
chpwang's Blog

September 13, 2016

本周最坑就是老师的网站 ontrackapp.com 的证书问题

这个问题导致的结果是，你无法访问 ontrackapp.com 网站，safari 显示 Safari can't establish secure connection

解决方案：

打开 Safari Preferences -> Privacy -> Remove All Website Data

然后重新访问 ontrackapp.com 即可

[我的编程老师是台妹](#)

September 11, 2016

无论是什么样的训练，造成训练效果之间巨大差异的原因通常是人。比如你的老师，比如你的同学，比如你自己。

今天的文字，就是讲述关于我的编程老师——台妹——的故事。

来自台湾的 Hacker

作为来自偏远地区的土包子，我对台湾的认知主要来自祖国母亲大人：

- 那里的人总在开各式各样看不懂的会议；
- 经常在桌子上打来打去；
- 他们谈论到和「钓鱼」有关的事情就很亢奋；
- 那里的女生漂亮又治愈；

带着这些对台湾的困惑，「卧槽！」便成了台妹老师给我的第一印象……

通常台妹对外的两个官方介绍是「两岸相当知名的开发者与软体教练」和「Facebook Hackthon 冠军」。虽然知名度没那么“相当”，而且冠军也是高度语境化的（highly contextualized），但之所以使用这样充满槽点的介绍，我觉得部分原因是台妹比较矜持，并且夸自己没那么重要，所以选择走朴实路线；而主要原因在于，使用文字很难表达出台妹教学上那种 sophisticated 的气质和效果。总之，隐藏在这平淡介绍背后的，是台妹深厚的开发功力，以及将这份「深厚的功力」传授于他人的强大指导力。

启动

台妹花了大量精力在设计针对新手的启动教学上，其中，课前作业时期就采用了恐吓，追杀，奖励和画大饼的方法。开课之后又新增了学员间互相帮助和日记这两大任务。每一个设计都从不同角度，以不同程度加速新手对启动知识的吸收。可谓用心良苦。

「启动」的设计思路来自于认知科学的研究，尽管认知科学目前还是比较有争议的学科，但如果你无论如何都要轻视前三周的「启动」训练，那么后半段很容易就会出现掉队的状况。台妹对于不听话的掉队者是非常不客气的，会直接退掉你。台妹说过「不要试图用你的努力来增加我退你课时候的愧疚感」，吐槽的就是掉队后的那些狼狈的努力。

学徒

学徒制是我很喜欢的学习方式。你可以把学徒制想象成知识的自助餐，你学到的东西取决于你胃的大小，以及饥饿程度。而台妹的实力是深不见底的，至少对我来说是这样，所以我不需要担心自助餐厅没东西给我吃，我只需要让自己不断地饿下来就可以了。

令我高兴的是，台妹本身也非常推崇这样的教学方式。这让我很感动，因为学徒制对学徒们虽然友好，

但是对老师傅来说就是噩梦。坦白说，我曾一度打算准备一套连哄带骗加贿赂的方案，用来在不知不觉中让台妹对我提供学徒式的教学，尽可能地压榨台妹的技术。后来非但不需要，而且台妹真的是尽心尽力倾囊而授，而且很少会看到她的不耐烦。也亏她能保持下来，我反正是会崩溃。

有段时间我觉得自己义不容辞的使命就是怂恿她对课程涨价。

GIF, Standup Meeting 和 ThoughtBot

使台妹投入大量精力的另一个设计就是连逼带哄催我们每天写日记，每周写周记，到了后面的合作项目，每天下午还需要上传 GIF 动态图来展示每天所做的成果，并且小组成员要在次日开 Standup Meeting 的会议。

开始我完全是按照对台妹的宗教信仰在做这些事情，直到台妹和我说了她和 [ThoughtBot](#) 之间的故事，我才完全意识到这些事的重要性，以及它们会怎样潜移默化地影响我们的成长。

[ThoughtBot](#) 是世界上排名第一的 Rails 外包公司，实力第一的雇佣兵集团。台妹老早就想偷学他们的做事方法，而给我们上课的过程中正好需要设计一个交作业的系统，于是台妹便花钱雇佣他们来帮忙写交作业的系统。期间，[ThoughtBot](#) 是按照小时收费的，并且每天都会有 Standup Meeting 的会议。台妹说，正是因为进行了这样频率的记录和汇报，项目才能保证持续推进，进而按时完成，不会跑偏。普通的外包公司顶多每星期一次会议听客户需求，沟通很容易不到位，所以普通外包公司经常加班加到累死还不能完成任务。

台妹强烈建议我们将来进入公司之后仍然保持这个记录和上传的习惯，她说别人不做你做，很快就能把他们杀爆.....

干货

遇到状况，台妹会伺机选择很没节操地说「关我屁事」来逃避一些作为教师的“责任”（台妹认为这些“责任”属于道德绑架），但更多的时候，台妹还是会慷慨地给我们分享她的「干货」，直到下半夜。

我个人觉得台妹的「经验型干货」比「技术型干货」要有用。比如：

下午 7 点 deploy 一定是晚上 9 点走，所以要避免在注意力分散的低效率期间 deploy

A 轮基本必死，但是是最有钱最有机会的时候，所以不要管死，先冲上去干一波再说，因为必死搞到一定程度就行了，剩下的交给最好的雇佣兵，不需要事必躬亲

.....

有一点需要说明的是，台妹会利用闲聊时间分享干货，但当台妹说要分享「干货」的时候，其实大家是要提高警惕的。台妹所分享的「干货」，其「干」的程度和围在她身边的学员的实力成正相关。学员实力越强，台妹的分享越「干」，或者说只有正确的问题，才能引出有价值的讨论和答案。但通常情况下，同学们.....你懂的.....我只能说全是信仰.....

还有一些八卦

台妹开发时喜欢坐 Aeron

台妹看书速度非常快，脑子里是没有默读声音的。她说声音会影响速度。

台妹喜欢玩全境封锁这类 FPS 游戏，还会请人帮她代练拿武器，以便直接享受杀爆的快感。

台妹谈到应聘话题时，会讲一个关于老板开除前台妹子后，公司 CTO 跟着辞职的故事。

台妹鼓励班里的女生去应聘时会提示，性别会让女生们在面试官眼里自动增加一年的开发经验。

但是台妹又说，除非实力超强，否则女生如果长得丑那还是吃亏。

朱萧木老师则说，别说自己不好看。要使劲儿捋飧。捋飧和不捋飧，效果能差个一百倍。捋飧完都好看。比如他自己。

台妹打了个喷嚏之后接着说，程序员要保存精力和注意力，可以骨折，但不可以感冒。

台妹还说，人很累的时候，内心的阴暗面就会出现。

通常，台妹的注意力北京时间下午 17:00 之后会开始涣散，晚上 21:00 左右才会开始恢复。
期间台妹有一定概率进化成阴暗台妹，但仍然会尽量克制，努力保持人类模式。

【CoCo 都可的百香绿茶——注意不能加珍珠】和【海底捞】可以辅助恢复台妹的活力，使其有概率进化成光明台妹。

台妹审核这篇文章之后，让我删了很多虽然有血有肉稀松平常，但她觉得会威胁到老大哥们(Big Brother)的文字。

台妹希望这个世界可以看到这个世界愿意看到的台妹。至于真正的台妹，台妹本人和这个世界都不是很关心。

第七周周记

September 6, 2016

本周几乎算是最后一周了，所以周记写一下跟着 xdite 在一起修炼的攻略，给今后的学员参考。

注意！ 这份攻略适合且仅适合对成长有着拼命的觉悟的人，也就是为了成长愿意努力到最后一口气并且**随时**都有勇气承认自己是傻逼的人——如果你竟然觉得自己不是傻逼，那通常有且仅有两种可能，要么你是个【万中无一的天才】，要么你就是个自以为是的【傻逼】。考虑到万中【无一的天才】是不会来 xdite 这里的，所以...你懂的.....：

简单粗暴的结论

- 1、修行过程中要严格听从老师的话，xdite 说啥，你做啥，最好能像宗教信徒一样不考虑其他。
- 2、凡事都要快，否则你会因信息过载而越来越落后，你会没时间练习，你会错过 xdite 那里套出更多内容机会，你会.....总之，速度带来的好处比你想象的要多，远非 get things done quickly 那么简单。
- 3、一定要把课前作业按时完成，否则你无法实现第 1 条里说的速度，或者说得严重一点，你没做作业就上课会直接完蛋。
- 4、每天都要搜集，都要写日记记录遇到的坑，并提交 Gif 展示每天做的功能。我的感觉是，坚持下来的人会比没坚持的要多成长 50%；
- 5、每天早上的到教室之后要思考当天各种事项的优先级。永远最优先把必要的事做完（比如下周作业本周做完，比如每天的日记早上写完，每周的日记周五写完），无论这件事有多无聊；
- 6、要住得离教室近，不要把宝贵的时间花在路上。我推荐使用【自如】的租房服务；
- 7、在住得近的大前提下，每天早上尽可能 6 点钟起床出发，避开早高峰；晚上 11 点之后才回家，避开晚高峰；而且还能获得更多在教室编程的时间。
- 8、第四周开始是团队合作，也就是全栈课程的精髓，强度会非常高，所以前三周要保存体力并打磨自己的代码速度。温馨提示：团队合作时，你会发现一个全栈工程师的实力和他写代码的能力几乎无关，有无数代码之外的巨坑在等着他。总之体力很重要。
- 9、第四周开始之后，如果可以的话，全程要避免充当产品经理的角色，争取只做一个代码雇佣兵安静地写代码。
- 10、对了，一定要做最后面的大桌子，并且插网线。网线保证网速，而做后方的大桌子主要是安静，不容易被别人打扰。这好处我到最后一周才发现，非常吃亏。

第六周周记

September 2, 2016

关于本周工作

我们组这周对产品的打磨主要集中在功能上，比如修 Bug 和增加各种诸如【开课】和【结课】的小功能。从功能上来说，已经比较齐全了，但是网站目前长得还很难看，需要将接下来的大部分时间投入到前端的设计上。

关于以战养战

除了重复之外，增加前进速度的方式就是整理。比如每天一定要提交 gif 演示图来展示自己的成果。还有就是每天都要写文章。

这样就能逼自己每天都要有东西可以产出，甚至有完整的成品可以产出，而不是所谓的身负技术但是只有尸体。

总之，通过不断推进来最大化训练效果的方法论中，每天的整理是至关重要的。理想状态下，每天的整理要像呼吸一样自然、迅速和不可或缺。

关于产品推进

这周我学到最醍醐灌顶的一句话是：

不要用战术上的勤奋和胜利在掩饰战略上的懈怠和失败

产品推进的过程中，优先级的判断是很重要的，这里的【优先级判断】就是属于战略的范畴。普通团队，尤其是团队里的技术人员，很容易把精力都花在酷炫但是优先等级很低的功能上，导致项目整个被拖垮，令人遗憾。

不过关于【优先级判断】的这个坑，老师这周已经教会了我们很好的方法，接下来就靠我们去实践了。

[第六周最棒](#)

September 2, 2016

本周学了很多很厉害的东西，但都不能细讲，能讲的是它们都是关于方法论和观念的。

还有一点因为体验深刻所以要吐血分享的是，对于以战养战的训练方式，无论如何住得离战场越近越好。

除了路上省下的时间和节约下来的体力之外，最重要的一点就是不再需要担心回家晚，从而可以和高手们待的久一点。不要小看【久一点】这个福利，它所带来的效益是很大的。我细想之下，觉得本周学到的很多东西本质上都跟我住得离战场近有关。

总之，再次推荐使用【自如】进行短租。

[第六周最坑](#)

September 2, 2016

本周最坑的就是 .gitignore 功能。事件经过大体如下：

由于要使用 Amazon S3 来实现上传图片服务，所以需要给部署的服务器设置 Amazon S3 的密钥。通常我们会通过 .gitignore 来忽略保存密钥的文件，以免密钥被上传到 Github 这类公开的地方。

然而 .gitignore 的功能并不像我们预期的那样忽略文件——我们的预期是在 .gitignore 里添加要忽略的文件，然后就能实现忽略功能，但其实不是这样。

真相是，编辑 .gitignore 文件并不是立即生效的，要看情况。如果你 ignore 的文件没有被追踪 (untracked)，比如新建的文件，则添加后 (git add .gitignore 并 commit 后) 可以生效。但如果是已经追踪的文件，需要通过以下清除缓存的命令来重新建立追踪。参考了[这里](#)

```
git rm -r --cached .
```

```
git add .
```

```
git commit -m "fixed untracked files"
```

第四组就是因为误解 .gitignore 文件的工作方式，不小心上传了密钥，导致信用卡被刷了 700 美金。

[第五周之最棒](#)

August 28, 2016

本周最大的收获是下面这个简短调查问卷：

问：我们店的快餐好吃吗？

答：No

问：服务好吗？

答：烂
问：你会再来吃吗？
答：会
问：为什么？
答：习惯了

第五周最坑

August 28, 2016

本周最坑就是团队合作还有业务逻辑了。这两大块的坑点在于它们是相互作用的，其中业务逻辑高度影响团队合作。

团队合作

第四周项目开始时，团队合作出现严重的问题，队员之间由于实力参差和观点差异，各种不服和吵架。在 xdite 老师的帮助和逼迫之下，团队问题得到很大程度的缓解，大家都收敛了自己的个性，气氛稍微和谐了。然而，问题并没有解决。目前团队的状态是，总觉得要做点什么，但都不知道做什么好，真要做点什么又容易偏离团队方向而遭到批评。也就是处于一个发呆状态。

业务逻辑

这两周我的经验是，业务逻辑是高度影响团队合作的。具体来说，团队成员都知道运营的大体方式是怎样的，但是软件的具体细节是怎样的，即点击之后跳转到哪？哪些地方应该加搜索框？主页好不好看？好看的标准？谁来给出最终决策？这些问题都没有很好地解决，并且在不同程度上影响团队合作和氛围。

我的观点是要有一个人来承担定义产品的所有职责，也就是他来做每个议题的最终决策人，他说的话大家必须绝对服从。换句话说，影响决策的只有两个途径，第一是大家可以提意见，尝试说服这个最终决策人；第二则是网站上线后，用户的行为影响决策。

但显然，这个方式不符合我们目前的情况，其中一个原因是没有人有这个实力来做这个产品经理。所以，我们就不得不选择聂师傅的方式，也就是所有方案都是大家讨论。但讨论的结果就是效率低下，并且激发队友之间的各种不服气。效率低下很容易就会产生疲惫感。

总之，希望团队在聂师傅的带领下能突破困境，实现逆转。

第五周周记

August 22, 2016

又到了周末，目前已经是第五周，感觉整个人已经出现疲惫了，主要是前进方向不明。当然，我们的组长聂师傅表示他对业务逻辑非常明确，所以目前也只能指望他了。此外，周五中午大家 Demo 完之后的闲聊加剧了这种疲惫，感觉注意力完全没法集中。写一下我这个星期的思考吧。

项目推进的两种模型

可能是由于我文化程度不高，这两个星期下来，我对一个项目的认识简化成了两种模型。第一种是【个人主义精英模式】，也就是项目组里存在一个拯救世界的超级英雄，大家团结在他周围，他几乎要做所有的决策。典型的例子就是乔布斯时代的苹果公司。第二种叫做【民主主义投票模式】，在这种模式里，项目的推进是靠大家一起讨论决定的，各种决策也是讨论的结果。

两种模式都有各自的特点，不考虑语境的话，很难说哪一种是更好的。

精英模式

在【个人主义精英模式】的特点是效率高，而且经常能做出令人惊喜的东西，真正改变世界和行业，但门槛极高。【精英模式】里，作为超级英雄的那个人需要有强大的实力或者魅力，而且往往要承担极大的责任，

所以他通常要做好【不是普通人】的觉悟，因为整个过程中需要贯彻类似【我这是对你好】、【我来拯救世界】这种在凡人看来高度自以为是的信念。这种模式的好处就是。不过就是团队会高度依赖这个超级英雄。

投票模式

【民主主义投票模式】的特点是效率低，难以做出惊喜，通常只能改造世界或行业，但实现的门槛低，几乎所有的团队都可以采用这个模式。产品失败了也是大家一起承担责任，【大家一起承担】的意思就是谁都没有责任的意思。雪崩嘛，没有任何一片雪花认为是自己的责任，尽管大家会认为雪崩是某一片雪花引起的，但这个【某一片雪花】永远会是个迷，所以相当于没有。

不过，【民主主义投票模式】并不是失败的代名词，它是能运作的，也能成就很多东西。它相当于我们普通人，或者说弱者，一步一步地在艰难前进。这种状态比不前进要好得多。再举个例子就是，我们穷得坐不起飞机，但颠簸的轮船也能让我们去到彼岸，或者至少忘那个方向前进了，离彼岸更近一些，尽管我们姿态难看，尽管我们衣衫褴褛。

此外，普通人如果要进入【精英模式】，是会惨败的。

结论

说了些自以为是的思考，不求大家同意。但如果你同意，我建议接下来的行动方针是，训练自己能在两种模式的团队中轻松快速地切换。如果没有训练过，会出现：

- 在【精英模式】里，你会认为老板是个自以为是的傻 X，然后做出伤害团队的事；
- 在【投票模式】里，你会嫌弃团队的效率和你的猪队友，并且无法理解在你所谓的猪队友眼里，你也是个猪队友，然后做出伤害团队的事；
- 在两种模式共存的混合模式里，做出伤害团队的事；

第四周大坑

August 21, 2016

不用说，本周最大坑就是让我熬夜一晚上的【Markdown 编辑器】，这东西我到现在还没解决，目前主要问题是无法使用编辑器上传图片。

值得感慨的是，我越是搜索尝试不同的 Markdown 编辑器，就越是感到这些前端工程师的命运之悲惨。完后再看看那些在网上指点江山的前端工程师，觉得这个世界愈发的残酷和凶险了.....除非找到更美好的真相，否则今后在他们面前只能装傻了。

后来 xdite 说去 Code School 学一下教程会有帮助，所以我目前要赶紧写完这些文章然后挤出时间去研究。

August 21, 2016

价值

第四周学到的最有价值概念是关于团队合作的攻略，来自 xdite 和聂师傅：

1. 不要因为急于得到大家的认可，去做那些很炫的东西浪费时间，要专注于 Basics，也就是最重要的那些东西；
2. 打开编辑器写第一行代码前，先考虑清楚：「为什么我们要开发这个功能」；
3. 要写别人最容易扩充的代码，最容易理解的代码，宁愿笨一点的方法；
4. 决策的标准只有一个，就是做这件事对项目有利；

工具

第四周最棒的工具是 Github：

这东西其实一直都在用，但本周 Github 用得比较多，已经上瘾了，具体体现在没事就想上传些什么。甚至

比起 Tower，我更愿意使用 Github 来提交那些待做的任务。

这其中包含的 Pattern 其实很有启发：

因为要经常使用，所以自己就被逼去看 Github 的说明书，然后明白怎么使用后因此获得极大的收益。简单来说就是，【逼迫】和【说明书】是对成长和进步极其有利的。

第四周周记

August 21, 2016

总结

本周的压力确实很大，还好我勉强顶得住。关于本周最需要反思的就是团队合作了，显然，马师傅事件是需要反思的。

据我的观察，马师傅的观点主要是希望自己能参与写代码然后得到锻炼，不希望自己被晾在一旁打杂。而为了得到锻炼，马师傅不介意项目的进度慢一点。此外，马师傅不觉得自己的代码写得烂。马师傅还认为我改她的代码属于窃取她的劳动成果。

聂师傅和我的共同观点是项目必须快速推进，不能浪费任何时间，尤其不能浪费在帮别人修改代码上。聂师傅我不知道，但我要迅速前进的理由有两个，第一就是能空出时间思考业务逻辑（业务逻辑是最花时间的，我觉得把再多的时间放在上面都是不过分的）；第二是我个人私心，即空出时间学习更多的东西。

以上的描述清楚地显示了双方冲突的根源：

1. 代码的认同问题；
2. 项目进度和个人成长的优先级；
3. 修改别人的代码后合并在主分支上属于窃取别人劳动成果的行为；

2 和 3 两点是三观问题，没什么好评论的，但关于第一点，也就是好代码和烂代码的区别我是通过【马师傅需要建立重复的 Model 才能开发】这个现象来判断的。当时的情况是，聂师傅建立的 Model 名字和马师傅的命名习惯不一样，然后马师傅自作主张新建属于她命名习惯的 Model 继续开发。而我当时希望的方式是，马师傅不要动数据库，继续使用聂师傅的 Model。最后名字如果要改变，由我这个管理合并的人来统一改名。总之这让我觉得马师傅的抽象能力不行，无法在她自己不认同的命名规则下工作，而我认为抽象能力不行的人写的代码是烂代码。

很容易看出，我上关于述烂代码的观点其实本质上不是在到底代码烂不烂（那属于我在烦躁时不理智的判断），而是双方冲突的隐藏根源：

- 马师傅拒绝并且很讨厌和我们沟通

整件事过后，我觉得这个隐藏冲突根源其实是价值观导致的，应该是绝症，无法调节。很难简单因为马师傅讨厌和我们沟通而责备她，因为我也有不愿意去沟通的人，相信聂师傅也有。所以今后还是比较尴尬的，哈哈。嗯.....就以马师傅拒绝别人的口头禅来结束本周周记吧，我有那么一瞬间还挺喜欢这句话的：

“好了好了好了，我们先不说了.....”

— mayalin

附录 —— 流水账整理一周事项

周一

- 上午 9:30 老师讲座教如何写 Landing Page
- 上午 10:00 到下午 1:00 左右，写完 Landing Page
- 下午 1:30 左右开始上台展示自己的 Landing Page
- 下午 4:00 开始投票选择自己喜欢的项目
- 下午 5:30 分组完成马上进休息室讨论 User Story 和要实现的功能
- 十分钟后老师进来一起讨论，并简略分配了任务

-
- 开始干活，等待聂师傅创建 Github 上的项目组并邀请，在此等待期间，我在看 Code School 上的 Ruby 教程
 - 搞定了各种部署、邀请和群组后，已经是晚上 8:30 左右，期间晚饭也消耗了近半小时的时间。
 - 老师提示今天写完 User Story，明天早上开会分配具体任务
 - Code School 上的 Ruby 教程学到了晚上 11:00 左右，回家睡觉

周二

-
- 上午 7:00 吃完早餐到教室，发现同组队友马师傅写的 User Story 太好了，是自己重新认识了 User Story 的作用，于是开始翻阅《[用户故事地图](#)》一书，直到组员到齐，然后开会。
 - 上午 9:40 小组开会继续讨论 User Story 和要实现的功能，老师到场后给我们分配了任务【我负责订单模块，聂师傅负责管理员课程创建模块，马师傅负责用户的登陆登出模块】
 - 上午 10:00 写代码的 3 人开始干活完成自己的任务
 - 下午 3:00 我开始合并聂师傅和马师傅的分支，遭遇冲突，但都是非常容易解决的冲突，一切顺利
 - 下午 4:00 完成了今天的任务，上传 gif 图片到 slack
 - 下午 4:30~6:40 和各组成员扯淡并吃晚饭
 - 晚上 7:00 继续观看 Code School 的教程
 - 晚上 10:30 回家睡觉

周三

-
- 上午 7:10 到教室，看到 Tower 上的指挥官任务，要求实现 Markdown 格式，于是开始研究 Markdown 的编辑器实现
 - 上午 8:20 到下午 2:30 左右，一直在搜索如何实现 Markdown 编辑器，并在几种不同的编辑器之间犹豫。
 - 下午 3:00 随便选了个凑合的 Markdown 编辑器放在了网站里，并笼统地实现了将 Markdown 格式的内容排版后输出（即 markdown 转成 html，但是无法实现 logdown 这里的编辑器的某些格式效果）
 - 下午 3:50 左右，派 YY 助教去偷问 xdite 写的 logdown 编辑器的实现，得到“是谁要问的？”和“我到时候再和他说”两句反馈，总之就是询问未果。
 - 下午 5:00 到 11:59 继续研究 Markdown 编辑器实现，测试各种网上的 Markdown 编辑器

周四

-
- 00:00 到 9:30 继续研究 Markdown 编辑器实现。期间了解到很多前端的知识和 logdown 的开发八卦，知道了两件事：第一是前端是个非常恶心的坑；第二则是 logdown 的编辑器大有来头，是众高手呕心沥血的成果，其中技术似乎远超网上那些开源的编辑器（怪不得 xdite 当时是那样回应的）。我们项目 Markdown 编辑器面临的困境主要在于无法上传图片，更不用说上传多张图片了
 - 上午 9:50 ~ 11:00 由于熬夜，开始犯困，于是睡觉
 - 中午 11:10 ~ 11:40 被叫醒开会，只记得今后由我来管控 Github 上的合并工作
 - 中午 12:00 ~ 12:30 午饭
 - 中午 12:40 xdite 说我们写的代码是渣渣，要求停工学习 CSS 之后再写
 - 下午 1:00 ~ 2:30 看教程不知不觉又犯困睡觉
 - 下午 2:40 xdite 把我们的代码批改完后，让我们拉下分支重构代码
 - 下午 3:00 ~ 5:00 重构我们的渣代码，期间马师傅的重构逻辑与我和聂师傅两人不同（她要创建聂师傅已经创建好的 Model，然而我和聂师傅认为这样会导致 Model 名字重复和混乱，希望她不要这么做），但沟通失败，此外马师傅对我似乎很不耐烦，认为我总是问她听懂了没有的举动非常讨厌，

她比较喜欢学霸。考虑到也许是我的错（毕竟高中文化程度），并且认为以我的实力，到时候合并时即使有冲突应该也能修复，所以没有继续争辩。

- 晚饭后的 7:00 我和聂师傅已重构完各自的代码，合并完成后，在等马师傅的重构。聂师傅找我出去谈话，表示不爽（我估计期间他去找马师傅沟通也吃了闭门羹）。我们商量明天早上一定要开会商量我们组的方向，不能再渣渣。
- 7:30 ~ 9:30 聂师傅先回去研究前端的页面布局，我分别通知到了蓝心和潘师傅说次日早上要早来，好开会，但没看到马师傅，于是继续研究代码等她。
- 10:00 马师傅过来找我解决问题，解决完后我通知她次日开会，并表示她要解决的这个问题明天会议上也会提到，是针对团队合作模式下的编程逻辑的探讨，但马师傅用她的口头禅阻止了我继续说话——“好了好了好了，我们先不讨论这个”。于是我让她做好上传，然后闭嘴。
- 10:15 左右 马师傅上传分支，我尝试合并，不过冲突太多，而且昨天熬夜导致太困，于是回家睡觉

周五

- 上午 7:10 到教室，继续合并马师傅的分支。
- 上午 8:50 聂师傅、潘师傅和蓝心都到了，马师傅还没来。聂师傅于是过来和我抱怨开会的事情，我说再等等，我也还没解决马师傅的合并冲突问题。
- 上午 9:30 聂师傅让我先合并他昨晚的前端代码，我照做了。然后继续研究马师傅的分支冲突
- 上午 9:50 聂师傅让我直接重写，不用再研究，我采纳了他的意见。
- 上午 10:00 ~ 11:00 我重新实现了马师傅的任务（即用户付费判断，付费才能看教学，否则看不到），然而由于太匆忙，Bug 非常多。期间马师傅到达。
- 上午 11:10 上台 Demo，失败，xdite 严厉批评我和我们组
- 上午 11:20 潘师傅帮我上台补充她做的 Demo，获得好评，拯救了整个小组
- 中午 12:30 Demo 结束，聂师傅找我私下讨论团队合作事宜，认为应该不再让马师傅参与开发任务，而是帮忙打杂，做些不影响主项目的活
- 下午 1:00 聂师傅召集大家开会，会上我提出了整个小组的前进方向，即由我负责后端逻辑，聂师傅负责前端页面布局，其他人边学习边做杂货的方案。马师傅对此表示非常生气，并对我早上更改她的代码提出异议，认为代码有问题应该和她说，由她去改，否则这段代码的贡献者不明。换句话说就是，我不仅盗用了她的劳动成果，而且由于是我来改代码而不是她改，她因此失去了学习和练习的机会。马师傅还指责我没通知她今天早上要开会的事情。会议在剑拔弩张的气氛中结束。
- 下午 1:30~2:30 我继续解决早上重写导致的 Bug，重写并完成了马师傅的之前的工作。
- 下午 4:00~4:40 马师傅遭到 xdite 批评，先行离开，我因此松了一口气。
- 下午 5:15 全队出发晚餐

[8/16 日记 被低估的用户故事](#)

August 16, 2016

今天有一个极大的成长，即重新认识了用户故事的功能。

之前说过，编程的过程中，脑子清楚最重要。而帮助我们理清思路的方法之一就是创建 To-Do Lists。但如果一股脑地把想到的事情列出到 To-Do List 上，也会乱掉，因为其中还有优先级需要考虑，在这种一股脑全列出来情况下，各项任务的优先级是非常模糊的。再就是，为了快速记录，To-Do List 上一般都是非常碎片的表达，并非完整的语句，这会让我们进一步迷失在取舍的决定过程中，白白浪费时间。

在做产品这一个具体的情境下，避免上述情况的方法之一就是写出用户故事。这招真的非常管用，之前一直被忽略了。因为都是老师在写，我只是看，从没意识到这个方法的效果是如此惊人。写完之后，所有混沌突然间变得井井有条。

总之，吐血建议一定要自己写一波。感谢老师放我们独立开发。不得不感慨，远离教程之后，才能更好地学会教程里的内容。

8/15 日记 五周冲刺

August 16, 2016

前三周七七八八的学习之后，Rails 大概怎么玩的已经略知一二了。接下来五周要开始真正的项目，非常刺激。也许因为人类是社会性动物，我发现人与人在一起是有增益的效果的，也就是自己各方面的潜力都不知不觉被激发了，至少我的状态相比之前一个人的时候要激情得多。比如目前我所在的小组做的线上课程，我目前就完全不知道怎么开始，但不知怎么的，我并没有很慌。如果是我一个人的话，很可能就洗洗睡了，不愿意或者说害怕去接这么麻烦的事情和由此产生的压力。

总之，我已经夸下海口要杀爆其他组了（万恶的竞争意识），必须要花上十二分的努力和认真。

第三周之最坑

August 13, 2016

本周经历的 Bug 不多，但大多是很烦人的。其中最悲剧的就是 Heroku 的云端部署了。这次主要卡在上传图片的功能实现上。

在 Heroku 上，我们采用的是 carrierwave 和 亚马逊的网络储存 S3 来实现上传图片的功能。其中，导致问题出现的巨坑就来自于亚马逊的 S3。具体来说就是 carrierwave.rb 这个配置文件里 region 这个参数的设置。国外，尤其是美国，这个参数通常不用设置，使用默认值就好了。然而在中国，region 是需要设置的，如果设置错误，将会导致上传功能不能用，heroku 会报错。但是，亚马逊把这个 region 参数的值放在了一个难以发现的地方：

```
CarrierWave.configure do |config|
  if Rails.env.production?
    config.storage :fog
    config.fog_credentials = {
      provider:          'AWS',
      aws_access_key_id: ENV['AWS_ACCESS_KEY_ID'],

      aws_secret_access_key: ENV['AWS_SECRET_ACCESS_KEY'],

      region:            'ap-northeast-1'
    }
    config.fog_directory = ENV['AWS_BUCKET_NAME'] # 你設定的 bucket name

  else
    config.storage :file
  end
end
```

第三周最棒

August 13, 2016

通过本周的学习，掌握了一些的方法论，虽然零碎，但是很精髓。个人觉得它们都非常棒：

1. 【迅速果断】有事要快速做完，新事物要快速尝试，所有等以后再做就会永远错过。具体来说就是先记在本子上，然后挤出时间去做。

-
2. 人们永远都是疑心重重的，不愿意改变，对害怕新事物。所以降低疑虑增加用户信心，产品才有更高的用户数量增长。
 3. Grow Hack 只有在 PMF 之后做才有意义。(PMF 是说能在市场上稳住阵脚，也就是产品能活下去了)
 4. 使用【监视器】来帮助判断影响业绩的因素
 5. 几乎没有创业公司是因为竞争激烈而倒闭的，倒闭的原因基本上都是产品烂，即对市场没有价值。

而本周最好用的软件是 [Airbrake](#)。该软件是用来对 Heroku 上部署的软件进行 Debug 的。类似 Heroku 的云端上部署的软件，如果出错，是不会有详细的错误信息的。这是出于安全考虑，不希望把整个软件的结构暴露出来。所以，debug 的难度就增加了。由于 Airbrake 可以拦截这些云端的错误信息，并且进行人性化的处理，让我们 Debug 的效率大大提高。正是依靠 Airbrake，我后来实现了使用 [carrierwave-aws](#) 这个插件来处理储存问题（原先的 fog 储存方式相对于 carrierwave-aws 速度较慢）。

宗教与学习

August 13, 2016

对于某个观念，或者说某些方法论，人类的处理方式一般有两种。第一种方法是，通过对这个方法论进行思考，并做理性分析，得出结论，然后据此判断是否接受它。第二种方法则是不问为什么，本能地接受，像宗教一样。两种方法各有优劣，宗教干净利落，节省时间，但有信错“教”的风险，即接受了错误的观念和理论；而思考可以很大程度防止信错“教”，心智也会有着极大的成长。不过思考的过程伴随着煎熬、挣扎、痛苦和迷失，而且耗时。此外，最终人们仍然很难保证自己的思考足够“正确”。

明白了这些，剩下的就看如何取舍了。

全栈第三周——写在闲聊之后

August 13, 2016

时间过得飞快，今天已经是第三周的星期五，下星期就要进入更加紧张的实战演练了。

本星期的主要成就是完成了购物车和订单系统，然而最大的收获还是今天的分享交流会。说实在的，每次分享会我都感觉像是国外的那种 confession 会，即一帮吸毒酗酒的人在一起忏悔，并互相帮助成长.....^_^

借口与理由

今天进行软件的 Demo 展示时不知为什么我整个人变得很怂，总是找各种理由和借口希望能逃避。自我反思之后觉得，也许是因为害羞、紧张和不知名的害怕，不自觉地通过各种理由和借口来给自己壮胆吧。太 low 了，以后绝对不能再次出现这样的怂。如果怕演示烂，那就在展示之前多多练习。如果怕产品烂，就加班加点把它做好。

炫耀的倾向

今天分享完之后和 xdite 扯淡，我突然发现一个令我震惊的现象，就是我会不自觉地经常提到我所购买的某些商品。感觉像是那种来自小地方的自卑所导致的装逼行为。特别 low，尤其还 TM 是和 xdite 装逼... 一个开着宝马进城的煤老板被人鄙视后，以他的理解能力，只能回家恶狠狠地换一辆兰博基尼再度进城。我想象不出比这更伤感的人生了.....

道德绑架党

用非常努力，非常刻苦的状态，来增加别人抛弃自己时的愧疚感，以此向人们求情，希望能再被给予一次机会的人，其实本质上是很 low 的。我自己也有类似的倾向，需要提高警惕。

无论多么拼命，如果没有成绩，那就是自己的大失败，与他人无关。没有实力的善良，是廉价的善，而无法精进的拼命，则是廉价的命。

及格的速度

本周我们一共完成了 20 个左右的任务，或者说功能，然而今天的交流中我知道了，真正合格的工程师每天要完成 20 到 25 个功能或者任务。换句话说，我和及格线还差了 7 倍的速度。下星期一定要加把劲，争取到达及格线。乔老大说优秀和普通之间差距会高达上百倍，那种差距估计要经年累月，不过目前离及格线的 7 倍我觉得还是有机会可以冲刺一下的。

优秀的代码

长期以来，我总觉得优秀的代码是可维护性高，不容易出错，并且通俗易懂的代码。但我所关注的焦点其实完全不对。真正优秀的代码其实是可以赚钱的代码。知道这一点，世界和命运就足以发生改变，而想通这一点，将这一点融入血液中，则会极大地提升进阶的效率。

最佳和最合适

在训练中，我每次遇到某个问题都会尝试自己去找方法解决（大部分时候是幸运地找到了），但总会去询问 xdite 这种解决方案是否是最佳实现。几乎没有意外，我每次找的方案总是稀奇古怪的，都不是最佳实现。然后我问 xdite 为啥不直接教最佳实现，她表示这些最佳实现很多情况下对新手来说不友好，也就是不容易理解，会使得脑子乱掉。需要循序渐进，新手才能保持成就感健康地前进，而不是被因难以理解新概念所导致的挫败感击倒。所以，最佳未必是最适合，就像【文明】这个系列的游戏，如果一开始不走奴隶制，直接走目前所谓的资本主义，甚至所谓的社会主义制度的话，必定会失败，尽管后两者也许是更加先进的制度。

[ruby 里的 delete_all 和 destroy_all](#)

August 12, 2016

主要差别如下：

1. delete_all 会忽略所有的 Callbacks 比如 before_destroy 和 after_destroy；
2. delete_all 不会删除相关联的 Objects，destroy_all 会；

参考：

[delete_all vs destroy_all](#)

还有[这里](#)

[8/11 日记 - 怎样提高编程速度](#)

August 12, 2016

在我这样的乡下人的心目中，一流全栈的突出特点就是编程非常快。所以我经常关注关于速度的修炼，而今天有了重大的领悟。关于速度的诀窍就是：

脑子越清楚，编程速度越快

得出这个结论是因为今天【有限状态机】方面的训练，说实话我其实是拖到夜深人静的时候才做的，部分原因是白天时间都花在 Debug 和帮别人 Debug 上了，主要原因还是今天白天我脑子不清楚，做事效率低。

【有限状态机】的练习，是一个开头是【啊！？】，中间是【这...】，结束后是【咦？】的过程。非常非常爽。这东西会让乡下人有智商上升的快感，或者说错觉，尤其是我。因为它其实包含了一种非常强大的思维方式。如果在练习过程中有仔细思考过状态机为什么要这么设计的话，你会清楚地察觉到它在重塑你的逻辑，换句话说，就是脑子变得更清楚了。

总之，我感觉自己的编程速度又上了一个小小的台阶。所以，在此推荐大家独立完成这个练习，过程真的很酷。

我预计累计到足够多的这类思维方式之后，大脑的速度就会再次超越手速，思考的速度将远超表达，那时候就爽了。让我们的大脑一起变得更清楚吧。

August 11, 2016

我们在 Heroku 上部署产品的时候，通常会需要调用各式各样的密码来访问第三方服务，而这些密码显然

不能明文上传，所以就有了各种管理密码的方法。

在这些密码管理方法中，其中一种思路是把密码设置到环境变量里，然后调用这些变量。这样就能在隐藏密码的同时，使用它们。

Heroku 上设置环境变量的方法可以参见这里 [Configuration and Config Vars](#)。不过，如果在 Heroku 网站上直接手动设置，会很麻烦，而且后期变动或维护就更麻烦了。所以，我们采用一个叫做 [figaro](#) 的 gem 来进行密码管理。

首先安装 gem

```
gem "figaro"
```

bundle install 完成后，运行一下代码安装 figaro

```
$ bundle exec figaro install
```

这段代码会生成一个文件 config/application.yml，并且会在 .gitignore 里加入配置，让 Git 不追踪该文件。如果是团队合作，在编辑 application.yml 文件写入密钥前，可以先复制一份该文件并重命名为 application.yml.example。这个 example 文件可以让 Git 追踪，因为密钥不在里面，它的作用是让队友知道你使用 figaro 来管理密钥。

以一个例子来说明一下使用方法：

1. 打开 application.yml 文件进行编辑，设置各种密钥：

```
config/application.yml
```

```
pusher_app_id: "2954"
```

```
pusher_key: "7381a978f7dd7f9a1117"
```

```
pusher_secret: "abdc3b896a0ffb85d373"
```

2. 将密钥的设置配置到 Heroku 上：

3. \$ figaro heroku:set -e production

[8/10 日记 压力的开始](#)

August 10, 2016

这周是全栈学习的第三周，学习上的压力明显增大了。

今天第一坑是王梦琪的键盘，她所遇到的 Bug 虽然不难，但由于按键习惯不符合我的直觉，常用的保存快捷键位不是正常的 cmd + s，导致改动之后始终没能成功保存。所以 Bug 一直存在。这使得天花乱坠进行讲解的我，自信心爆棚的我，当时一下子就怀疑人生了。因为怎么看都正确的代码，各种报错，我也几近崩溃。

但说起今天最大的坑，还得算我在 heroku 上部署 jd-store 时遇到的 AWS 配置坑，不记录不行。

在 heroku 上部署 jd-store 是噩梦，carrierwave 总是会出问题，其中，以目前的实力最难以发现的是亚马逊的云储存——也就是 AWS 的配置。[这里](#)是提供解决方案的参考链接之一。

虽然各种搜索，做了各种改动，然而一整天过去之后，我还是没找到方案。最后是 xdite 老师提醒我要注意代码中的 Region 的值：

```
CarrierWave.configure do |config|
```

```
  if Rails.env.production?
```

```
    config.storage :fog
```

```
    config.fog_credentials = {
```

```
      provider:          'AWS',
```

```
      aws_access_key_id: ENV["AWS_ACCESS_KEY_ID"],
```

```
      aws_secret_access_key: ENV["AWS_SECRET_ACCESS_KEY"],
```

```
      region:            'eu-west-1'
```

```
}  
config.fog_directory = ENV["AWS_BUCKET_NAME"] # 你設定的 bucket name
```

```
else  
  config.storage :file  
end  
end
```

我用的是日本地区的 AWS，但其 Region 的值不是 Tokyo，而是要在这里查看：

就这么一个小细节，卡了我一天。我觉得这种 Bug 我是很难找出来的，还是需要求助于老师。

[8月9日 时刻提醒自己](#)

August 10, 2016

1. 永远知道当前要实现的功能是什么，这个功能必须非常具体，如果抽象说明该功能很复杂，就要继续分解这个大功能；
2. 切记四面开花，切记想到什么就做什么。列表格非常重要；
3. Must have 还是要精简，30 分钟没做完就放弃；
4. 速度仍然非常重要，速度越快，练习越多，越熟练；

[全栈第二周最棒的概念](#)

August 8, 2016

本周学的东西很杂，吸收了很多新概念。其中，印象最深的概念是关于文件命名的。

现象

当我们在开发 Rails 时使用类似 `<%= image_tag "rails.png"%>` 这个代码的时候，通常的理解是它会生成一个 html 的代码：

```

```

但是实际上，image_tag 的代码生成的是这样一个 html：

```

```

注意到 html 代码中图片的命名了吗？我们的图片文件名本来是 rails.png 的，然而 Rails 在生成 html 的时候给图片文件赋予了新的名字：rails-af27b6a414e6da000035031.png。

概念解释

现代浏览器都用缓存来提高用户的浏览速度，所以图片都被缓存在用户的本地电脑里，这样用户的浏览器就不用每次都去访问服务器获取图片，毕竟图片都一样嘛。但如果服务器上的图片改变了，这些用户本地的缓存图片怎么知道这个改变，并进行更新呢？这就是图片文件名后面那一串数字的作用了。这串数字被称作 MD5 hash，相当于文件的一个“指纹”，独一无二。如果图片文件改变了（但图片名字还是 Rails.png）的话，由于这个“指纹”随着文件的改变而改变，用户的浏览器就能知道这个改变，进而更新。所以，Rails 通过这样的重命名，就能让用户终端更好地发挥缓存的效果，只在必要时才走流量访问服务器，然后最大限度提高浏览速度。

[全栈第二周的大坑](#)

August 8, 2016

回顾了一下, 本周最大的坑当属 csrf_meta_tags 所导致的 ActionController::InvalidAuthenticityToken 报错。具体的出错和解决过程我记录在了[这里](#)。

坦白说如果没有人指点, 可能要好几天才能发现这个错误。关于这里说的“几天”, 我想进一步记录一下: 大部分新手其实有个致命弱点, 就是对信息的筛选的能力还未建立。其实如果一定要寻找这个错误, 直接 Google 错误关键词 ActionController::InvalidAuthenticityToken, 然后对比自己所有的代码就好了, 但因为对信息的筛选能力不行, 正确的解决方案放在面前都会被无视, 所以类似这种 Bug, 真的是需要老师出来指点一下的, 否则确实效率太低了, 也会导致挫败感, 进而影响新手的上进心。

[Ruby 全栈第二周周记](#)

August 8, 2016

感觉上一次来 Logdown 写东西已经是很久之前了。感觉做了很多事, 但又好像什么都没做。

完成第二周的任务【job-listing】和【jd-store】之后, 剩下的大部分时间我都在补充学习各种课外知识, 比如 html 和 css。作为一个乡下来的土包子, 发现什么都要学的时候, 心情是沮丧而兴奋的, 但无论如何, 这是新手要经历的混沌阶段。关于这周的混沌状态, 确实有些零碎的感受想谈:

功能的实现其实远比想象的要简单

通常的状态下, 我的思考速度是非常快的, 以至于经常被实现速度制约。以写作来说, 我的打字速度就跟不上我的思考速度。然而全栈学习两周以来, 由于持续写作 (尽管不是每天都写), 我渐渐发现, 我的大脑已经开始拖后腿了。编程也是如此。当 CRUD 写得越来越快的时候, 我发现实现功能其实非常简单, 难的是要实现什么样的功能。就这周的作业来说, 我大部分编程的时间都花在排版和构思整个网站的逻辑架构上, 总是各种微调, 但是每次想好要怎么调整的时候, 实现的速度都是很快的, 以至于能马上看到微调之后的结果。

也许成长的过程就是思考和输出这两者轮番牵扯的过程。每次总会有一个拖后腿的, 然后会被快的那个拉一把, 接着它变成快的那个, 然后继续拉慢的一把.....

就我的情况来说, 这次轮到思考速度方面的训练了。

视频的效率低

由于作业做得快, 自以为有空余的时间, 所以在 [CodeSchool](#) 学习补充各种编程知识。学了一星期之后, 最大的感受是看视频学习的效率很低。我也不知道具体是什么原因, 也许还是练得太少吧。听视频上讲着讲着总是不容易记住, 以至于重复听。事后想想, 还是写他个滚瓜烂熟之后, 再听讲, 学习效果才好。

Hackthon 的项目管理无处不在

这星期的各种乱学导致我的进度已经远远落后于所有人。我反思了一下, 主要是自己胃口太大了, 除了什么都想学的欲望之外, 最糟糕的是觉得短时间内可以吃个大胖子。比如 [CodeSchool](#) 上的某个课我以为一个下午能学完, 谁知花了两天时间。然后我还不知悔改, 继续专研, 不知不觉中忽视了重点。

这点上需要向 @xdite 老师学习。这家伙每次都只教一点点简单得让人觉得智商被侮辱的知识, 然后布置的任务也不多, 可是这样导致每天大家都能处于完成任务的状态, 而不是每天都“负债累累”。这正是项目管理的精髓。

我现在开始理解为什么老是要求我们要全职学习了, 尽管这些知识看起来非常简单, 但分心的代价是巨大的。人们总是高估自己的做事效率, 这是个很大的隐患。从新的一周开始, 我也要开始以最保守的心态面对自己的效率了。

[8/8 日记](#)

August 5, 2016

比起听几场讲座和看视频, 和高手们经常待在一起的收益是更大的。如果你是个上进的人, 他们的一言一行其实都会对你潜移默化, 而且是不是就会有一些很重要的观念传给你。这些观念其实司空见惯, 但你很可能

从来没有从高手们的角度去思考和正视它们。

今天发生的一件小事就让我受益很大：

我尝试通过一个循环语句 (loop) 来删除购物车里的商品，但是各种出错。最后没办法，只好找了 @xdite 。简单几下解决完之后，@xdite 和我闲聊起了项目和任务的管理方法论。她教我要克制自己的编程冲动，哪怕是看起来很简单一个功能，也要暂时搁置下来，先做必须要做的事情。她建议 30 分钟无法解决的问题就直接改变思路，不要继续一头扎进去。这个思路本质上就是要把 Must Have 精简到不能再精简的地步。我对此其实是非常感慨的，因为上个星期我就是犯了同样的错误，导致本来比较流畅的作业进度一下子被拖慢了很多。今天这个 Loop 循环也是一样，我以为很简单，但弄了一个多小时，这个时候其实应该停下来思考一下是否要改变方案暂时搁置了。和 xdite 讨论之后，我发现自己要实现的功能比她设想的要多，这是导致我做得慢的重要原因。

总之，永远不要高估自己的实力，永远把任务精简精简再精简。取舍之后所带来的速度其是会让我们更快的。(原理参见[这里](#))

8/4 日记 Hackthon、设计工作室和 Scope

August 4, 2016

今天最大的收获有两个，一个是听到了老师关于 Hackthon 的经验，另一个收获是从我们组的设计师那里了解到各种影视作品相关的工作室的八卦。

Hackthon 干货

PPT 和最后的 Presentation 要做得好；

要政治正确，适合主办方发新闻稿（毕竟 Hackthon 是本质上商业性的）；

限制的时间做出【能动的产品】即 demo；

只实现一个完美的功能，而不是十个残缺的功能；

预留三分之一的时间来应对紧急情况，只用三分之二的时间（有意让自己处于资源稀缺的情况下，这样容易专注于最重要的东西）

工作室

Maya 可以做动画，是非常值得学习的软件

Scope

以下是聂师傅的代码：

```
def publish!
  if is_hidden
    self.is_hidden = false
    self.save
  end
end

def hide!
  if !is_hidden
    self.is_hidden = true
    self.save
  end
end
```

这里的 self 一词非常微妙，如果直接说结论就是：

Reading attributes doesn't need "self", but setting attributes needs "self".

整个原理就是，读取的时候，在 method（或者说函数）内层如果读不到某变量，那程序就会在外层寻找，最终，它会找到处于最外层的属于 Model 本身的 attributes；然而赋值时，如果不指定 self，那程序就会在 method（或者说函数）内层创建一个新变量，然后赋值给它。当 method 调用结束，该变量也会被注销，无法从外层被访问。

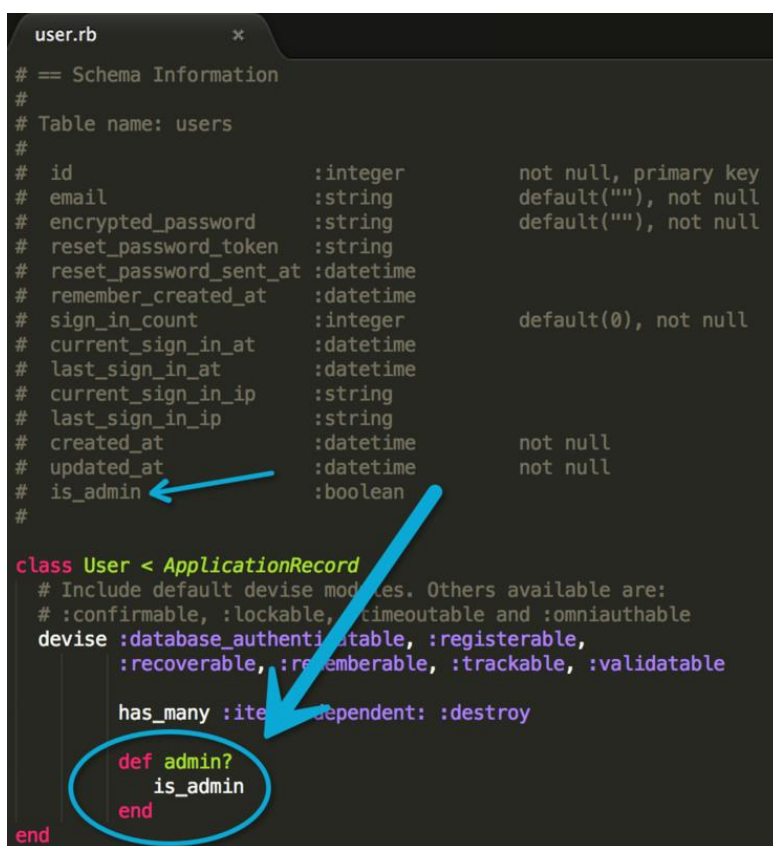
[8/3 日记 封装与数据架构思路](#)

August 4, 2016

封装(wrap it in a method)的重要性

下面这段代码，把 is_admin 封装在了 admin? 的函数里，然后别的地方调用 current.admin?

虽然不封装，调用的时候使用 current.is_admin 也可以，不过将来判定 admin 的条件可能会变，这样的话，很多调用的地方就要重新改了，非常麻烦。综上，编程过程中要时刻思考封装问题。



```
user.rb
# == Schema Information
#
# Table name: users
#
# id :integer not null, primary key
# email :string default(""), not null
# encrypted_password :string default(""), not null
# reset_password_token :string
# reset_password_sent_at :datetime
# remember_created_at :datetime
# sign_in_count :integer default(0), not null
# current_sign_in_at :datetime
# last_sign_in_at :datetime
# current_sign_in_ip :string
# last_sign_in_ip :string
# created_at :datetime not null
# updated_at :datetime not null
# is_admin :boolean
#

class User < ApplicationRecord
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable

  has_many :items, dependent: :destroy

  def admin?
    is_admin
  end
end
```

多角色的管理

尽管网站可能有多个角色，甚至今后还会增加角色，数据库里也只需要建立一个 devise user 就好。角色之间的区分通过 user.is_admin? 这种 column 来区分。

[8/2 日记 InvalidAuthenticityToken](#)

August 2, 2016

今天最值得记录的就是这个叫做 InvalidAuthenticityToken 的 Bug，这个 Bug 是 2 组的同学遇到的，卡了一个下午。我去看过之后，也自问绝对无法在短时间内解出。最后 @xdite 出马，才搞定。

Bug 现象描述

注销登出还有任何切换和刷新页面的操作都会报错，出现以下界面：

ActionController::InvalidAuthenticityToken in Devise::SessionsController#destroy

ActionController::InvalidAuthenticityToken

Extracted source (around line #195):

```
193
194   def handle_unverified_request
195     raise ActionController::InvalidAuthenticityToken
196   end
197 end
198 end
```

Rails.root: /Users/TinkerInn/Documents/full_stack_training/jdstore

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

```
actionpack (5.0.0) lib/action_controller/metal/request_forgery_protection.rb:195:in 'handle_unverified_request'
actionpack (5.0.0) lib/action_controller/metal/request_forgery_protection.rb:223:in 'handle_unverified_request'
devise (4.2.0) lib/devise/controllers/helpers.rb:253:in 'handle_unverified_request'
actionpack (5.0.0) lib/action_controller/metal/request_forgery_protection.rb:218:in 'verify_authenticity_token'
activesupport (5.0.0) lib/active_support/callbacks.rb:182:in 'block in make_lambda'
activesupport (5.0.0) lib/active_support/callbacks.rb:169:in 'block (2 levels) in halting'
actionpack (5.0.0) lib/abstract_controller/callbacks.rb:12:in 'block (2 levels) in <module:Callbacks>'
activesupport (5.0.0) lib/active_support/callbacks.rb:170:in 'block in halting'
activesupport (5.0.0) lib/active_support/callbacks.rb:454:in 'block in call'
activesupport (5.0.0) lib/active_support/callbacks.rb:454:in 'each'
activesupport (5.0.0) lib/active_support/callbacks.rb:454:in 'call'
activesupport (5.0.0) lib/active_support/callbacks.rb:101:in 'run_callbacks'
activesupport (5.0.0) lib/active_support/callbacks.rb:760:in 'run_callbacks'
activesupport (5.0.0) lib/active_support/callbacks.rb:760:in 'run_callbacks'
activesupport (5.0.0) lib/active_support/callbacks.rb:760:in 'run_callbacks'
```

Bug 原因和解决

因为 html 缺少了 `<%= csrf_meta_tags %>` 导致（一般这串代码会在 layout 里的全局输出 html.erb 文件里，比如 application.html.erb 或者 admin.html.erb）。`<%= csrf_meta_tags %>` 是生成验证 token 的一个机制，为了防止机器人过来批量操作网页。

PS:

相关内容还可以参考 stackoverflow，见[这里](#)

August 2, 2016

解释一个东西的最好方法是举例，所以我们直接看使用的范例。要注意的是，Controller 里的 render 和 View 里的 render 是不一样的，要区分开来。

render :template - render 特定的模板

假设我们在这个目录下有个 html 的模板：views/foo/bar.html.erb，我们要渲染这段代码可以用以下任一种代码，两种代码等价（注意双引号）：

```
render views/foo/bar.html.erb
```

```
render "foo/bar"
```

如果是同一个 Controller 下的 action 的 view（注意这句话的表达，是 action 的 view 哦——“...render 的特性是不跑 controller action，而是直接将该 action 下预设的模板传出来，而我们可以指定我们自己的特定模板”，可以使用以下表达，两种代码等价（此处 render 后面接的是 action 的名字）：

```
render :foobar
```

```
render "foobar"
```

render :text - render 纯文字

用 render 直接输出纯文字，是一种不常用的方法，不过某些时候也许是一种输出的解决方案，比如代码中单双引号冲突时。

```
render text: "This text is from render - Twentynight"
```

render options - render 纯文字

Controller 里的 render 还可以设置两种常用的 options：

:content_type

render 默认会找 text/html 类型的文件，但可以通过 :content_type 这个 option 来指定其他类型的档案，比如：

```
render file: filename, content_type: "application/rss"
```

PS：content_type 所支持的类型可以 Google 搜索 MIME 这个关键词

Rails 官方说明

By default, Rails will serve the results of a rendering operation with the MIME content-type of text/html (or application/json if you use the :json option, or application/xml for the :xml option.). There are times when you might like to change this, and you can do so by setting the :content_type option

:layout

Rails 默认的 layout 文件是 app/view/layouts/application.html.erb，但我们也可以指定一个其他的 layout，比如 app/view/layouts/admin.html.erb，然后只要在 controller 中指定使用这个新的 layout 就可以了：

```
class AdminsController < ApplicationController
```

```
  layout "admin"
```

```
end
```

有时，我们也可以选择高级用法。比如我们可以指定 Controller 中的某个 action 要使用 admin layout：

```
class AdminsController < ApplicationController
```

```
  layout "admin", :only => :new
```

```
  # 另外也可以在 render 的时候就指定要使用哪一个 layout
```

```
  def show
```

```
    render :layout => "admin"
```

```
  end
```

```
  # 甚至可以指定模板再指定 layout
```

```
  def index
```

```
    render :template => "others/weired_topics", layout: "admin"
```

```
  end
```

```
end
```

补充知识 - redirect_to 和 render

在练习中，会看到以下两种代码：

```
redirect_to root_path
```

```
render :new
```

两者的效果都似乎是引导到某个页面，不过它们是不一样的。最关键的区别在于是否运行了 Controller 中相应的 action。

redirect_to 是真正地跳转，通常是在需要从当前页面跳转到另一个页面时使用，它会执行目的地页面所对应的 Controller 中的 action。

render 是将所对应的页面（或者所指定的 action 对应的页面，以上述例子来说，就是 :new 这个 action 所对应的页面）的模板拿出来刷新而已，并没有执行相应的 action（以上述例子来说，:new 这个 action 没有被执行）。

通常 render 的使用时机是让使用者回到同一个页面，例如表单填写不完全时再重回表单填写页，这样做的原因是 render 会传模板给使用者，而这个模板在使用者第一次送出表单时就已经被存起来了，所以 render 同一个模板的时候就会保留刚刚使用者打的表单资料，不用全部重打。

— @xdite

若使用 `redirect_to` 跳转到同一个表单页面就会是一个全新的模板，不会有任何送出前填写的资料，所以适合在跳转到不同页面时使用。

[8/1 日记 - 新学到的 debug 的方法](#)

August 2, 2016

今天一天都在 Debug，有一些不大不小的收获。今天要记录这些 Bug，还有由此学会的 Debug 的方法。这些内容价值 5 万元。

Bug #1

【描述】

想新建一个 migration 来增加栏位 (column)，但 `db:migration` 总是失败。

【解决】

问题出在单复数

`add_column :group_relationship` 应该写成 `add_column :group_relationships`

Bug #2

【描述】

job-listing app 中表单一直无法提交，停留在 `render:new` 和 `render:edit`，并且 `publish` 和 `hide` 按键功能失效，之后还发生了无法注册 User 的状况（点击 `submit` 按钮后，chrome 停留在注册页无响应，诡异的是 safari 表现正常）

【解决】

通过在 `publish` 函数中的 `.save` 后增加感叹号！造成中断报错，进而得知是缺少 User 导致的无法提交问题。缺少 User 的原因有两个，第一是没有在 controller 的 `create` 中将 User 保存（`@job.user = current_user`），第二是因为 Job 和 User 这两个 Model 之间建立了从属关系，然而 Job 这个 Model 里没有 `user_id` 栏位，导致 Rails 的 `convention over configuration` 特性无法发挥作用。

Rails 中的感叹号（如 `@topic.save!`）会有中断效果，是 debug 的利器，在不报错的时候可以考虑使用。但是产品部署时不要使用，中断会让用户感觉你有问题。在 `Develop` 阶段使用。

Debug tips #1

用浏览器查看生成的 html 代码，观测是否符合预期。

Debug tips #2

使用 `flash[:notice] = "Fuck"` 来在各种可疑地方插入讯息，观测程序行为。

Debug tips #3

在 Rails 中使用感叹号！来触发中断，以查看错误信息。例子：`resume.save!`

[View 里的 render](#)

August 2, 2016

View 中的 `render` 一般是用于输出 `partial` 的指令。所以明白了 `partial`，就明白了 view 里 `render` 的用法。
partial

Rails 中的 `partial` 是 View（也就是 html）页面中被抽象出来的一段代码。这些 `partial` 中的代码之所以被抽象出来，是为了更好地整理整个 html 的结构、增加页面效能、或者使该代码得以在多个 html 页面之间进行重用。

partial 的使用例子

可以看到，以下 new.html.erb 和 edit.html.erb 两个文件里的 html 代码存在高度重复的 block：

```
edit.html.erb
<div class="col-md-4 col-md-offset-4">
  <h2>Edit a Job</h2>
  <hr>
  <%= simple_form_for @job do |f| %>
    <div class="form-job">
      <%= f.input :title, input_html: { class: "form-control"} %>
      <%= f.input :description, input_html: { class: "form-control"} %>
      <%= f.input :wage_lower_bound, input_html: { class: "form-control"} %>
      <%= f.input :wage_upper_bound, input_html: { class: "form-control"} %>
      <%= f.input :contact, input_html: { class: "form-control"} %>
      <%= f.input :is_hidden, as: :boolean, checked_value: true, unchecked_value: false %>
    </div>
    <%= f.submit "Submit", class: "btn btn-primary", data: { disable_with: "Submitting..." } %>
  <% end %>
</div>
```

```
new.html.erb
<div class="col-md-4 col-md-offset-4">
  <h2>Add a Job</h2>
  <hr>
  <%= simple_form_for @job do |f| %>
    <div class="form-job">
      <%= f.input :title, input_html: { class: "form-control"} %>
      <%= f.input :description, input_html: { class: "form-control"} %>
      <%= f.input :wage_lower_bound, input_html: { class: "form-control"} %>
      <%= f.input :wage_upper_bound, input_html: { class: "form-control"} %>
      <%= f.input :contact, input_html: { class: "form-control"} %>
      <%= f.input :is_hidden, as: :boolean, checked_value: true, unchecked_value: false %>
    </div>
    <%= f.submit "Submit", class: "btn btn-primary", data: { disable_with: "Submitting..." } %>
  <% end %>
</div>
```

这时我们可以通过建立 _form.html.erb 来将这段重复的代码搬过来：

```
_form.html.erb
<%= simple_form_for @job do |f| %>
  <div class="form-job">
    <%= f.input :title, input_html: { class: "form-control"} %>
    <%= f.input :description, input_html: { class: "form-control"} %>
    <%= f.input :wage_lower_bound, input_html: { class: "form-control"} %>
    <%= f.input :wage_upper_bound, input_html: { class: "form-control"} %>
    <%= f.input :contact, input_html: { class: "form-control"} %>
```

```
<%= f.input :is_hidden, as: :boolean, checked_value: true, unchecked_value: false %>
</div>
<%= f.submit "Submit", class: "btn btn-primary", data: { disable_with: "Submitting..." } %>
<% end %>
```

然后 new.html.erb 和 edit.html.erb 两个文件里的代码改成：

edit.html.erb

```
<div class="col-md-4 col-md-offset-4">
  <h2>Edit a Job</h2>
  <hr>
  <%= render "form"%>
</div>
```

new.html.erb

```
<div class="col-md-4 col-md-offset-4">
  <h2>Add a Job</h2>
  <hr>
  <%= render "form"%>
</div>
```

于是，render 就被用来输出被抽象到 partial 里的代码。语法为：<%= render "xxx"%>

[Rails 中的 Scope](#)

August 1, 2016

scope 是什么

Rails 中的 scope 是用来打包复杂语法，或者说把复杂语法抽象成简单符号的 API。这里打包的语法通常是数据库的查询语法。

举例来说，练习中我们用 @group.posts.order("created_at DESC") 来对 posts 进行排序，而且很多地方都用到这段代码。所以我们可以使用 scope :recent, -> { order("created_at DESC")} 来将 order("created_at DESC") 封装成 recent。然后我们就可以把原来冗长的代码缩写成 @group.posts.recent。

什么情况下使用 scope

一般来说，一下两种情况下可以考虑使用 scope：

- 有过于复杂的数据库查询语法代码
- 有过于重复使用的数据库查询语法代码

scope 的使用方法

带参数的使用方式

```
class Post < ActiveRecord::Base
  scope :created_before, ->(time) { where("created_at < ?", time) }
end
Event.created_before(Time.now)
```

无参数的使用方式

```
class Post < ActiveRecord::Base
  scope :recent, -> { order("created_at DESC")}
end
@group.posts.recent
```

组合的使用

```
class Event < ActiveRecord::Base
  scope :recent, -> { order("created_at DESC")}
  scope :created_before, ->(time) { where("created_at < ?", time) }
end

Event.recent.created_before(Time.now)
```

全栈学习第一周周记

July 29, 2016

第一周在学习新知识和找住房中度过。感觉这一周过得非常快，好像什么事情都没做。因为搞来搞去其实就写了两个程序。而且还非常小。我最大的感受就是，这些东西让教练来写顶多需要 2 个小时，然而我却花了半个多月（因为开课前已经写了很久）。

以下事情我本该在课前完成的：

1、住房问题：其实需求很简单，就想离教室近一些，但又不希望是天价。然而直到开课了，住房还没搞定，弄的我非常狼狈。最后是通过自如客这个 app 搞定的，我来之前根本不知道还有这么好的租房服务。这次长经验了，以后不会再在住房上浪费时间。

2、显示器问题：因为我用的是 12 寸的 Macbook，只有一个 USB-C 插口。它的特殊性在于无法支持 DisplayPort 接口，只能支持 HDMI，然而教室的显示器是不支持 HDMI 的，只支持 DisplayPort、VGA 和 DVI 三个口（是的，我也觉得非常诡异，毕竟 VGA 是上个时代的接口了）。最后我不得不去 Apple Store 购买了这种[接口](#)，这过程其实浪费了很多时间。还有就是市面上所有的 DisplayPort 转 HDMI 线都是单向的，也就是数据流只能从 DP 到 HDMI，不要妄想买这种线来接 Macbook。总之，显示器问题应该和助教之前就沟通好，不应该开课了还没解决，以后需注意。大屏显示器能提高阅读效率，所以是必须的。

3、我的 12 寸视网膜 Macbook 是 2015 年的顶配版，之前看评论，以为不能胜任开发的工作，但知乎那帮傻逼其实都是扯淡，15 年版 Macbook 不仅完美胜任 rails 开发，而且非常流畅。浪费我一堆时间去看配置准备购买新电脑，然后今年（2016）即将出现新版 Mac 的这一事实让我在购买新电脑的决策上浪费了更多的时间——因为想省下钱等九月份它出之后购买。

以上三件事是本周浪费我最多时间的三件事，写出来让大家参考，不再重走我的弯路。

在之前看到 xdite 老师快速地实现搜索功能之后，今天又看到一篇文章 [Why working quickly is more important than it seems](#)，里面关于速度的方法论，或者关于速度的观念让我非常震惊。我希望分享这篇文章来作为我第一周对编程的感受，并确定今后努力的方向。

如果不出意外，它应该是整个疗程中（没错，我确实觉得自己是花钱来李老师的软件学院这治疗的），属于我的最好鸡血：

The prescription must be that if there's something you want to do a lot of and get good at—like write, or fix bugs—you should try to do it faster.

That doesn't mean be sloppy. But it does mean, push yourself to go faster than you think is healthy.

That's because the task will come to cost less in your mind; it'll have a lower activation energy.

还有一些我认为精彩的段落：

...

If you work quickly, the cost of doing something new will seem lower in your mind. So you'll be inclined to do more.

...

The converse is true, too. If every time you write a blog post it takes you six months, and you're sitting around your apartment on a Sunday afternoon thinking of stuff to do, you're probably not going to think of starting a blog post, because it'll feel too expensive.

...

...

It's now well known on the web that slow server response times drive users away. A slow website feels broken. It frustrates the goer's desire. Probably it deprives them of some dopaminergic reward.

...

...

The general rule seems to be: systems which eat items quickly are fed more items. Slow systems starve.

...

...

Slowness seems to make a special contribution to this picture in our heads. Time is especially valuable. So as we learn that a task is slow, an especial cost accrues to it. Whenever we think of doing the task again, we see how expensive it is, and bail.

That's why speed matters.

— James Somers

速度就是力量，速度带来加速度，速度产生复利效应.....本周已经适应学习环境，今后每天都要争取比昨天快那么一点点.....【go faster than you think is healthy】

[全栈学习第一周的最棒概念和工具](#)

July 28, 2016

本周学习最棒的软件当属 [Sublime](#)，个人觉得比 Atom 好用，至少对像我一样的新手而言比较亲切。准备利用周末好好探索一下，释放出整个 App 的全部潜力以进一步提升效率。

概念是学习和成长过程中那些精髓而重要的东西，然而本周接触到的那些最棒的知识，与其说是概念，不如说是思考方式和思考的视角。

第一个最棒是把注意力当成一种资源看待，这一观点来自课程的发起人李老师。尽管和让我知道了他的发小这件事比起来微不足道，但这一个观念的转变依然是有巨大影响力的，至少对我这种乡下来的新人是这样的。

第二个最棒知识是知道了如何对已经建立的 Model 进行各种改动。这些改动是要通过 migration 来进行的。我觉得我知道了这个知识之后，自己的自信心一下就起来了。这个小知识很多人可能很不在意，但对我来说，让我理解了一个 Cascading 的方法论。我之前就不明白什么叫 CSS，为啥是 Cascading，当 xdite 老师和我讲解改动 Model 要利用 migration 已经这么做的原因之后，我一下就理解了为啥是 Cascading，以及这一看似简单的方法论背后隐藏的强大力量。有那么一瞬间，我觉得发明这个概念的先人们真的非常厉害。

[全栈学习第一周的三个大坑](#)

July 28, 2016

本周的编程过程中遇到过各种各样的问题，其中印象深刻的大坑有三个：

第一个坑是关于 job_params 函数的，也就是被称为 Strong Parameter 的过滤器的坑。我在之前的日志里提到过：[Rails 里的参数一直为 nil 的现象](#)

第二个坑是第二小组的谢萍老师向我求助的 Bug。当时的她做到招聘网站练习中的 Step 3，这一步需要新增「薪资上限」与「薪资下限」和「联络方式」。

当时她使用 rails g migration add_more_detail_to_job 来新增，但在接下来的 rake db:migrate 这步运行中却遭遇了报错。当时包括我和助教 yy 老师在内的三个人都没能解决，我们用了包括 db:drop、db:create、db:migrate 重建数据酷在内的各种作为新人所能想到的“最强”大招，都没能解决。

当晚北京下了很大的雨，大家郁闷而归。但第二天，谢萍老师自己把问题解决了，不过她采用的方式是重头再做一遍。她这一举动使我没法继续追踪这个大 Bug 了。我因此非常伤感。

无论如何，谢萍老师这个 Bug 绝对算得上本周一大坑，尽管已经无从重现。

最后一个坑是属于不起眼，但如果因此无视，很可能造成极大损失的坑。它就是关于 `method: :post` 的坑，我把具体细节更新在[这里](#)了。

在这个坑里，我发现使用 `method: :publish` 程序也能正确运行。如果我不是多个心眼，去问了一下 xdite 老师，我可能以后都会写 `method: :someMethod` 了。从与 xdite 老师的交流看来，如果不纠正回来，这很可能在将来的某个时期造成难以预料的后果。因为这属于[第一印象错误](#)。

感谢这三大坑（也许不应该谢第二个坑），我又成长了对高手们来说的一小步，对我来说的一大步。

PS：

更新第四个坑

第三组的 @nieyicheng 大神遇到了一个问题，就是关于他的 `<footer>` 的位置问题。他的 `<footer>` 每次都会显示在页面的最上方，而且好几位同学都没找出原因。我也因过于得瑟，debug 失败。不过最后我还是找到原因了，他的 `index.html.erb` 文件里的 `</table>` 标签拼写错误，写成了 `</tbale>`。

感想：1、拼写错误造成的 Bug 其实对新手来说也是很大的坑；2、对新手来说，看起来很复杂乃至诡异的 Bug，通常都是很简单甚至是傻不拉几的小失误造成的；3、在 `admin.html.erb` 文件里，也容易出现 `<footer>` 位置不对的现象，产生的原因是没给 `<div>` 标签增加 `class='container-fluid'`

此外，可参考[这里](#)学习 Bootstrap

[7/28 日记 - 关于 method: :post](#)

July 28, 2016

新知识：用 `link_to` 激活 controller 里的 methods 时，需要增加 `method: :someMethods` 这个参数指定。由于之前关于 Edit 的代码如下：

```
<%= link_to("Edit", edit_admin_job_path(job)) %>
```

于是我以为关于 Publish 和 Hide 这两个自定义的 method 的代码也是类似：

```
<%= link_to("Publish", publish_admin_job_path(job), class: "btn btn-sm btn-default") %>
```

```
<%= link_to("Hide", hide_admin_job_path(job), class: "btn btn-sm btn-default") %>
```

对比课前中级训练作业的代码后发现需要增加不对，需要增加 `method: :someMethods` 这个参数指定。正确代码如下：

```
<%= link_to("Publish", publish_admin_job_path(job), method: :publish, class: "btn btn-sm btn-default") %>
```

```
<%= link_to("Hide", hide_admin_job_path(job), method: :hide, class: "btn btn-sm btn-default") %>
```

经过 @xdite 老师的指点，真正的正确代码如下（不是 `method: :someMethods` 而是 `method: :post`，尽管上面的代码也能运行）：

```
<%= link_to("Publish", publish_admin_job_path(job), method: :post, class: "btn btn-sm btn-default") %>
```

```
<%= link_to("Hide", hide_admin_job_path(job), method: :post, class: "btn btn-sm btn-default") %>
```

[7/27 日记](#)

July 27, 2016

今天几乎一天都在帮组员解决问题。然后根据老师的解答重头做了一遍 job-listing，并且和自己做的进行了对比（老师的整个网站结构的思路和我完全不同，目前还在尝试理解中）。

主要收获是解决一个新增的问题【对 attribute 新增默认值】。

问题描述：

原文件

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

新增一个 default: true 之后变成

```
add_column :jobs, :is_hidden, :boolean, default: true
```

执行 rake db:migrate 之后，没反应

新生成的 job 实例 (instance) .is_hidden 的值仍然是 nil

解决方案：

通过 rails g migration add_default_value_to_show_attribute 增加一层 migration 来叠上最新的改动

参考[这里](#)

结论：每次对 model 进行操作，比如新增变量，或者增加默认值 default，都需要通过新增 migration 来实现，而不是修改现有的 migration 来实现。所以直接修改 db/migrate 文件夹下的文件是不行的

注意 增加默认值属于修改变量的范畴，要用 change_column 而不是 add_column。

[7/26 日记](#)

July 27, 2016

今天独立写完了招聘网站的基本功能。

今天的情绪高峰期是解决一个关于过滤器的 Bug 问题，即 job_params 的问题

今天巩固了 Rails 的基础知识

今天对自己的评价是：还是太慢，注意力不够集中和。

明天需要继续把老师的答案看一遍，和自己的对比。还要增加自己的编程速度，这样可以空出时间教组员成长。

July 26, 2016

Bug 描述：

最开始通过以下代码生成 model

```
rails g model job title:string description:text wage_upper_bound:integer wage_lower_bound:integer
```

之后用 rails g migration add_contact_to_job 来增加一个 attribute（具体增加 attributes 方法参见教程<https://xin.ontrackapp.com/posts/70>）

然后 rails s 运行服务器后发现新增的 contact 无法存入参数，rails console 分析显示，参数 contact 一直是 nil。

Bug 原因：

增加新的参数之后，没有在 controller 中的 job_params 函数中增加它

解决方案：

在 job_params 函数中，也就是被称为 Strong Parameter 的过滤架构中增加新的 attribute：

```
params.require(:job).permit(:title, :description, :wage_upper_bound, :wage_lower_bound)
```

变成

```
params.require(:job).permit(:title, :description, :wage_upper_bound, :wage_lower_bound, :contact)
```

[7/25 日记](#)

July 25, 2016

说实在的，今天老师讲的内容我几乎全忘记了，只留下一些碎片：

碎片一：自学的英文不是 self-learning，而是 self-teaching；

碎片二：比时间更重要的资源是我们的注意力；

碎片三：用 User Story 的思考方式来开发产品；

碎片四：重新做人，看看能坚持多久。

碎片五：/eks/dite 其实叫 叉 dite 也行。

其实不记得也很正常，注意力不在那些地方嘛。以前的英语老师就说过，他去外面听课，别人讲的什么全都忘完了，但扯的那些淡，栩栩如生.....

所以今天的高峰自然是笑来老师的出现，不过我也已经忘了他讲什么了，只记得他的气场很煽动。不过我喜欢他的煽动性，也享受这种煽动性——估计宗教狂热分子们和我彼刻的感受是一样的，都有涅槃的幻觉。

尽管感觉一天里什么也没学到，不过我观念确实发生了一些改变，目前只想赶紧做事。观念的改变对我这种人来说，通常比明确掌握某项技能更重要，所以还算满意——有很多瞬间，我觉得付出的代价，其实在听到某一句话就已经值回来了（当然，还是要尽可能多捞一点，压榨笑来老师的宝贵经验）。

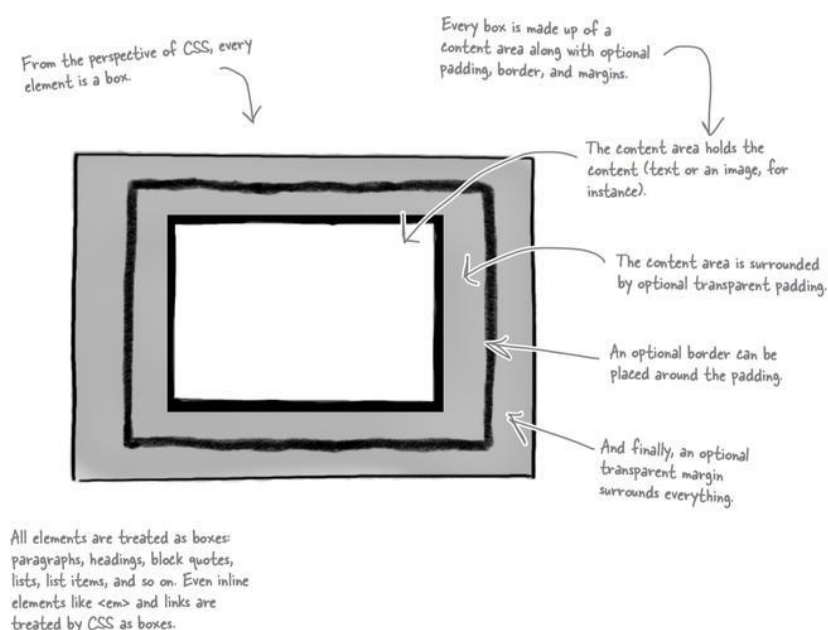
若要用一个词形容今天的工作，那就是【好玩】。

明天要起床早一点，提前到，把今天写完的代码再多过几遍。因为【胡搞瞎搞】的关键是【重复胡搞瞎搞】。

[CSS 自学作业](#)

July 25, 2016

margin、padding 还有 box model 结构：



为何要使用 em 而非 px 来定义字的大小：

```
h1 {font-size: 16px;}
```

```
h1 {font-size: 1em;}
```

px 是精确地指定字体的尺寸大小（通过像素指定），而 em 则是字体的相对尺寸大小，相对于浏览器对该字体的默认尺寸。所以使用 em 可以使字体更好地适应多种浏览器（自适应浏览器）。

与内边距的设置相同，这些值的顺序是从上外边距 (top) 开始围着元素顺时针旋转的：

margin: top right bottom left

所以以下 CSS 的所设定的 margin 参数为：margin-top = 10px；margin-right = 0px；margin-left = 5px；margin-bottom = 15px。

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

[HTML 自学作业](#)

July 25, 2016

<div> 和 ：

这两者的最主要区别在于是否另起一行——<div> 需要另起一行，而 是接着前面的内容。

把内容看成一种流体的话，<div> 包裹的流体和前后流体是断开的，而 包裹的流体则是与前后连接在一起的，是连续的。

这也正是 <div> 被称为【Block（块）】元素， 被称为【inline（内联）】元素的原因。

class 和 id：

一个 class 可以对应多个 HTML 元素，但一个 id 能且仅能对应一个特定的 HTML 元素。id 属性只能在每个 HTML 文档中出现一次，class 可以出现多次。

```
#sidebar p {
    font-style: italic;
    text-align: right;
    margin-top: 0.5em;
}
```

上面的样式只会应用于出现在 id 是 sidebar 的元素内的段落。这个元素很可能是 div 或者是表格单元，尽管它也可能是一个表格或者其他块级元素。

<p> 和
：

<p> tag 一般用来标记段落，而
 则专门用来空行的。
 没有结束标记 </br>。

<table>：

```
<table border="1">
<tr>
<th>Heading</th>
<th>Another Heading</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

以上代码在浏览器显示如下

Heading Another Heading

row 1, cell 1 row 1, cell 2

row 2, cell 1 row 2, cell 2

<tr>, <th>, <td> 和分别对应行、标题化的列（上面例子是把内容变粗体），和普通列（不变粗体）。

胡搞瞎搞的关键

July 24, 2016

以前的一位李恩师说过，如果你进入一个新领域，最快熟悉它的方式就是去胡搞瞎搞，关键是要 搞。

我非常赞同他的观点，直到遇到了 @xdite 老师。

尽管只有非常短的时间，但 @xdite 老师让我重新认识了 胡搞瞎搞。

胡搞瞎搞 的关键不是 搞，而是 重复 搞。

三遍确实重要，每次重复都有新发现。不过重复多了会因为不断深入的理解而产生愚蠢的优越感，这也是需要警惕的。

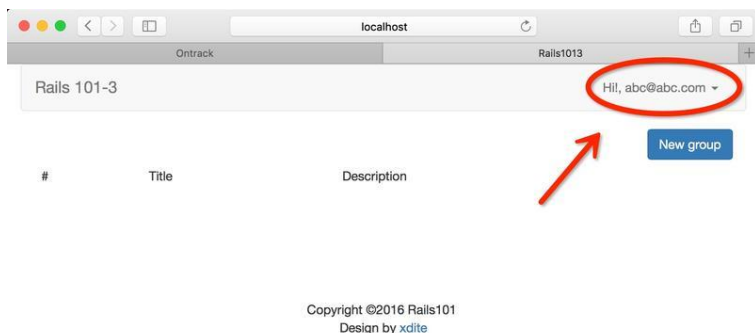
PS：

对了，这次没有这位李恩师，我也不知道 @xdite。然后虽然我是你们中的最后一名，但我已经迫不及待地要秒杀你们啦（这就是刚说的愚蠢的优越感）.....^_^

下拉菜单没反应了

July 24, 2016

编辑完成 app/views/common/_navbar.html.erb 文件之后，刷新测试网页，发现本应实现的下拉菜单鼠标点击没有反应：



仔细研究发现是 app/assets/javascripts/application.js 文件中没有加上：

```
//= require bootstrap/dropdown
```

文件中加上 `//= require bootstrap/dropdown` 之后，一切恢复。

PS：

之前一直以为 application.js 文件中，前面加了 `//` 这些符号的行都是注释，没想到居然也会影响网页的显示。看来它们中有些不是注释，很可能那些不包含 `=` 的 `//` 才是注释，有等于号的 `//=` 就不是注释。

有时候你检查了半天也没发现为什么连不上网页

July 24, 2016

.....然后你忍不住开始怀疑国家。有“忍不住”这一迟疑的动作，是因为这个网页是你用 Rails 搭建的本地服务器上的，如果真是国家，那也太可怕了。

检查到最后，发现原来是教程上的网址打错了：

目標

在這一節，我們會完成以下事項：

- 使用者在首頁，可以直接看到「討論群一覽表」

步驟

Step 1. 修改 routing

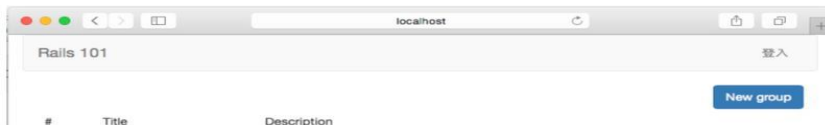
把 `root 'welcome#index'` 刪掉，換成 `root 'groups#index'`

(記得 `+` `-` 不要打進去，他們各代表「加這一行」 / 「減這一行」的意思。)

```
config/routes.rb
Rails.application.routes.draw do
  resources :groups
+  root 'groups#index'
-  root 'welcome#index'
end
```



然後你就可以在 <http://localhost:3000/> 看到 Group 一覽表了



原本是 localhost，教程上错写为 locahost。所以如果直接点击教程上的网址，会怎么也连不上。还是自己手动输入，利用 Safari 的自动补充功能，补全之前其所记忆的正确网址吧。

PS：

考虑到教程上的这个错误链接在很前面，所以如果你不小心第一次就点击这个链接，那自动补全功能将会记住这个错误链接，导致之后每次都是错误的。之前[关于第一次](#)记录过这个现象。

[一个普通但新手时期得注意的 Bug](#)

July 24, 2016

在使用 rails g model group title:string description:text 建立 group model 时发生了如下错误：

```
rails101-3 -- -bash -- 100x35
19:in `<top (required)>':
  from /Users/TinkerInn/.rvm/rubies/ruby-2.3.1/lib/ruby/2.3.0/rubygems/core_ext/kernel_require
.rb:55:in `require'
  from /Users/TinkerInn/.rvm/rubies/ruby-2.3.1/lib/ruby/2.3.0/rubygems/core_ext/kernel_require
.rb:55:in `require'
  from -e:1:in `<main>'
TinkerInn:rails101-3 $ rails -v
Rails 5.0.0
TinkerInn:rails101-3 $ rails g model group title:string description:text
/Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing/mapper.rb:32
8:in `check_part': Missing :controller key on routes definition, please check your routes. (Argument
Error)
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
/mapper.rb:308:in `check_controller_and_action'
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
/mapper.rb:251:in `normalize_options!'
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
/mapper.rb:115:in `initialize'
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
/mapper.rb:68:in `new'
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
/mapper.rb:68:in `build'
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
/mapper.rb:1698:in `add_route'
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
/mapper.rb:1670:in `decomposed_match'
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
/mapper.rb:1634:in `block in match'
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
/mapper.rb:1617:in `each'
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
/mapper.rb:1617:in `match'
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
/mapper.rb:1929:in `match_root_route'
  from /Users/TinkerInn/.rvm/gems/ruby-2.3.1/gems/actionpack-5.0.0/lib/action_dispatch/routing
```

仔细检查发现的输入的指令，没错之后，我仔细看了一下报错信息，注意到这句话：

.....

Missing :controller key on routes definition, please check your routes. (ArgumentError)

.....

于是去检查 config/routes.rb 文件。删掉我提前编辑输入的 `root groups#index` 之后，一切恢复正常。group

model 得以创建。

看来有些东西不能太过超前。

需要多次记忆的碎片

July 23, 2016

这些类似目录结构的关系需要多次记忆：

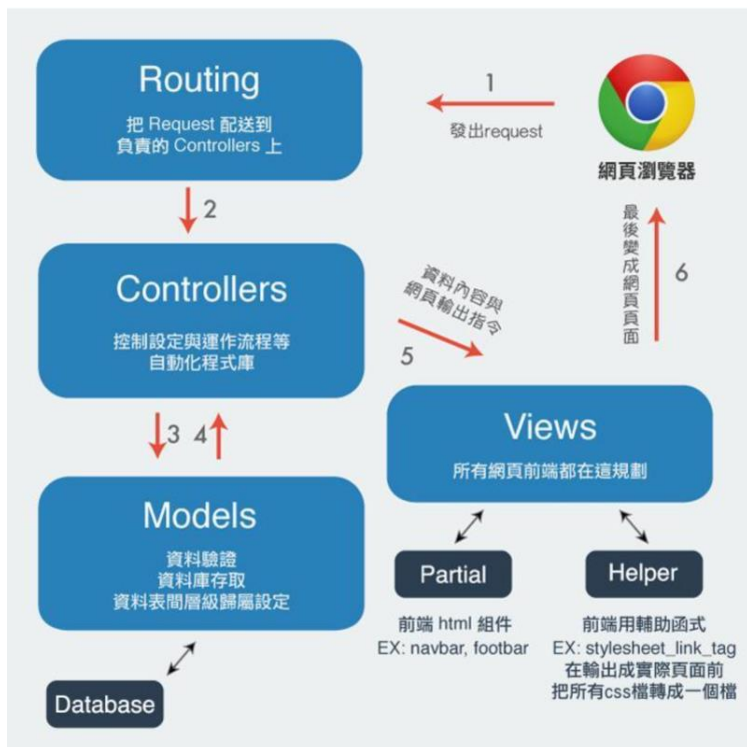
params[:group_id] 從哪來的？

	PATCH	/users(.:format)	devise/registrations#update
	PUT	/users(.:format)	devise/registrations#update
	DELETE	/users(.:format)	devise/registrations#destroy
group_posts	GET	/groups/:group_id/posts(.:format)	posts#index
	POST	/groups/:group_id/posts(.:format)	posts#create
new_group_post	GET	/groups/:group_id/posts/new(.:format)	posts#new
edit_group_post	GET	/groups/:group_id/posts/:id/edit(.:format)	posts#edit
group_post	GET	/groups/:group_id/posts/:id(.:format)	posts#show
	PATCH	/groups/:group_id/posts/:id(.:format)	posts#update
	PUT	/groups/:group_id/posts/:id(.:format)	posts#update
	DELETE	/groups/:group_id/posts/:id(.:format)	posts#destroy
groups	GET	/groups(.:format)	groups#index
	POST	/groups(.:format)	groups#create
new_group	GET	/groups/new(.:format)	groups#new
edit_group	GET	/groups/:id/edit(.:format)	groups#edit

Helpers (以Routes: group為例)	Controllers (輸出後的網址)			
HTTP Verb (HTTP的request動詞)	GET 讀取 (Read)	POST 新增 (Create)	PUT 更新 (update)	DELETE 刪除 (Destroy)
groups_path	index action (http://your_project_name.com/groups)	create action		
group_path(group)	show action (http://your_project_name.com/groups/[id])	update action	destroy action	
edit_group_path(group)	edit action (http://your_project_name.com/groups/[id]/edit)			
new_group_path	new action (http://your_project_name.com/groups/new)			

以上輔助函式(helper)
放在Views裡就能
轉成正確的連結網址

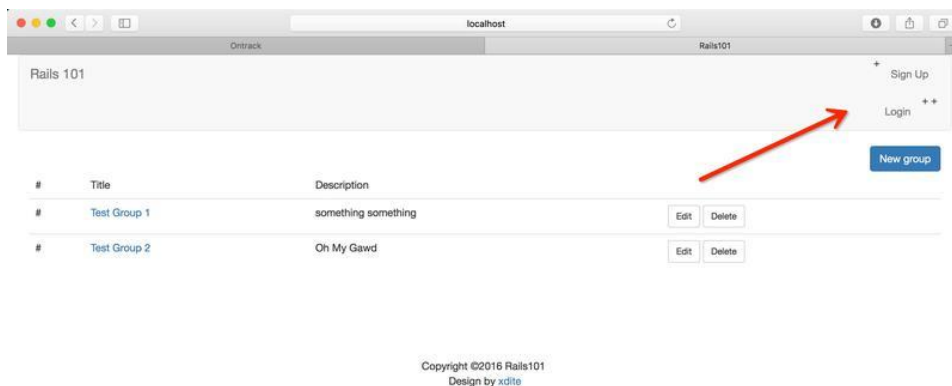
所有GET的action都有對應
各自專屬的view



神秘的“加号”

July 23, 2016

遇到了一個神秘的 Bug，它是如此的神秘，以至於一整天都沒解決，令人非常抓狂。這個 Bug 就是網頁上隨機出現的符號 +：



我仔細檢查，沒有發現我輸入的字符與自己 Github 上的代碼有任何區別，但是複製 Github 上 /app/views/common/_navbar.html.erb 的代碼過來運行 Bug 就消失，一複製教程上的代碼就出現 Bug 符號 + ...

就在我接近崩潰的時候，我突然發現問題所在：

Step 1: 修改 app/views/common/_navbar.html.erb



PS :

每次纠正自己这边的 Bug 时, 都觉得自己是天下第一白痴.....

July 21, 2016

跌跌撞撞地完成了第一遍, 感觉仍然好多坑。

不过我是独自一人完成的, 全程所有令我绝望的问题 (事后发现其实也没那么难) 都是通过自己的观察和推理解决的, 甚至没有到 slack 上询问和搜索。

但是这也许付出了惨重的代价:

- 1、缺少和队友的交流;
- 2、错过了帮助队友的机会;
- 3、第一遍耗时过久;

关于最后一点, 我希望能追上大家不掉队, 哈哈。

[一切看似正常之后.....](#)

July 21, 2016

在本地测试一切都没问题后, 上传到了 Heroku, 谁知报错了:

We're sorry, but something went wrong.

If you are the application owner check the logs for more information.

马上使用 Heroku logs 查看, 显示 can't write unknown attribute, 不知是什么原因:

```
rails101 -- -bash-- 120x48
f88fac74] Rendering groups/new.html.erb within layouts/application
2016-07-21T14:26:49.401388+00:00 app[web.1]: I, [2016-07-21T14:26:49.401325 #3] INFO -- : [762d126e-0868-4b61-a843-f393
f88fac74] Processing by GroupsController#new as HTML
2016-07-21T14:26:49.444637+00:00 app[web.1]: I, [2016-07-21T14:26:49.444516 #3] INFO -- : [762d126e-0868-4b61-a843-f393
f88fac74] Rendered groups/_form.html.erb (16.1ms)
2016-07-21T14:26:49.444778+00:00 app[web.1]: I, [2016-07-21T14:26:49.444711 #3] INFO -- : [762d126e-0868-4b61-a843-f393
f88fac74] Rendered groups/new.html.erb within layouts/application (16.6ms)
2016-07-21T14:26:49.413962+00:00 app[web.1]: D, [2016-07-21T14:26:49.413847 #3] DEBUG -- : [762d126e-0868-4b61-a843-f393
f88fac74] User Load (11.1ms) SELECT "users".* FROM "users" WHERE "users"."id" = $1 ORDER BY "users"."id" ASC LIMIT $
2 [{"id", 1}], [{"LIMIT", 1}]
2016-07-21T14:26:49.400233+00:00 app[web.1]: I, [2016-07-21T14:26:49.400145 #3] INFO -- : [762d126e-0868-4b61-a843-f393
f88fac74] Started GET "/groups/new" for 119.9.104.162 at 2016-07-21 14:26:49 +0000
2016-07-21T14:26:49.446300+00:00 app[web.1]: I, [2016-07-21T14:26:49.446201 #3] INFO -- : [762d126e-0868-4b61-a843-f393
f88fac74] Rendered common/_navbar.html.erb (0.4ms)
2016-07-21T14:26:49.446937+00:00 app[web.1]: I, [2016-07-21T14:26:49.446872 #3] INFO -- : [762d126e-0868-4b61-a843-f393
f88fac74] Rendered common/_footer.html.erb (0.1ms)
2016-07-21T14:26:49.447512+00:00 app[web.1]: I, [2016-07-21T14:26:49.447423 #3] INFO -- : [762d126e-0868-4b61-a843-f393
f88fac74] Completed 200 OK in 46ms (Views: 32.1ms | ActiveRecord: 11.1ms)
2016-07-21T14:26:49.446637+00:00 app[web.1]: I, [2016-07-21T14:26:49.446550 #3] INFO -- : [762d126e-0868-4b61-a843-f393
f88fac74] Rendered common/_flashes.html.erb (0.1ms)
2016-07-21T14:26:49.449495+00:00 heroku[router]: at=info method=GET path="/groups/new" host=evening-scrubland-17501.hero
kuapp.com request_id=762d126e-0868-4b61-a843-f393f88fac74 fwd="119.9.104.162" dyno=web.1 connect=0ms service=53ms status
=200 bytes=4314
2016-07-21T14:27:03.883218+00:00 app[web.1]: I, [2016-07-21T14:27:03.883144 #3] INFO -- : [28b7358d-03e4-4fd5-a7d3-1c8e
f5e35dcd] Parameters: {"utf8"=>"✓", "authenticity_token"=>"uLo5K843XPjZyFfCuyQ3GXq9h5Cho8xgpJaVPG4LyFicB6lwsP1/ld4ALG
lb7hcbTmngYxOyFHWrg6D+YrNLA==", "group"=>{"title"=>"First Rails trip", "description"=>"Twentynight's first Rails trip"},
"commit"=>"Submit"}
2016-07-21T14:27:03.898739+00:00 app[web.1]: F, [2016-07-21T14:27:03.898670 #3] FATAL -- : [28b7358d-03e4-4fd5-a7d3-1c8e
f5e35dcd]
2016-07-21T14:27:03.889191+00:00 app[web.1]: D, [2016-07-21T14:27:03.889112 #3] DEBUG -- : [28b7358d-03e4-4fd5-a7d3-1c8e
f5e35dcd] User Load (1.6ms) SELECT "users".* FROM "users" WHERE "users"."id" = $1 ORDER BY "users"."id" ASC LIMIT $2
[{"id", 1}], [{"LIMIT", 1}]
2016-07-21T14:27:03.897697+00:00 app[web.1]: I, [2016-07-21T14:27:03.897296 #3] INFO -- : [28b7358d-03e4-4fd5-a7d3-1c8e
f5e35dcd] Completed 500 Internal Server Error in 14ms (ActiveRecord: 1.6ms)
2016-07-21T14:27:03.898905+00:00 app[web.1]: F, [2016-07-21T14:27:03.898821 #3] FATAL -- : [28b7358d-03e4-4fd5-a7d3-1c8e
f5e35dcd]
2016-07-21T14:27:03.898815+00:00 app[web.1]: F, [2016-07-21T14:27:03.898747 #3] FATAL -- : [28b7358d-03e4-4fd5-a7d3-1c8e
f5e35dcd] ActiveRecord::MissingAttributeError (can't write unknown attribute "user_id"):
2016-07-21T14:27:03.879593+00:00 app[web.1]: I, [2016-07-21T14:27:03.879509 #3] INFO -- : [28b7358d-03e4-4fd5-a7d3-1c8e
f5e35dcd] Started POST "/groups" for 119.9.104.162 at 2016-07-21 14:27:03 +0000
2016-07-21T14:27:03.883119+00:00 app[web.1]: I, [2016-07-21T14:27:03.883053 #3] INFO -- : [28b7358d-03e4-4fd5-a7d3-1c8e
f5e35dcd] Processing by GroupsController#create as HTML
2016-07-21T14:27:03.898939+00:00 app[web.1]: F, [2016-07-21T14:27:03.898882 #3] FATAL -- : [28b7358d-03e4-4fd5-a7d3-1c8e
f5e35dcd] app/controllers/groups_controller.rb:16:in `create'
2016-07-21T14:27:03.900266+00:00 heroku[router]: at=info method=POST path="/groups" host=evening-scrubland-17501.herkua
pp.com request_id=28b7358d-03e4-4fd5-a7d3-1c8ef5e35dcd fwd="119.9.104.162" dyno=web.1 connect=0ms service=24ms status=50
0 bytes=1669
Twentynight: rails101 $
```

进一步调查显示, 我在执行 rails g migration add_user_id_to_group 之后, 编辑其生成的文档时, 把 integer

错输入成了 interger :

```
db/migrate/20160720142303_add_user_id_to_group.rb
class AddUserIdToGroup < ActiveRecord::Migration[5.0]
```

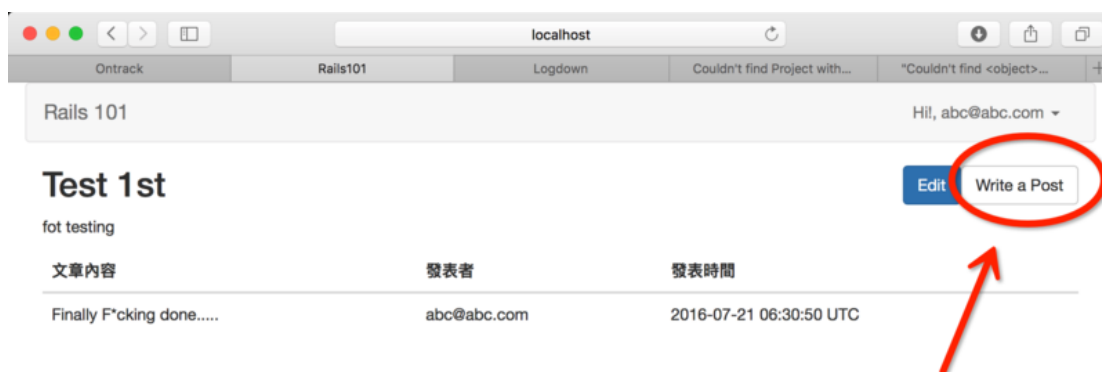
```
  def change
    add_column :groups, :user_id, :integer
  end
end
```

复制与粘贴

July 21, 2016

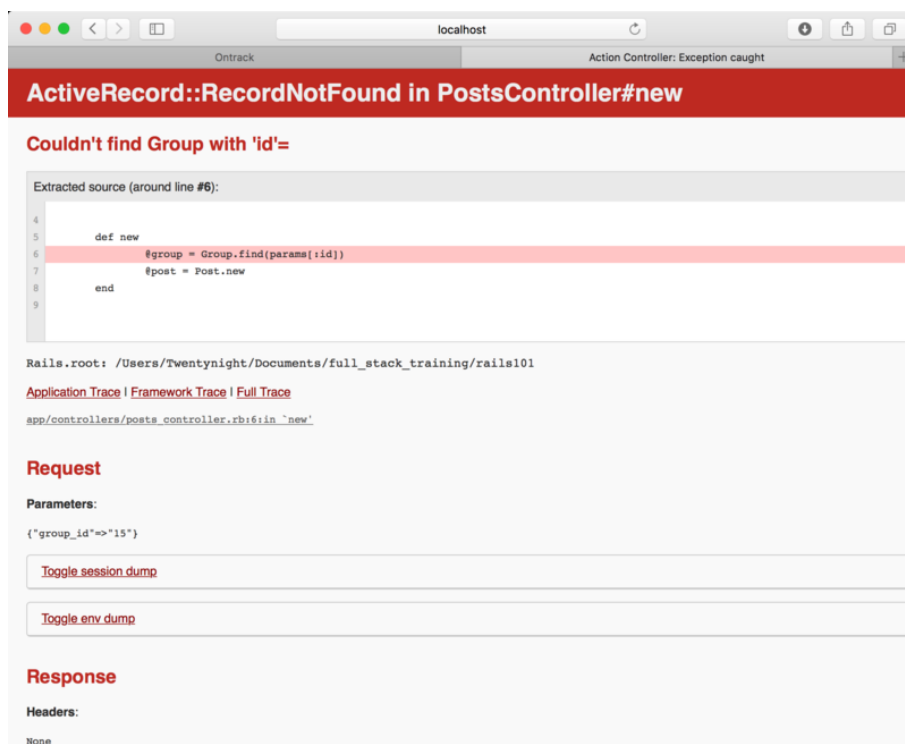
由于希望上来就多练习，严格要求自己，没有复制粘贴老师的代码，而是手打。导致有些 typo 不可避免地出现，并且导致 bug 之后还很难被发现。

下面是今天的一个例子：



Copyright ©2016 Rails101
Design by [xdite](#)

我做到第五章第二节的这步，点击 Write a Post 之后，出现了以下报错：



我查看 Terminal 后发现 `id = nil`，也就是参数没有传递过去：

```
Started GET "/groups/16/posts/new" for ::1 at 2016-07-21 14:28:11 +0800
DEPRECATION WARNING: before_filter is deprecated and will be removed in Rails 5.1. Use before_action instead. (called from <class:PostsController> at /Users/Twentyntight/Documents/full_stack_training/rails101/app/controllers/posts_controller.rb:3)
Processing by PostsController#new as HTML
Parameters: {"group_id"=>"16"}
  User Load (0.2ms) SELECT "users".* FROM "users" WHERE "users"."id" = ? ORDER BY "users"."id" ASC LIMIT ? [{"id", 1}, {"LIMIT", 1}]
  Group Load (0.1ms) SELECT "groups".* FROM "groups" WHERE "groups"."id" = ? LIMIT ? [{"id", nil}, {"LIMIT", 1}]
Completed 404 Not Found in 3ms (ActiveRecord: 0.3ms)
```

我百思不得其解，又查看了自己在 `app/views/groups/show.html.erb` 文件里的相关代码：

```
<%= link_to("Write a Post", new_group_post_path(@group), class: "btn btn-default pull-right") %>
```

内心各种草泥马：“没有错啊，就是 `new_group_post_path(@group)` 啊！他妈的，难道是老师的教材错了？”在网上搜索各种信息无果，并卡在这一整天之后，我最终绝望地去查看那些运行正常的网页，一步一步点击，然后在终端里观察它们的参数传递行为。

突然间，我看到了某些规律：

The terminal screenshot shows three HTTP requests. The first request is a GET to `/groups/16` with parameters `{"id"=>"16"}`. The second request is a GET to `/groups/16/posts/new` with parameters `{"group_id"=>"16"}`. The third request is a GET to `/groups/16/posts/new` with parameters `{"group_id"=>"16"}`. Red arrows point from the `id` parameter in the first request to the `group_id` parameter in the second and third requests. The second and third requests show a `404 Not Found` error, indicating that the `group_id` parameter is not being passed correctly.

我发现每次发生参数传递时，似乎都有个 `Parameters: {something => somevalue}` 出现，图中是 `Parameters: {"id"=>"16"}` 和 `Parameters: {"group_id"=>"16"}`。其中 `Parameters: {"group_id"=>"16"}` 是我那个报错网页的参数传递过程的信息。

于是我尝试把 `app/controllers/posts_controller.rb` 里的 `:id` 改成 `:group_id`。

`app/controllers/posts_controller.rb`

```
def new
```

```
-       @group = Group.find(params[:id])
+       @group = Group.find(params[:group_id])
  @post = Post.new
```

end

‘奇迹’终于出现了，一切变得正常起来。

我内心再次骂娘，并伴随着傲视天下的优越感：“他妈的教程上真是乱写，这种错误新手根本发现不了，估计也就只有我这种智商的推理才能发现问题。垃圾 xdite。”

于是我再回去看教程，想在万无一失的情况下继续鄙视 xdite。哪知，教程上原本写的就是 @group = Group.find(params[:group_id])，而我因为手打，并且之前都是 xxx.find(params[:id])，我想当然地就以为 params[] 的中括号里都是填的 :id。

我异常惭愧地又翻出 xdite 老师 [放下你的無效學習方式](#) 一文。

「傲慢」是失敗的根源

...

所謂『傲慢』是指：

...

我是否懷疑老師能力，不照教材實作，而是繞彎路去看一堆亂七八糟的補充教材，把自己的腦袋搞得更混亂

...

— xdite

总结：

- 1、此后要小心自己愚蠢的优越感了，不要像傻逼一样乱骄傲。
- 2、以后不懂的 Bug，首先通过直接复制粘贴老师的代码来寻找。因为类似今天这个 Bug，是怎么检查都检查不出的。它属于第一印象 Bug 或者说 先入为主 Bug。和李笑来老师[看不见自己的错字](#)是类似的原理。之前的文章 [关于第一次](#) 已经描述了这种现象。
- 3、再次体会到程序的精确。

[变量名少打一个“s”](#)

July 20, 2016

练习到了第三章的第一节【實作討論群「新增」功能】，在完成【Step 1. 建立 new action】和【Step 2. 建立 new action 的表單】之后，打开 <http://localhost:3000/groups/new>，发现网站提示错误 First argument in form cannot contain nil or be empty。

经过仔细检查之后，发现 app/controllers/groups_controller.rb 中 @group 这个变量被我错输入成 @groups。就这么个小错误就卡了我半天，真是不得不感慨计算机的精确，容不得马虎。听说结构化编辑器可以避免这些 typo，也不知道是不是真的。将来成长之后再了解吧。

```
class GroupsController < ApplicationController
```

```
  def index
```

```
    @groups = Group.all
```

```
  end
```

```
+ def new
```

```
+   @group = Group.new
```

```
+ end
```

```
end
```

```
app/views/groups/new.html.erb
```

```
<div class="col-md-4 col-md-offset-4">
```

```
  <h2>新增討論版</h2>
```

```
  <hr>
```

```
  <%= form_for @group do |f| %>
```

標題

```
<%= f.text_field :title %>
```

```
<br>
```

敘述

```
<br>
```

```
<%= f.text_area :description %>
```

```
<br>
```

```
<%= f.submit "Submit", :disable_with => 'Submitting...' %>
```

```
<% end %>
```

```
</div>
```

重新捣鼓 first_app 发生错误

July 19, 2016

从 Github 上 clone first_app 到 Mac Pro 上运行，发生了如下错误：

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

按照提示运行 bin/rails db:migrate RAILS_ENV=development 然后问题解决。

重走了一遍作业，感觉 bin/rails db:migrate RAILS_ENV=development 和 rake db:migrate 似乎是一样的功能。

关于第一次

July 19, 2016

第壹次很重要。

在 Safari 瀏覽器裏輸入壹個網址（比如：<http://www.daydays.com/products-tiantianluke/>）

但是卻輸入錯誤了——【daydays】寫成了【daysdays】——則瀏覽器將會記住這個錯誤，在妳沒發現並糾正之前壹直幫妳用妳第壹次輸入的錯誤自動補完（autofill），使得妳每壹次都無法跳轉到正確的頁面。

練習鋼琴時，如果不從慢節奏開始，快板彈到壹般通常會因為指法不熟練而卡在某個音，接下來每次彈到這壹小節大腦都會發出警告信號，提醒自己接下來這個音很難彈，然後不出意外卡在這個音上，或在這個音符上發生停頓。然後每次練習，這個停頓都會發生，以至於好像是在專門練習這個錯誤的停頓壹樣。最後就再也改正不了了，該停頓已被肌肉永久記憶。

學習和練習也是壹樣的，第壹次做壹定要以非常正確的方式進行，否則之後難以糾正，因為大腦會在之後的每次練習和熟悉的過程中，不自覺地使用第壹次的錯誤來“自動補完”，然後錯誤被多次重複，進而加強，最後不僅難以糾正，並且代價很大。

...第壹次很重要。

Miscellaneous references about full stack training

July 19, 2016

[Fish tutorial](#)

[Terminal/iTerm2 的快捷鍵](#)

[各种功能的 gem](#)

[What to do if libxml2 is being a jerk ...](#)

July 19, 2016

在第二台 Mac 上重頭走了一遍第一次作業。到了安裝 rails 這步，出現了和其他學員類似的問題，gem install rails -v 5.0.0 安裝顯示出錯。

经过慢慢查看 rails 的安装信息，发现当安装到一个叫做 nokogiri 的东西时，问题开始出现。nokogiri 表示需要 libxml2 这个东西才能顺利解析和安装。

于是我参考了 Github 上的一篇名为 [What to do if libxml2 is being a jerk](#) 的说明，找到了解决方法：

一、用 brew 安装 libxml2 和 libxslt；

二、安装 nokogiri 这个东西时，手动指定 libxml2 和 xslt 这两个库来帮助解析（parsing）。以我的情况为例：

```
gem install nokogiri --  
--use-system-libraries  
--with-xml2-lib=/usr/local/opt/libxml2/lib  
--with-xml2-include=/usr/local/opt/libxml2/include/libxml2  
--with-xslt-lib=/usr/local/opt/libxslt/lib  
--with-xslt-include=/usr/local/opt/libxslt/include
```