，自己的注意力更重要。”我会不断用这个方法践行，因为我发现这个方法真的会带来很多好处，也能让我的选择更加理性和果断。

[Tweet](#)

[fullstack-course 7/30.31](#)

- 表格和表单的区别？(1)表格 (table) 表格用来布局。页面上要放上丰富的网页元素，如文字、图形、各种按钮、菜单、滚动新闻、动画等等，如果没有表格来安排它们，它们在网页上就会不工整，而且不能比较准确的定位，网页就不能做到赏心悦目。越是复杂的网站和网页，就越是要考虑大量应用表格。(2)表单 (form) form 表示一个表单,用来把一系列的控件包围起来,然后再统一发送这些数据到目标,比如最常见的注册,你说需要填写的资料,都是被封装在 form 里的,填写完毕后,提交 form 内的内容,如果不在 form 内则不会提交。表单中常见的元素主要包括：文本输入框、下拉选择框、单选按钮、复选按钮、文本域和按钮等。

二者在概念上是不同的,通常发挥作用的是 form，但是 form 内那些无规则随意摆放的文本框怎么看上去很整齐呢，所以通常是一个 form 里嵌套 table

- form_for 是什么？

打包表单到 model。form_for 是 Rails 的表单產生器，Rails 會智能的根據你指定的「物件」@group 與欄位 f.text_field :title 展開，產生表單，

While this is an increase in comfort it is far from perfect. If Person has many attributes to edit then we would be repeating the name of the edited object many times. What we want to do is somehow bind a form to a model object, which is exactly what form_for does.

- 在 new 的页面，新增资料点击提交不能成功提交？（点击提交后没有反应，表单没有提交，页面也没有跳转）

rails 101Hill,123@123.123 ▾

新增讨论版

* Title

123

Description

123

Submit

copyright ©2016 Rails101
design body xdite

原因：新增登陆验证后，没有相应让 create 紀錄 @group.user = current_ser。

```
app/controllers/groups_controller.rb

def create
  @group = Group.new(group_params)
  @group.user = current_user

  if @group.save
    redirect_to groups_path
  else
    render :new
  end
end
```

- touch 、 rails g 和 mkdir 的区别
touch 建立的是一个空白文档， rails g 建立的是已经有一些内容的文档，见图：

```
posts_controller.rb — /Users/lin/rails101-5

Gemfile  application.scss  application.js  20160731075123_add_user_id_

1  class PostsController < ApplicationController
2  end
3
```

一般来说，controller/model 都是通过 rails g， controlller 一般是复数， model 一般是单数。view 一般是通过 touch 建立。migrate 一般也是通过 rails g。
mkdir 是新建文件夹。

[fullstack-course 第一周/最棒的概念或工具](#)

1. 用 SourceTree 来回到 git 的某个状态超好用
2. 先知道怎么做，多做几遍，做熟练了，以后在某个时间点自然会懂。
3. 把该犯的错都犯过一遍，把该掉的坑都掉过，甚至还要去看别人犯的错，去弄懂，自然就学会了，自然效率就高了，也就成成长了。所以我的心态从看到错误疯了，变得现在很变态的期待新的错误，解决后记下来，就又懂了新的东西。

fullstack-course 第一周/最大的坑

Q:为什么会有这种神奇的东西出现 <http://localhost:3000/admin/jobs.5?>

A:正确的应该是

```
<%= link_to(job.title, admin_job_path(job)) %>
```

如果错误加了个 s，即

```
<%= link_to(job.title, admin_jobs_path(job)) %>
```

就会出现“神奇”的错误。

总结：message_thread_path(1) 這樣路徑就會是/

message_threads_path(1) 這樣路徑會是。

[fullstack-course 7/29](#)

Q:想修改 migration 要怎么做？



A:migration 不能直接修改，修改了也起不了作用。

方法：新建一个 rails g migration change_is_hidden_to_job,然后添加 change_column_default :jobs, :is_hidden, true

![Screen Shot 2016-07-29 at 1.59.40 PM.png](http://user-image.logdown.io/user/18420/blog/17955/post/775447/Fz84ITgCTMi9iuXeUL49_Screen%20Shot%202016-07-29%20at%201.59.40%20PM.png)

也可以 google：rails change column default`更详细了解

注意：不用轻易执行 rake db:drop / rake db:create / rake db:migrate,这会把之前的资料库都删掉，虽然也能起到一定作用，但是很麻烦，也容易因为没有及时更新数据产出不必要的 bug。

如果真的采用这种方法，可以考虑撰写 seed.rb 档来快速产生一些必要的数据库数据，详见：

[撰写 seed.rb 档](#)

[fullstack-course 7/28](#)

Objective

课程：针对所有基础知识进行答疑

完成：HTML 和 CSS 的练习，重新做了 job-listing，做到了 CRUD 完成的部分

Reflective

今天情绪：挺嗨的

高峰期：可以说是听老师扯蛋的时候么

低点：好像也没什么低点

Interpretive

课程主要内容

1、今天课程只要是回答一些答疑，主要内容都收集在 github 的 issue 里面，[issues](#)

2、老师的一些分享

- 写作或者说是成长的四个好建议：搜索搜索再搜索、付费付费再付费、整理整理再整理、分享分享再分享
- 时间管理的精髓就是精简时间，把多余的时间砍掉，这样人就会自己进行优先级排序，效率就会高。（比如 3 个小时写完文章，就砍成 1 个小时，人自然就会筛选最重要的事去做，就能发挥最佳效能）

重要领悟

“做正确的事比懂更多的事更重要”

Decisional

形容今天的工作：觉得自己慢慢懂了很多东西

明天努力方向：继续向前努力

Q&A

主要记录在 HTML 和 CSS 的笔记中：

[HTML](#)

[CSS](#)

Q：gemfile 重装后要重开 server？

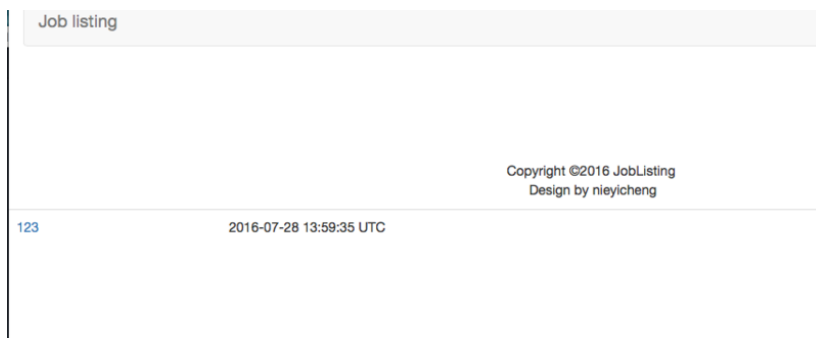
A：必须要重开啊！！必须！！必须！！！！每次 gemfile 重装后，bundle install，然后就去重开 rails s！！！！

Q：contain-fluid 是什么？

A: fluid 指的是那个容器（如导航栏）会随着浏览器的拉伸而拉伸变动。

Q:layout 页面排版问题(同学的问题和解答)

A:昨天重做 step1 打完 CURD 和制作完 layout 之后发现页面排版出现了问题



然后弄了好久都没发现出了什么问题，最后同学帮助解决了。。。。

原因是 views\Jobs\index.html.erb 里的 table 打成了 tbale。以后再出现这种问题，就要思考一下导致错误有哪些原因，看似有问题的地方并不一定是问题的关键。这个问题一开始我们一直现在一个思路里面，以为是 footer 的位置错了，一直排查发现不到问题。其实是思维陷入了一个死胡同，因为真正的问题是 index 下沉导致 footer 跑到上面，而不是 footer 自己跑到上面。

```
1 <table class="table table-bordered">
2   <% @jobs.each do |job| %>
3     <tr>
4       <td>
5         <%= link_to(job.title, job_path(job)) %>
6       </td>
7       <td>
8         <%= job.created_at %>
9       </td>
10    </tr>
11  <% end %>
12
13 </tbale>
14
```

错误变形：



error：

```
application.html.erb — /Users/lin/rails101-5
Gemfile application.scss application.js _flashes.html.erb flashes_helper.rb mailer.html.erb groups_controller.rb edit.h
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Rails101</title>
5     <%= csrf_meta_tags %>
6
7     <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track':
8     <%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
9   </head>
10
11   <body>
12
13     <div class="container-fluid">
14       <%= render "common/navbar" %>
15       <%= render "common/flashes" %>
16       <%= yield %>
17       <%= render "common/footer" %>
18     </div>
19   </body>
20 </html>
21
```

ok :

```
application.html.erb — /Users/lin/rails101-5
Gemfile application.scss application.js _flashes.html.erb flashes_helper.rb mailer.html.erb groups_controller.rb edit.h
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Rails101</title>
5     <%= csrf_meta_tags %>
6
7     <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track':
8     <%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
9   </head>
10
11   <body>
12
13     <div class="container-fluid">
14       <%= render "common/navbar" %>
15       <%= render "common/flashes" %>
16       <%= yield %>
17     </div>
18     <%= render "common/footer" %>
19   </body>
20 </html>
21
```

[Tweet](#)

CSS 基础知识

margin 與 padding 的差異

- padding 是内边距, 内边距在边框和内容区之间。margin 是外边距, 是围绕在元素边框的空白区域, 设置外边距会在元素外创建额外的“空白”。 图

什麼是 box model

- 框模型 (Box Model) 规定了元素框处理元素内容、内边距、边框和外边距的方式。 图

為何要使用 em 而非 px 來定義字的大小

- em 可以避免在 Internet Explorer 中无法调整文本的问题
- 1em 等于当前的字体尺寸。如果一个元素的 font-size 为 16 像素, 那么对于该元素, 1em 就等于 16 像素。在设置字体大小时, em 的值会相对于父元素的字体大小改变。浏览器中默认的文本大小是 16 像素。因此 1em 的默认尺寸是 16 像素。可以使用下面这个公式将像素转换为 em :

pixels/16=em

margin 示例

```
h1 {margin : 10px 0px 15px 5px;}
```

margin-top:10px

margin-right:0px

margin-left:5px

margin-bottom:15px

class 选择器第一个字符不能使用数字 !!!

[Tweet](#)

【转载】升级你的操作系统答疑——郑伊廷

BITTORRENT 式协议更新如何实际操作？

其实我平常就蛮喜欢去做分享的嘛。那为什么去做分享？

因为就会像现在这样有很多人问我问题，很多问题是我以前都没有想过的，那我就会试着去解答这些，大家就会给我更多的 Feedback 那我就有办法把自己的知识系统去完善。

这件事情对我来时是很有意义的，因为我自己只有一个人的人生，我没有想到那么多的问题，但如果有十个人、二十个人甚至几百个人给我 Feedback 那我就“活了几百个人的人生”，那我的速度就比以前快了几十倍甚至是几百倍，

所以这就是 BitTorrent 式更新，我通过分享和答疑，所以就可以学到更多的东西。

学习全栈工程最少必要知识是什么呢？

我觉得：

第一个是你一定要买一台 Mac，接下来 HTML 和 CSS 你一定要去了解，然后想办法找到自己学习的方式跟规律，在我的部落格里面，有一篇文章叫做《自学与自教》，我鼓励你去看看，这是一般来说我自学的一个很好的 pattern。

到外地工作，是离开家乡出去工作么？还是每天通勤去外地？

我推荐直接住在外地，因为你唯有了解这个城市是怎么样发展的，这个城市的生活习惯，而且你更多的时间经营在工作上面，那你得到的 Feedback 更大，像我年轻的时候，一天的工作是 12 个小时、16 个小时的，我把人生所有的时间都花在工作上面，我年轻的时候很有体力，所以那时候我得到的成长是很巨大的，要是我喜欢退休的生活没那么我当时成长的就不会这么迅速了。

培养人忠诚度会更高，外请大牛你怎么保证能留下来？

我最近在看樊登读书会里面有一本《联盟》，我觉得写得很好，你不要把员工想成你的奴隶啊，你应该想成员工是你的联盟，每个员工应该是任期制的，这个任期他做了哪一些东西可以帮你的事业达到更大的成长，那这样子就足够了。

你不要去奴隶嘛，奴隶就是会变成你讲什么，他动什么，这是你不要的嘛。

再有就是他被限制的时候会很郁闷的，他会在你的公司里面搞破坏的，所以呢和前员工最好的相处方式就是，是他想创业，你去鼓励他，甚至给他投资，他在外面帮你赚钱，那更好啊。所以你不要说想去控制人。我以前爱控制别人，后台我发现很好的员工在外面会给我带来生意的，甚至我投资他我会有更好的回报。

程序员需要向全栈发展吗？

我觉得话不能说死。

有些人是擅长全栈的，有些人是擅长前端的。全栈的好处是当你做产品的时候可以很快地发展出另一个模型。后来我年纪大了，编程速度比较慢了，我就觉得也不用真的自己搞全栈啊，我就是专心把这个模式做好，等赚到了钱我就请比我更牛的人来做这件事情，那就会更快了。

这是我变成生意人之后的另外一个想法。全栈可以让你接触到更多的语言，和他们的思维，我觉得还不错。

程序员现在“最好的网路环境”和“最好最丰富的下载论坛”在哪里？

这个会牵扯到程序员之间的战争，但我还是政治不正确地回答一下好了。

我觉得你要看就看 Ruby on Rails 和 Node.js 这两个社区，这两个社区牛人特别多，资源交流特别广泛，那我觉得你在里面可以学到最多的视野，我喜欢 Ruby on Rails 的原因是硅谷很多公司都用这套框架，他们会分享出来他们的套件，我从他们 release 出来的程式学到很多方法，好像我有很多牛逼大神同事一样，所以那个时候我得成长是非常快的。现在 Node.js 火起来了，所以我建议两个社群都关注一下。

您怎么去带领专业技术上比您更厉害的人，有哪些注意的点？

我觉得你要小心他的价值观，如果他的价值观不正确的话，即便他再厉害，你也不能留他在身边，除此之外你就放他自由开打就行。

这 5 点，你觉得哪一点是最重要的？

我自己觉得是去外地工作，但没一个人不尽相同，但我觉得你千万不要泡在你的朋友旁边，你的朋友会害死你的。我通常都会结交比我更优秀的人，他们经常会点出我的盲点，我这个人最不一样的就是，比我厉害的人跟我说什么我会照做的，因为他会看到我不知道的事情，我每次这样做后都会得到一个最新的天地，那我觉得这个是对我最大的帮助。

那个黄金通道没听太懂，可以稍微重复下么？

可以去看我以前的文章。

在原先的朋友圈交朋友，这么做对成长真的有帮助吗？老的朋友圈的朋友系统没有更新，那从中学到的东西会不会反而更慢？

没有帮助！没有帮助！

这是我领悟到的一个方式，但我觉得这是低效的，你应该去外地结交工作的朋友，这是高效的。

写代码的人怎么提高营销能力？

你可以看运营的书，或者去看硅谷 Growth Hack 的书。

怎么样说服比自己更牛的人为自己工作？

也不用说服啊，拿钱给他就好了。然后签一个明确的合作协议，你拿到你想要的，他拿到他想要的，这样就好了。但是最重要的是他真心想要和你做同一个产品。

用 BITTORRENT 类比，特别精妙。我觉得新生大学这里就是很好的 BT 管道，但是当下这里的 FEEDBACK 很有限。比如我之前写的文章有 300 阅读 23 赞，但是只有 5 条评论。那我在“上传”时，应该怎样做可以鼓励大家给更多 FEEDBACK？

你发到知乎啊，知乎上面一大堆人会给你 Feedback 的，或是你直接去回答人家问题，有些人就会 Feedback 你，但有些人会骂你，我以前是不喜欢人家骂的，后来我也喜欢别人来骂我。

为什么呢？

因为这样其实我不知道这么多东西，感谢别人给我看出不足的地方，像我以前做产品的时候，我一开始就收钱，那别人就开始骂我，你这个产品没做好你就想要收钱，你还缺这个那个呢，你这样太不道德了。我才知道，原来要收钱还要做三个功能才算道德，那我做了之后这个东西就开始赚钱，所以我就感谢大家的 Feedback。

怎么更新防火墙规则，你的判断标准是什么？

我有一套自己的价值观：你要诚实，你要透明。

这是我的价值观。

学习新的技能或者语言，最好是“付费”去学习？这个观点怎么看待？

你不付费是不会痛的，学习最好的方式是你认真对待这件事情，一般免费的通常你是不会认真对待的。因为虽然每个人的时间是最贵的，但是对大家来说，潜意识里最贵的还是钱。你给了钱，就逼着你认真学习，我发现我之前认真学习的都是我付了很贵学费去学的。

后来我自己教学生，也是发现同样的，我之前做那个免费推广 Rails 的时候，教了 60 个学生，但是只有最后一个人有兴趣，那么他升为职业的动力就不足，那我着急了 20 多个精英助教但是得到的效果就很差，后

来我开付费版的时候每个人都是付大钱学习的，每个人都变成职业 developer 因为他们够用心。

如何衡量自己是在走在下一个高峰的路上还是已经达到某个瓶颈了？

你每天都很开心地上班做这件事情，就是往高峰前进，每天做这件事情很无聊就是达到瓶颈。

目前的工作环境没有办法改变，自己的核心竞争力也不够强大。那除了靠自学还有其他方式么？刚才听到自学那部分只听到了靠自学并不够。多谢

没有，赶快换工作。

当你已经完成一次升级，通常有什么感觉或表征让你自己能马上知道呢？

通常是我灵感很多的时候，我一个礼拜发三四篇精华的文章，就表示我猛烈的升级，如果没有发文章的话就表示没有升级。

对年刚入职的年轻人，初始系统的升级有什么建议吗？

我建议你找有牛逼的人所在的工作，无论工资怎样都要去，你的目的是要更新你的系统。

[Tweet](#)

【转载】升级你的操作系统——郑伊廷

大家好！我叫鄭伊廷，大家在網路上都叫我 Xdite。我目前在新生大學跟笑來老師開設第一期的全棧工程師學習營。

我是個程序員，跟人家比較不一樣的是我除了熱愛程序寫作之外，我也熱愛文章寫作。

今天想跟大家分享的是一個我最近沈澱出來的一套系統性成長方式：

分享將分為五個部分：

- 第一部分 快置身到一個更優質的協作關係
- 第二部分 人的大腦就是一個操作系統
- 第三部分 從版本系統去看升級這件事
- 第四部分 改造自己的升級系統以及形塑的價值系統
- 第五部分 把自己的傳輸更新系統，更新為 BitTorrent 式的傳輸方式

第一部分：趕快把自己換掉到一個更優質的協作關係

最近一期，第 176 期的羅輯思維，羅胖請了他的老友張泉靈代班。這集節目裡面的內容講了什麼對我們的主題來說不重要，但這集最後 5 分鐘的結尾給了我很大的省思，羅胖給出來了一段分享，他對於家裡有孩子給高考的家長只給一種建議，就是讓孩子去上城市上大學。

而這個大學最好要在城裡邊，這樣才能很快適應中國的一線城市的生活，他認為這是大學必須要收穫的一項成就，即便選的專業差一點也是可以接受的。

為什麼呢？他從張泉靈身上體察的一點，一個人想要進步最好的方法是什麼？絕對不是玩命的讀書自學，在自己的體內累積信息和知識，而是把自己置身到一個更優質的協作關係中。比如說你今天在北京，上海，或者深圳，廣州的城市，在核心地帶，每天光做訂外賣、叫出租車這些事情。就這種響應和協作的速度，就會重新雕塑你的人生感覺。

對協作怎麼樣提高效率，人群應該怎麼樣互相打交道，對於這個所有圍繞你的服務和產品，怎麼樣高速的迭代，你會建立一個全新的感覺，在不知不覺當中，就已經完成升級。

這一段分享，對我來說，有很大的感受，這不就是我最近一個月的感覺嗎？我是上個月 16 號來北京的，這是我繼 2012 年中國的 Ruby 年會後再次重訪北京，笑來老師只不斷的跟我說，我真的得來北京發展。

於是我就搬過來了，這一個月我真是不斷的猛爆性升級，各位最近應該在新生大學公眾號不斷地看到我的文章出現，其實這也不是什麼軟廣，而是在我的腦袋閃光爆發的速度到了相當可怕的境界。我每天都有全新的沈澱，於是許多文章就這樣生出來了。

第二部分：人的大腦就是一個操作系統

這就讓我重新思考出另外一個比喻，人們要怎樣有效升級自己的作業系統。笑來老師曾經將人類的大腦比喻成一套電腦作業系統，我也想用這個角度更深的去解釋「升級」這件事。

在我的角度來看，其實每個人都有自帶的一套操作系統，出廠可能就是預設的。所謂的這個預設值呢？就是他誕生的那（前後五年的社會文化），以及（父母作為青年時期養育子女）幫時的價值觀。

比如說可能有人出場就是 iOS 5.1，有人出場就是 Android Cupcake ..有人是 OSX Leopard。這無所謂。因為每個國家文化都有自己不同的價值觀與文化，每個人都有自己人生的任務。

但是我們都知道，手機剛出廠不持續 upgrade 是一件很恐怖的事。為什麼呢？因為輕的很可能最新的 App 裝不上，但是重則是，社會上很多惡意人，會開發的「病毒」特別針對沒有下過補丁的手機作業系統下手。然後這些機器要不就是癱瘓了，要不就是變得很慢，要不然就是被當提款機.....

OK，這對我們來說都是常識。再接下來想，更可怕的事就來了，你平時升級你的大腦嗎？

很多人終其一生就是不升級他的大腦的，而且他還控制他周遭的大腦不准升級....。

第三部分：從版本系統去看升級這件事

在我們程序員界呢？有一個比較通用的版號控制發布規則，叫做 Semantic Versioning。好的程序都會用這套規則去發布他的軟體，讓其他人知道這次他的軟體做了哪些的變更，以讓他人決定要不要更新。

比如說版本號 8.2.1

- 8 是 Major version，指的是這個版本做了「不相容的 API 變更」。
- 2 是指 Minor version，指的是做了「向後相容的 API 變更」
- 1 是指 PATCH，指的是「向後相容的 Bugfix」

那常見的人生升級方式有哪些呢？

1. 在原先的朋友圈交朋友

做常見的可能就是交朋友吧。不過交朋友很看你的朋友圈以及當前你的價值觀。最常出現的狀況最多就是只到 minor version 階段的變更。

因為一般人很少跟自己不相容的人做朋友吧。所以更多的可能性是做到 bugfix，或者是產生更多的 bug

2. 工作

很多人出社會，快速提升的管道往往是透過工作。為什麼是透過工作呢？因為在公司的時間超級長，遠比自己的自學時間長。就算你頻寬再小呢，每天 8 個小時的下載，總也可以下載不少的補丁吧。

所以為什麼有人說，一定得挑自己喜歡而且有牛人的公司呢，因為有機會這樣半年一年呢，就升級了超多 minor 版本，甚至是累積到達 major 版本

3. 自學

其實在很多程度上，我是有點 anti 自學的。很多人總認為要升級總得「刻苦的自學」，我不以為然。我人生第一次最大的升級，是偶然跟大神級人物吃飯，從他身上我意識到了我的「自學」patch 速度是多麼可笑，那時候我天真的覺得自學應該是最有效的事，結果人家在牛逼的公司工作週週升一個 minor version。

那個禮拜當下，我就從我公司辭職了。這是我人生下過最值得一個決定，從那個決定開始後，我的成長速度像火箭一般的起飛。

4. 到外地工作

但是就算連 Major 版號開始顯著的變更，我還是不滿意這樣的速度。我的版本很快地就升到當地的頂點。於是我只好創業，但是創業又遇到了同樣一個問題，市場太小，升級太慢。偶然的一個機會下，我的公司加入了一間矽谷的 O2O 公司，我以前總天真的認為認真的開發程序，提高公司產品品質，公司就會賺錢。當公司成長時，項目陷入混亂，導入厲害的項目管理技巧，公司就會賺錢。

我在到矽谷時，工作所需我開始研究了 GrowthHack。是一套用技術去做運營以及營銷的一套科學。這個觀點從此改變了我開發產品的 major version 認知。從此我的 major 版本變更速度開始變得不一樣。

而笑來老師叫我搬到北京來後，同樣的版本快速前進一樣在發生。

5. 雇用牛逼的員工以及引入比自己厲害的同事

以前我作為經理，甚至是具有優秀能力的經理呢？也是迷信自己有相當不錯的能力，甚至是一度迷失在我應該將我的版本能力複製許多份，於是這樣就可以 scale。

所以我的方向總是挑選優秀的人加入團隊，再施予培訓。的確他們也有很好的成長速度，但是始終無法突破一個高度。就連帶我覺得我做培訓久了，我自己的更新速度也開始下降。

後來我的生意開始賺錢了，我開始嘗試僱用一些矽谷優秀的服務，雖然價格是我們本地的兩三倍高，但我心中總想知道，為什麼他們值得收這麼貴的價錢？而且還可以持續擴張。後來開始合作後，他們的結果真是讓我喜出望外，不僅品質高，我更學習到他們許多優秀的工作方法以及獨特洞見。

從那一刻我開始知道，要花錢要僱用人，就必須僱用比自己牛逼的人，而不是試圖去培訓員工，僱用牛逼的人，才能下載更多東西。而不是僱用比自己差但好控制的人，把自己的頻寬佔滿在上傳，而非下載。

第四部分 改造自己的升級系統以及形塑的價值系統

雖然我前面講了很多怎麼樣提升下載機會的方式。

但是還有一件事很重要，下載那麼東西也不是不可能下載到「病毒」。所以更重要的一個問題在於，當自己在拓展自己的下載頻寬以及下載機會時，你的網路防火牆規則是什麼？甚至是，你的系統核心演算法，又名「價值觀」到底是什麼。

價值觀是自己看待這個世界，以及做決策的方式。也是自己重複做一件事的準則。價值觀不是不會變的，甚至在系統升級過程中，它會一點點的被改變。

但是你真的得一直去思索「什麼是自己的價值觀」，甚至設下規則去保護最核心的價值觀，才不至於在社會網路中，被病毒趁隙感染摧毀。

第五部分 把自己的傳輸更新系統，更新為 BITTORRENT 式的傳輸方式

在前面的四個部分，我談到了很多東西，但是唯獨沒談「更新系統」。大家都喜歡「下載」，但卻忘了如今最高速的下載方式，不是「HTTP 下載」而是 BitTorrent 式的下載。

在 BitTorrent 式傳輸更新，有幾個特點：

- 只有大家都多多提供上传，下载的速度才能更快
- 所有客户端程序都对上传速度快的用户提供优先服务
- 如果你对上传速度进行了限制，实际上也就是变相限制了自己的下载速度

所以最後我總結一下呢，人人總是覺得更新自我的系統，是相當困難的。但是抓檔案這件事，我想應該是相對簡單的，以計算機的角度去比喻就是：

- 置身更好的網路環境
- 找到更厲害更豐富的下載論壇
- 維護防火牆規則、做好病毒碼更新，以及建構更強健的核心演算法
- 付費取得黃金下載通道
- 改為 BitTorrent 式協議更新

希望我的分享對各位有啟發！

[HTML 基础知识](#)

div/span 的不同

- <div>和一样可作为容器，没有特定意义。
- <div>元素是块级元素，可用于对大的内容块设置样式属性，还可以用于文档布局。
例如：
 - <div style="color:#00FF00">
 - <h3>This is a header</h3>
 - <p>This is a paragraph.</p>
 - </div>
- 元素是内联元素，可用于为部分文本设置样式属性。 例如：
 - <p>some text.some other text.</p>

class/id 的不同

- class 用来指向样式列表中的类，id 是用来给 HTML 元素设置唯一的编号。
- 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。
- 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。可以这么理解，id 相当于 iphone 的序列号，每台 iphone 都有一个唯一的序列号；而 class 相当于 iphone 的型号，一个型号可以有很多台机子。

p/br 的不同

- <p>是定义段落的元素，<p>元素里必须有内容。
则是空标签，里面没有内容。
- <p>有 opening tag 和 closing tag，
只有 opening tag,也可以写成
。
-
 标签只是简单地开始新的一行，而当浏览器遇到<p>标签时，通常会在相邻的段落之间插入一些垂直的间距。

如何使用 table 排版

- 将整个文档布局用表格的方式排版，通过调整单元格大小和位置来显示内容，然后将 border 设置为 0。
- 表格定义布局是老式方法，使用<table>元素进行文档布局不是表格的正确用法。<table>元素的作用是显示表格化的数据。建议布局还是使用<div>。

HTML 表格常见命令

- colspan 跨行（合并列）
- rowspan 跨列（合并行）
- cellpadding 单元格边距
- cellspacing 单元格间距
- bgcolor 背景颜色
- background 背景图像
- frame “框架”属性（①box 全框 ②above 上划线 ③below 下划线 ④hside 上下划线 ⑤vside 左右划线）

列表项目符号

- ul（①disc 实心点【默认】 ②circle 空心圆 ③square 方块）
- ol（①A 大写排序 ②a 小写排序 ③I 罗马数字排序 ④i 小写罗马数字排序）

[Tweet](#)

[fullstack-course 7/27](#)

Objective

课程：针对所有基础知识进行答疑

完成：实做 JOB 栏位（step3）、设置职缺隐藏（step4）、is_hidden 和 sidebar 的设置（step5）

Reflective

今天情绪：今天是三天来觉得最混乱的一天，但觉得也是学到最多东西的一天，感觉一天下来整个头脑很乱，但就像郑老师推荐的那篇文章说的那样，感觉自己今天拼了好多个拼图，所以还是想想还是挺兴奋了

高峰期：听到老师表扬我进步很快的时候，哈哈哈

低点：7 点左右，肚子超饿，头脑都不转了的感觉，情绪就很 down。感觉以后下午茶的时候一定要多吃点东西

Interpretive

课程主要内容

今天课程只要是回答一些答疑，主要内容都收集在 github 的 issue 里面，此外，我在补充一点点东西。

1. CRUD 里面有 ID，需要加 (params[:id]) 的有四个，show/edit/update/destroy 2.link_to 的写法 link_to("a", root_path) ok link_to ("a",root_path) error (次结构 link_to 后面不加括号) 3.ruby 里面大写的意义
2. def show
3. @group = Group.find(params[:id])
4. @posts = @group.posts.recent.paginate(:page => params[:page], :per_page => 5)
5. end

Group 是骆驼式写法，对应 model 4.order 顺序和倒叙的写法 ASC 顺序 DESC 倒叙 5.关于 method 和 action controller 里面的 method 叫 action 其他里面的 method 不叫 action 6.两个学习方法 ①多去看 log，log 里面有程序运作的所有的逻辑 ②在自己做的页面可以右击-inspect，这样就可以看到每个地方对应什么代码。一方面可以知道自己写的某部分代码呈现什么效果，另一方面更重要的是，可以知道某种效果是通过什么代码来实现的。如果看到某个代码不理解，直接 google 就可以学习积累了。此外，inspect 还可以用来检查错误，遇到报错，可以去 inspect 看看没有实现预期效果的地方的代码，去看问题出现在哪里。 7.redirect 和 render redirect 是重定向到某个页面，render 指的是回退 8.关于 model 的描述 model 就是把复杂的语言包装成人话，用 model 写就会 ruby 会将其翻译成底层的语法 9.gem "annotate" 这个 gem 超好用（据郑老师说），它可以直接显示 model 生成了哪些表单和路径。 使用方法：gemfile 里面添加-bundle install-在 iterm 输入 annotate 运行

重要领悟

1.先知道怎么做，多做几遍，做熟练了，以后在某个时间点自然会懂。（“很多程序员工作第一年只知道按老板的要求完成任务，都不知道自己在做什么”——郑老师）

2.反思了一下最近三天的学习，有很多不懂的东西还没有弄懂，但是我又知道不能过于把精力放在搞清楚这些东西上，因为现阶段懂的太少，用未知解释未知，大脑会疯掉。如何找到这个平衡很重要。所以想了个办法，从明天开始实践看看。具体方法写在“明天努力方向”里。

“学不学的好不重要，不用什么都懂，也不可能，会用最重要”

“不理解又重要的地方，先背起来 !!! 背起来 !!!!! 背起来 !!!!! ”

“不要着急，不要和别人比，记得关注自己的成长就可以。”

Decisional

形容今天的工作：混乱，但学到了很多。

明天努力方向：

“how-why-how”学习方式：先把任务做完，然后去看一些自己不懂的地方，查懂之后（可以用 rails 或者 ruby 加关键词搜索），再把代码写一遍。不用求所有东西一次都查到懂，也不可能，相信“日拱一卒，功不唐捐”。

阅读同学的学习笔记和 issue。

阅读之前做过的教材，从 step 中找一些关键步骤，记下来。

Q&A

Q：在做 job-listing 时出现点击 add a job 添加的内容却没有显示？

A：检查 controller/jobs_controller.rb，发现是这样的

```
def index
  @jobs = Job.where(:is_hidden => false)
  @jobs = @jobs.recent
end
```

这样的意思是被隐藏的文章是看不到的

改成以下代码即可解决

```
def index
  @jobs = Job.all
  @jobs = @jobs.recent
end
```

Q：view 出现错误的提醒

ActionController::UnknownFormat in Admin::JobsController#edit

Admin::JobsController#edit is missing a template for this request format and variant. request.formats: ["text/html"] request.variant: [] NOTE! For XHR/Ajax or API requests, this action would normally respond with 204 No Content: an empty white screen. Since you're loading it in a web browser, we assume that you expected to actually render a template, not... nothing, so we're showing an error to be extra-clear. If you expect 204 No Content, carry on. That's what you'll get from an XHR or API request. Give it a shot.

Extracted source (around line #56):

```
54     "That's what you'll get from an XHR or API request. Give it a shot."
55
56     raise ActionController::UnknownFormat, message
57   else
58     logger.info "No template found for #{self.class.name}##{action_name}, rendering head :no_content" if logger
59     super
```

Rails.root: /Users/lin/job-listing

[Application Trace](#) | [Framework Trace](#) | [Full Trace](#)

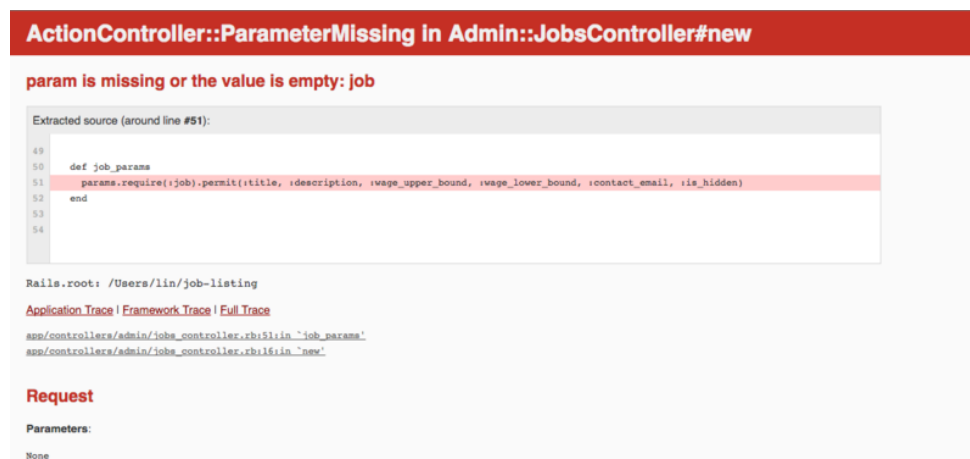
```
actionpack (5.0.0) lib/action_controller/metal/implicit_render.rb:56:in "default_render"
actionpack (5.0.0) lib/action_controller/metal/basic_implicit_render.rb:4:in "block in send_action"
actionpack (5.0.0) lib/action_controller/metal/basic_implicit_render.rb:4:in "tap"
actionpack (5.0.0) lib/action_controller/metal/basic_implicit_render.rb:4:in "send_action"
```

A：命名错误。

OK: edit.html.erb

ERROR: edit.html.erb

Q:controllers 里面 def new 的错误



A:CRUD 错误。new 的 define 不需要 (job_params)

OK :

def new

@job = Job.new

end

ERROR:

def new

@job = Job.new(job_params)

end

Q:如何添加按钮？

A：例如：

<%= link_to("Edit", edit_job_path(job), class: "btn btn-sm btn-default")%>

btn 是按钮的意思， btn-sm 是按钮的大小， btn-default 是按钮的样式。

按钮的大小可参考：搜索“grid”

按钮的样式可参考：[按钮样式](#)

Q:.和=的区别

A:.是执行， =是赋值。

<table class="table table-bordered">

<%= @jobs.each do |job| %>

Q:为什么会有这种神奇的东西出现 http://localhost:3000/admin/jobs.5?

A:正确的应该是

<%= link_to(job.title, admin_job_path(job))%>，

如果错误加了个 s， 即

<%= link_to(job.title, admin_jobs_path(job))%>

就会出现“神奇”的错误。

总结：message_thread_path(1) 這樣路徑會是/
message_threads_path(1) 這樣路徑會是.

Q:怎么理解 migrate 里的文件？

A:

```
class AddIsHiddenToJob < ActiveRecord::Migration[5.0]
  def change
    add_column :jobs, :is_hidden, :boolean, default: true
  end
end
```

add_colume 新建一个栏位，migrate 是指挥 model 的，所以要增加新栏位，一定要先在 migrate 里做（view 只是前端呈现，给浏览者看的，migrate 加了，数据库才能知道）

：jobs 指对象

：is_hidden 指名称

:boolean 指类型

Q:col-md-* 和 col-md-offset-*的关系

A：col-md-*指栏位的大小，col-md-offset-*指栏位向右移动多少

栏位一共分为 12 个，所以 col-md-10 col-md-offset-2 可以呈现这种效果

Job Listing			Hit, 123@123.123 -			
					Add a job	
123	2016-07-27 04:29:03 UTC		Edit	Destroy		
23	2016-07-27 04:27:23 UTC		Edit	Destroy		
123	2016-07-27 04:24:17 UTC		Edit	Destroy		
1	2016-07-27 04:23:37 UTC		Edit	Destroy		
123	2016-07-26 14:57:34 UTC		Edit	Destroy		

Copyright ©2016 Rails101
Design by linzhewei

因为大小为 10，向右 2 格，总数刚好为 12，所以效果是大小为 10，且靠右。
单纯的靠右还可以通过<div class="pull-right">来实现

Hil, 123@123.123 ▾

Add a job

	Edit	Destroy
	Edit	Destroy
	Edit	Destroy
	Edit	Destroy
	Edit	Destroy

Q:建立错误的 migrate 如何删除

A：可以用 rails d migrate 名称来删除

```
lin@linzheweiMacBook-Pro: job-listing ~ ruby=2.3.1 v1rsion3 rails g migration add_more_detail_to_job
Running via Spring preloader in process 5317
invoke active_record
conflict db/migrate/20160726145243_add_more_detail_to_job.rb
Another migration is already named add_more_detail_to_job: /Users/lin/job-listing/db/migrate/20160726144518_add_more_detail_to_job.rb. Use --force to replace this migration or --skip to ignore conflicted file.
```

要删除，只要输入 rails d migration add_more_detail_to_job 就可以了

[Tweet](#)

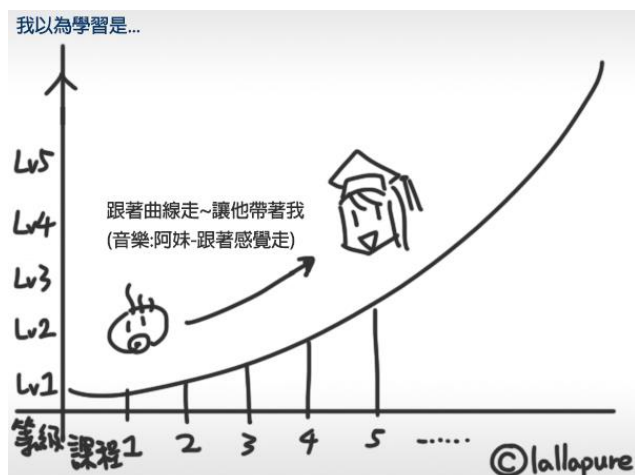
【转载】混亂是學習的常態

【原創文】混亂是學習的常態

今天要分享我學習語言、繪圖等事物上得出的心得，文章開始前先聲明這是純粹的個人經驗，如果你不贊同或對於個案分享沒有興趣的，請不要花費寶貴的生命往下看，或看完後來找我筆戰，我不會花任何時間討好、說服不相信的人，對於爭論也沒有興趣，那麼我們開始吧。

我從自身在學習遇到的瓶頸、困境以及解脫方案得出：

混亂是學習的常態。



一直以來我有一個學習上的迷思，認為學習是搬磚，按課本的進度帶領，我每學會一課便獲得一塊磚頭，磚

頭越堆越高，我就越爬越高，總走一天越過城牆到達天際我就成仙了，或突然會開口說流利的外語。

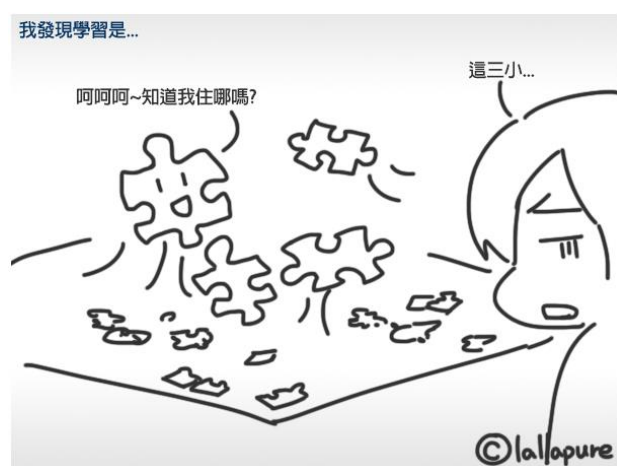
而我們的義務教育餵養學子的方式，也錯植並加深 " 學習 = 線性 " 的錯誤觀念。

以至於很多人到達第五塊磚，越不過去第六塊便放棄，因為在他們的大腦裡，始終深信學習就是一條往上的曲線，線不能斷，線一旦斷了，晉級/進步的可能亦隨之終止，因此主動放棄學習、放棄進化的可能。

我算幸運，資質尚可，在求學途徑沒碰過什麼實際困難。只是卡在一些有興趣的項目，如繪畫和學習寫程式許久，上不去就是上不去。今年 3 月開始學習韓文後又碰到一樣的困境：我爬上了第二塊磚，第三塊上不去，翻開課本發現自己是文盲，覺得要死了要死了。我繳的學費在哭泣，每堂課輪到我發言口說像公開處刑，第一次期末考不及格，好像 50 幾分吧...慘吶！

以前人家跟我說看不懂英文，我覺得對方在唬爛，現在人家跟我說看不懂英文，我感同身受！我收回以前對背不出英文單字同學的鄙夷，因為在困境中頓悟了學習的道理：

真正的學習不是線性，而是拼圖狀的。



今天你懂了一個概念，在這個概念上拚上一角，明天懂了那個概念，又拚上一角，手中的拼圖多了，自然就會串聯在一起，拼成一幅完整的版圖，知識越多，拚得越大，如此而已。

比如在韓語剛開始學發音和簡單詞彙的地方我就痛苦萬分，百思不得其解，為什麼有的助詞是에요，有的是

이에요，有的是입니다；到後來，一樣是地名後面的助詞，有的是接에，有的是接에서，這件事我也糾結了好一陣子差點不能自己。(好在我糾結的點，在幾堂課後就講解到，編教材的人滿厲害的嘛...哼哼)

後來我發現在不是主要學習目標上出現的文法或概念，只要用紅筆圈起來，記得有這回事，不要太糾結，先認同"它的狀態是正確的"並提起精神往下走，就能順利通過這一關卡要你學會的東西。

這種乍看有點隨便的學習態度，對於"學習 = 線性"是很大的衝擊，而且我們的社會又這麼要求正確答案，要求準確率，若抱著疑惑和不確定性繼續往走，很難敵過心中那個 " 我還沒搞懂、我還不會呀 " 的自我批評音浪。加上大家臉皮薄、又玻璃心，怎能容忍犯錯呢？乾脆兩手一攤，放棄求生，中斷學習成長之路，一了百了。這段敘述對多少人而言是慘痛又真實的人生經歷。(憐憫眼神)

以前寫網頁卡關，搞不懂為什麼這串 CSS 要這樣寫，我那樣寫不行嗎？我就會蹭很久，然後生出一堆自我懷疑的小惡魔站在肩膀上罵自己笨。

現在不揪了，先 copy/paste，東西能 work 就好，剪貼久了有一天，總有那麼一天！只要你持續在這個領域上耕耘，持之以恆努力不懈的學習(不能總是 copy-paste 耶!)，總有一刻你學的東西夠多了，或是剛好接觸到解開你結的關鍵、你就會頓悟。(講得很玄)

經驗是，當你到達下一個 level，前一個 level 困住的地方都會自然而然解開，所以真的不用太擔心學不會或不懂，除非你沒心想學啦(攤手)

學新的東西一定有很多不懂，在懵懂的狀態中要像逃難一樣死命拽著自己往前跑，因為你不跑，等洪水淹過來就掰掰了，沒有下輩子，學習失敗的心靈創傷，極少人有能力東山再起！這段混亂期就是我說的：混亂是學習的常態。怕死就別來玩！

不過混亂也是有等級之分的，對於新接觸的領域是一片空白，宛如一幅拼圖，從頭摸索一定是一片茫然，聰明點的人也是先翻找出邊邊角角或類似的花樣，一點一點湊上去。越熟悉的領域越得心應手，自然不若初期那樣瞎子摸象。

因此在學習新事物時，碰到鬼打牆期，絕對不要就此停下，找出你不懂什麼，為什麼不懂，並且記下來，這是一個很重要的步驟，學習最大的問題是：你根本連自己不懂什麼都不知道。(相信我，很多人連自己懂不懂也不知道，所以學習還是要花腦子的！)

一旦找出自己的困惑點，就能在排除這個因素的影響下繼續前進。等到有一天資源到位或你開竅的時候，把遺漏的拼圖補上，便能由老鷹般更高的角度飽覽美麗的風光，優遊翱翔於其中，天水共一色，快哉、快哉！人生樂事，莫過於此。

結論：放掉學習 = 線性的錯誤觀念，接受混亂是常態，擁抱學習新事物的混亂期，接著避免犯同樣的錯誤，用拼圖法補足空缺，然後享受漸漸上手的成就感吧 ~ Enjoy it !

寫作時間：約 90 分鐘，比想像中快耶，開心~

後話：

我是個水星在雙魚座，比較擅長感性想像和跳躍式思考的人，要抓緊腦中飛躍的思緒重組成有邏輯性的文章，對我是很大的考驗，文章用了一些比喻，這是我慣用且擅長的方式，希望這些比喻對於讀者理解我的意思有幫助。另外還加入冷笑話...矮由~人蔘就是要這樣才有趣呀

【感悟】学习还像画画，要先把轮廓画出来，先初步完成，再慢慢填充细节。最大的坑就是一开始就想追求完美的细节，最后连坚持下去的耐心都没有。(一开始细节够不够完美不重要，重要的是先画完)

学技术不一定要懂很多为什么 就像木工会干活达成目的就好，不用了解锯子锯到木头上摩擦系数多少 做功多少

[Tweet](#)

[fullstack-course 7/26](#)

Objective

课程：terminal 基本命令、git 基本命令、rails 结构简介、html/erb 基础知识

完成：JOB CRUD 的实做 (step1)、JOB admin CRUD 的实作 (step2)

Reflective

今天情绪：比较平静吧，心态平和地学习

高峰期：发现自己能比较熟练做 CRUD 的时候

低点：上课听到一些让自己懵逼的部分的时候

Interpretive

课程主要内容

1. terminal 基本命令

cd

ls -al

pwd

irb

rails s

rails g

rails c

rails c 和 irb 的区别？

irb 是模拟 ruby 的一些运行，即使看到执行 ruby 的结果。

rails c 会直接对资料库的内容进行修改。

2. git 基本命令

git init

git add README

git add .

git commit -m"message"

git remote add origin git://github.com/xxx.git

git push origin master

git push -u origin master

git status

git pull

git checkout -b "ch01"

git checkout "ch01"(这是一个大坑，会清空所有的修改，慎用 !!!)

3. html/erb

<%= @post.title %> 印出@post.title

<% @post.title %> 执行@post.title

4. 拆解任务

这一部分老师主要教我们怎么通过 user story 的方法来分析任务，怎么通过任务优先级和任务归类的方法来整理任务，最后通过任务细分的方法将任务拆解到可以开发的阶段，然后就可以开始写程序来开发了。

重要领悟

初学编程的过程中会遇到很多不懂的地方，最好的方法是先抑制住自己的好奇心，先做完，遇到实在不懂的就背下来。之后等较为熟练之后再深究为什么。

记住“学不学的好不重要，能用会用更重要”

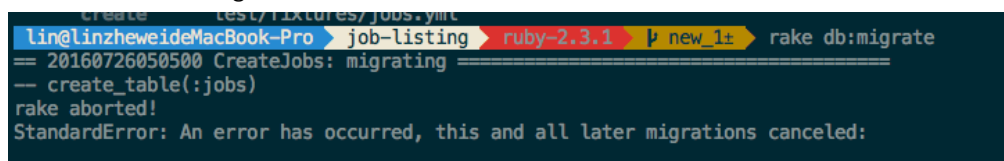
Decisional

形容今天的工作：之前的东西懂了一些，不懂还多，但没事，先做再说

明天努力方向：努力搞懂 job-listing 这个 project 的步骤和背后的思考方式

Q&A

1. rake db:migrate 的错误可以用三个步骤来解决



```
lin@linzheweideMacBook-Pro ~ % rake db:migrate
== 20160726050500 CreateJobs: migrating ==
-- create_table(:jobs)
rake aborted!
StandardError: An error has occurred, this and all later migrations canceled:
```

- rake db:drop
- rake db:create
- rake db:migrate **注意:** 这会清空掉资料库，包括之前提交的资料、admin 的资格等等都会没有掉。

2. 编程排除错误的重点

遇到 ERROR，一定要先排查完才能往下做，不然会变得很复杂。

养成打完代码就看下是否有错误的好习惯。一般来说，打完代码出现一行或几行提示的是正确；打完代码出现一堆 N 行提示的一般就是有 ERROR 了。

3. 在 iterm 中怎么看是否 commit 完成了？

```

lin@linzheweideMacBook-Pro job-listing ruby-2.3.1 p version-1 atom .
lin@linzheweideMacBook-Pro job-listing ruby-2.3.1 p version-1 git add .
lin@linzheweideMacBook-Pro job-listing ruby-2.3.1 p version-1 git commit -m"add validation to title"
[version-1 9b1e656] add validation to title
3 files changed, 14 insertions(+), 4 deletions(-)
lin@linzheweideMacBook-Pro job-listing ruby-2.3.1 p version-1 git add .

```

绿色表示 commit 完成，如果用 git status,会提示 nothing to commit.

黄色表示代码有增减修改，还没有 commit。

转载

- 关于 git 版本控制(回滚) 关于 git 的问题:之前设置 commit 节点,总想着 reset 回去.是因为对 branch 功能理解有误,它是在当前文件目录基础(包括所有改动)上 copy 一份,进行分支,所以通过 git checkout [branch_name] 也具备"恢复"的功能.
- before_action 和 before_filter 是同样的用法.
- 每一个 view 对应一个 action,对应一个地址
- 资料库里面的表格(table)是 model 的 classname 的负数形式.例如:Model 叫做 Duck,rails 就会自动去资料库读写'ducks'这个表格

[Tweet](#)

[fullstack-course 7/25](#)

Objective

课程：工程师思想（详见 Interpretive）、搭建网站框架雏形（bootstrap/devise/simple form）

完成：fork、网站框架雏形（bootstrap/devise/simple form）、pull request

Reflective

今天情绪：困并且兴奋

高峰期：笑来老师的分享

低点：早上挤电梯/(T o T)/~~

Interpretive

-课程主要内容

xidte

1.Computational thinking:

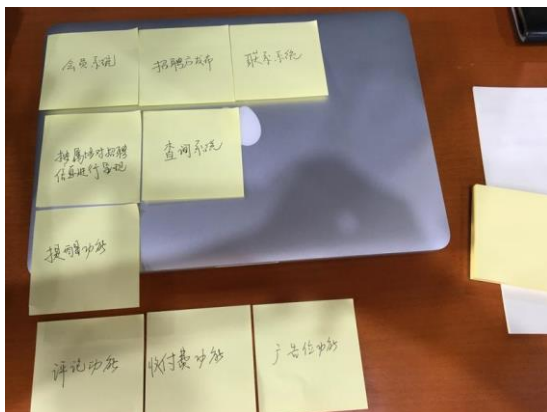
用電腦解決問題的思維

把大問題拆解成可以實作的小問題

在實作過程中找到資源

问题拆解实践—————

招聘网站拆解



说明：拆解网站，并按 must have,should have,could have,nice to have 进行时间层级划分。

2.debug 的方法：

通过五个问题来矫正自己的思维方式提问——

我遇到了什麼問題？

你想要怎麼解決

我希望別人怎麼幫助我

如果別人現在無法幫助我的話，我的解決方案

這個問題卡住的話，會讓我接下來無法繼續下去嗎？

(問題——自己的思路——求助方向——退而求其次的自我方案——問題是否影響進程)

3.User story

通过 user story 的思考方式，进行角色带入，更全面的布局整体结构框架。通过把握整体，防止过分陷入细节部分，而且更能促进项目协作与沟通。

李笑来

1.笑来老师讲话的一个重要结构：什么不重要，什么更重要。

[详见此文](#)

2.全栈工程师类比：全栈工程师可以独立做一个可以装水的“桶”（独立完成各种功能开发），不需要拥有所有木板，只要有组成这个“桶”需要的木板即可。十八般武艺样样精通是以后的事，现在先保证能把东西做出来。

3.学习方法论：不管学什么，都要用业余时间去研究那个东西的历史。

4.人生三大坑：①凑热闹②随大流③操别人的心

5.全栈班的一些学习建议：

①只看英文文档（“国内工程师的瓶颈不是别的，而是英文水平”）

②学编码背后的思考模式

③学的好不好不重要，能用就行

④教是最好的学，教别人能经历别人的各种错误（全方位而创新的错误），是性价比最高的学习方式

6.!!!! 人生最宝贵的不是钱，甚至不是时间，而是**注意力**!!!! 注意力觉得了为什么你是今天的你。（与信息格局乱论类似）

-重要领悟

编程的本质是解决问题，所以提出问题和解决问题的能力比编代码这件事更重要。

编码过程中遇到挫折要先思考，即使不会也要尽量先形成自己的思路（哪怕是错的）。然后在和助教、同学沟通，这样才是一直切磋、讨论，才能更快进步。拒绝做伸手党。

Decisional

形容今天的工作：初步接触、不是很懂、慢慢摸索

明天努力方向：努力搞清楚之前照着打的 project 的一些基础知识

Q&A

1.rails 的错误退出

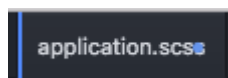
rails server 的正常退出：ctrl+c,没有 ctrl+c 直接退出 iterm 也是错误退出。

2.rails 错误退出如何处理？

ls -i tcp:3000 然後找到 PID 之後用 kill -9 <PID>(指读取到的正在运行的 PID)

3.atom 如何看文件是否已保存？

标签出现小蓝点，即未保存



标签无小蓝点，即未改动或已保存


```
application.scss
```

[heroku 上传出现错误](#)

ActionView::Template::Error (Missing partial groups/_group_item, application/_group_item with {:locale=>[:en], :formats=>[:html], :variants=>[], :handlers=>[:raw, :erb, :html, :builder, :ruby, :coffee, :jbuilder]}).

這邊的錯誤是說找不到_group_item 這個 partial 的檔案

用 command + t 檢查一下專案裡面有沒有這個檔案

发现檔名打錯字

`group_item_html.erb`

應該是

`_groupitem.html.erb`

[Tweet](#)

[ruby 1.9 and the new hash syntax](#)

旧写法：

`has_many :members, :through => :group_relationships, :source => :user`

新写法：

`has_many :members, through: :group_relationships, source: :user`

旧写法：

`new_hash = {simon => "Talek", :lorem => "Ipsum"}`

新写法：

`new_hash = {simon: "Talek", lorem: "Ipsum"}`

[Tweet](#)

[migration 的检查](#)

ActiveRecord::PendingMigrationError

Migrations are pending. To resolve this issue, run: `bin/rails db:migrate RAILS_ENV=development`

Extracted source (around line #572):

```
570 # Raises <tt>ActiveRecord::PendingMigrationError</tt> error if any migrations are pending.
571 def check_pending!(connection = Base.connection)
572   raise ActiveRecord::PendingMigrationError if ActiveRecord::Migrator.needs_migration?(connection)
573 end
574
575 def load_schema_if_pending!
```

Rails.root: /Users/lin/rails101-2

[Application Trace](#) | [Framework Trace](#) | [Full Trace](#)

```
activerecord (5.0.0) lib/active_record/migration.rb:572:in "check_pending!"
activerecord (5.0.0) lib/active_record/migration.rb:548:in "call"
actionpack (5.0.0) lib/action_dispatch/middleware/callbacks.rb:38:in "block in call"
activesupport (5.0.0) lib/active_support/callbacks.rb:97:in "run_callbacks"
activesupport (5.0.0) lib/active_support/callbacks.rb:790:in "run_callbacks"
activesupport (5.0.0) lib/active_support/callbacks.rb:95:in "run_callbacks"
actionpack (5.0.0) lib/action_dispatch/middleware/callbacks.rb:36:in "call"
actionpack (5.0.0) lib/action_dispatch/middleware/exceptions.rb:12:in "call"
actionpack (5.0.0) lib/action_dispatch/middleware/remote_ip.rb:79:in "call"
actionpack (5.0.0) lib/action_dispatch/middleware/show_exceptions.rb:49:in "call"
web-console (3.3.1) lib/web_console/middleware.rb:131:in "call_app"
web-console (3.3.1) lib/web_console/middleware.rb:28:in "block in call"
web-console (3.3.1) lib/web_console/middleware.rb:18:in "catch"
web-console (3.3.1) lib/web_console/middleware.rb:18:in "call"
actionpack (5.0.0) lib/action_dispatch/middleware/show_exceptions.rb:31:in "call"
railties (5.0.0) lib/rails/rack/logger.rb:36:in "call_app"
railties (5.0.0) lib/rails/rack/logger.rb:24:in "block in call"
activesupport (5.0.0) lib/active_support/tagged_logging.rb:70:in "block in tagged"
activesupport (5.0.0) lib/active_support/tagged_logging.rb:26:in "tagged"
activesupport (5.0.0) lib/active_support/tagged_logging.rb:70:in "tagged"
railties (5.0.0) lib/rails/rack/logger.rb:24:in "call"
sprockets-rails (3.3.1) lib/sprockets/rails/quiet_assets.rb:13:in "call"
actionpack (5.0.0) lib/action_dispatch/middleware/request_id.rb:24:in "call"
rack (2.0.1) lib/rack/method_override.rb:22:in "call"
rack (2.0.1) lib/rack/runtime.rb:22:in "call"
```

可以通过 `rake db:migrate:status` 来看是否每个 migration 都被执行了，如果没有，再检查是否因为其他原因导致，比如 key 的输入错误等，例如：

```

→ rails101-2 git:(ch05) rails db:migrate
== 20160719014641 CreatePosts: migrating =====
-- create_table(:posts)
rails aborted!
StandardError: An error has occurred, this and all later migrations canceled:

undefined method `inreger' for #<ActiveRecord::ConnectionAdapters::TableDefinition:0x007fdbf58b9c68>

```

关于 MVC，table 和 model 等概念



， table 就是在 database 裡， Group model 對應 Group table，， 要看 table， 可以 command + t 找一下 schema.rb

之前建出來的 table 全在裡面了

table 是做什么的， 他和 model 的关系是什么？

database 裡有很多表單，表單會有很多欄位記錄著不同資料，在 Rails 你可以透過 model 使用 ORM 的方式，讓你不用寫 sql 語法也可以去存取或修改表單內容，但如果要做複雜的 query 還是要寫 sql。這些表單就是 table，簡單來說 model 就是幫你跟 table 溝通去拿資料。

[Tweet](#)

@group = Group.find(params[:id])，Group 的 G 为什么是大写的？

在 define edit、create、update 等的时候，@group = Group.find(params[:id])，Group 的 G 大写的原因：因為在這個 edit action 底下，並沒有定義 group，而 Group 是呼叫 Group model，是拿資料是藉由 model 這邊去跟資料庫拿，所以需要大寫的 Group，除非有另外定義 group 是什麼。

groups_path 含义？

這個是 rails 裡面 url 的 helper，groups_path 會變成/groups，所以你的 url 會變成 localhost:3000/groups，然後會指向 groups 的 index action

你可以 rake routes 看一下

[Tweet](#)

@group = Group.find(params[:id])，Group 的 G 为什么是大写的？

在 define edit、create、update 等的时候，@group = Group.find(params[:id])，Group 的 G 大写的原因：因為在這個 edit action 底下，並沒有定義 group，而 Group 是呼叫 Group model，是拿資料是藉由 model 這邊去跟資料庫拿，所以需要大寫的 Group，除非有另外定義 group 是什麼。

groups_path 含义？

這個是 rails 裡面 url 的 helper，groups_path 會變成/groups，所以你的 url 會變成 localhost:3000/groups，然後會指向 groups 的 index action

你可以 rake routes 看一下

[Tweet](#)

@group = Group.find(params[:id])，Group 的 G 为什么是大写的？

在 define edit、create、update 等的时候，@group = Group.find(params[:id])，Group 的 G 大写的原因：因為在這個 edit action 底下，並沒有定義 group，而 Group 是呼叫 Group model，是拿資料是藉由 model 這邊去跟資料庫拿，所以需要大寫的 Group，除非有另外定義 group 是什麼。

[Tweet](#)

groups_path 含义？

這個是 rails 裡面 url 的 helper，groups_path 會變成/groups，所以你的 url 會變成 localhost:3000/groups，然後會指向 groups 的 index action

你可以 rake routes 看一下

@group = Group.find(params[:id])，Group 的 G 为什么是大写的？

在 define edit、create、update 等的时候，@group = Group.find(params[:id])，Group 的 G 大写的原因：因為在這個 edit action 底下，並沒有定義 group，而 Group 是呼叫 Group model，是拿資料是藉由 model 這邊去跟資料庫拿，所以需要大寫的 Group，除非有另外定義 group 是什麼。

groups_path 含义？

這個是 rails 裡面 url 的 helper，groups_path 會變成/groups，所以你的 url 會變成 localhost:3000/groups，然後會指向 groups 的 index action

你可以 rake routes 看一下

@group = Group.find(params[:id])，Group 的 G 为什么是大写的？

在 define edit、create、update 等的时候，@group = Group.find(params[:id])，Group 的 G 大写的原因：因為在這個 edit action 底下，並沒有定義 group，而 Group 是呼叫 Group model，是拿資料是藉由 model 這邊去跟資料庫拿，所以需要大寫的 Group，除非有另外定義 group 是什麼。

groups_path 含义？

這個是 rails 裡面 url 的 helper，groups_path 會變成/groups，所以你的 url 會變成 localhost:3000/groups，然後會指向 groups 的 index action

你可以 rake routes 看一下

@group = Group.find(params[:id])，Group 的 G 为什么是大写的？

在 define edit、create、update 等的时候，@group = Group.find(params[:id])，Group 的 G 大写的原因：因為在這個 edit action 底下，並沒有定義 group，而 Group 是呼叫 Group model，是拿資料是藉由 model 這邊去跟資料庫拿，所以需要大寫的 Group，除非有另外定義 group 是什麼

groups_path 含义？

這個是 rails 裡面 url 的 helper , groups_path 會變成 /groups, 所以你的 url 會變成 localhost:3000/groups , 然後會指向 groups 的 index action

你可以 rake routes 看一下

[@group = Group.find\(params\[:id\]\), Group 的 G 为什么是大写的 ?](#)

在 define edit、create、update 等的时候, @group = Group.find(params[:id]), Group 的 G 大写的原因 : 因為在這個 edit action 底下, 並沒有定義 group,

而 Group 是呼叫 Group model, 是拿資料是藉由 model 這邊去跟資料庫拿, 所以需要大寫的 Group , 除非有另外定義 group 是什麼。