Table Of Contents

2009年度运动总结， 2010年度运动计划

2009运动总结

4月19日, 海南70.3铁人 118名 07小时30分4秒的成绩 对我来说不算理想，但这个经历绝对是我一生中最难忘也是最重要的一段。无论是赛前拼命准备，赛中的绝望和想放弃，到后来的坚持与冲刺，以及赛后被送进医院，还有最后的海鲜大餐。现在仍难以想象在短短的时间里发生过这么多事情。真心的感谢妈妈爸爸，虽然并不支持我参加比赛，但是他们的关心是我永远的精神支柱；感谢 adrian，还有他的妈妈和弟弟，不是他们我一定会挂在海南；感谢我的朋友们，他们的从开始的不信任到后来的鼓励都是我坚持到底的动力。
具体细节我不可能比 adrian 描述 的更清楚。

赛前准备

赛后兴奋

7月4日，2009年北京首届永定河穿越赛 100公里，全军覆没在半程。任何事情，如果没有足够的了解和准备，都是十分危险的。这个道理应该在4月份的第一次铁人赛就体会到，但直到这时才深深的理解。如果还会举办，2010年我还会继续。

难忘的回忆

7月26日的北京铁三洲际杯赛 建立自信，享受比赛，绝对是一段愉快的经历。 3小时07分， 232的排名， 明年还会继续。

拐弯

在终点

10月18日北京马拉松 早早就报了名，可是由于前一天受伤，最后决定了放弃...

2010运动计划

4月底
北京 —— 秦皇岛 自行车拉练
这个会很有趣，5个好朋友已经开始准备。

5月8日
The North Face 100®2010北京国际户外耐力跑挑战赛（50公里）
去年没有参加是个遗憾，过完年开始准备。

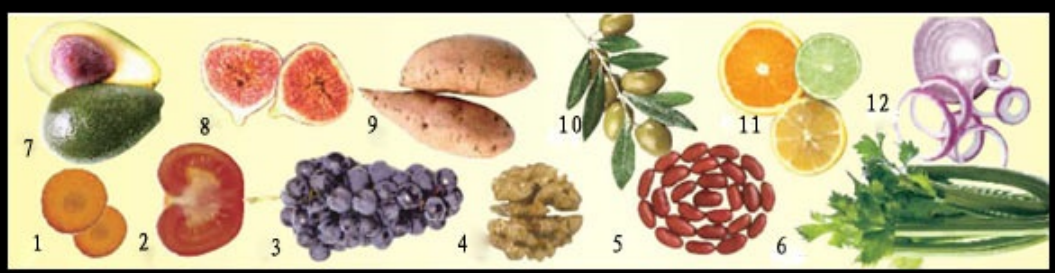
7月3日至4日
国际铁人三项洲际杯赛暨全国铁人三项冠军杯系列赛
继续铁人之旅。

9月5日
北戴河铁人三项赛
想第一次在家乡做铁人。

10月17日
北京国际马拉松赛

神奇的吃啥补啥(分享)

分享一篇来自《生命时报》的文章——“中医养生：神奇的吃啥补啥”。



1. 切开的胡萝卜就像人的眼睛，有瞳孔、虹膜，以及放射的线条。科学研究表明，大量胡萝卜素能促进人体

血液流向眼部，保护视力，让眼睛更明亮。

2. 番茄有四个腔室，并且是红色的，这与我们的心脏一样。实验证实，番茄饱含番茄红素，高胆固醇患者要想降低心脏病和中风危险，不妨多吃点。

3. 悬挂的一串葡萄具有心脏的形状，而每一颗葡萄就像红血球。葡萄汁中含有丰富的多元酚类，能帮助身体对抗心血管疾病。

4. 核桃就像一个微型的脑子，有左半脑、右半脑、上部大脑和下部小脑，甚至其褶皱或折叠都像大脑皮层。目前人类已经知道，核桃含有36种以上的神经传递素，可以帮助开发脑功能。

5. 蚕豆等豆类的形状看起来很像人的肾脏，它们也的确可以帮助维持肾脏功能。

6. 芹菜等很多根茎类蔬菜看起来就像人的骨头，而它们确实能强化骨质。人骨头中含有23%的钠，而这些食物也含有23%的钠。

7. 鳄梨长得很像子宫，能够保护女性的子宫和子宫颈健康。研究表明，女性每星期吃一个鳄梨，就能平衡雌激素、减掉分娩产生的多余体重，防止宫颈癌。奇妙的是，鳄梨从开花到成熟结果的生长期，也恰恰是9个月。

8. 无花果就像男人的睾丸一样，无花果充满了籽，而且它们生长时也是成对的。研究表明，无花果可增强男性精子活力，增加精子数量，并治疗男子不育症。

9. 甘薯看起来像胰腺，事实上，它确实能平衡糖尿病患者的血糖指数。

10. 卵巢仅有橄榄大小，但却是肿瘤最易发生的器官，并且肿瘤的种类也最多，有30多种。多吃橄榄有助于卵巢健康，预防各种卵巢肿瘤。

11. 柑橘类水果长得像乳腺，橘子的抗氧化剂含量是所有水果中最高的，含170多种不同的植物化学成分。食用时橘络不要扔掉，可缓解乳腺增生症状。

12. 洋葱的纹路看上去像人体细胞。研究表明，它能清除身体所有细胞里的垃圾物质和危害性的游离基。

rails 的 cloud_tag plugin

在 andy 处发现很酷的 tag cloud 动画，很喜欢，于是封装了这个 plugin ， 并加入到 博客 中。

flash 拷贝自 http://www.tag-cloud.de/wpcloud/?r=tag_cloud

使用:
1.放置 cloud_tag 到项目中 *vendor/plugins* 文件夹下
2.在 view 中使用:

```
<%= cloud_tag(tag_link_hash, height = "240", width="200") %>
```

height 和 **width** 是可选的。
tag_link_hash 的格式为:

```
{"tag1" => "www.tag1.com", "tag2" => "www.tag2.com"}
```

例子:
http://zhangzhe.heroku.com/example/cloud_tag.html

Github地址:
http://github.com/zhangzhe/cloud_tag

cloud_tag plugin for rails

Found this cool flash tag cloud from andy's blog. Readly like it, so create this plugin , and add to my blog.

flash copied from http://www.tag-cloud.de/wpcloud/?r=tag_cloud

use:
1.puts cloud_tag folder into project *vendor/plugins* folder
2.use in view :

```
<%= cloud_tag(tag_link_hash, height = "240", width="200") %>
```

height and **width** are optional.
tag_link_hash should be formatted like:

```
{"tag1" => "www.tag1.com", "tag2" => "www.tag2.com"}
```

example:
http://zhangzhe.heroku.com/example/cloud_tag.html

Github address:
http://github.com/zhangzhe/cloud_tag

评估 rails 代码质量

在 rails 中评估代码质量，有 Saikuro, Flog, Flay, Rcov, Reek, Roodi, Churn 等工具。metric-fu 把这些通通集成在一起。

安装

```
sudo gem install metric_fu
```

创建 metric_fu.rake 文件

```
begin
  require "metric_fu"
rescue LoadError
end
```

运行

```
rake metrics:all
```

flog评估结果：

Total Flog score for all methods: 438.7

Average Flog score for all methods: 6.7

File	Total score	Methods	Average score	Highest score
/app/controllers/blogs_controller.rb	86	9	10	41
/app/models/pinyin.rb	49	7	7	37
/app/helpers/application_helper.rb	70	6	12	31
/app/controllers/comments_controller.rb	23	1	23	23
/app/models/blog.rb	21	2	10	18
/app/models/coderay_string.rb	17	1	17	17
/app/models/formatable_string.rb	78	13	6	15
/app/controllers/upload_files_controller.rb	22	2	11	15
/app/controllers/blog_groups_controller.rb	13	2	7	8
/app/controllers/sessions_controller.rb	10	2	5	7
/app/controllers/application_controller.rb	22	6	4	4
/app/models/blog_group.rb	4	1	4	4
/app/controllers/tags_controller.rb	4	1	4	4
/app/models/upload_file.rb	4	1	4	4

另一个很棒的插件 rails best practices 从代码规范，编码设计更细节的方面提出建议。
安装

```
sudo gem install rails_best_practices --source http://gemcutter.org
```

运行

```
rails_best_practices .
```

我的博客程序获得的建议如下，

```
./app/views/blog_groups/_show.html.erb:8 - replace instance variable with lo
./app/views/blog_groups/_show.html.erb:8 - replace instance variable with lo
./app/views/blogs/_admin_tool.html.erb:2 - replace instance variable with lo
./app/views/blogs/_form.html.erb:3 - replace instance variable with local va
./app/views/blogs/_form.html.erb:3 - replace instance variable with local va
./app/views/blogs/_form.html.erb:13 - replace instance variable with local v
./app/views/blogs/_show.html.erb:2 - replace instance variable with local va
./app/views/blogs/_show.html.erb:4 - replace instance variable with local va
./app/views/blogs/_show.html.erb:6 - replace instance variable with local va
./app/views/comments/_show.html.erb:5 - replace instance variable with local
./app/controllers/blogs_controller.rb:26,33 - use before_filter for new,edit
./app/controllers/upload_files_controller.rb:2,12 - use before_filter for cr
./app/controllers/blogs_controller.rb:42 - move model logic into model (@blo
./db/migrate/20100129115908_acts_as_taggable_on_migration.rb:10 - always add
```

rails code evaluation

metric_fu

There are tools like Saikuro, Flog, Flay, Rcov, Reek, Roodi, Churn to evaluate rails code. metric-fu Mix them together.

Install

```
sudo gem install metric_fu
```

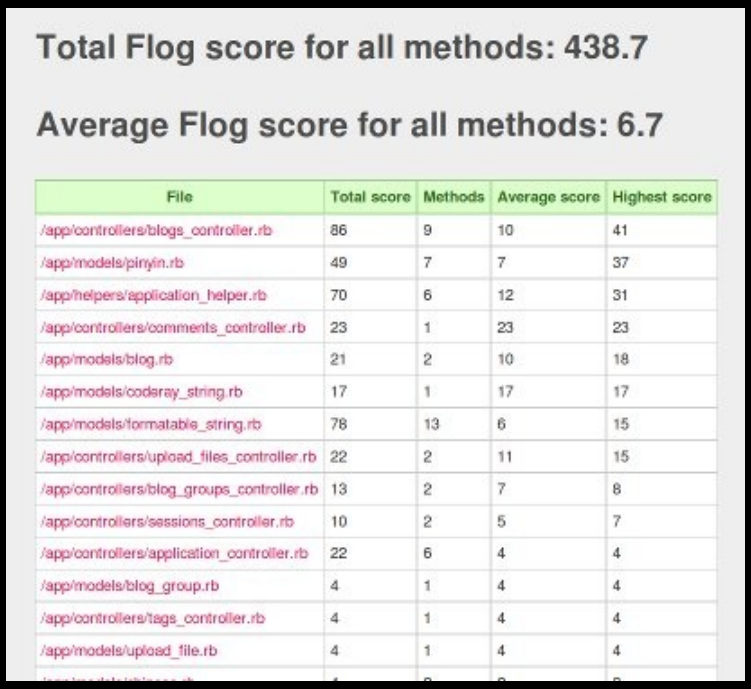
create metric_fu.rake file

```
begin
  require "metric_fu"
rescue LoadError
end
```

run

```
rake metrics:all
```

Result of flog



Total Flog score for all methods: 438.7

Average Flog score for all methods: 6.7

File	Total score	Methods	Average score	Highest score
/app/controllers/blogs_controller.rb	86	9	10	41
/app/models/pinyin.rb	49	7	7	37
/app/helpers/application_helper.rb	70	6	12	31
/app/controllers/comments_controller.rb	23	1	23	23
/app/models/blog.rb	21	2	10	18
/app/models/coderay_string.rb	17	1	17	17
/app/models/formatable_string.rb	78	13	6	15
/app/controllers/upload_files_controller.rb	22	2	11	15
/app/controllers/blog_groups_controller.rb	13	2	7	8
/app/controllers/sessions_controller.rb	10	2	5	7
/app/controllers/application_controller.rb	22	6	4	4
/app/models/blog_group.rb	4	1	4	4
/app/controllers/tags_controller.rb	4	1	4	4
/app/models/upload_file.rb	4	1	4	4

rails best practices

Another gem rails best practices gives more advices from code standards, and detail design.
install

```
sudo gem install rails_best_practices --source http://gemcutter.org
```

run

```
rails_best_practices .
```

The advice to my blog:

```
./app/views/blog_groups/_show.html.erb:8 - replace instance variable with lo
./app/views/blog_groups/_show.html.erb:8 - replace instance variable with lo
./app/views/blogs/_admin_tool.html.erb:2 - replace instance variable with lo
./app/views/blogs/_form.html.erb:3 - replace instance variable with local va
./app/views/blogs/_form.html.erb:3 - replace instance variable with local va
./app/views/blogs/_form.html.erb:13 - replace instance variable with local v
./app/views/blogs/_show.html.erb:2 - replace instance variable with local va
./app/views/blogs/_show.html.erb:4 - replace instance variable with local va
./app/views/blogs/_show.html.erb:6 - replace instance variable with local va
./app/views/comments/_show.html.erb:5 - replace instance variable with local
./app/controllers/blogs_controller.rb:26,33 - use before_filter for new,edit
./app/controllers/upload_files_controller.rb:2,12 - use before_filter for cr
./app/controllers/blogs_controller.rb:42 - move model logic into model (@blo
./db/migrate/20100129115908_acts_as_taggable_on_migration.rb:10 - always add
```

在 ubuntu 下使用 *test notifier* 辅助 autotest

Ubuntu 下面无法 使用 Growl 辅助 autotest 测试 ， test_notifier 提供了更好地解决方案。

安装 gem test_notifier

```
sudo gem i test_notifier
```

在项目根目录 *~/.autotest* 文件中加入

```
require "test_notifier/autotest"
```

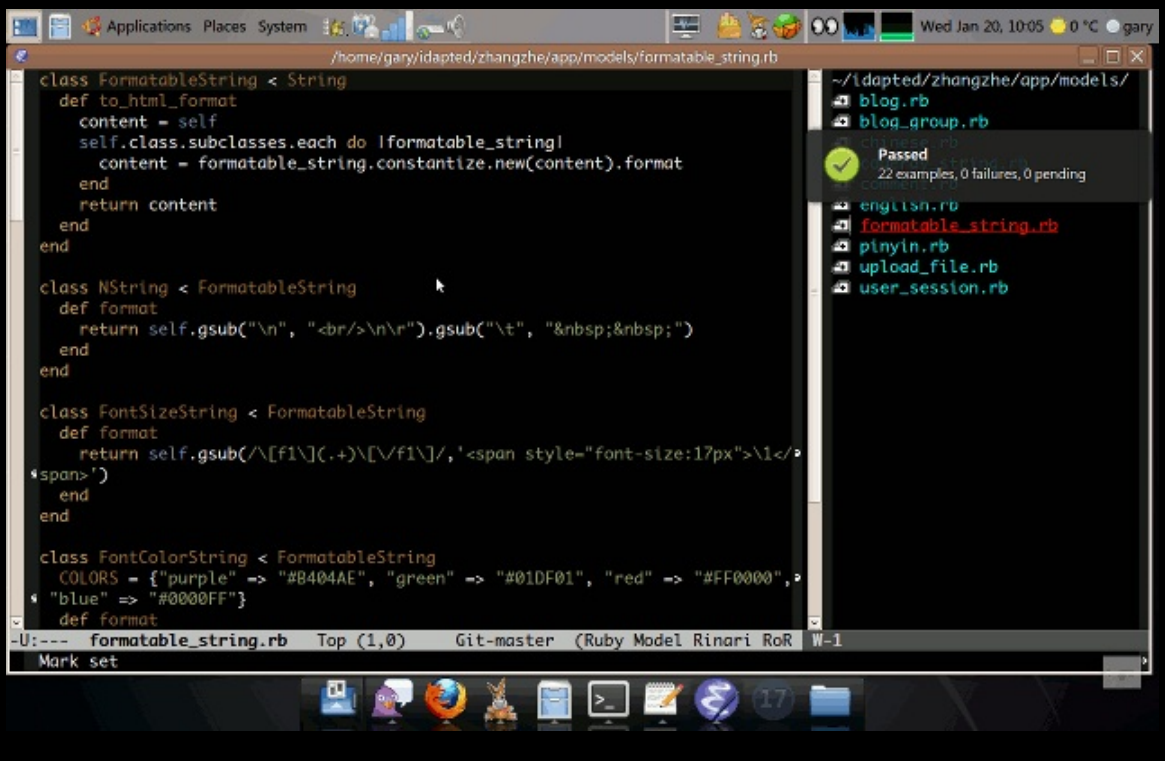
如果之前没有安装信息提示软件，现在会有安装提示。这里推荐 libnotify-bin

```
sudo apt-get install libnotify-bin
```

安装完毕运行

```
autospec
```

自动测试结果会显示在屏幕右上方：)



set up autotest to use test_notifier

It is a pity that Ubuntu can not set up autotest to use Growl.

There is a better solution: test_notifier

```
sudo gem i test_notifier
```

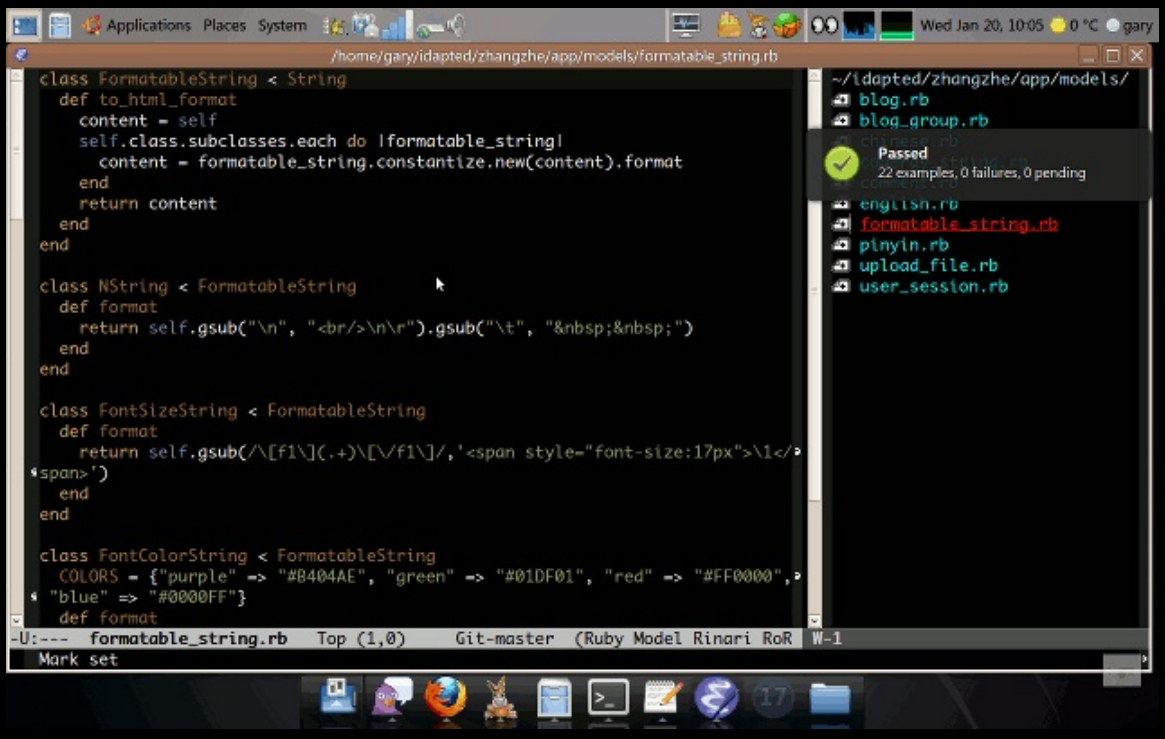
and add to the file *~/.autotest*

```
require "test_notifier/autotest"
```

If do not have notify thing installed, may need libnotify-bin

```
sudo apt-get install libnotify-bin
```

So run autospec, and the test result should be shown on right top of screen.



Nice, isn't it:)

公司图书出版

公司图书《31种雅思口语高分必背公式》正式出版。希望可以大卖：）

北京，2010年1月9日 — 外语教学与研究出版社（FLTRP）和EQ英语日前共同推出了一本具有革命性意义的雅思备考教材——《31种雅思口语高分必背公式》，该书适用于全国准备参加雅思考试的考生。

《31种雅思口语高分必背公式》一书中所提到的雅思备考学习方法具有革命性的创新和突破，并且正在受到包括中国著名的海外学习教师徐小平和杨继等在内的整个英语培训界的认可和好评。“我读了这本书后发现，该书中给出的高分公式涵盖了所有的雅思口语考类型，这能够有效地帮助学生学习如何运用英语答题逻辑来组织自己的观点。而且它还能大大提高学习的效率。如果你掌握了这31种高分公式，你的雅思口语分数在1个月内可以提高1分”，徐小平对记者说。

“这本书是英语口语培训的一大突破，”斯坦福大学英语语言培训主任、菲尔·赫巴德博士说，“学生通过该书将学会如何用西方人的思维方式回答问题。学生从书中获取的知识和帮助越多、成绩越优秀，也就越有可能被斯坦福这样的大学录取，并在学业上取得更大的成功。”

《31种雅思口语高分必背公式》能够指导学生如何回答所有雅思口语考试题型，无论遇到什么样的话题，都可以使用书中的31种高分公式来应对。“高分公式”包括以下几个方面：“答题思路”、教会你“说什么”，以及“高分语言点”、教会你“如何说”。

据悉，“31种高分公式”已经开始在EQ英语在线学习平台上进行应用，加上EQ英语1对1雅思外教的辅导，能够确保学生提高雅思分数。同时校方还表示，如果学习效果不理想还可全额退款。帕里补充道，该学习方法非常行之有效，能够保证学生在一个月内雅思分数至少提高1分。EQ英语也愿意开此先例，为学生提供口语培训的退款承诺。

本书的出版方外研社是中国最大的外语教育出版机构，这次携手雅思专业培训机构EQ英语的专家团队，可谓强强联手，力求打造出令广大读者满意的精品图书。在刚刚结束的新书发布会上，出版方表示对本书的市场前景充满信心。

《31种雅思口语高分必背公式》的作者是首席运营官和合伙创办人Jonathan Palley、首席执行官和合伙创办人李国栋，主编是EQ英语内容主任Oliver Davies。本书由外语教学与研究出版社出版，在全国各地书店有售。要想获得关于本书的更多信息，请登录www.EQEnglish.com/book。

外研社综合英语分社社长张黎新、本书作者之一李国栋、EQ英语内容主任Oliver Davies在发布会现场

《31种雅思口语高分必备公式》作者之一李国栋在发布会现场

关于EQ英语

EQ英语是一家基于互联网的实时1对1英语学习服务公司，所属公司为美国最大的在线1对1语言指导供应商—— 一德有限公司。

作者简介

Jonathan Palley，美国人，在斯坦福大学主修物理学和戏剧表演。他始终致力于教育与技术的双重领域，将互联网技术和培训完美结合，创建了世界领先的语音交流平台，使在线语言培训实现了革命化巨变。同时也是31种雅思口语高分公式的创始人之一。

Adrian Li，英国人，曾获剑桥大学经济学学士和斯坦福大学MBA硕士学位。曾供职于世界知名企业摩根大通（JPMorgan）管理层，并在语言学习领域多年不辍研究。2006年来到中国创建了EQ英语，联合多位语言学专家，独创EQ英语逻辑学习法，是31种雅思口语高分公式的创始人之一。

Oliver Davies，英国人，曾获威尔士大学宗教与哲学学士学位，拥有英国伦敦圣三一学院英语教学师资证书。从事英语教学工作长达10年，先后在多个国家的大学和国际学校里任教。多元文化下的工作经历以及对语言教学理念的创新，让他在EQ英语成功地把语言教学搬上了互联网。

Company book published

北京，2010年1月9日 — 外语教学与研究出版社（FLTRP）和EQ英语日前共同推出了一本具有革命性意义的雅思备考教材——《31种雅思口语高分必背公式》，该书适用于全国准备参加雅思考试的考生。

《31种雅思口语高分必背公式》一书中所提到的雅思备考学习方法具有革命性的创新和突破，并且正在受到包括中国著名的海外学习教师徐小平和杨继等在内的整个英语培训界的认可和好评。“我读了这本书后发现，该书中给出的高分公式涵盖了所有的雅思口语考类型，这能够有效地帮助学生学习如何运用英语答题逻辑来组织自己的观点。而且它还能大大提高学习的效率。如果你掌握了这31种高分公式，你的雅思口语分数在1个月内可以提高1分”，徐小平对记者说。

“这本书是英语口语培训的一大突破，” 斯坦福大学英语语言培训主任、菲尔·赫巴德博士说，“学生通过该书将学会如何用西方人的思维方式回答问题。学生从书中获取的知识和帮助越多、成绩越优秀，也就越有可能被斯坦福这样的大学录取，并在学业上取得更大的成功。”

《31种雅思口语高分必背公式》能够指导学生如何回答所有雅思口语考试题型，无论遇到什么样的话题，都可以使用书中的31种高分公式来应对。“高分公式”包括以下几个方面：“答题思路”、教会你“说什么”，以及“高分语言点”、教会你“如何说”。

据悉，“31种高分公式”已经开始在EQ英语在线学习平台上进行应用，加上EQ英语1对1雅思外教的辅导，能够确保学生提高雅思分数。同时校方还表示，如果学习效果不理想还可全额退款。帕里补充道，该学习方法非常行之有效，能够保证学生在一个月内雅思分数至少提高1分。EQ英语也愿意开此先例，为学生提供口语培训的退款承诺。

本书的出版方外研社是中国最大的外语教育出版机构，这次携手雅思专业培训机构EQ英语的专家团队，可谓强强联手，力求打造出令广大读者满意的精品图书。在刚刚结束的新书发布会上，出版方表示对本书的市场前景充满信心。

《31种雅思口语高分必背公式》的作者是首席运营官和合伙创办人Jonathan Palley、首席执行官和合伙创办人李国栋，主编是EQ英语内容主任Oliver Davies。本书由外语教学与研究出版社出版，在全国各地书店有售。要想获得关于本书的更多信息，请登录www.EQEnglish.com/book。

外研社综合英语分社社长张黎新、本书作者之一李国栋、EQ英语内容主任Oliver Davies在发布会现场

《31种雅思口语高分必备公式》作者之一李国栋在发布会现场

关于EQ英语

EQ英语是一家基于互联网的实时1对1英语学习服务公司，所属公司为美国最大的在线1对1语言指导供应商—— 一德有限公司。

《31种雅思口语高分必背公式》

作者简介

Jonathan Palley，美国人，在斯坦福大学主修物理学和戏剧表演。他始终致力于教育与技术的双重领域，将互联网技术和培训完美结合，创建了世界领先的语音交流平台，使在线语言培训实现了革命化巨变。同时也是31种雅思口语高分公式的创始人之一。

Adrian Li，英国人，曾获剑桥大学经济学学士和斯坦福大学MBA硕士学位。曾供职于世界知名企业摩根大通（JPMorgan）管理层，并在语言学习领域多年不辍研究。2006年来到中国创建了EQ英语，联合多位语言学专家，独创EQ英语逻辑学习法，是31种雅思口语高分公式的创始人之一。

Oliver Davies，英国人，曾获威尔士大学宗教与哲学学士学位，拥有英国伦敦圣三一学院英语教学师资证书。从事英语教学工作长达10年，先后在多个国家的大学和国际学校里任教。多元文化下的工作经历以及对语言教学理念的创新，让他在EQ英语成功地把语言教学搬上了互联网。

我的 rails 测试配置和心得

使用 Rspec 已有一年多，越来越感觉得心应手。
下面介绍我的测试环境配置和一些测试心得，希望有所帮助:)

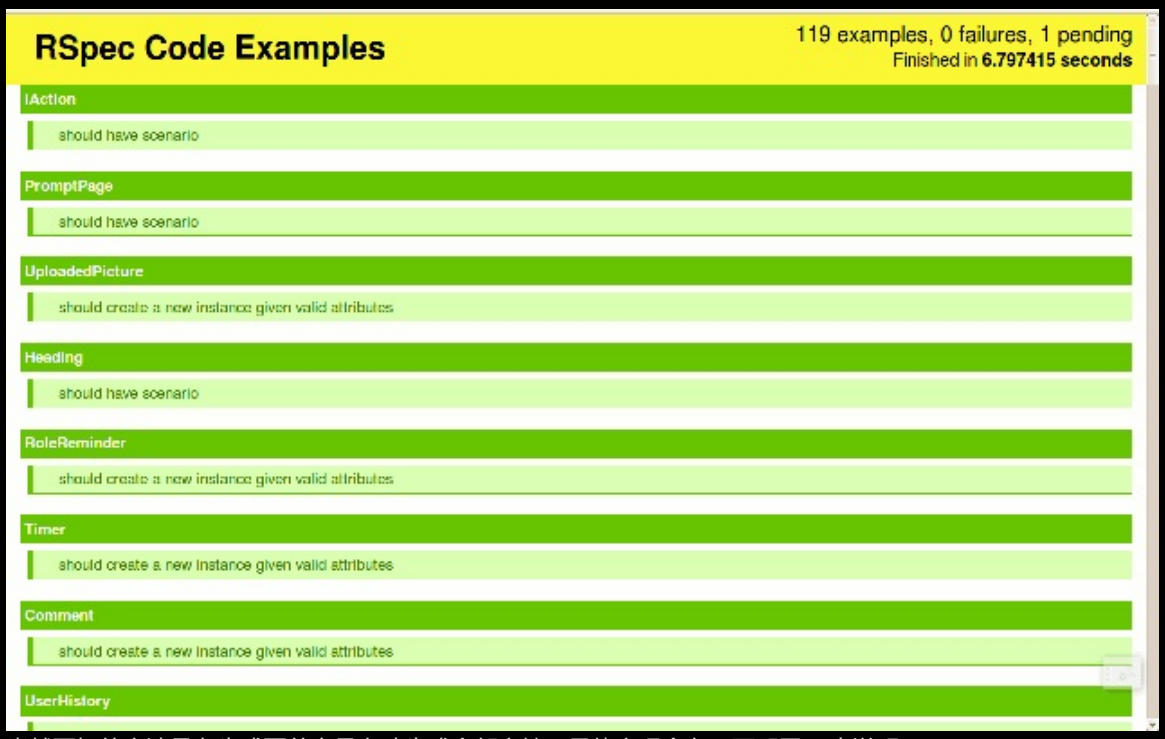
1.自动生成测试文档

通过附加参数 --format specdoc 每次测试时在终端输出测试文档

```
spec spec/models/blog_spec.rb --format specdoc
```

也可以使用 rake spec:doc 在终端生成全部测试代码的文档
我更喜欢的方式是利用 --format html 将测试文档输出到指定 html 文件

```
spec spec/models/blog_spec.rb --format html:test.html
```



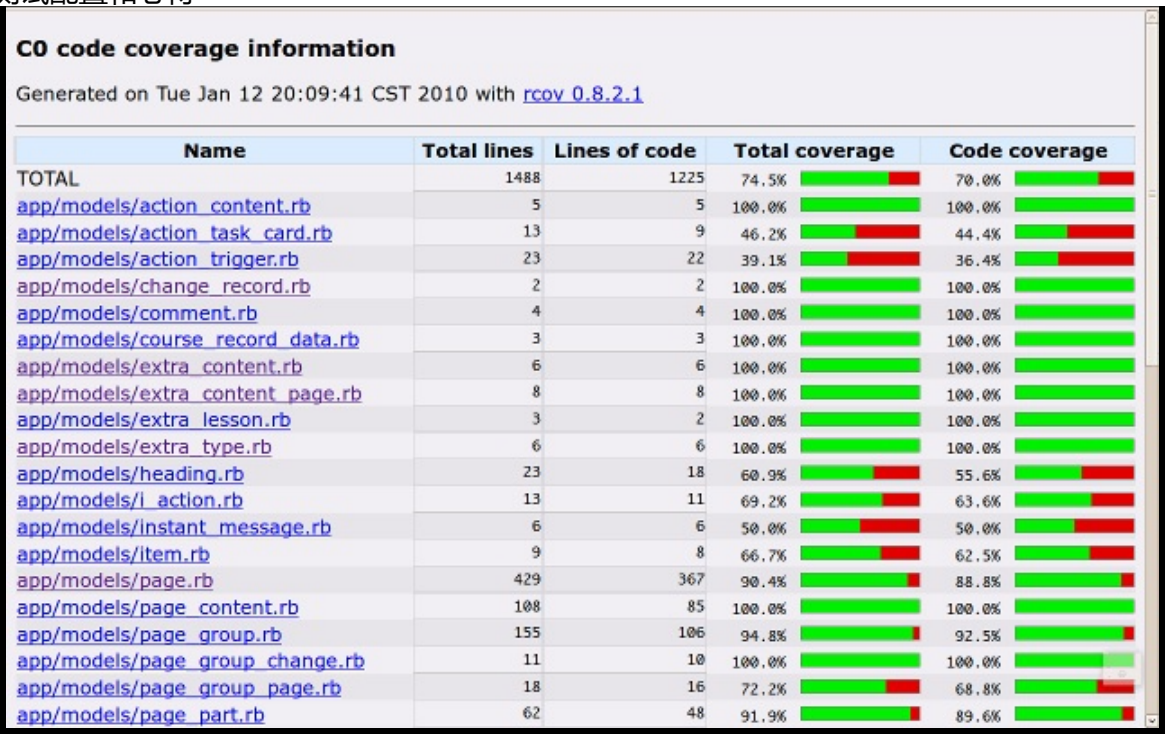
当然更好的方法是在生成覆盖率是自动生成全部文档，具体实现会在下面配置 4 中说明。

我认为如果代码覆盖率足够高并保证代码和测试组织合理的话，这个测试文档完全可以做为代码级别文档使用。

2.利用 rcov 生成代码覆盖率

```
rake spec:rcov
```

会生成 html 格式的代码覆盖率文档，非常方便查看。



C0 code coverage information

Generated on Tue Jan 12 20:09:41 CST 2010 with rcov 0.8.2.1

Name	Total lines	Lines of code	Total coverage	Code coverage
TOTAL	1488	1225	74.5%	70.0%
app/models/action_content.rb	5	5	100.0%	100.0%
app/models/action_task_card.rb	13	9	46.2%	44.4%
app/models/action_trigger.rb	23	22	39.1%	36.4%
app/models/change_record.rb	2	2	100.0%	100.0%
app/models/comment.rb	4	4	100.0%	100.0%
app/models/course_record_data.rb	3	3	100.0%	100.0%
app/models/extra_content.rb	6	6	100.0%	100.0%
app/models/extra_content_page.rb	8	8	100.0%	100.0%
app/models/extra_lesson.rb	3	2	100.0%	100.0%
app/models/extra_type.rb	6	6	100.0%	100.0%
app/models/heading.rb	23	18	60.9%	55.6%
app/models/i_action.rb	13	11	69.2%	63.6%
app/models/instant_message.rb	6	6	50.0%	50.0%
app/models/item.rb	9	8	66.7%	62.5%
app/models/page.rb	429	367	90.4%	88.8%
app/models/page_content.rb	108	85	100.0%	100.0%
app/models/page_group.rb	155	106	94.8%	92.5%
app/models/page_group_change.rb	11	10	100.0%	100.0%
app/models/page_group_page.rb	18	16	72.2%	68.8%
app/models/page_part.rb	62	48	91.9%	89.6%

spec:rcov 默认会为app/下所有文件生成对应覆盖率文档，即使没有对应的测试代码。往往我更专注于model的测试，因此通过配置 spec/rcov.opts 来过滤其他代码覆盖的干扰

```
--exclude "spec/*,gems/*,app/controllers/*,app/helpers/*,app/sweepers/*,app/
```

代码覆盖率很直观，但 100% 的测试覆盖率并不是就完全解决了问题。正如 The Rspec Book 中所说的：低的代码覆盖率说明程序有问题，但高的代码覆盖率并不是说程序一定是完美的。

3.测试运行时附加参数

常用参数

```
--colour                  为测试输出加颜色
--format o                列出测试时间
--format html:test.html   输出测试文档到test.html
--diff                    如测试失败，以diff模式查看期望结果和所得结果的不同
```

diff 需安装 gem

```
sudo gem install diff-lcs
```

以上参数均可选，可搭配
例如

```
spec spec/models/formatable_string_spec.rb --format specdoc --color
```

4.设置 opts

如果不希望在每一次测试中加入测试参数，可以在 *spec/spec.opts* 中加入默认参数

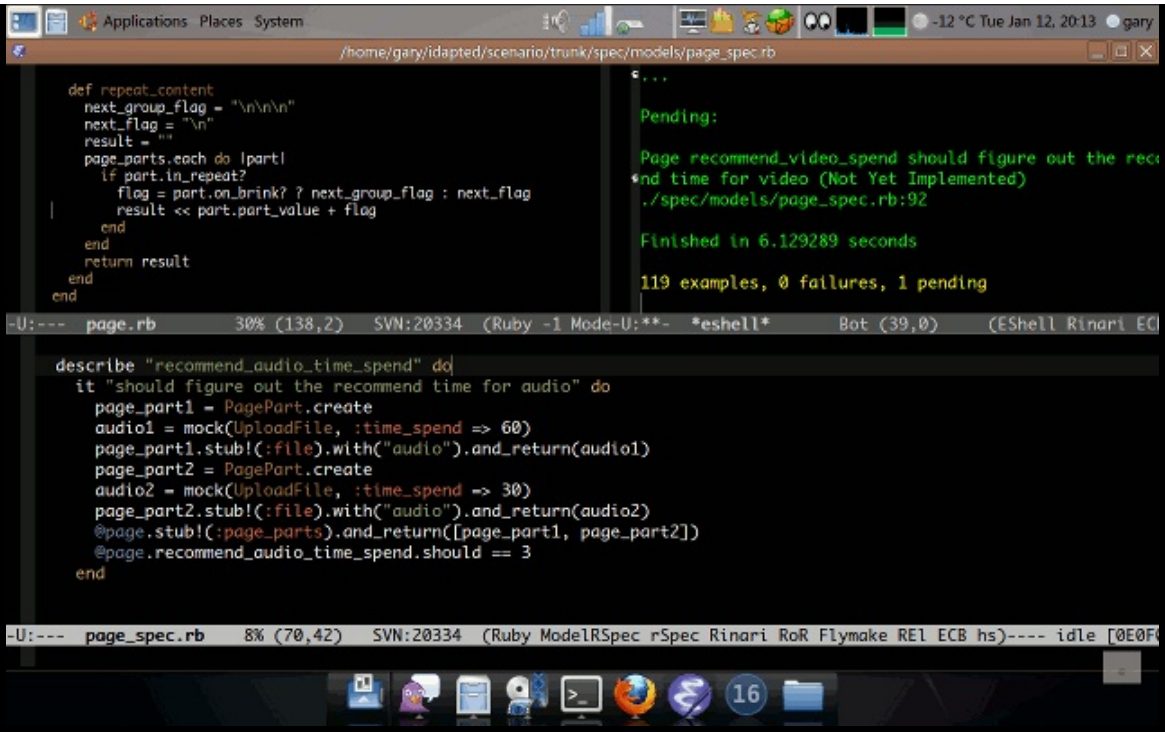
```
--colour
--format html:test.html
--diff
```

结合 1 中生成测试文档的方法，在 *spec/spec.opts* 中加入参数 --format html:test.html，即可在每次生成代码覆盖率时自动生成 html 格式的测试文档，文档会输出在 test.html 中。

5.Autotest

强烈推荐。
autotest 的运行原则：每次修改代码或对应测试代码都会自动运行一次相应测试，如果测试失败，则等待下次修改；如果测试成功，则再将所有测试运行一遍。
在公司项目中配置出现过一些问题，解决方式见前一篇博客。
autotest 结合 test_notifier 更可以实现测试的屏显效果，具体安装方式。

以上的配置，加上emacs的多窗口，这就是我的测试环境。



（左上角是程序代码，右上角是autotest，下面是测试代码）

测试很重要，但并不能解决软件构建的全部问题。正如代码大全 中描述的：
即使考虑到了各种可用的测试手段，测试仍然只是良好软件质量计划的一部分。高质量的开发方法至少和测试一样重要，这包括尽可能减少需求和设计阶段的缺陷。

my rails test environment

Rspec and TDD are great.
Would like to share my configuration and some experience, enjoy:)

1.generate test doc

use params *--format specdoc* will output test doc in term

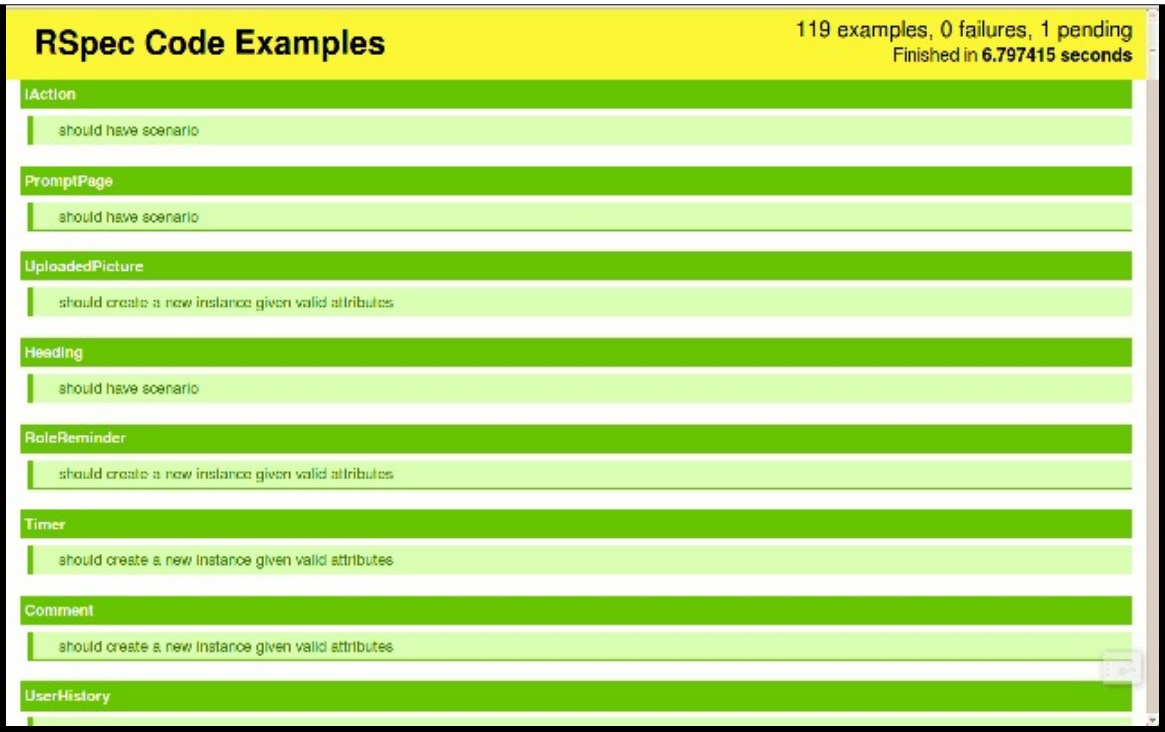
```
spec spec/models/blog_spec.rb --format specdoc
```

Also can output test doc for all tests using commend

```
rake spec:doc
```

But my favourite way is use *--format html* to output doc into a html file

```
spec spec/models/blog_spec.rb --format html:test.html
```



When code coverage and organization are good enough, this doc even can use as documentation for code.

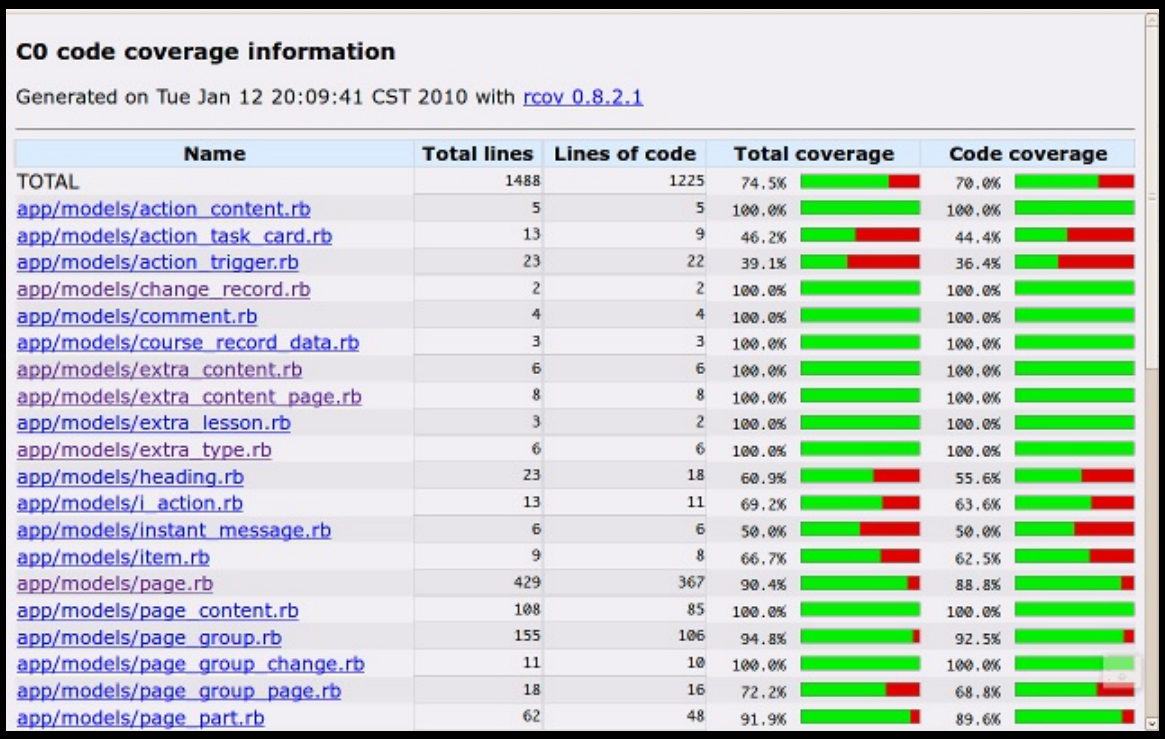
A more convenient way is automate generate this doc after test or rcov, detail see 4 bellow.

2.generate code coverage

rcov

```
rake spec:rcov
```

will generate html format code coverage.



C0 code coverage information

Generated on Tue Jan 12 20:09:41 CST 2010 with rcov 0.8.2.1

Name	Total lines	Lines of code	Total coverage	Code coverage
TOTAL	1488	1225	74.5%	70.0%
app/models/action_content.rb	5	5	100.0%	100.0%
app/models/action_task_card.rb	13	9	46.2%	44.4%
app/models/action_trigger.rb	23	22	39.1%	36.4%
app/models/change_record.rb	2	2	100.0%	100.0%
app/models/comment.rb	4	4	100.0%	100.0%
app/models/course_record_data.rb	3	3	100.0%	100.0%
app/models/extra_content.rb	6	6	100.0%	100.0%
app/models/extra_content_page.rb	8	8	100.0%	100.0%
app/models/extra_lesson.rb	3	2	100.0%	100.0%
app/models/extra_type.rb	6	6	100.0%	100.0%
app/models/heading.rb	23	18	60.9%	55.6%
app/models/i_action.rb	13	11	69.2%	63.6%
app/models/instant_message.rb	6	6	50.0%	50.0%
app/models/item.rb	9	8	66.7%	62.5%
app/models/page.rb	429	367	90.4%	88.8%
app/models/page_content.rb	108	85	100.0%	100.0%
app/models/page_group.rb	155	106	94.8%	92.5%
app/models/page_group_change.rb	11	10	100.0%	100.0%
app/models/page_group_page.rb	18	16	72.2%	68.8%
app/models/page_part.rb	62	48	91.9%	89.6%

spec:rcov by default will generate code coverage for all code under app/ and /lib, even there is no test file for them. It is not good in some condition, we can config *spec/rcov.opts* to change this.
Just generate code coverage for models:

```
--exclude "spec/*,gems/*,app/controllers/*,app/helpers/*,app/sweepers/*,app/
```

Code coverage is great way for checking the quality of code, but 100% coverage does not mean your code is perfect. As says in The Rspec Book:
So while low code coverage is a clear indicator that your specs need some work, high coverage does not necessarily indicate that everything is honky-dory.

3.rspec options

most useful options for me

```
--colour                                Show coloured (red/green) output
--format o                            Specifies format for output: Profiling
--format html:test.html          Specifies format for output: A nice HTML repo
--diff                                    Show diff of objects that are expec
--help                                   Help
```

diff need gem diff-lcs

```
sudo gem install diff-lcs
```

All options are asstable, for example

```
spec spec/models/formatable_string_spec.rb --format specdoc --color
```

4.opts

Do not need to add options every time when test, just do configuration to *spec/spec.opts*

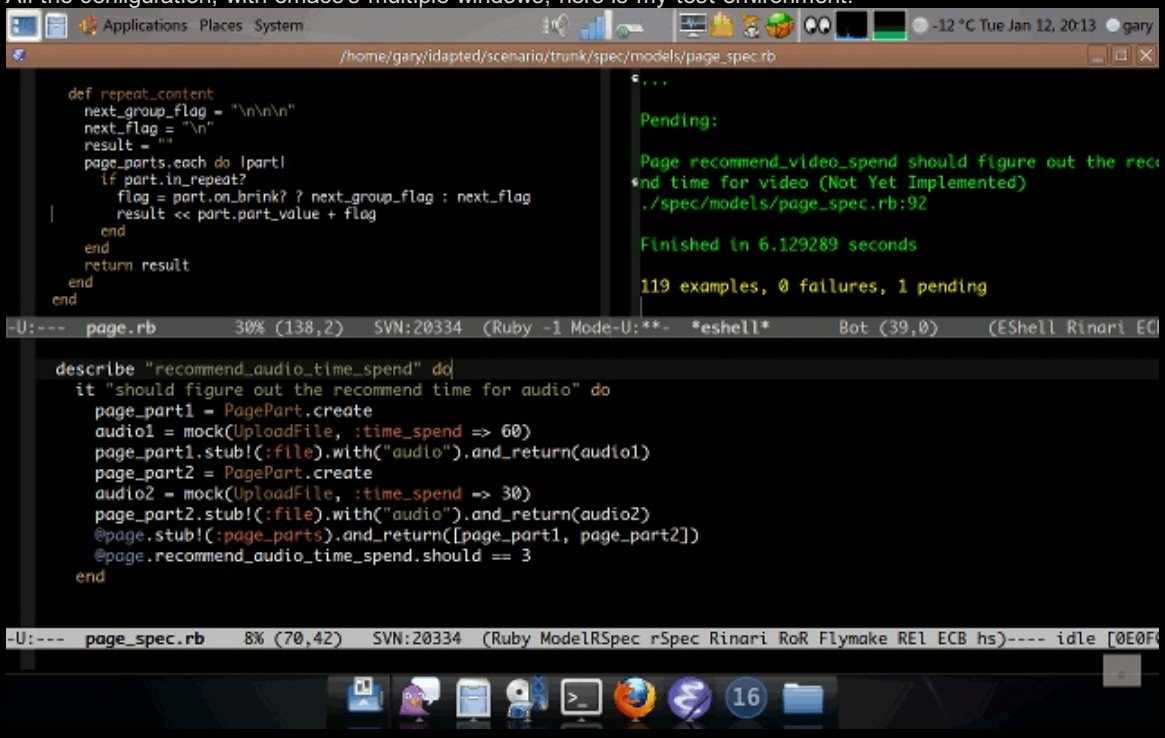
```
--colour
--format html:test.html
--diff
```

Combine the way of "generating test doc" in 1 , add *--format html:test.html* to *spec/spec.opts* , will automate generate test doc after each test or rcov , the doc will output into test.html.

5.Autotest

Autotest is very cleaver, every time you save a test file, autotest will run that test file; and every time you save a library file, autotest will run the corresponding test file. If it sees that the previous failures are now passing, it loads up the entire suite and runs all of the examples again.
When tried autotest on company project, there was a problem. Solution see solve autotest always rerun problem.

All the configuration, with emacs's multiple windows, here is my test environment:



(code top left, autotest top right, test code underneath)

Test is good, but as says in Code Complete:
Even considering the numerous kinds of testing available, testing is only one part of a good software-quality program. High-quality development methods, including minimizing defects in requirements and design, are at least as important.

Do not excessive relay on test.

解决 autotest 不停执行的问题

运行 autospec ，发现即使不修改保存任何文件，测试也会不停在后台执行。
调查发现原因是某个公司管理 gem(idp_gem) 的问题。这个gem在测试运行时，会自动更新 **spec/spec_helper.rb** 文件，而这个文件的更新会触发下一轮测试的执行。于是就造成了:

运行次测试 => 自动更新文件 => 运行测试 => 自动更新文件...

的死循环。

处理方法很简单，在项目目录下 *.autotest* 文件中加入

```
Autotest.add_hook :initialize do |at|
  %w{spec/spec_helper.rb}.each {|exception| at.add_exception(exception)}
end
```

重新运行

```
autospec
```

问题解决：)

solve autotest always rerun problem

Run autospec，found even not change and save happens, autotest will rerun time and time...

After research I found it happens because of a gem (idp_gem): every time when running a test, this gem will update file **spec/spec_helper.rb**, and this file will trigger next time's autotest.
So endless loop

autotest => file updates => autotest => file updates...
happens.

The solution is easy，add to .autotest

```
Autotest.add_hook :initialize do |at|
  %w{spec/spec_helper.rb}.each {|exception| at.add_exception(exception)}
end
```

rerun

```
autospec
```

Problem solved:)

使用 emacs 搜索 ruby 文件文本

参考 Doug Alcorn的方法。首先安装 exuberant ctags。
创建 rake 任务，用于为当前 project 创建 tag。

```
module Tags
  RUBY_FILES = FileList['**/*.rb']
end
namespace "tags" do
  task :emacs => Tags::RUBY_FILES do
    puts "Making Emacs TAGS file"
    sh "/usr/bin/ctags -e #{Tags::RUBY_FILES}", :verbose => false
  end
end
task :tags => ["tags:emacs"]
```

运行任务生成tags。

```
rake tags:emacs
```

现在可以使用 *tags-search* 来查找 tags; 使用 *tags-loop-continue* 切换下一个 tag， 很方便：）
当然最好还是指定快捷键，我指定为

```
(global-set-key (kbd "C-c C-c C-s") 'tags-search)
(global-set-key (kbd "C-c C-c C-n") 'tags-loop-continue)
```

经过上面的操作可以查找.rb文件，但还是不能查找 view 文件：（

两个很方便的快捷键。

```
C-h w         where-is
C-h k         describe-key
```

顺便重新梳理了一遍 emacs配置和快捷键。
目前最常用的自定义快捷键：

```
C-c C-c C-c               rinari-find-controller
C-c C-c C-m               rinari-find-model
C-c C-c C-v               rinari-find-view
C-c C-c C-r               rinari-find-rspec
C-c C-c C-f               rinari-find-file-in-project
C-c C-c C-h               hide-all-blocks
C-c C-c C-t               toggle-hiding-block
C-c C-c C-o               comment-region
C-c C-c C-u               uncomment-region
C-c C-c C-g               goto-line
```

"find in project" in emacs

Reference Doug Alcorn's method. First install exuberant ctags
create a rake task for creating tags for current project.

```
module Tags
  RUBY_FILES = FileList['**/*.rb']
end
namespace "tags" do
  task :emacs => Tags::RUBY_FILES do
    puts "Making Emacs TAGS file"
    sh "/usr/bin/ctags -e #{Tags::RUBY_FILES}", :verbose => false
  end
end
task :tags => ["tags:emacs"]
```

run task

```
rake tags:emacs
```

Now can use *tags-search* to search tags; and use *tags-loop-continue* to loop tags, great:)
I will be more handy using shortcut, what I did:

```
(global-set-key (kbd "C-c C-c C-s") 'tags-search)
(global-set-key (kbd "C-c C-c C-n") 'tags-loop-continue)
```

Right now it can search all the *.rb files, but not others. It works, but not perfect:(

Two very handy shortcut

```
C-h w         where-is
C-h k         describe-key
```

Refreshed emacs config and shortcut again.
Most popular shortcuts:

```
C-c C-c C-c               rinari-find-controller
C-c C-c C-m               rinari-find-model
C-c C-c C-v               rinari-find-view
C-c C-c C-r               rinari-find-rspec
C-c C-c C-f               rinari-find-file-in-project
C-c C-c C-h               hide-all-blocks
```

```
C-c C-c C-t                toggle-hiding-block
C-c C-c C-o                comment-region
C-c C-c C-u                uncomment-region
C-c C-c C-g                goto-line
```

腐败第一站 --- 江边城外巫山烤全鱼

first FuBai --- JiangBianChengWai WuShan Fish

一德腐败联盟成立

公司的第四个联盟成立:)
详情见:

亲爱的大家伙：
　今天基于大多数吃货的强烈要求，我们决定成立腐败联盟，简称**FBA**（Fu Bai Association of Idapted）。联盟的目标是本着交通方便，物美价廉的原则，组织大家尝遍北京美食。
　联盟将定期组织聚会，首先以公司附近的美食餐馆为目标，以后逐渐扩大到全北京市。目前的目标包括，巫山烤鱼，京港百汇海鲜。欢迎大家推荐好的餐馆。

联盟的好处
1.共享，发现美食：任何人都可以推荐自己最喜欢的餐馆。推荐获得大家认可会有相应**奖励**。
2.相对便宜：人多价格相对**便宜**，同时可以享受更多种类的美食。
3.方便快捷：可以一起拼车出发，免去寻找的麻烦；会有指定人提前排队等候，可以**节省时间**。

4.加强交流：通过共同享受美食熟悉同事，**朋友**。
5.留下美好回忆：我们每次合影留念，记下**美好的瞬间**。
6.更好地熟悉北京：很多北京美食里都包含**北京文化**在里面。

7.其他 …

　联盟会暂时每人收*30*元作为储备经费。而每次活动预计金额会控制在50-60元左右，多退少补在储备经费里。
　联盟秘书——超级谨慎的**andy**同学会负责保存活动经费，并在每次活动后汇报经费余额。呱唧呱唧…

　基于煮蛋协会经营不善，业已关闭，我决定将剩余30元左右煮蛋基金，打入**FBA**作为**初始基金**。同时感谢兄弟联盟篮球协会和台球协会的大力支持：各捐赠三十元作为**互助基金**。
　联盟目前加入成员包括：denny, seven, guolei, andy, buzz, sonic, sherry, angela, scott, gary。 热烈欢迎大家加入。想加入请在后面跟贴说明即可：)

　联盟首次活动定在下周（或大下周），地点为交大的 **江边城外巫山烤全鱼**。图片见附件。我可以很负责任的说，这是我目前吃过的最好吃的**烤鱼**，太好吃了。强烈烈烈烈烈烈烈烈烈烈烈烈烈烈烈推荐！！！ 由于很难定位子，我们需要**sherry**同学帮忙提前定位子。呱唧呱唧…

　如果有任何不清楚，请告诉我。欢迎大家提出宝贵意见。

　谢谢:)

Fu Bai Association of Idapted

The fourth Company Association established:)
Details:

亲爱的大家伙：
今天基于大多数吃货的强烈要求，我们决定成立腐败联盟，简称**FBA**（Fu Bai Association of Idapted）。联盟的目标是本着交通方便，物美价廉的原则，组织大家尝遍北京美食。
联盟将定期组织聚会，首先以公司附近的美食餐馆为目标，以后逐渐扩大到全北京市。目前的目标包括，巫山烤鱼，京港百汇海鲜。欢迎大家推荐好的餐馆。

联盟的好处
1.共享，发现美食：任何人都可以推荐自己最喜欢的餐馆。推荐获得大家认可会有相应**奖励**。
2.相对便宜：人多价格相对**便宜**，同时可以享受更多种类的美食。
3.方便快捷：可以一起拼车出发，免去寻找的麻烦；会有指定人提前排队等候，可以**节省时间**。

4.加强交流：通过共同享受美食熟悉同事，**朋友**。
5.留下美好回忆：我们每次合影留念，记下**美好的瞬间**。
6.更好地熟悉北京：很多北京美食里都包含**北京文化**在里面。

7.其他 ...

联盟会暂时每人收*30*元作为储备经费。而每次活动预计金额会控制在50-60元左右，多退少补在储备经费里。
联盟秘书——超级谨慎的**andy**同学会负责保存活动经费，并在每次活动后汇报经费余额。呱唧呱唧...

基于煮蛋协会经营不善，业已关闭，我决定将剩余30元左右煮蛋基金，打入**FBA**作为**初始基金**。同时感谢兄弟联盟篮球协会和台球协会的大力支持：各捐赠三十元作为**互助基金**。
联盟目前加入成员包括：denny, seven, guolei, andy, buzz, sonic, sherry, angela, scott, gary。 热烈欢迎大家加入。想加入请在后面跟贴说明即可：)

联盟首次活动定在下周（或大下周），地点为交大的 **江边城外巫山烤全鱼**。图片见附件。我可以很负责任的说，这是我目前吃过的最好吃的**烤鱼**，太好吃了。强烈烈烈烈烈烈烈烈烈烈烈烈烈烈推荐！！！ 由于很难定位子，我们需要**sherry**同学帮忙提前定位子。呱唧呱唧...

如果有任何不清楚，请告诉我。欢迎大家提出宝贵意见。

谢谢:)

include 和 extend 的区别

请看下面的代码：

```ruby
module Gary
  def hello
    puts "hello"
  end

  def self.world
    puts "world"
  end
end

class A
  include Gary
end

class B
  extend Gary
end

A.new.hello #=> "hello"
B.hello #=> "hello"
A.world # throw error: undefined method `world' for A:Class (NoMethodError)
B.world # throw error: undefined method `world' for B:Class (NoMethodError)
```

结论是:

include 让目标类可以以实例方法的形式调用包含的module中的实例方法;
extend 让目标类可以以类方法的形式调用包含的module中的实例方法;
include 和 **extend** 都无法调用包含的module中的类方法。

Difference between include and extend.

Difference between **include** and **extend**.
Check bellow code:

```
module Gary
  def hello
    puts "hello"
  end

  def self.world
    puts "world"
  end
end

class A
  include Gary
end

class B
  extend Gary
end

A.new.hello #=> "hello"
B.hello #=> "hello"
A.world # throw error: undefined method `world' for A:Class (NoMethodError)
B.world # throw error: undefined method `world' for B:Class (NoMethodError)
```

So solution is:

include makes the instance methods in source module as instance method for target class;
extend makes the instance methods in source module as class method for target class;
neither **include** nor **extend** can use class methods of source module for target class.

解决 rails 代码创建索引名过长的问题

在rails中为数据加索引很简单，
在 migration中

```
add_index(:test_statistics, [:page_id, :study_record_id, :student_id])
```

但如果联合索引，很可能造成索引名过长mysql无法处理：

```
Mysql::Error: Identifier name
'index_test_statistics_on_page_id_and_study_record_id_and_student_id'
is too long: CREATE  INDEX
index_test_statistics_on_page_id_and_study_record_id_and_student_id`
ON `test_statistics` (`page_id`, `study_record_id`, `student_id`)
```

解决方法是进入dbconsole, 简短索引名创建：

```
CREATE  INDEX `my_pss_index` ON `test_statistics` (`page_id`, `study_record_
```

问题解决：)

solve the index "Identifier name too long" problem

It is easy to add **index** in rails
in migration:

```
add_index(:test_statistics, [:page_id, :study_record_id, :student_id])
```

But sometimes mysql can not handle it because of "Identifier name too long"

```
Mysql::Error: Identifier name
'index_test_statistics_on_page_id_and_study_record_id_and_student_id'
is too long: CREATE  INDEX
index_test_statistics_on_page_id_and_study_record_id_and_student_id`
ON `test_statistics` (`page_id`, `study_record_id`, `student_id`)
```

solution is go into dbconsole, manual create index using shorter identifier name:

```
CREATE  INDEX `my_pss_index` ON `test_statistics` (`page_id`, `study_record_
```

solve：)

使用视图解决 rails 跨数据库操作的单表继承问题

需要做跨数据库读取操作，创建model page_group.rb

```
class PageGroup < ActiveRecord::Base
  ActiveRecord::Base.establish_connection(
    :adapter  => "mysql",
    :host     => "localhost",
    :username => "myuser",
    :password => "mypass",
    :database => "somedatabase"
  )
end
```

如果源程序中PageGroup使用了单表继承，在console中使用

```
PageGroup.first
```

得到出错信息:

```
ActiveRecord::SubclassNotFound: The single-table inheritance mechanism
failed to locate the subclass: 'sss'. This error is raised because the
column 'type' is reserved for storing the class in case of inheritance.
Please rename this column if you didn't intend it to be used for
storing the inheritance class or overwrite PageGroup.inheritance_column
to use another column for that information.
```

解决方案是在源程序中为使用单表继承的表创建 视图：

```
execute("CREATE VIEW extra_lessons AS SELECT id, title, blabla FROM page_gro
```

问题解决:)

补充:
正如 ruby on rails code quality checklist 中所说的， 单表继承不是共享代码的好方法。而且 type 字段在 rails 中还会引起以上类似问题。
所以，尽量少使用单表继承吧。

use view to sovle "single-table inheritance problem" when rails app cross database

I want to read data cross database, so I create a model page_group.rb and code like:

```
class PageGroup < ActiveRecord::Base
  ActiveRecord::Base.establish_connection(
    :adapter  => "mysql",
    :host     => "localhost",
    :username => "myuser",
    :password => "mypass",
    :database => "somedatabase"
  )
end
```

If PageGroup model in source app use single-table inheritance, then in console:

```
PageGroup.first
```

will got error:

```
ActiveRecord::SubclassNotFound: The single-table inheritance mechanism
failed to locate the subclass: 'sss'. This error is raised because the
column 'type' is reserved for storing the class in case of inheritance.
Please rename this column if you didn't intend it to be used for
storing the inheritance class or overwrite PageGroup.inheritance_column
to use another column for that information.
```

Solution is create view for the table use single-table inheritance(PageGroup) in source app:

```
execute("CREATE VIEW extra_lessons AS SELECT id, title, blabla FROM page_gro
```

solved:)

如何调试 ruby 代码

调试代码在编程中有很重要的作用，一个好的调试器有助于迅速发现和解决问题。
我的调试方法如下:

方法一， 在任何需要的位置写入

```
puts something
```

以输出调试信息到logs。这是最基本的调试方法，还可以使用tap不修改代码结果，使调试过程变得更简单。

方法二， 我在 lib/my_puts.rb 写入如下的方法：

```
def gp(s = "")
  puts "\n\n<<<<<<<<<<<<<<<<<<<<<<<<<<"
  puts s
  puts "<<<<<<<<<<<<<<<<<<<<<<<<<<\n\n"
end
```

并在 enviroment.rb 中写入：

```
require 'my_puts'
```

这样就可以在rails环境中使用

```
pg "I am strong"
```

会获得输出

```
<<<<<<<<<<<<<<<<<<<<<<<<<<
I am strong
<<<<<<<<<<<<<<<<<<<<<<<<<<
```

使调试信息更容易找到。

3.ruby-debug 是更好的解决方法。使用debugger甚至可以调试测试代码和rake任务。需要做的只是在想调试的位置加入 **debugger**.
最常用的命令ruby-debug

```
list
irb
p
```

不要在调试完毕后忘了删去debugger：)
4.如果使用TDD或是BDD的方法编程，并达到100%的测试覆盖率，理论上再不用调试。
遗憾的是目前很难达到。

How I debug ruby code

Debug code is so important.
I would like to share my ways. Enjoy:)

1. put in anywhere of the code need to debug

```
puts something
```

it will output *something* into logs.
This is the basic way but very easy, further more you can use tap to make it easier.

2.later I wrote a method in lib/my_puts.rb like

```
def gp(s = "")
  puts "\n\n<<<<<<<<<<<<<<<<<<<<<<<<<<"
  puts s
  puts "<<<<<<<<<<<<<<<<<<<<<<<<<<\n\n"
end
```

and put into enviroment.rb

```
require 'my_puts'
```

So in Rails enviroment, use

```
pg "I am strong"
```

will output

```
<<<<<<<<<<<<<<<<<<<<<<<<<<
I am strong
<<<<<<<<<<<<<<<<<<<<<<<<<<
```

It make the debug info more easy to be found.

3.Debugger is the final debug solution. It even can be use in writing test code or rake task. Just puts **debugger** in your code.
Most useful commands in ruby-debug

```
list
irb
p
```

Do not forget to delete the debugger afterward:)

4.Of course, if use TDD or BDD and meeting 100% code coverage, no need debug any more.
But we need...

我的2009

Ubuntu, Emacs, Rails, Rspec, IdpV2，两次铁人比赛， 永定河穿越，泰山， 白洋淀, 幸福而苦涩的感情...,
不足以描述。太多的点点滴滴，时间却转瞬即逝。

my 2009

Ubuntu, Emacs, Rails, Rspec, IdpV2, Ironman, Triathlon, YongDing river traverse , Mount Tai , Baiyang Lake, full of happiness and bitter's love ..., can not pictured. so many things, time passes in a twinkling.

用 whenever 做定时任务

统计数据，需要从多个数据库，多个表读取数据，计算组合得出统计结果。
速度很慢，做了一些优化，但效果不佳。
于是想改成非实时统计：每天在指定时间进行一次任务统计，将计算后的统计结果存入数据库。这样每次查看统计结果时只需读取计算好的数据就好。
使用whenever可以很方便的做定时任务。

安装

```
$ sudo gem install whenever
```

在 config/environment.rb 写入

```
Rails::Initializer.run do |config|
  config.gem 'whenever', :lib => false, :source => 'http://gemcutter.org/'
end
```

进入程序目录

```
$ cd /my/rails/app
```

生成cron文件

```
$ wheneverize .
```

会在config下面生成 schedule.rb 文件
在文件中写定时任务：

```
set :output, "/home/app/applications/scenario/current/log/whenever.log"
every 1.minutes do
  runner "puts ('still working')"
  command "date"
end

every 1.day, :at => '02:00 am' do
  rake "update_course_record"
end
```

注意：
1. set :output 会把任务的输出打印到指定log文件中，但如果文件不存在，不会有出错信息。
2. 使用 command "date" 可以打印出任务执行的当前时间。 使用 runner "puts Time.now"只会每次打印出首次将whenever命令转换成系统任务的时间。
3. rake "update*course*record" 是我要执行的定时任务，会计算数据，并将结果存入统计数据表。我让它在每天的凌晨两点执行（为服务器空闲并且负载最小的时间）。

翻译并在/var/spool/cron/crontabs生成系统任务

```
$ whenever --update-crontab scenario
$ crontab -l
```

完成：)

如果想查看更多的记录信息, 这里有个很好的方法。

use whenever do cron job

Doing statistic things, need to statistic data from several databases and tables.
So slow and optimization seems not effective.

I want to make it not real time statistic: do cron job to count data at assign time and save the statistic result into database.
It will be very easy to do this using whenever.

install

```
$ sudo gem install whenever
```

write into config/environment.rb

```
Rails::Initializer.run do |config|
```

```
  config.gem 'whenever', :lib => false, :source => 'http://gemcutter.org/'
end
```

go to directory of app

```
$ cd /my/rails/app
```

generate cron file

```
$ wheneverize .
```

will generate schedule.rb at config/
in schedule.rb

```
set :output, "/home/app/applications/scenario/current/log/whenever.log"
every 1.minutes do
  runner "puts ('still working')"
  command "date"
end

every 1.day, :at => '02:00 am' do
  rake "update_course_record"
end
```

importent
1. set :output will output log into assigned log file, but if the file not exists, there is not error message.
2. command "date" will print current time; runner "puts Time.now" will print time the first time when whenever task converts to system task.
3. rake "update*course*record" is the cron task I want to run, it will statistic and save the result into database. I want it run at 2:00 am very day(when system is idle).

Generate system task into /var/spool/cron/crontabs

```
$ whenever --update-crontab scenario
$ crontab -l
```

done:)

If you need more logs about what happened, see here

添加缓存并比较性能提升

刚刚为博客添加了缓存，并想进一步查看性能提升了多少。

Rails新的缓存机制很棒, 只需要把想要缓存的代码放入block中.

```
<% cache @blog do %>
  <ht>
    <%= link_to h(@blog.title), blog_path(@blog) %>
    <%= link_to "(#{@blog.brother.type} version)", blog_path(@blog.brother)
  </ht>
  <hd><%= @blog.created_at.to_date %></hd>
  <div>
    <%= g_textile @blog.content %>
  </div>
<% end %>
```

New Relic是很好的性能评测工具。
用plugin安装多次没有成功。
使用gem安装：

```
sudo gem install newrelic_rpm
```

并把下面的代码加到environment中：

```
config.gem "newrelic_rpm"
```

下面的图片是依据newrelic得出的缓存前后访问首页十次的性能对比

缓存前

Timestamp	Resp. Time	URL	
08:00:08	667 ms	/blog_groups	[Detail] [SQL (15)]
08:00:08	605 ms	/blog_groups	[Detail] [SQL (15)]
08:00:06	515 ms	/blog_groups	[Detail] [SQL (15)]
08:00:06	572 ms	/blog_groups	[Detail] [SQL (15)]
08:00:05	552 ms	/blog_groups	[Detail] [SQL (15)]
08:00:05	545 ms	/blog_groups	[Detail] [SQL (15)]
08:00:04	542 ms	/blog_groups	[Detail] [SQL (15)]
08:00:03	546 ms	/blog_groups	[Detail] [SQL (15)]
08:00:02	500 ms	/blog_groups	[Detail] [SQL (15)]
08:00:00	765 ms	/blog_groups	[Detail] [SQL (15)]

缓存后

Timestamp	Resp. Time	URL	
08:00:09	205 ms	/blog_groups	[Detail] [SQL (15)]
08:00:08	212 ms	/blog_groups	[Detail] [SQL (15)]
08:00:07	206 ms	/blog_groups	[Detail] [SQL (15)]
08:00:07	149 ms	/blog_groups	[Detail] [SQL (15)]
08:00:06	142 ms	/blog_groups	[Detail] [SQL (15)]
08:00:05	150 ms	/blog_groups	[Detail] [SQL (15)]
08:00:04	210 ms	/blog_groups	[Detail] [SQL (15)]
08:00:03	206 ms	/blog_groups	[Detail] [SQL (15)]
08:00:02	206 ms	/blog_groups	[Detail] [SQL (15)]
08:00:01	464 ms	/blog_groups	[Detail] [SQL (15)]

性能大约有300%的提升：)

add profiling and profiling tool to blog

Just added **Cache** in my blog. Want to check the performance enhancements.

New cache in rails is so good, just put code to be cached in a block is OK.

```
<% cache @blog do %>
  <ht>
    <%= link_to h(@blog.title), blog_path(@blog) %>
    <%= link_to "(#{@blog.brother.type} version)", blog_path(@blog.brother)
  </ht>
  <hd><%= @blog.created_at.to_date %></hd>
  <div>
    <%= g_textile @blog.content %>
  </div>
<% end %>
```

About Profiling tool, New Relic is a god choise.
Install newrelic as a plugin seems not work. so install the gem

```
sudo gem install newrelic_rpm
```

and add to environment

```
config.gem "newrelic_rpm"
```

Below is the comparation of time spend on home page without cache and with cache for 10 times from newrelic.

Before cache

Timestamp	Resp. Time	URL	
08:00:08	667 ms	/blog_groups	[Detail] [SQL (15)]
08:00:08	605 ms	/blog_groups	[Detail] [SQL (15)]
08:00:06	515 ms	/blog_groups	[Detail] [SQL (15)]
08:00:06	572 ms	/blog_groups	[Detail] [SQL (15)]
08:00:05	552 ms	/blog_groups	[Detail] [SQL (15)]
08:00:05	545 ms	/blog_groups	[Detail] [SQL (15)]
08:00:04	542 ms	/blog_groups	[Detail] [SQL (15)]
08:00:03	546 ms	/blog_groups	[Detail] [SQL (15)]
08:00:02	500 ms	/blog_groups	[Detail] [SQL (15)]
08:00:00	765 ms	/blog_groups	[Detail] [SQL (15)]

After cache

Timestamp	Resp. Time	URL	
08:00:09	205 ms	/blog_groups	[Detail] [SQL (15)]
08:00:08	212 ms	/blog_groups	[Detail] [SQL (15)]
08:00:07	206 ms	/blog_groups	[Detail] [SQL (15)]
08:00:07	149 ms	/blog_groups	[Detail] [SQL (15)]
08:00:06	142 ms	/blog_groups	[Detail] [SQL (15)]
08:00:05	150 ms	/blog_groups	[Detail] [SQL (15)]
08:00:04	210 ms	/blog_groups	[Detail] [SQL (15)]
08:00:03	206 ms	/blog_groups	[Detail] [SQL (15)]
08:00:02	206 ms	/blog_groups	[Detail] [SQL (15)]
08:00:01	464 ms	/blog_groups	[Detail] [SQL (15)]

From the pictures, there is about 300% performance enhancement. good:)

没有实现在 model, rake 中调用helper方法

我要在controller, model, 甚至rake中使用helper方法。
使用

```
include ApplicationHelper
```

可以调用ApplicationHelper中定义的方法。但如果这些方法使用了其他helper方法，会抛出"undefined method", 提示未定义的方法是rails提供的helper方法 。
问题很奇怪，因为在controller使用include方法中调用helper在我的gem中工作很好。
于是我仿照gem的调用方法，将我的helper包含到base helper中：

```
ActionView::Base.send(:include, Student::CourseRecordsHelper)
```

这次不再有"undefined method"错误，但取而代之的是

```
The error occurred while evaluating nil.url_for
/usr/lib/ruby/gems/1.8/gems/actionpack-2.3.4/lib/action_view/helpers/url_hel
/usr/lib/ruby/gems/1.8/gems/actionpack-2.3.4/lib/action_view/helpers/url_hel
```

继续尝试将所需module逐一include进来，得到了和上面同样的错误信息。

找到另外两种在在helper外调用helper方法的办法:

```
1.ActionController::Base.helpers.link_to("", "")
2.@template.link_to("", "")
```

第一种方法可以用在任何地方；第二种方法只可以用在controller。
更多细节
但上面两种方式都不能解决我的问题。首先我想调用的是自己创建的方法，方法一不好使；其次我使用的地方是在rake中，方法二不好使。

===

最后用其他解决方案来绕过这个调用问题，并使实现过程更合理，代码更整洁。
我想以后遇到类似问题应该跳出问题重新思考。因为往往难以实现的技术问题有时也可能是因为设计的不合理。

===
使用 helper 可以在 console 中调用 helper 方法

```
helper.truncate "Big sentence", 5
```

use helper methods in controller, model, or even rake(not achieved)

I wanted to use helper methods in controller, model, or even rake.

So I add

```
require 'helpers/application_helper'
include ApplicationHelper
```

Then I can use methods of ApplicationHelper. But if these methods call other helper methods from out side, I got "undefined method" error.

Wired thing is , the "include" way works well in gem "ldp::File" when include helper. So I try the way gem include module and add in controller:

```
ActionView::Base.send(:include, Student::CourseRecordsHelper)
```

The "undefined method" did not show up, but got

```
The error occurred while evaluating nil.url_for
/usr/lib/ruby/gems/1.8/gems/actionpack-2.3.4/lib/action_view/helpers/url_hel
/usr/lib/ruby/gems/1.8/gems/actionpack-2.3.4/lib/action_view/helpers/url_hel
```

I also tried include all need modules, but got the same error.

Trying to find other way, and got two:

```
1.ActionController::Base.helpers.link_to("", "")
2.@template.link_to("", "")
```

The first way can be used in controller, model or any where in rails; the second way can only be used in controller. more details

But neither of them can solve my problem. First, methods I want to call are defined by myself , first way not work ； Second, where I want to call helper methods is in rake , second way not work either.

Stop here, I did not find a way to solve it.

====
JP and seven advise me other way to achieve my requirement. Which make it easier and cleaner.
I think next time meet this kind of problems, I should jump out and rethink. Some function difficult to achieve may because it is not reasonable. Just try other idea.

===
Use helper can call helper methods in console

```
helper.truncate "Big sentence", 5
```

对比 Screen Emacs Terminator 打开多个窗口

认识screen

之前在服务器上执行长时间任务时，需要一直开着命令行，还要担心网络。screen很好地解决了这个问题。
详细使用说明

基本命令

```
screen -S name          start screen with name
screen -r name          resume a screen
screen -D name          detach an attached screen

Ctrl -a  c              create new screen
Ctrl -a  n              next screen
Ctrl -a  d              detach screen left it running in background
```

使用screen同时打开多个窗口

有时在服务器上调试，需要在多个目录切换。发现也可以利用screen：创建多个screen, 使用 Ctrl + a + n 切换。缺点是需要同时打开多个screen，使用完毕经常忘记关；无法同时显示多个screen，有时为编辑带来不方便。

使用emacs同时打开多个窗口

Ctrl + x + 2(3) 打开多个窗口, Ctrl + x + 左右键 切换。 优点：方便快捷.缺点：需要记快捷键（如果爱用emacs没有缺点）。

使用terminator同时打开多个窗口

优点和emacs一致，而且不用记住emacs特殊的快捷键。缺点是需要多次登陆服务器（等效于同时打开多个命令行）。
安装及使用方式

结论

screen可以很好的解决后台任务，也可以勉强满足多窗口需求；但更好的解决方案是emacs和terminator。如果工作时习惯使用emacs,这就是最好选择；terminator同样可以很好地满足多窗口需求，而且不用记emacs命令。

Screen VS Emacs VS Terminator on opening multiple screens

first meet screen

when run migration or rake task on server, should keep the connection and worry about the net. Screen solves this problem well.

Basic screen commends

```
screen -S name          start screen with name
screen -r name          resume a screen
screen -D name          detach an attached screen

Ctrl -a  c              create new screen
Ctrl -a  n              next screen
Ctrl -a  d              detach screen left it running in background
```

use screen opening multiple windows

When work on remote server, sometimes need to switch from several directories. I found this also can be done with screen: create several screens, using "Ctrl + a + n" to change screen. Disadvantage : need to open multiple screens at the same time, and always forget to close; can not show all screens at the same time.

use Emacs opening multiple windows

Ctrl + x + 2(3) open several windows, Ctrl + x + right(left) to switch. Advantage: handy; Disadvantage ：need to remember emacs's short cuts(if you love emacs, it is perfect).

use Terminator opening multiple windows

Advantage: the same as emacs，and need not to remember emacs's commands; Disadvantage: need ssh several times to login server(just like open multiple terms).

conclusion

screen can solve back end task well, and can get by multiple windows requirement; but better solution are emacs and terminator. If be used to emacs, it is best chiose; terminator is also a good choise, and no need to remember many commands.

Textile 的所见即所得编辑器

公司内容组需要 Textile & WYSIWYG（所见即所得）编辑器来更好地输入课程。google找到Sanskrit可以满足绝大部分的需求，但不具备列表，颜色，超链接和字符转义的功能。还好代码清晰，不难添加新功能。

这里是修改后的例子. 更多信息请访问这里。 我同样将这个编辑器用在这个博客上：）

共享一些用于textile和html相互转换的javascript的正则表达式。写了这些痛苦，难于测试的代码才真正感受到用 ruby on rails 编程的快乐。

```
textilize: function(html, escape){
        html = html.replace(/<br ?\/?>/gi, "\n");
        html = html.replace(/<(?:b|strong)>((.|[\r\n])*?)<\/(?:b|strong)>/gi
        html = html.replace(/<(?:i|em)>((.|[\r\n])*?)<\/(?:i|em)>/gi, '_$1_'
        html = html.replace(/<(?:u|ins)>((.|[\r\n])*?)<\/(?:u|ins)>/gi, '+$1
        html = html.replace(/<a href="(.*?)">((.|[\r\n])*?)<\/a>/gi, '"$2":$
        html = html.replace(/<font color="#0000ff".*?>((.|[\r\n])*?)<\/font>
        html = html.replace(/<font color="#FF0000".*?>((.|[\r\n])*?)<\/font>
        html = html.replace(/<font color="#088A4B".*?>((.|[\r\n])*?)<\/font>
        //list
        var r = /([\s\S]*)<ul>([\s\S]*?)<\/ul>([\s\S]*)/;
        while (r.test(html))
            {
                p = RegExp.$1;
                c = RegExp.$2;
                e = RegExp.$3;
                c = c.gsub(/<li>([\s\S]*?)</li>/, "* #{1}\n");
                html = p + "\n" + c + e;
            }
        html = html.replace(/^[\r\n]+|[\r\n]+$/g, '');
        return html;
    },
```

```
htmlize: function(textile, escape){
        //escape
        textile = textile.replace(/\/gi, "0142540974231612");
        textile = textile.replace(/\\_/gi, "07500238936923338");
        textile = textile.replace(/\\*/gi, "06430048798350 27");

        textile = textile.replace(/\n/gi, '<br/>');
        textile = textile.replace(/\*(.+?)\*/gi, (this.internetExplorer ? '<
        textile = textile.replace(/_(.+?)_/gi, (this.internetExplorer ? '<em
        textile = textile.replace(/(.+?)/gi, '<u>$1</u>');
        textile = textile.replace(/"(.+?)":([^\s\n<]+)/gi, '<a href="$2">$1<
        textile = textile.replace(/%{color:blue}(.+?)%/gi, '<font color="#00
        textile = textile.replace(/%{color:red}(.+?)%/gi, '<font color="#FF0
        textile = textile.replace(/%{color:green}(.+?)%/gi, '<font color="#0
        textile = textile+'<br\/>';
        textile = textile.replace(/<p>[\s\n]*<\/p>/g, '');
        //unescape
        textile = textile.replace(/0142540974231612/, "\");
        textile = textile.replace(/07500238936923338/, "\\_");
        textile = textile.replace(/06430048798350 27/, "\\*");
        //list
        var r = /(<br\/>\* .*?<br(\/)*>)(?!\* )/;
        while (r.test("<br/>" + textile))
            {
                l = RegExp.leftContext;
                m = RegExp.lastMatch;
                e = RegExp.rightContext;
                m = m.gsub(/\* (.*?)<br(\/)*>/, "<li>#{1}</li>").gsub("<br/>
                textile = l + "<ul>" + m + "</ul>" + e;
            }
        return textile;
    },
```

注意:

"JavaScript and VBScript do not have an option to make the dot match line break characters. In those languages, you can use a character class such as [\s\S] to match any character." 这里 更详细的说明。

因此在ruby中:

```
/(.*)<ul>(.*?)<\/ul>(.*)/
```

在javascript中需要写成:

```
/([\s\S]*)<ul>([\s\S]*?)<\/ul>([\s\S]*)/
```

痛苦 ...

======
后记
在 Redmine 中发现了更轻量的 textile 编辑器，经过简单修改，例子。如果不是编辑很复杂的文件，更推荐使用这个。

Textile WYSIWYG editor

Working on Textile & WYSIWYG editor these days.

What I want is a WYSIWYG editor for content team inputting learning content.
The content need styles like bold, cold, list and so on. I do not want to save HTML direct into database(which is also very difficult to edit). So Textile seems a good choose. Sanskrit can meet most commands, except list, colors, link, image and word escape. Which are what I need to work on.

Here is the demo. Check here for more details. I will also use the editor on this blog:)

I would like to share some javascript regular expression code. After writing these code, what I felt is how lucky I am working with Ruby On Rails.

```
textilize: function(html, escape){
        html = html.replace(/<br ?\/?>/gi, "\n");
        html = html.replace(/<(?:b|strong)>((.|[\r\n])*?)<\/(?:b|strong)>/gi
        html = html.replace(/<(?:i|em)>((.|[\r\n])*?)<\/(?:i|em)>/gi, '_$1_'
        html = html.replace(/<(?:u|ins)>((.|[\r\n])*?)<\/(?:u|ins)>/gi, '+$1
        html = html.replace(/<a href="(.*?)">((.|[\r\n])*?)<\/a>/gi, '"$2":$
        html = html.replace(/<font color="#0000ff".*?>((.|[\r\n])*?)<\/font>
        html = html.replace(/<font color="#FF0000".*?>((.|[\r\n])*?)<\/font>
        html = html.replace(/<font color="#088A4B".*?>((.|[\r\n])*?)<\/font>
        //list
        var r = /([\s\S]*)<ul>([\s\S]*?)<\/ul>([\s\S]*)/;
        while (r.test(html))
            {
                p = RegExp.$1;
                c = RegExp.$2;
                e = RegExp.$3;
                c = c.gsub(/<li>([\s\S]*?)</li>/, "* #{1}\n");
                html = p + "\n" + c + e;
            }
        html = html.replace(/^[\r\n]+|[\r\n]+$/g, '');
        return html;
    },
```

```
htmlize: function(textile, escape){
        //escape
        textile = textile.replace(/\/gi, "0142540974231612");
        textile = textile.replace(/\\_/gi, "0750023893692338");
        textile = textile.replace(/\\*/gi, "0643004879835027");

        textile = textile.replace(/\n/gi, '<br/>');
        textile = textile.replace(/\*(.+?)\*/gi, (this.internetExplorer ? '<
        textile = textile.replace(/_(.+?)_/gi, (this.internetExplorer ? '<em
        textile = textile.replace(/(.+?)/gi, '<u>$1</u>');
        textile = textile.replace(/"(.+?)":([^\s\n<]+)/gi, '<a href="$2">$1<
        textile = textile.replace(/%{color:blue}(.+?)%/gi, '<font color="#00
        textile = textile.replace(/%{color:red}(.+?)%/gi, '<font color="#FF0
        textile = textile.replace(/%{color:green}(.+?)%/gi, '<font color="#0
        textile = textile+'<br\/>';
        textile = textile.replace(/<p>[\s\n]*<\/p>/g, '');
        //unescape
        textile = textile.replace(/0142540974231612/, "\");
```

```
        textile = textile.replace(/0750023893692338/, "\\_");
        textile = textile.replace(/0643004879835027/, "\\*");
        //list
        var r = /(<br\/>\* .*?<br(\/)*>)(?!\* )/;
        while (r.test("<br/>" + textile))
            {
                l = RegExp.leftContext;
                m = RegExp.lastMatch;
                e = RegExp.rightContext;
                m = m.gsub(/\* (.*?)<br(\/)*>/, "<li>#{1}</li>").gsub("<br/>
                textile = l + "<ul>" + m + "</ul>" + e;
            }
        return textile;
    },
```

Importent:
"JavaScript and VBScript do not have an option to make the dot match line break characters. In those languages, you can use a character class such as [\s\S] to match any character."
see here for more details.

So when in Ruby we use:

```
/(.*)<ul>(.*?)<\/ul>(.*)/
```

In Javascript we need code like:

```
/([\s\S]*)<ul>([\s\S]*?)<\/ul>([\s\S]*)/
```

What a pain ...

======
Afterward
later on, I found a more light weight textile editor from Redmine. After simple modification, here is the demo. I would like to recommend this one unless editing complex format articles.

rails 代码片段记录

1.简短的正则表达式写法

```
"I am strong..."[/(.*?)(\.+)/, 1]
```

2.简短的rescue写法($!)

```
a.downcase rescue puts $!
```

3.动态调用方法

```
send("puts", "I am strong")
```

4.动态定义方法

```
define_method "simple?" do |number|
  number < 3
end
```

5.在rails中使用sql（这里用于改变单表继承的类型）

```
Page.connection.execute("update pages set type='PreparationPage'")
```

6.在rails中使用避免回调

```
PagePart.update_all("page_id = NULL", {:id => part.id})
```

7.给任务传递参数

```
desc 'For test params'
namespace :gary do
  task :test => :environment do
    puts "I am strong, I am #{ENV['who']}"
  end
end
```

使用方法

```
rake gary:test who="gary"
```

8.用正则表达式判断中文

```
puts "Chinese" if "中文" =~ /[\xa0-\xff]/
```

9.数组和哈希的相互转换

```
fruit_array = ["apple","red","banana","yellow"]
fruit_hash = Hash[*fruit_array]
#fruit_hash  =>  {"apple"=>"red", "banana"=>"yellow"}
fruit_array = [*fruit_hash].flatten
#fruit_array  =>  ["apple","red","banana","yellow"]
```

rails code snippet

1.simple regular expression

```
"I am strong..."[/(.*?)(\.+)/, 1]
```

2.simple rescue (with $!)

```
a.downcase rescue puts $!
```

3.call method dynamically

```
send("puts", "I am strong")
```

4.define method dynamically

```
define_method "simple?" do |number|
  number < 3
end
```

5.use SQL in rails(here change 'type' of single table inherit)

```
Page.connection.execute("update pages set type='PreparationPage'")
```

6.ignore call back rails

```
PagePart.update_all("page_id = NULL", {:id => part.id})
```

7.send params to task

```
desc 'For test params'
namespace :gary do
  task :test => :environment do
    puts "I am strong, I am #{ENV['who']}"
  end
end
```

use

```
rake gary:test who="gary"
```

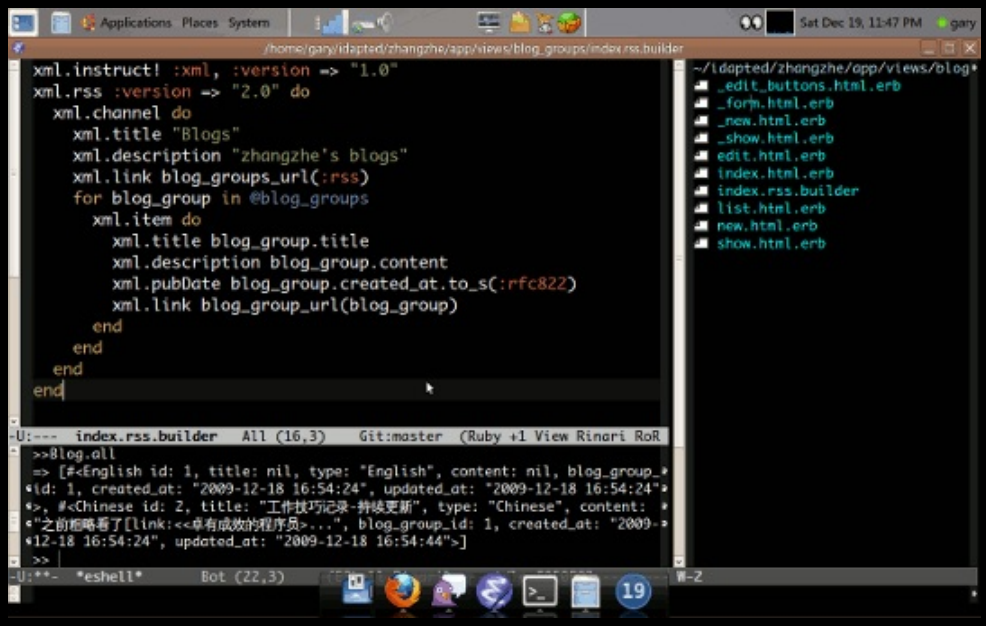
8.use regular expression to find Chinese char

```
puts "Chinese" if "中文" =~ /[\xa0-\xff]/
```

9.switch between array and hash

```
fruit_array = ["apple","red","banana","yellow"]
fruit_hash = Hash[*fruit_array]
#fruit_hash  =>  {"apple"=>"red", "banana"=>"yellow"}
fruit_array = [*fruit_hash].flatten
#fruit_array  =>  ["apple","red","banana","yellow"]
```

我的 emacs 配置



学习ruby on rails 两年了，期间使用过Eclipse和Netbeans，半年以前开始使用emacs。

经过刚开始的不适，半年来不断到处拷代码，改配置。最近感觉逐渐使得顺手了。越来越喜欢上这个高可配置的编辑器。

这里把快捷键和配置贴出来留下一个记录，也希望能对其他人有启发或者帮助：）

对源码和配置感兴趣可以访问这里。

常用快捷键

```
C-v                     Scroll down (toward end of buffer)
M-v                     Scroll up (toward beginning of buffer)
C-u #                   Prefix numeric arg # to next cmd
C-g                     Stop a command in progress
C-x C-c                 Exit emacs
C-a                     move-beginning-of-line
C-e                     move-end-of-line
C-l                     recenter-top-bottom
C-n                     next-line
C-r                     isearch-backward
C-s                     isearch-forward
C-up                    backward-paragraph
C-down                  next-paragraph
C-x C-f                 find-file
C-x C-w                 write-file
C-x 1                   delete-other-windows
C-x 2                   split-window-vertically
C-x 3                   split-window-horizontally
C-x s                   save-some-buffers
M-<                     beginning-of-buffer
M-=                     count-lines-region
M->                     end-of-buffer
<f1> k                  Info-goto-emacs-key-command-node
```

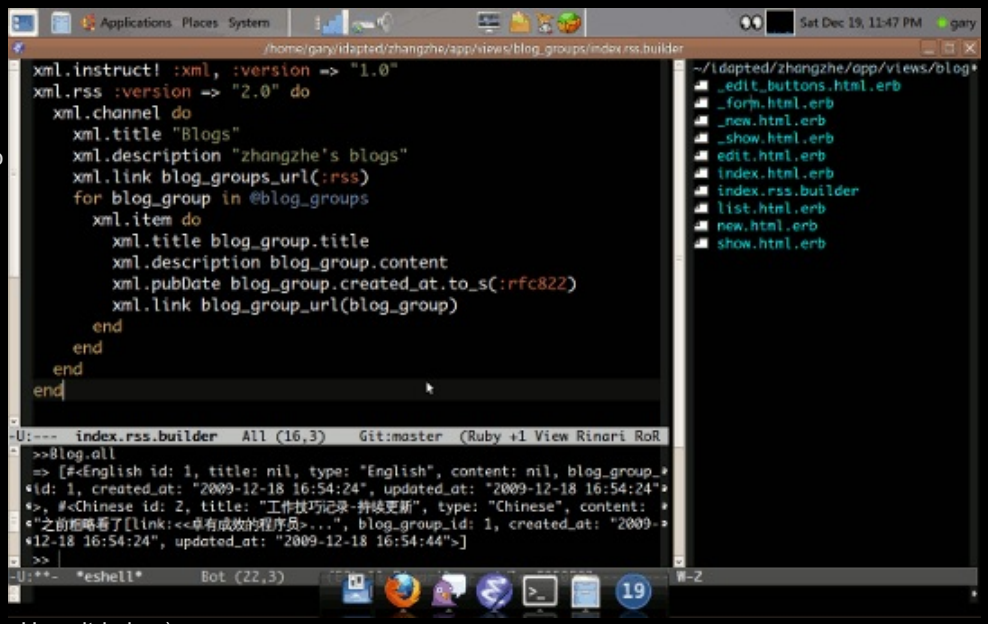
```
<f1> w                  where-is
```

自定义常用快捷键

```
C-c C-c C-s             tags-search
C-c C-c C-n             tags-loop-continue
C-c C-c C-c             inari-find-controller
C-c C-c C-m             rinari-find-model
C-c C-c C-v             rinari-find-view
C-c C-c C-r             rinari-find-rspec
C-c C-c C-f             rinari-find-file-in-project
C-c C-c C-h             hide-all-blocks
C-c C-c C-t             toggle-hiding-block
C-c C-c C-o             comment-region
C-c C-c C-u             uncomment-region
C-c C-c C-g             goto-line
<f5>                    smart-compile
<f8>                    ecb-toggle-ecb-windows
<f12>                   kill-this-buffer
C-<f12>                 kill-all-rinari-buffers
```

my emacs configuration

Learning ruby on rails for more than 2 year , used to use Eclipse and later Netbeans. 1 years ago switch to Emacs.



At first it was a big pain struggling to learn and config it. Then I fall in love with Emacs.

Here record my configuration. Hope it helps:)

If interesting, please visit here for more information.

cheat sheet

```
C-v                     Scroll down (toward end of buffer)
M-v                     Scroll up (toward beginning of buffer)
C-u #                   Prefix numeric arg # to next cmd
C-g                     Stop a command in progress
C-x C-c                 Exit emacs
C-a                     move-beginning-of-line
C-e                     move-end-of-line
C-l                     recenter-top-bottom
C-n                     next-line
C-r                     isearch-backward
C-s                     isearch-forward
C-up                    backward-paragraph
C-down                  next-paragraph
C-x C-f                 find-file
C-x C-w                 write-file
C-x 1                   delete-other-windows
C-x 2                   split-window-vertically
C-x 3                   split-window-horizontally
```

```
C-x s                   save-some-buffers
M-<                     beginning-of-buffer
M-=                     count-lines-region
M->                     end-of-buffer
<f1> k                  Info-goto-emacs-key-command-node
<f1> w                  where-is
```

self config cheat sheet

```
C-c C-c C-s             tags-search
C-c C-c C-n             tags-loop-continue
C-c C-c C-c             inari-find-controller
C-c C-c C-m             rinari-find-model
C-c C-c C-v             rinari-find-view
C-c C-c C-r             rinari-find-rspec
C-c C-c C-f             rinari-find-file-in-project
C-c C-c C-h             hide-all-blocks
C-c C-c C-t             toggle-hiding-block
C-c C-c C-o             comment-region
C-c C-c C-u             uncomment-region
C-c C-c C-g             goto-line
<f5>                    smart-compile
<f8>                    ecb-toggle-ecb-windows
<f12>                   kill-this-buffer
C-<f12>                 kill-all-rinari-buffers
```

工作技巧记录-持续更新

之前粗略看过<<卓有成效的程序员>>，理解能力有限，但也觉得受益良多。
今天读到<<追求神乎其技的程式設計之道（十）>>:程式設計師的生產力之謎。很认同作者对生产力的差异原因的理解。

会在这篇分享记录工作中的点滴技巧。持续更新。

1.创建ubuntu短命令

修改 ~/.bashrc
加入

```
alias s='script/server'
alias c='script/console'
alias gz='cd /home/gary/idapted/zhangzhe/'
```

重启命令行。之后只需输入 **gz** 即可进入/home/gary/idapted/zhangzhe/目录， 在rails目录下，输入 **s** 即可打开server, **c** 即可打开console。

2.使用快捷键

ubuntu 快捷键
使用 Compiz Config Settings Manager根据习惯设定。
我设定为

```
Alt + f       打开 Firefox
Alt + e      打开 Emacs
Alt + p      打开 Pidgin
Alt + g      打开 Geditor
```

Firefox 快捷键

```
ctrl + shift + t      打开最后关闭的tab
ctrl + k                进入google搜索输入焦点
ctrl + w               关闭当前tab
ctrl + t                打开一个新tab
ctrl + l                进入地址栏
ctrl + tab            循环进入前tab
ctrl + shift + tab  循环进入后tab
ctrl + f                查找
```

emacs 快捷键
Emacs 是快捷键设置的典范，配置方法见 我的 emacs 配置 。

3.用脚本简短命令输入

举例说明，我使用 git 来进行版本控制，部署服务器在 heroku， 每次叠代完毕需要输入命令:

```
git add .
git commit . -m "some info"
git push heroku master
```

以提交代码并部署。

以上命令可以通过简单 ruby 脚本简化输入。
在项目根目录创建文件 cmh，写入 ruby 脚本

```
#!/usr/bin/env ruby
def exe_cmd(cmd)
  puts cmd
  system cmd
end

cmd = "git add ."
exe_cmd cmd

args = ARGV.join(' ')
cmd = "git commit . -m '#{ARGV[0]}'"
exe_cmd cmd

cmd = "git push heroku master"
exe_cmd cmd
```

现在可以通过在命令行输入 ./cmh "some info" 来提交并部署程序。

另一个例子, 备份 heroku 程序数据库。在项目根目录创建文件 bf，写入 ruby 脚本

```
#!/usr/bin/env ruby
def exe_cmd(cmd)
  puts cmd
  system cmd
end

cmd = "heroku db:pull mysql://user@localhost/zhangzhe_bf?encoding=utf8"
exe_cmd cmd
```

现在可以通过在命令行输入 ./bf 以备份数据库到 zhangzhe_bf 表。

4.加强版的 rails console

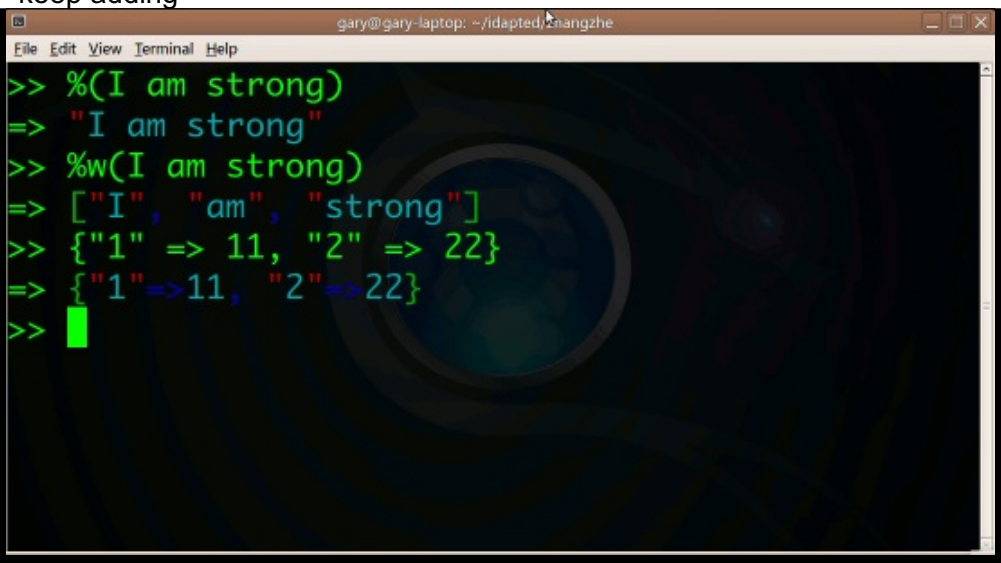
使用 Wirble，可以为 rails 命令行增加代码高亮和代码记忆功能。
安装

```
sudo gem install wirble
```

然后在 ~/.irbrc 中加入

```
require 'rubygems'
require 'wirble'
Wirble.init
Wirble.colorize
```

代码高亮效果图：



这里 有更详细的说明。

work tricks -keep adding

Read << The productive programmer >> before , though can not understand a lot , I think it was useful to me.
Today read <<追求神乎其技的程式設計之道（十）>>:程式設計師的生產力之謎. Identify with author's opinin about productivity.
And this blog is my dog food:)

Record tricks from work, hope helps:)

1.create ubuntu shorter command

edit ~/.bashrc
add

```
alias s='script/server'
alias c='script/console'
alias gz='cd /home/gary/idapted/zhangzhe/'
```

restart terminal.
Now input **gz** goes to /home/gary/idapted/zhangzhe/ folder ,
under rails project folder , input **s** opens server, **c** open console.

2.Use shortcut

ubuntu
Use Compiz Config Settings Manager config with habit.
My config:

```
Alt + f       open Firefox
Alt + e      open Emacs
Alt + p      open Pidgin
Alt + g      open Geditor
```

Firefox

```
ctrl + shift + t      open last closed tab
ctrl + k                into google search
ctrl + w               close current tab
ctrl + t                open a new tab
ctrl + l                into address field
ctrl + tab            loop go to next tab
ctrl + shift + tab   loop go to pre tab
ctrl + f                search
```

emacs
Emacs is the model of shortcuts, check my emacs configuration for more details.

3.Use script to shorter command

For example, I use git do version control , deploy app to heroku. So after every iteration, I need to input into term:

```
git add .
git commit . -m "some info"
git push heroku master
```

to commit and deploy.
All the command above can be simple by ruby script.

I create a file cmh in app root, with ruby script

```
#!/usr/bin/env ruby
def exe_cmd(cmd)
  puts cmd
  system cmd
end

cmd = "git add ."
exe_cmd cmd

args = ARGV.join(' ')
cmd = "git commit . -m '#{ARGV[0]}'"
exe_cmd cmd

cmd = "git push heroku master"
exe_cmd cmd
```

so I can call ./cmh "some info" in term for short to deploy.

Another example, create a file calls bf in app root, with ruby script

```
#!/usr/bin/env ruby
def exe_cmd(cmd)
  puts cmd
  system cmd
end

cmd = "heroku db:pull mysql://user@localhost/zhangzhe_bf?encoding=utf8"
exe_cmd cmd
```

so I can call ./bf in term for short to backup.

4.improve ruby console

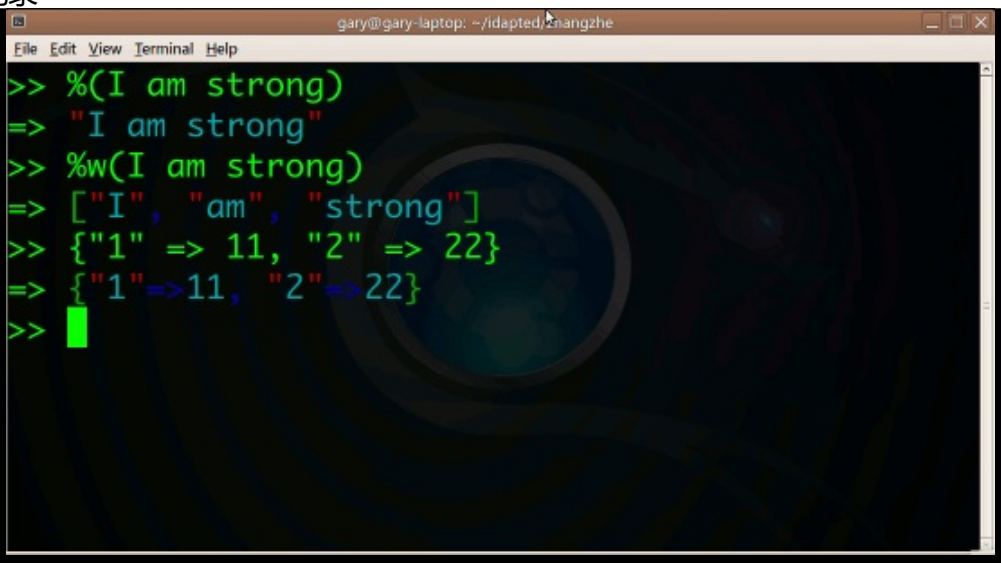
Wirble, a gem that improve your irb console with syntax coloring of output and input history.
install

```
sudo gem install wirble
```

and put the following into ~/.irbrc

```
require 'rubygems'
require 'wirble'
Wirble.init
Wirble.colorize
```

syntax coloring console:



check here for more details.

电脑升级记录

去年十一买的**R61e**如今已经不堪重负，终于升级。
之前已经升级了内存到3G(实际买的两根4G内存，无奈装的32位系统支持不好，换成了3G)。
现在开发时切换程序经常需要较长的等待时间，有时系统甚至暂时没有反应。由于工作的娱乐都使用这一台电脑，原配的120G硬盘常常捉襟见肘了。
因此想换一块好一点的处理器，并加一块大硬盘，升级下来的硬盘做移动硬盘继续使用。

处理器看好**t7300**（淘宝）,性能不俗，而且完美支持小黑的**GM965**主板。硬盘看好日立的**7K320**，7200转（京东）。硬盘盒看好元谷星钻**iPD-USB**（京东）。
提交订单，到货，安装。一切顺利：）

但安装系统和软件过程出乎意料的麻烦，以为曾经装过的会很顺利，没想到竟耽误了近10个小时。
下面是安装过程以及一些心得。

ubuntu

因为着急用，手头又没有9.10的安装盘。就想安装了8.10之后升级上去。后来缓慢的升级过程证明这是一个非常笨的错误。
发现Ubunut Tweak很好用阿：）

ruby on rails

或许因为是逐渐升级的缘故，安装过程中遇见很多困难，解决时使用一些命令和新的：

```
apt-cache search ruby
dpkg -L ruby：检查装了什么
dpkg -S：查依赖关系
which ruby：调用ruby的命令地址
ls -L /usr/bin/ruby：显示符号链接
gem 装在/usr/lib/ruby/gems/1.8
gem list
/etc/hosts：本机域名配置地址
```

gem的安装让我崩溃，最后怒把老硬盘里gems原封不动考了过来。

重要心得：都在变化，所以，缺什么，装什么..

github和heroku

升级完之后发现github和heroku账户都无法commit了。
解决方法如下，
1.生成key
ssh-keygen -d

2.加入key
heroku
运行heroku keys:add
输入邮箱和密码，heroku解决。

github
拷贝/home/gary/.ssh/id_dsa.pub中内容到github账户“Deploy Keys”中,github解决。

升级后系统性能的提升让我非常满意。
升级后的首次亮相：）

Computer update

My **R61e** bought last National Day is kind of slow. I am trying update it.
The memory is already update to 3 G， so I focus on CPU and hard disk.

CPU I like **t7300**（from taobao）,high speed and compatible **GM965** main board; hard disk I like Hitachi **7K320**（from jingdong）; hard disk cartridge I like 元谷星钻**iPD-USB**（form jingdong）.
Placing order ，delivered，install. So far so good：）

The trouble came when install system and soft, it cost me about 10 hours.
Below are some records.

ubuntu

Was Kind of in a hurry, and do not have ubuntu 9.10 live cd. So I desided to install 8.10 and upgrade later, which later prove very stupid.
Ubunut Tweak is so good：）

ruby on rails

Maybe because the upgrade thing, the installation met lots of problems.
Below are commands help me solved the problem.

```
apt-cache search ruby       search app
dpkg -L ruby                check what was installed
dpkg -S ruby                check dependency relationship
which ruby                  the location to use ruby
ls -L /usr/bin/ruby         show symbolic link
gem list                    show gems
/etc/hosts                  local domain config
```

The installation of gem crashed me. At last I copied all of them from old system(gems installed in /usr/lib/ruby/gems/1.8).

impotent: all will change, so meet problem, solve it..

github and heroku

After all upgrade and installations, github and heroku account cannot commit anymore。
here are how to solve:
1.generate key
ssh-keygen -d

2.add key
heroku
in Ternimal

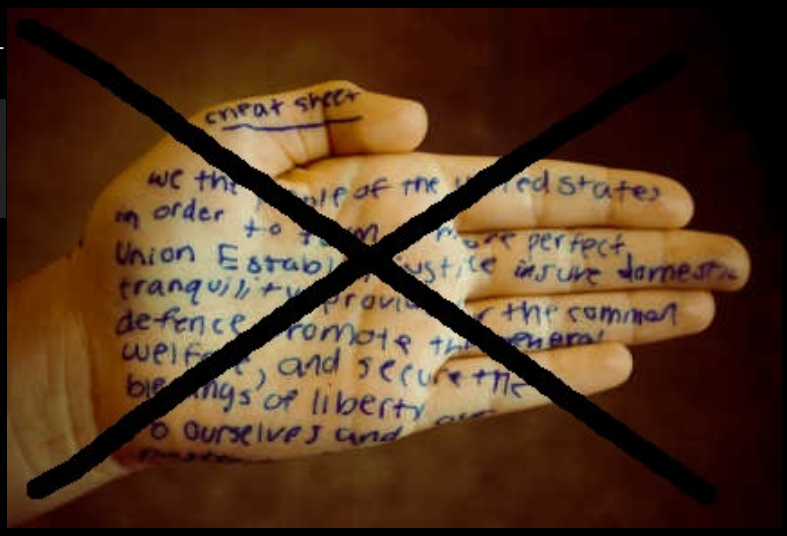
```
heroku keys:add
```

input mail and password, solved.

github
copy /home/gary/.ssh/id_dsa.pub content into github **Account Settings**: **SSh public Keys**, solved.

I am pleased with the performance after the upgrade. first show up:)

不再需要"小纸条"



工作以来一直用小纸条来记忆一些容易遗忘的命令。

```
RAILS sheet
MYSQL sheet
PROTOTYPE sheet
GIRL sheet...
```

以后这些小黄条可以都撕了!!!

因为cheat.

安装
sudo gem install cheat

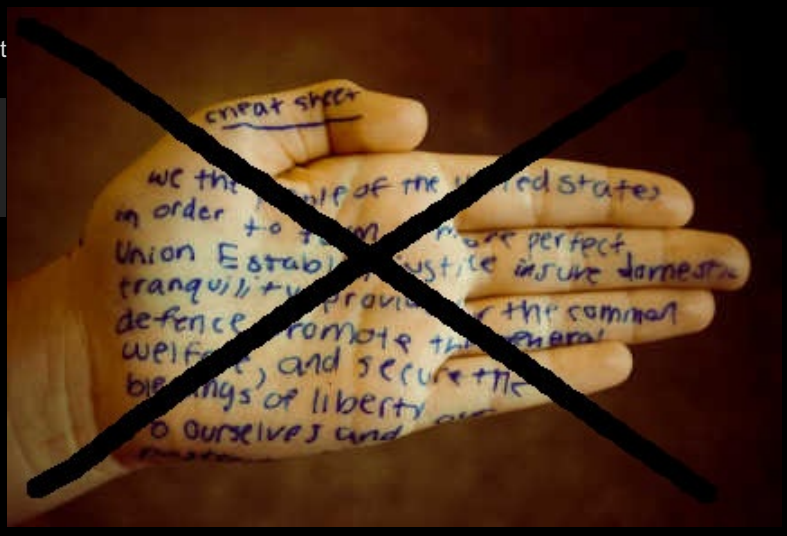
使用
cheat <小纸条名称>

列出已有的所有小纸条
cheat sheets

帮助
cheat cheat

例子
cheat mysql
cheat svn
cheat ruby
cheat console

no cheat sheet anymore



All the times I used cheat sheet to struggle my bad memory.

```
RAILS sheet
MYSQL sheet
PROTOTYPE sheet
GIRL sheet...
```

No need anymore!!!

cheat Comes.

install
sudo gem install cheat

use
cheat

check all

```
cheat sheets

help
cheat cheat

examples
cheat mysql
cheat svn
cheat ruby
cheat console
```