September 11, 2016 | Posted by 王梦琪

16 年最重要的决定

一个重要的决定

2016 年的七月份，我还是一只有理想的产品汪，尽管被现实屡屡打击，一直希望给自己成长升级大换血，看到新生大学的发文，我特别的激动，觉得这是我的机会：
我是编程 0 基础，大部分的 0 基础培训看上去都是培训基础知识，而我想实战；我希望学习全栈的技能，而不是单纯的前端后端 ;google 了教练 xdite 老师，浏览了 Xdite 老师的 blog，发现教练非常的牛；相信李笑来举办新生大学软件学院的诚意…
但同时对于报名这件事，我是很纠结的：
这个价格对于毕业两年的我还是非常高的，几乎是大半年的收入了；毕业之后我就直接回了家乡（二线城市），内心对去一线城市还是有很多的恐慌和不确定性；去得话几乎只能离职，而这是我付出很多心血的第一份工作，内心是有很多感情的；学完 rails 之后我所在的城市几乎没有对口的职位，不知道出路在何处…
留在原地是安稳的选择，风险小，无收益，去远方是冒险的选择，风险大，收益大。最终我决定给自己一个机会，放弃安全感，拥抱可能性，于是我报名了。

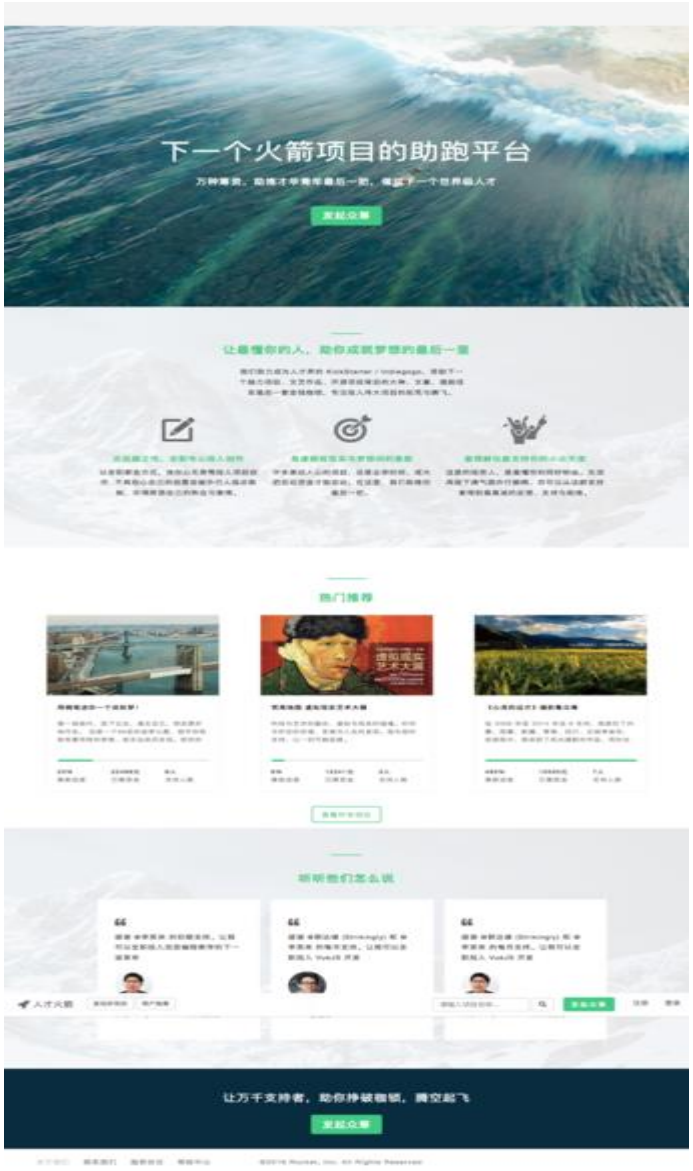
疯狂的过程

刚来的第一天，北京下了 2016 的最大的一场暴雨，提着行李箱，爬着地铁坏了的电梯，想到被子床单都在奔波的路上…第一天真是凄惨无比，北京欢迎我的方式真独特。
培训开始了，没搬家之前，我几乎每天就睡 5-6 小时，剩下时间都在学习，尽管累，但是每天都非常兴奋，刚进入编程领域，万事万物都是全新的，创造是这个世界上最能带来快感的事。
两个月的高强度训练，写 code，修 bug，记录成为我的日常，朝 9 晚 10，晚 11，晚 12 都是常有的事，经常写着写着就忘记了时间。周末休息一天就感觉非常愧疚。这种状态从最初持续到了最后。

我的改变

编程能力

我现在可以独立开发 web 端中型网站，会写前端 css，html，query，熟练运用 ruby on rails。
最后 4 周，我们团队 5 人完成了人才火箭项目，完成了从最初的使用 user story 分析拆解需求,到开发,到使用 on boarding 收尾,seo 优化,及测试,性能优化及上线的整个流程，以下是我们的作品（下图是我实作的 landing page）：



地址：https://talent-rocket.herokuapp.com

观念

和 code 能力同等重要的是观念的大幅改变。甚至，在我看来这是更重要的事。之前有多少观念在脑海里洗涤不清，乱做一团。

xdite 老师我是我接触过的大大牛，而慢慢我才发现，牛不仅是技能，而是在观念上就与我远远岔开了。接触到这样的牛人，为我的世界打开了一扇窗，让我看到了窗外的世界，与屋内的世界有多不一样。如果没有跳出来，就永远看不到差距。

做对的事

做对的事比作对的是更重要。但是往往你是不知道什么是对的事，什么是对的观念。而在这里，不间断地接受正确的理念。

比如幸运的公式是“1.看说明书。2.做更高级的决定”。更高级的决定是你当下无法通过自己的判断得出，而是比你厉害比你专业的人给予你的建议，例如，如果你的工作不能让你在感兴趣的领域成长，马上换工作，哪怕待遇好。可能你当时并不能理解，实际做了，才能体会到这个决定带来的红利。其实道理很简单，如果你从事感兴趣的事业，每天投入的时间是

“8+2”小时，如果你只在业余时间从事感兴趣的事业，每天投入的时间是 2 小时。所以每天的差距是 5 倍。日积月累，能力积累何止翻 5 翻？
比如，安全感不是来自于你的工作，你的另一半，你的银行卡上的数字，甚至是你和别人一样，而是你的能力，你的视野。井底之蛙当然会担心没有食物，因为它只守着一口井，不能控制井的干涸，天气的情况，只能听天由命。当它跳出来，才发现安全感来自于捕食的能力。
学习理念
编程学习就是拼图学习+联系套路（肌肉记忆）+成就感
很多人认为学习是登山，但如果真正去学的时候，大多时间没有攀登的感觉，不是匀速向上，而是混乱。
这是由于学习不是线性学习，而是拼图学习，混乱是常态。
在拼图学习中，最重要的快速拼出框架，而不是纠结于细节。快速联系套路是掌握拼图框架的最好方式。在混乱的过程中，正反馈是必须的，这样才能坚持不断地前进。
真正高效的学习法——
喂给学生能够有效、够多量「启动思考」的基础知识
这句话的重点在于「启动思考」而不是「基础知识」
甚至是要教给新手「可启动」的「套路」，而「事实型知识」留待新手自己补完
所谓「可启动的套路」，实质上是 一个有效连结「高频存取知识」（由教师统计出）的完整成果
然后，练习练习再练习，总共练三遍。烧进长期记忆中。
甚至迁移迁移再迁移，使得学习者建立可以在陌生场景中，轻易辨识一个新问题（即便以前没有建构）的内在结构的能力。
https://gist.github.com/xdite/bb0dd0aa0e3f986089a5481bf5c013b7
这套模式让我开始可以独立开发，而且不可思议的是，学习了 rails 之后，再学 css 和 jquery 居然没有在感受到陡峭的难度，完全可以自学，通过重复练习小的套路，然后上手，实际效果很不错。
商业理念
xdite 老师的 growth hack 整套理念改变了我的商业价值观。原来如果努力做一件还是赚不到钱的话，不是要更加努力，而是要思考转变道路，如果还是不能成功的话，就要考虑所做的事情是否正确。
这套源自硅谷的概念，经 xdite 不断深挖，成熟可实操，第一次听到振聋发聩。比如传统的互联网营销注重的是拉新，而在 growth hack 体系里，这只是 acquisition，远没有 referal 和 activation 重要，后两者可以改变商业模式的成长系数，让收益成本比翻几倍。
回顾
不敢相信，时光过去了两个月。这两个月是我人生中感受最长的两个月，时光仿佛无限拉长，现在谈起一件发生在 5 月份的事，仿佛发生在去年。我们感受的时光不是 24 小时制，而是时光的密度，事件的浓度，所以这两个月发生了大多，更像是是一年的强度。
2 个月后，我庆幸做了这个决定，这无疑是我 2016 年最重要的决定。我发现，来全栈培训营是一个投资回报率超高的投资，收益比我想象中大，无风险。
2 个月的时间足以让我焕然一新。
September 9, 2016 | Posted by 王梦琪
第七周的总结
团队的利益远高于个人利益
rocket 项目上线了。特别感谢我的战友们，5 个 rails 0 起点的队友能把网站从 0 在四周内搭

建起来。

我认为原因在于成员三观一致，**认同团队利益远高于个人人利益，个人的成果通过团队展示，所以没有人炫技，每个人都负责任，只为把网站做好。**在实际开发过程中，当然大家会产生争执，但是由于三观一致，所以对事不对人，可以用最小的成本把事情解决。尽管上周我说过更喜欢做后端，但是团队目标需要我做前端的时候，一定义不容辞。

地址：https://talent-rocket.herokuapp.com

只有每个人都以团队为先，大家在一起才能做出最棒的成果。

给自己最利于成长的外部环境

你最想要的是什么，如果是成长，请给自己最利于成长的外部环境！对一个最在意的是自我成长的人来说，对自己好不意味着最好的物质享受、声色犬马，**而意味着最好的外部环境，**

最好的队友，持续给你正反馈的环境。在这种环境中，能放大你的成果，让你的成长速度提升至 200%。所以为什么不花时间？不花钱？不放弃代表稳定的安全感？你值得最好的！
不要追求完美主义，时间成本使最昂贵的，优化的工作应该在迭代的时候做。
应该用最短的时间完成 mvp，前期不应该花太多时间追求完美主义，做到 85%即可，因为 85%-99%之间是成倍的工作量，而从 99%-100%几乎是不可能的。做到 85%可以极大地压缩时间成本，相当于给自己留下了巨大的时间资产，才有可能在固定的时间内完成整理，测试，根据运营迭代等有价值的事，最重要的是按时达成有质量完成上线目标。
在迭代前要开始进行前后端代码整理工作，便于团队合作改 bug 及功能迭代，不然后续加功能会很痛苦，易出错。其实代码整理是很爽的事，可以回顾之前的工作，删繁就简，大大提升网站性能。
最后是感谢
2 个月马上就结束了，有点伤感，更多的是收获还有感激。
感谢 xdite 老师，你升级了我的观念，让我看到更大世界的可能性，是我的 role model。
感谢 rocket 团队，学霸，nfree 哥，liber 和李师傅，能够和你们合作，我感觉很幸运。感谢给我鼓励和帮助的小红老师，eve 老师和 yy 老师。感谢曾经帮助过我的同学们：DJ，班长，lily，聂师傅，思宁，国锋....感谢全栈营的你们。
最后，感谢自己。感谢自己的努力带来的收获，在不断成长的过程中不断地增加信心。继续加油，成长无止境！

September 5, 2016 | Posted by 王梦琪

第六周的总结

1. 只学最小必要知识并实践是节省时间和快速上手的最佳方法，如果想要进阶，需要学更多的内容。
   本周一半时间大战 css，由于 css 学了 code school 上的一个课程（css cross country）就直接开始写，写的过程是结合学的内容，使用 chrome 的检查研究别人的网站，使用工具画图算间距，然后动手。
   如果不是学完一个课程就开始写，到今天应该还是在学习。
   但同时，由于学的内容较少，所以一旦局部页面复杂程度增加一倍，时间成本至少会变双倍。所以一天差不多写一个 3 屏页面。
   另外由于学的比较浅，前端页面代码可维护性比较差，结构比较冗余，如果改版，需要进行代码重构。
   如果有时间的话，应该不断进阶，再继续写页面，这样后续能够节省更多的时间。
2. 确定规则的讨论才有意义，否则是无效讨论。
   比如头像功能，我们讨论了很久，讨论不出结果的原因是因为大家没有统一标准，各自的观点建立在自己的标准上。
   因为讨论的虽然是具体功能，但是实际问题在于价值排序， 比如说可能有些人认为用户体验更重要，有些人认为页面信任度更重要，有些人认为实现成本高低更重要。自然很难确定。
   此时统一标准就显得尤为重要。
3. 做对的事情比把事情做对更重要，所以要确定什么是对的事情
   方法论特别重要，没有方法去做事，事倍功半。比如回到头像功能，xdite 老师周五分享了 growth hack 的高阶方法。先算这个功能的价值，再算这个东西的实现成本，

再去更其他功能进行比较，做性价比最高的事。如果大家都认同这套方法论，将所有的功能一一列出算出得分，这种细节功能就没有争论的必要了。

4. 尝试了才知道自己喜不喜欢
   之前很多研发同学说前端有趣，好玩，易上手，推荐女生转前端。
   但是实际去做后我发现，比起前端我更喜欢后端。前端更琐碎，更多条条块块，细节更多。
   后端更重逻辑和流程，做起来会让我更有成就感。

September 4, 2016 | Posted by 王梦琪

[第六周好用的工具]

atom pakages, atom 插件

tree-ignore

隐藏文件：创建一个.atomignore 文件可以将 treeview 不常用的文件夹隐藏。

git-time-machine

使用快捷键 alt+t 查看该文件的所有历史版本，并和当前版本对比。

特别适用于协作。

git 快捷键设定

在 atom 里的/.git/config 文件里加入

[alias]

ci = commit -m

co = checkout

st = status

br = branch

此时就可以在 iterm 里用缩写打出对应命令。

iterm 命令

rails c 里查看一个对象的所有方法。

例如：a = User.all

a.methods.sort

September 4, 2016 | Posted by 王梦琪

[第六周的大坑]

xdite 老师分享说有把自己之前实现问题的整个过程都记录下来，不仅记录对的还有记录错的，好处是：

1. 帮助攻克难题 :方便查看之前的步骤, 避免步骤太多忘记之前的思路和尝试的方法。
2. 避免以后犯同样的错误：错的东西一旦记录下来以后会犯得可能性大大降低了。
3. 便于回顾：以后可以回顾自己是如何解决问题的，回顾当时的思路和方法。
4. 可以帮助遇见相同问题的开发者。

实现步骤

本周有一半时间在做前端，前端的坑主要还是排版的问题，这里记录一个后端的功能。

功能：admin 创建一个项目的时候通过输入 user 的 email 的方式指定用户（说明是哪个用户创建的）。

我的想法是：admin 在新建项目页输入 email 提交时，将 email 参数传入 controller，在数据库里找对应的 user_id，找到了就将 user_id 存到 project 表里，找到后进行其他栏位的判断，找不到在当前页面报错。

步骤：

1. 第一步：在 view 里使用 simple_form 将 email 这个参数传到 controller
   因为在 project 这个 model 里没有 email 字段，所以 google "rails simple form not"，此时出现最佳匹配句"rails simple form field not in model"
   找到用法：在 model 里添加 attr_accessor:user_email，在 view 里就可以写<%= f.input :user_email %>
2. 第二步：在 controller 里取出 email 的对应 value,并进行判断。
   第一稿：

```
def create
@project = Project.new(project_params)
@user = User.where(email: params[:user_email])
if !@user.present?
    render :new
    flash[:alert] = "无此用户"
else
    @project.user = @user
    if @project.save
        redirect_to admin_projects_path
    else
        render :new
    end
end
end
```

此时出现坑

使用：@user = User.where(email: params[:user_email])

取不出来 user_email 参数，去看 rails log 出现:

<ActionController::Parameters {"utf8"=>" ✓ ", "authenticity_token"=>"kAzFJ7l7dzx9j+pZxlQbNT+9CEexlmT/6wfjRgL6w5hZmOKt+R/ep6KVuC8M9se46fapQ1Ulyy53sMlNXvSDJA==", "project"=><ActionController::Parameters {"name"=>"22", "description"=>" 22 ", "user_email"=>"1", "category_id"=>"1", "fund_goal"=>"33", "video"=>"3"} permitted: false>, "commit"=>"创建项目", "controller"=>"admin/projects", "action"=>"create"}

发现是 hash 套 hash 结构，应该现将:project 对应的 value 取出，再将:user_email 对应的 value 取出。如下：

@user = User.where(email: params[:project][:user_email])

此时又发现

报错 AssociationTypeMismatch @project.user = @user

使用 binding.pry 发现可以取出@user，但是取不出 @user.email.看来是数据类型不对，发现使用 where 得到的是一个 array，而使用 find_by 得到的是一个值。

在 rails log 里 使用 where 是

SELECT "users".* FROM "users" WHERE "users"."id" = ? ORDER BY "users"."id" ASC LIMIT ? [["id", 1],

而使用 find_by 得到的是

SELECT "users".* FROM "users" WHERE "users"."id" = ? ORDER BY "users"."id" ASC LIMIT ?

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

至此出了第二稿，正确。

```
 def create
@project = Project.new(project_params)
@user = User.where(email: params[:project][:user_email])
if !@user.present?
   render :new
   flash[:alert] = "无此用户"
else
   @project.user = @user
   if @project.save
      redirect_to admin_projects_path
   else
      render :new
   end
end
end
```

August 28, 2016 | Posted by 王梦琪

最快做出 mvp 的方法是什么

rocket 项目从开始到如今近 3 周了，在没有前端，没有设计的 5 个 ror 新手努力下（好吧，有一个做了 2 年 java 的学霸），居然进展了 80%，到了修 bug，完善用户体验的阶段，实在是很神奇。

这一切来源于 xdite 老师的方法。

1. 每天拿出一个完整展示当天成果的产品 gif，
2. 每周拿出一个能完整展示本周成果的 presentation
3. 每周结束时确定下周任务，
4. 每天完成 tower 任务加开晨会
5. done.

这个方法最重要的是一二两条！正是这些简单的方法，带来了莫大的好处。

很多互联网公司做产品的进展很慢，一种经常出现的情况是，产品做完交接给设计，设计做完交接前端，研发自己默默地写，直到一个月研发套完页面后，我的天，产品发现逻辑跟设想的完全不一样！

而做以上的方法的好处是:

1. 目标感：团队目标统一，每个人的目标都是做成“能动的程序”，提供团队作战能力，保证大家对当前的项目有贡献，避免团队成员走偏。
2. 解决沟通问题：每天/每周都有结果汇总，团队成员都可以看到最新进度，避免上述的消息不对称及沟通不畅的问题，可以快速纠偏，避免一个月之后才发现问题。
3. 成就感 :增强团队成就感，每天每周的展示让每个成员都意识到自己对项目的贡献，成就感本身就是对开发者的莫大激励。

事实上，上述方法的本质就是**“展示所有成果”**，所谓一招制胜，想到之前开发过程中的辛酸泪，真是无语凝噎呀。

August 28, 2016 | Posted by 王梦琪

第五周的大坑

ror 入门之后发现程序出错，踩到坑也不算什么大不了的了。因为只要是坑必然可解，只要能解，都不算特别了不起的大坑。所以本周最大的坑不是程序方面的。而是更深入领悟了一个道理：

学 code，状态非常重要。

状态分为两个方面：一方面是注意力，一方面是心态。

注意力：最影响注意力的是体力，所以千万不要熬夜。人的精力（体力）是有限的，第一天透支了，第二天势必会效率下降。

所以错误一：这个月停止运动，整取使用尽可能多的时间写 code。

错误二： 第二天熬夜至 2-3 点，第二天 7 点起，导致第二天效率极大下滑，第一天的夜算是白熬了。

要想保证注意力，体力是根本保证，要好好吃饭，好好睡觉，好好锻炼，尽量减少生病的概率。

心态：心态是意愿，想保持好的心态，秘诀很简单。不要被闲杂事干扰情绪，不停地找成就感。在每天晚上拿出一个让自己有成就感的成果，就是对自己最好的激励~ 同时在明白这个道理的时候，也是提醒自己每天要特别关注注意力管理，把当天重要的事情做完，做好。

August 17, 2016 | Posted by 王梦琪

第四周的大坑

1. 建立 model 关系后导致的坑

   分别建立了 plan 和 project 的 model 后，在 plan 内加入了 project_id 栏位。分别作了 plan 和 project 的 CRUD，功能均可以用。
   当加入了 belongs_to 和 has_many 的关系时，plan 的 create 不可用，原因是此时未在 controller 里加入 plan 和和 project 的关联，所以 plan 新建时找不到 project_id
2. git 命令

   git 命令一定要看说明书，不能看见命令想当然。
   比如 git pull origin master
   git push origin master
   此处第一处 master 指的是远端的 master 分支，第二处指的是本地的 master 分支。
3. 将项目文件夹丢掉 trash 导致
   将项目文件夹丢到了 trash 之后，重新 clone 远端后，忘记重开 rails server，所以此时 server 捞的是 trash 里的。所以无论修改什么 localhost 都不显示。

建议在问题处将内容输出检查，比如

在判断式@plan.price < @project.fund_goal 判断式不生效时，执行输出，测试变量是否正确。

puts "#{@plan.price} ----- #{@project.fund_goal}"

August 17, 2016 | Posted by 王梦琪

git pull、git fetch、git push

合并远程分支
简单说“pull = fetch + merge”
fetch（推荐）
把远程分支 C(远程主机名为 origin)拉到一个分支 D，确认无误后再合并到自己的原来分支 B。

1. git fetch origin C
2. git branch -a (查看全部分支，包括远端分支，假设里面有远端分支缓存 remotes/orgin/E)
3. git checkout -b D origin/E （远端分支必须装到本地分支才能取出）
4. 有冲突，git status
5. 在 atom 里面改对应文件。head 下是你的，往下是远程的。
6. 如果 schema 错了， rake db：migrate， 重开 rails server
7. 提交 git add . git commit -m “”
8. 切换至 B git merge D

pull 高级版
把远程分支 C(远程主机名为 origin)拉到一个分支 D，查看后再合并到自己的原来分支 B。

1. 切至 D，git pull origin C
2. 有冲突，git status
3. atom 里面改对应文件。head 下是你的，往下是远程的。 4.如果 schema 错了， rake db：migrate， 重开 rails server 5.提交 git add . git commit -m “” 6.git checkout B 7. 执行 git merge D

pull 简易版
把远程分支 C 拉到合并所在分支 B。

1. git pull origin C
2. git status
3. 在 atom 里面改对应文件。head 下是你的，往下是远程的。
4. 如果 schema 错了， rake db：migrate， 重开 rails server
5. 提交 git add . git commit -m “”

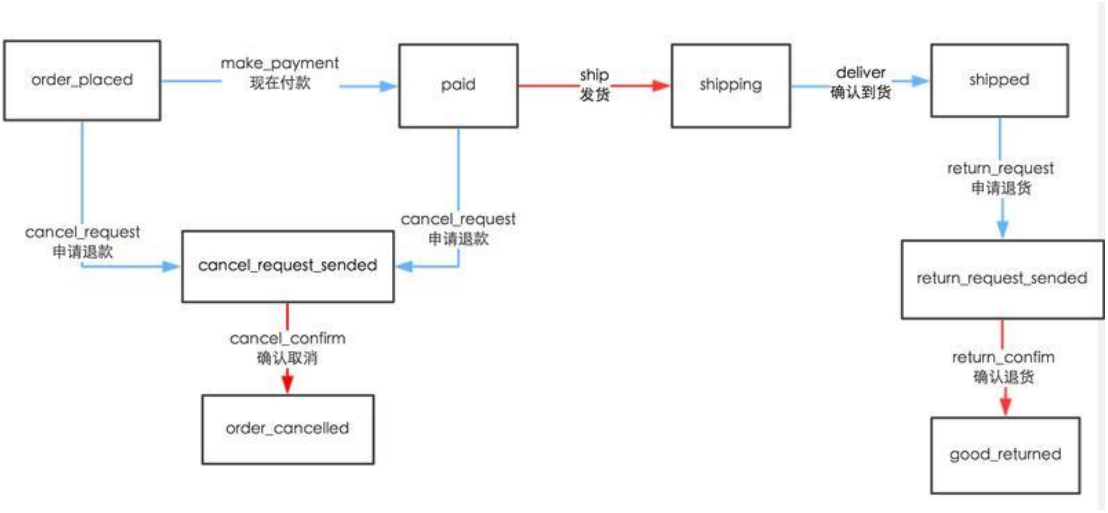
push
本地分支 B 推送至远程终端
1、将远端拉下来（参考 fetch 推荐）
2、再 git push origin B

August 13, 2016 | Posted by 王梦琪

第三周总结

本周上课第一天就完全懵了，什么，购物车？各种条件限制？ 我还只会做 CRUD 啊，真后悔上周周末休息了一天半。
拼命一周，到第五天回顾的时候，发现自己弄懂了参数传递，会看 rails log，弄明白了数据

库基础语法，搞明白了状态机（下图为订单流程图）终于可以给自己点个赞。不由得说前几天惶惶不可终日，连睡眠都睡不够的状态太夸张了些。



于此明白了**学习就是"懵懵懵懵懵"然后"哦哦哦哦哦"的循环往复的状态，懵是因为给自己高于处理能力的信息量，每天大概只懂了 70%；哦是因为把所有的知识处理完毕，将所有的拼图拼了起来（知识练了起来，融会贯通）。而这种是快速学习的捷径。**

最重要的心态是**"不要慌，不要慌，不要慌"**，最重要的态度是**"专注，拼命往前"**。

截止目前已经完成了一个有"admin 后台管理产品，管理用户，管理订单（确认退款，确认退货，发货）；user 前台查看产品，购物车，下单，收到邮件通知，管理我的订单（付款（api 没有做），申请退款，确认收货，申请退货）的一个 project。从一行代码没写过，到现在出了一点内容，对自己慢慢有了信心~。这个世界上的天花板往往是自己给自己设定的，下个礼拜继续和伙伴们一起成长~，加油~~~~

第三周最棒的概念或工具

推荐工具：

1. process on
   画流程图禁止不能更棒！
   详细内容见：https://www.processon.com/
2. 生成表关系的 gem Rails-ERD
   会有一个表关系结构图（表里有重要字段）
   1.安装 Graphviz： brew install graphviz
   2.安装 Rails ERD,在 Gemfile 里面修改如下：
3. group :development do
4. gem "rails-erd"
5. end
6. bundle install

3.在终端里执行 rake erd

会生成一个 pdf 文件 erd.pdf，在你的项目文件夹里~

详细内容见：http://rails-erd.rubyforge.org/install.html

本周重要概念：

总结！现在可以开始总结了，方法的应用，参数的传递这一步都有开始总结清楚！！！现在不总结，下一步写的时候会糊涂。

此处先填个坑，回来补参数和方法之我的看法。
August 12, 2016 | Posted by 王梦琪
8 月 10 日总结
今天有两个重要收获：
1.搞明白了参数是如何传递的！
明白了参数的传递路径，重点是怎么取出来，怎么存进去。
2.会看 log 了，并且搞懂了基础数据库的语法！
Started PATCH "/carts/1/change_quantity" for ::1 at 2016-08-10 18:50:04 +0800
Processing by CartsController#change_quantity as HTML
Parameters: {"utf8"=>" ✓ ", "authenticity_token"=>"fV/6ERuggXqHgOcqr8D478rpk+1tmBGmK4ftHQ8jIEXRSML/ysQq/rAIHelyXzWuQ25ALVXepeSJrHSoS3JieQ==", "cart_item"=>{"quantity"=>"2"}, "id"=>"1"}
DEPRECATION WARNING: `redirect_to :back` is deprecated and will be removed from Rails 5.1. Please use `redirect_back(fallback_location: fallback_location)` where `fallback_location` represents the location to use if the request has no HTTP referer information. (called from change_quantity at /Users/apple1/jdstore/app/controllers/carts_controller.rb:41)
Redirected to http://localhost:3000/carts
Completed 302 Found in 1ms (ActiveRecord: 0.0ms)
这一段的重点内容是参数，其中要在 controller 取出 quantity 的值要如下取
@quantity = params[:cart_items][:quantity]
Started GET "/carts" for ::1 at 2016-08-12 18:50:04 +0800
Processing by CartsController#index as HTML
Rendering carts/index.html.erb within layouts/application
Cart Load (0.3ms) SELECT "carts".* FROM "carts" WHERE "carts"."id" = ? LIMIT ? [["id", 41], ["LIMIT", 1]]
CartItem Load (0.1ms) SELECT "cart_items".* FROM "cart_items" WHERE "cart_items"."cart_id" = ? [["cart_id", 41]]
Product Load (0.1ms) SELECT "products".* FROM "products" WHERE "products"."id" = ? LIMIT ? [["id", 7], ["LIMIT", 1]]
Rendered carts/index.html.erb within layouts/application (9.8ms)
(0.2ms) SELECT COUNT(*) FROM "products" INNER JOIN "cart_items" ON "products"."id" = "cart_items"."product_id" WHERE "cart_items"."cart_id" = ? [["cart_id", 41]]
User Load (0.3ms) SELECT "users".* FROM "users" WHERE "users"."id" = ? ORDER BY "users"."id" ASC LIMIT ? [["id", 1], ["LIMIT", 1]]
Rendered common/_navbar.html.erb (8.1ms)
Rendered common/_flashes.html.erb (1.6ms)
Rendered common/_footer.html.erb (1.1ms)
Completed 200 OK in 83ms (Views: 80.9ms | ActiveRecord: 1.0ms)
这段是基本的数据库语法，比如前两句的意思是筛选出 id=41 的 carts， 在 cart_items 表中筛选出 cart_id=41 的 cart_items
August 12, 2016 | Posted by 王梦琪
8 月 11 日总结—两个大坑

两个大坑。

1.写了人生第一个死循环,一定要是最后一个。

```
  def pay_with_wechat
    @order = Order.find(params[:id])
    if @order.is_paid?
      flash[:alert] = '你已经付过款了'
    else
      @order.is_paid = true
      @order.payment_method = pay_with_wechat
```

本来希望存一个字段，但是因为没有加'',所以程序认为是调用方法，成了死循环。

2.routes 路径没写 member。

在设定 action 路径时，没有写在 member 下，导致参数传的有问题。

```
resources :orders do
        post :pay_with_alipay
        post :pay_with_wechat
  end
```

August 12, 2016 | Posted by 王梦琪

作业：destroy_all delete_all 区别

具体内容见：http://stackoverflow.com/questions/6698207/delete-all-vs-destroy-all

“If you want to delete the User and all associated objects -> destroy_all However, if you just want to delete the User without suppressing all associated objects -> delete_all

According to this post : Rails :dependent => :destroy VS :dependent => :delete_all

destroy / destroy_all: The associated objects are destroyed alongside this object by calling their destroy method

delete / delete_all: All associated objects are destroyed immediately without calling their :destroy method

“

Destroy_all 彻底删除，delete_all 是比 destroy_all 更加高效，delete_all 是一条 SQL DELETE 声明，直接用于数据库，不会实例化记录，不会调用 destroy 方法，不会引用回调。

```
u = User.find_by_name('Jq')
u.usage_indexes.destroy_all
u.user_stats.destroy_all
u.delete
```

但如果把 destroy_all 换成 delete_all，只会删掉用户自己表单中的 ID，其他表单中这个用户的记录不会被删掉，只是在 ID 栏位变为空。

August 11, 2016 | Posted by 王梦琪

heroku 管理密码

安装 figaro

修改

gemfile

```
gem"figaro"
```

执行

```
bundle install
```

figaro install
cp config/application.yml config/application.yml.example
查看
.gitignore 里有么有 config/application.yml
设定 hroku 的 key value
方法一
figaro heroku:set -e production
heroku config 列出所有设定
方法二
打开 heroku 对应的 app
在 setting 里找到
Config Vars，手动添加 key value
August 10, 2016 | Posted by 王梦琪
当日知识总结
快捷键 command+/ #

=begin
=end
多行注释
to_sym 转化为符号
to_s 转化为 symbol
symbol 可以简单理解为轻量级的字符串
创建符号 在前加：
条件判断
真假值
nil 方法不存在
nil 和 fails 都为假
返回真假值都要用？结尾
逻辑运算符（多个条件表达式）：
条件 1 && 条件 2 （且）
条件 1 || 条件 2 (或）
！条件 （相反的条件）
条件判断语句
if 语句
if 条件 1
处理 1
elsif 条件 2
处理 2
elsif 条件 3
处理 3
end
case 语句（比较对象是 1 个）
when 值 1

处理 1
when 值 2
处理 2
else
处理 4
end
比较运算符（处理判断语句）
== != < >
循环
对象.each do |变量|
希望循环的处理
end
方法
调用方法的语法如下
对象.方法名（参数 1，参数 2）
receiver（接收者），面对对象的世界中，调用方法被称为“向对象发送 message”，
结果是”对象 receive 了消息“
方法的调用时把参数连同消息发给了对象的过程
修改数据表
def change
change_table :products do |t|
  t.remove :description, :name
  t.string :part_number
  t.index :part_number
  t.rename :upccode, :upc_code
end
end
Product Load (0.2ms)

数据库语句
cart.products
SELECT "products".* FROM "products"
INNER JOIN "cart_items"
ON "products"."id" = "cart_items"."product_id"
WHERE "cart_items"."cart_id" = ? [["cart_id", 1]]
find_by 如果没找到不会报错
find 只能用 id 没找到会报错
Order.find(1) => ActiceRecord::NotFound
Order.find_by_id(1) => nil
order = Order.find_by_token(“xxx”)
order = Order.where(:token=>”xxx”).first
find_by 与 where
order = Order.find_by_name_and_email(“xxx”, “xxx@xxx.com”)

order = Order.where(name: “XXX”, email ：”xxx@xxx.com“).first
action build/new
@order = Order.new
@order.user = current_user
与以下等价
@order = current_user.orders.build
August 7, 2016 | Posted by 王梦琪

第二周最棒的概念

问自己，什么最重要

在资源有限的情况下，你不能什么都想要，什么都做的后果就是手忙脚乱，什么都做的乌七八糟，你只能将有限的资源集中在少数几件事情上才能得到好的结果。

人生也是这样，本质上是一场资源稀缺的条件下的优化配置的游戏。这里的诀窍就是，问自己，什么最重要。

在人生中问自己什么最重要，可以帮助你实现自己最重要的目标。

在每天的计划里问自己什么最重要，可以帮助自己明确今日目标，有效利用时间。

给自己的最利于成长的外部条件

让经济条件和时间为你自己服务。

不要忽略展示的重要性

内容见下：

点击这里

August 7, 2016 | Posted by 王梦琪

第二周最大的坑-csrf

bug

出现错误 ActionController::InvalidAuthenticityToken in Admin::UsertypesController#set_admin,调试了大约 2 个小时,一直没找到原因. 最后是 XDite 老师发现

application.html.erb 里没有写

<%= csrf_meta_tags %>

这个是用来防止机器攻击的,如果不写,就会出现 **InvalidAuthenticityToken** 错误

启示

不懂得时候不要乱删代码，每一行代码都是有意义的，你现在觉得没效果是因为当前视觉上没效果，不代表这个没有意义。

August 7, 2016 | Posted by 王梦琪

我是怎样从零基础学习程序的

从 0 基础开始的同学，往往和这两种情绪不期而遇：

慌，啊我怎么什么都不会！恐惧,为什么别人什么都会，我是不是太笨了？

这非常非常的正常，每个人都一样，大神也是这么过来的，只是因为人的记忆力机制，他们已经不记得了 XD。上周的我也深陷两种情绪的反复中，现在来分享一下我的方法，希望对你有帮助。

从零基础开始学习的**核心是合理分配自己的注意力！**将自己的注意力放置在核心内容上非常重要，不要流于细枝末节。

模仿

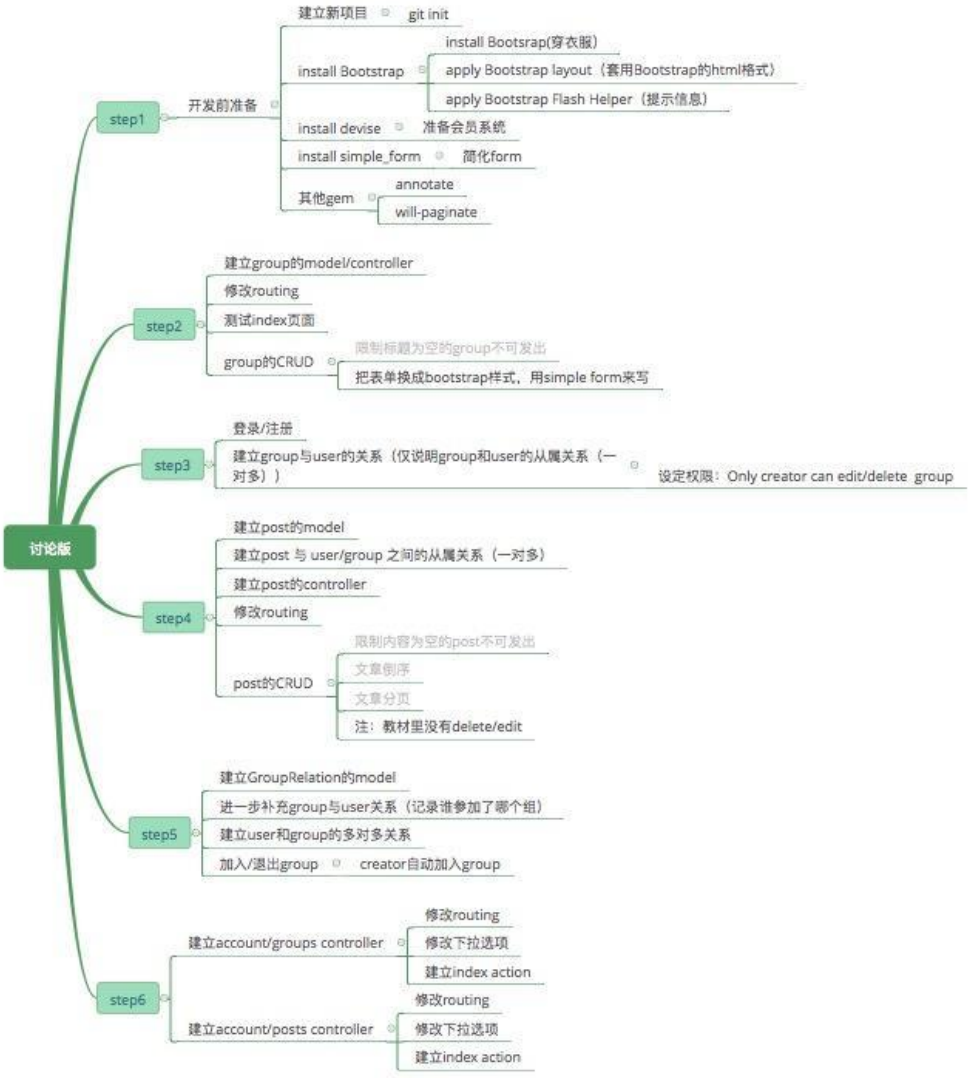
将 Xdite 老师的中级教程模仿 3~4 遍，直到你觉得你会做了为止。请注意：不要到懂了为止，而是要到你觉得**会写了为止！**

在每一遍做的时候一定要问自己，什么最重要？去关注最重要的事情，比如第一遍是整个流程（建议 copy&paste）；第二遍是代码；第三遍是教程是怎样实现这个功能的。这时你什么都不会，重点在于每一遍降低自己的学习成本，不被恐惧吓住。对于零基础而言，模仿一定是最快的捷径！！！

梳理

当你觉得你会了的时候，就可以开始梳理了，此时重要的是**梳理流程而不是总结规律**，因为目前知识量还是太少了，总结规律往往会陷入钻牛角尖的困境。

这里向大家推荐一个工具帮助梳理思路 Xmind。你可以从不同的方面来梳理你的思路，比如功能，User Story，实现步骤，附上我的步骤梳理图。



自己动手写

根据自己梳理的步骤，现在开始动手吧。会写的一定自己写，你一定会遇到想不到的坑，这是提高自己 debug 能力的好机会。

遇到不会的地方可以参考书籍写，**但一定要记录下来**，只言片语就可以。

解决自己不会的

把上一步写不出来的地方（样式可以先忽略），一处一处解决。此时的重点依然不是教程为什么要这么设计，而是我要怎样做才能实现这个功能~！

重新写一遍中级教程/或者做一个类似的功能

你开始写的时候，一定有“哦哦哦~”的流畅感，成就感爆棚！恭喜你完成了。

P.S 如果你没有流畅感的感觉，请循环以上步骤 XD

三个大坑：

好奇心太旺盛，什么都想搞懂

你最重要的资源是你的注意力。程序世界深不可测，奇妙无比，想想一个小学生拿着一道高等数学题拼命要解是什么场景吧。

所以要先建立主干，再填枝叶。重点要在怎么样实现这个功能上。

总结规律停不下来

如果说学习是拼图而不是线性的话，此时的知识量如下图，科学的规律也是建立在大量样本上的，比如孟德尔种了 8 年的豌豆才发现了遗传规律，这个阶段与其多想，还不如提高自己的知识量。

会写 CRUD，理解 Restful、网站运行原理在初级阶段才是最重要的。

不动手

如果觉得自己会了就不动手，开始 copy&paste。这样怎么提高自己的 debug 能力？怎么把别人的变成自己的？

我只能说，醒醒，该吃药了~
祝你程序之旅愉快，一起进步！Fighting~
August 5, 2016 | Posted by 王梦琪
永远不要忽略展示的力量
Xdite 老师分享在 facebook hackathon 的比赛里 presentation 的重要性让我很受启发!
很多中国人不愿意展示/分享,一部分人可能暗藏私心(好东西要自己留着),另外很大一部分人提起分享,内心总是有点害羞又有点尴尬,怕别人嘲笑自己的内容,也怕别人嘲笑自己的展示行为.
传统教育弘扬谦虚和含蓄的美德,似乎只有像诸葛亮或者姜太公一样在深山老林里隐藏才华被挖掘才值得被人称道,展示自己的人都会被嘲笑"王婆卖瓜,自卖自夸","班门弄斧""自傲".他们坚信:

"是金子总会发光的"

"酒香不怕巷子深"
然而以上两句话成立是有前提条件的:

1. 优秀人才是稀缺的.
2. 他人有时间和精力发掘你的才华.

还有一层隐藏的背景,过去上升的渠道太少,所以有才华人只能坚信如此,所以多少含有一点无奈的味道.
现在以上的条件基本不成立了:

1. 顶级人才依然稀缺,但是优秀人才已经不稀缺了. 一个美女在众人中很显然,但是一个美女在维密天使里呢?
2. 信息量爆炸的时代,他人的注意力是非常稀缺的.你不展示,等着别人去发掘你,就是在给别人增加成本,你喜欢给自己增加成本的人么?

此外,时代背景不同了,今天的上升渠道如此之多,只通过"被人发现"这一条渠道是不是有点少呢,通过多条渠道不是刚能给自己增加成功概率么?
前提条件和背景都不成立的时候,结论自然也不成立了.
在这个时代,要展示分享,要学会好好的展示和分享.

"presentation"有展示和分享两层含义,presentation 作为综合性人才的重要能力之一需要不断的练习.
好的展示往往就是好的分享,这么利人又利己的事为什么不做呢? 所以从今天开始改变心态,开始练习吧.
August 4, 2016 | Posted by 王梦琪
8 月 3 号总结
超级有成就感
今天自己完成了第一个网站,完全自己写的!!尽管只有产品的增删改查,前台展示及用户权限管理,
但是能自己写出来特别有成就感,xdite 老师还给我加了 5000 分"最佳进步奖",很受鼓励.
我发现每天给自己规定 deadline 很管用,几点之前必须做完什么,会帮助自己充分提高效率
Xdite 老师分享
每天多成长 1% 1.01^365= 1.378, 一年多成长 37.8%, 这个成长速度是非常惊人的,所以不用

着急.
分享书籍"师傅", 同样的错误绝对不要犯 3 次, 写下来会帮助你不犯错
成长的最快方法

- 换工作,报酬再低也要去

最大的坑:
如果你从事的是自己不喜欢的工作,而选了一份很闲的工作,利用自己的剩余时间自学,最后一定会被坑到,这相当于深山练剑.因 为别人做这个的时间是每天: 8+2=10 小时,你做的时间是每天 2 小时,别人的有效时间是你的 5 倍.

跟着大神练功你会很快的模仿学习, 你可能花半年的时间就自己提高一倍(大神一般接收和自己差距 3 倍以内的同事)

- 抗起责任
- 帮助别人,回答别人问题
- 公开分享

  分享(写作)可以帮自己整理思路,发现自己没想清楚的问题
  会收到反馈,帮助自己改进思路.

  可能找到潜在的工作伙伴(机遇)

August 4, 2016 | Posted by 王梦琪
[8 月 2 号总结]
总结: 关于 Rails 符号的应用 及惊天 bug
符号应用

1. 判断式

   == 是判断式的等于

   != 是不等于
2. !和?的差别

   !XXXX 不是

   XXXX! 执行会改变自己的状态, 如:
3. User.create! => ActiveRecord::RecordInvalid
4. User.create => false

XXX? 预期回传的是 true 或 false
惊天 bug
出 现 错 误 ActionController::InvalidAuthenticityToken in Admin::UsertypesController#set_admin,调试了大约 2 个小时,一直没找到原因. 最后是 XDite 老师发现
application.html.erb 里没有写
<%= csrf_meta_tags %>
这个是用来防止机器攻击的,如果不写,就会出现 **InvalidAuthenticityToken** 错误

August 3, 2016 | Posted by 王梦琪

render 用法

在 controller 中 render 用法

1. 渲染动作
   render :new
2. 打出文本
   render :text => “ok”
3. 使用另一个 model 的全部"衣服"
   render :layout => admin

在 view 里 render 的用法:

1. 调用 partial
   render :partial => “welcome/item”
2. 等价于循环
   <%= render :collection=>@jobs %>
   与以下循环等价
   <%= @jobs.each do |job|%>
   <%= render :partial => "job", :collection=>@jobs , :as=>:job%>

August 3, 2016 | Posted by 王梦琪

scope 用法

scope 使用场景:

- 负责的资料查询
- 重复使用的资料查询

scope 是类的一个方法，是用来查询数据库的一种方法:
要想定义简单的作用域，可在类中调用 scope 方法，传入执行作用域时运行的代码：

```
class Post < ActiveRecord::Base
scope :published, -> { where(published: true) }
end
```

上述方式和直接定义类方法的作用一样，使用哪种方式只是个人喜好：

```
class Post < ActiveRecord::Base
  def self.published
    where(published: true)
  end
end
```

August 2, 2016 | Posted by 王梦琪

8 月 1 号总结

感谢笑来老师的猪蹄和啤酒,意外之喜,完全被收买了.

今天,XDite 老师分享了两个让人变得幸运的方法:

1. 看说明书
   前人把踩过的坑和指引都写在了说明书里,所以看说明书就是避免跳进这个坑,减少80%的不幸.
2. 面临选择时,做高一级的决定,哪怕你不认同
   当你不理解的时候去做比你智慧的,有经验人提供的建议和决定,哪怕你不认同或者不理解,因为你的方法一定不是全世界最好的方法,这样会获得很多意想不到的红利.

千万不要自己躲在深山里面去练,因为这样下去你的速度和牛人会越差越多,分享和交流才能帮助你更快成长.

我之前总是有很强的"自尊心",怕被笑或者被骂,但是 XDite 老师说的很好,被笑没有关系,被笑能换来解决方法非常值,这样你的成长是最快的.

August 1, 2016 | Posted by 王梦琪

本周总结

1. 一周回顾

   本周工作 7 天,大部分时间状态不错, 从之前的云里雾里到现在的会写,感觉非常的神奇.debug 能力也有所提升,也可以帮同学 debug 了.

   本周需要注意的是:要早睡才能提升注意力(提高有效时间),不要熬夜,要按时吃饭,因为状态好的时候 1 个小时能顶 3 个小时用.
2. 三个写程序我需要养成的习惯:
   - 记得改完 atom 的任何一个文件随时 save,不要全部 save,不然往往会忘记保存一两个文件,如果全部保存在正式开发的时候是很危险的,
   - 开始和结束一起做,例如写完 def 马上写 else,end,不然往往会忘记写结束.
   - 记得将代码正确缩进,不然代码长的时候很难辨认出逻辑关系,往往会出错,特别是写了很多 end 的时候.
3. 学习方法

   记得是 self-teaching,而不是 self-learning,比如研究 restful 概念的诞生历史就对这个概念理解很有帮助.人生本质是注意力管理(有效时间的产出).
   - 搜索,记得问 google,不停摸索.
   - 付费,付费内容往往更优质,最宝贵的是自己的注意力,一定不要浪费.
   - 整理,重复,特别是整理,之前特别缺乏.
   - 分享,分享是帮助理解的最好的老师.
4. 学到的东西

   CRUD,Restful(7 个 action),网页运作原理,html 的基本写法.

August 1, 2016 | Posted by 王梦琪

本周最棒的概念: 关于 CRUD, Http Verb, Restful 及网页运作原理

本周我逐渐理解了这三个概念及之间的关系,以下是我自己的理解.(附上网页运作概念的理解)

1. CRUD

   基于 database 的网站的本质上是对 database 的处理,具体来说把 database 存起来之后在 database 里做这四种操作（Create Data, Read Data, Update Data, Delete Data),这也就是 CRUD.
2. Http Verb

   我们在网上进行操作 (进入到一个不同的 url), 叫做发出 http request, http 响应后, 叫做 http response。

   request 分为四种类型（我们会发出下 4 种 http request）: get, post, put/patch, delete

我们叫它 4 种 Http 动作（verb）：Get、Post、Upload、Delete

3. RESTful
   我们想让这四种请求通过一些方式执行对应的 datadase 的四种操作，但是大家执行 CRUD 和网址设定都不一样，所以我们想要建立规范,所以产生 RESTful 这个概念.
   rails 里我们通过 Action 去对应。

   最常用的是以下 7 种 action，以建立讨论组（group 的新建，编辑，删除）来举例：

   index：http get 通过这个 action 及后续一系列操作执行了 read data 的操作，对应首页

   new：http get 通过这个 action 及后续一系列操作执行 read data 的操作， 对应新增分组页面

   create：http post 通过这个 action 及后续一系列操作执行了 create data 的操作，对应新增一个分组
   edit： http get 通过这个 action 及后续一系列操作执行 read data 的操作，对应编辑分组页面

   update： http put 通过这个 action 及后续一系列操作执行了 update data 的操作，对应修改一个分组
   show： http get 通过这个 action 及后续一系列操作执行了 read data 的操作，对应单个 group 页面

   destroy： http delete 通过这个 action 及后续一系列操作执行 delete data 的操作，对应单个 group 删除页面

   （注：之所以不用一个单词我猜测的是希望不要把 destroy 和 delete 混淆，一个是 http request 一个是 action）
4. 网页运作原理
   用户执行操作, 浏览器发出 http request（get, post, put, delete）, 网站解析 routes.rb, 找到网址，然后通过 controller 找到对应的 action，action 会去 models 调出所用的资料（model 会去 database 找所用的资料）。然后 action 会去按 views 的规则（views 调用 partial 和 helper）输出，成为我们看的网页

August 1, 2016 | Posted by 王梦琪

本周最大的坑: routing error.

现在基本能排除掉拼写的错误,但是其他错误往往会绊倒很久,本周的大坑在是排除 routing error.

场景再现: 在重写中级作业的时候,写成了建立 model group 与 controller group,事实上是 controller groups,所以一直提示 routing error.

最后解决方案:使用 rake d controller XXXX 删除 controller 后,重建 controller 可解.

总结: 如果提示 routing error.

1. 检查文件夹/文件名称(controller, view, model 里的),看看是不是有误.
2. 检查 routes.rb 的规则是否正确
3. 检查按钮/链接的地址是否正确.
4. 检查 controller 和 view 的代码是否正确.

July 29, 2016 | Posted by 王梦琪

css 自学作业

1. margin 和 padding 的差别
   margin(外边距）是边框到边缘的距离，padding（内边距）是元素到边框的距离。
2. 什么是 Box Model
   Box Model 是 CSS 框架，这个框架规定了元素框处理内容、内边距、边框和外边框的方式。
3. 为什么要用 em 而非 px 定义字体大小
   如果使用 px 定义，IE 不会重新调整文字大小。
   1em 等与父元素的默认字体大小。 em= pixels/父元素的大小
4. h1 {margin : 10px 0px 15px 5px;}
   margin-top: 10px
   margin-right: 0px
   margin-left:5px
   margin-bottom： 15px

July 28, 2016 | Posted by 王梦琪

7 月 28 日总结

今日感觉很累，完成了 step5 两遍，开始重做中级教程。有如下两点体会：

1. 关于学习过程
   前两天看了 Xdite 老师推荐的文章，里面讲学习过程的是拼图似的，刚开始手里拿着一片拼图，没有一点头绪，学着学着，突然有一天，哦！原来是这样，我懂我在做什么了。这篇文章解了我长久以来的困惑，因为主流的观点是学习的过程是线性的，而我自身体会并不如此。
   这篇文章未提到的是，往前走又会进入一个新的循环，没有止境的，所以挫败感是常有的，当你习惯了，就可以以平常心对待了。
   另外，不断学习还会有一个 extra bonus，就是你的拼图技巧的提高（学习能力的提升）会让你提升速度，越来越自信。
2. 关于精力（其实是注意力）
   每个人的注意力上限是由精力决定的，每个人的精力是不一样的（往往精力上限其实是体力决定的），我们应该遵循自己的精力做事，不要透支自己。当然，更长远的做法是提高自己的体力及注意力，比如运动和冥想。
   我前两天由于觉得任务没有完成，拼命加班加点到晚上 2 点，由于要错开早高峰又要 6 点多起床加上早晚高峰挤地铁。前几天还好，今天特别的疲惫，不断出现基础错误，导致进度很慢，所以前两天的强度基本是浪费的节奏。

July 27, 2016 | Posted by 王梦琪

7 月 27 日总结

之前一直云里雾里的，今天突然懂了之前学的，明白了 CRUD，controller,view,model 之间

关系，每一行代码都有了清晰的含义。感觉特别开心，很有成就感。

昨天由于解决环境问题花了不少时间导致进度严重滞后，今天补上了进度，做了 step2-step4。

今天发现的小技巧：

- 在 atom 编辑器修改后，记得全部保存，不然可能会漏掉，全部保存的快捷键为“optinon + command + s”
- atom 编辑器搜索文件的快捷键为“command + p”
- google 搜索 比如 ruby 的问题 搜索 ruby each

总结下遇到的拼写的坑：

标点符号

- 一定要注意中英文切换，特别是在输入中文字后！千万不要输入中文空格。
- "" '' 两者区别在于""里的内容有可能会被转译， ''则不会被转译。
- ：这个符号的用法很多，这个符号永远是有意义的，不要乱用。
    - 在 symbol 前往往有箭头的意思，表示特指这个 sybmol。如下： only: [:new, :create, :update, :edit, :destroy]
    - 有时可能是缩写的一种格式
      <%= link_to("Add a job", new_admin_job_path, :class => "btn btn-default") %>
      等价于
       Edit
    - 空格 空格本身就是断开的含义，但用错位置，特别与 :, . 一起用的时候，会出现错误。
    - _ 有一些函数是定义好的；可运作自定义函数。

拼写错误

- 大小写 是不可以随便大小写的。比如 @job = Job.new, J 不可小写。
- 字母拼写错误 注意这个实在没有办法了少年，写的时候多注意，多检查。有些单词不是英文单词不要混淆。

July 27, 2016 | Posted by 王梦琪

7 月 26 日总结

今日主要是回顾加搞明白之前的常用指令，包括 git、rails 和 ruby 的基本指令。
自学了 html 和 css。
下午环境搞乱了，rake db:migrate 持续出问题提示 routing 规则有错，XD 老师帮忙发现是 spring（rails 的加速器）的问题。在 iterm 执行 spring stop 可以解决。
于是重新做了昨天的任务和今天的 step1 job 的 CRUD。
今日所学指令有：
rake db:drop 如果 migrate 出错可用 drop
rails d controller XXXXXXX 在 generate 出错时，把 g 改为 d 可 destroy 刚刚 generate 的。
pwd 查看所在目录
rm -rf 文件夹名称
cd .. 返回一层目录
cd ../.. 返回两层目录
做错了，想回到之前的版本：
方法一：使用 sourcetree
方法二：

git log 查看 git 上传记录，退出按 q
git reset --hard 编号 强制回到指定版本
port 被占用时：
方法一：解除占用
lsof -i tcp:3000
kill -9 你的 PID
方法二：启用 port4000
rails s 4000
方法三：
重启

July 26, 2016 | Posted by 王梦琪

html 自学作业

1. div/span 的不同
   区别在于：
   div 元素是是块级元素，是用于组合其他 html 元素的容器。由于是块级元素，浏览器在其前后会显示折行。
   span 元素是内联元素，是用于组合文本的容器。浏览器在其前后不会显示折行。
2. class/id 的不同
   class 属性规定元素的一个或多个类名，id 属性规定元素的唯一 id。
   两者的区别在，class 用于元素组（某一类元素），id 用于唯一的元素。
3. p 与 br 的不同
   尽管两者浏览器在前后都会显示折行，不同之处在于：
   p 元素用于定义段落，有开始标签和结束标签，格式为： XXXXXXXXX
   br 元素为插入一个换行符，是空便签，没有结束便签，格式为:
4. 如何使用 table 排版
   由<table>定义，表头由<th>(table head)定义，行由<tr>(table row)定义，每个单元格由<td>（table data）定义。
   加边框，使用 border 属性，格式为:<table border="1">
   单元格为空，输入<td> </td> (no breaking space)
   加标题，使用 caption 标签，格式为:<caption>XXX</caption>
   排列，使用 align 属性，格式为:<th align="left">xxxx</th>

July 25, 2016 | Posted by 王梦琪

7 月 25 日总结及 git 常用指令（一）

You not only deserve a second chance，but also a third chance, a forth chance, and all the chances you can get.
笑来老师的一番话让我感受深刻。
真正对自己好的人,是相信自己，给自己机会，且严格要求自己的。这两者并不矛盾。
人生本质上不是时间管理，而是注意力管理。
不懂得注意力管理的人往往会落入三大陷阱：
莫名其妙的凑热闹
心急火燎的随大流
操碎别人的心肝
这就是大多数人为什么没有成长的原因，注意力重心放错了。

git 常用指令

1. git 初始化
2. cd 文件夹名称
3. git init
4. git add .
5. git commit -m "Note"
6. git 存档
7. git add .
8. git commit -m "Note"
9. 新建/删除一个 branch
    - 新建： git checkout -b branchname
    - 删除：先去其他分支，再删除 git checkout otherbranch git branch -d branchname
10. 本地 push 至 github
    创建新的 repository
    git remote add origin git@github.com:用户名/repository 名称.git
    - 如果是单个分支 git push origin branchname
    - 如果是多个分支 git push --all origin
11. 如果 push 到 github 上失败

- 如果执行 git remote add origin git@github.com:用户名/repository 名称.git
  出现错误：fatal: remote origin already exists
  则执行：git remote rm origin
  再往后执行 git remote add origin git@github.com:用户名/repository 名称.git 即可。
    - 在执行 git push origin master 时， 报错：error:failed to push som refs to.....
      则执行：git pull origin master 先把远程服务器 github 上面的文件拉先来，再 push 上去。

July 16, 2016 | Posted by 王梦琪

Hello World

Hi, This a demo post of Logdown.

Logdown use Markdown as main syntax, you can find more example by reading this document on Wikipedia

Logdown also support drag & drop image uploading. The picture syntax is like this:

Bloging with code snippet:

inline code

Plain Code

puts "Hello World!"

Code with Language

puts "Hello World!"

Code with Title

hello_world.rb

```
puts "Hello World!"
```

MathJax Example

Mathjax

Inline Mathjax

The answser is .

Table Example

Tables	Are	Cool
col 1	Hello	$1600
col 2	Hello	$12
col 3	Hello	$1