Taskium 系统设计说明书

非常 4+1

2012.06.09

小组成员：

招蕴豪(09388327)
何金城(09388299)
吴 垚(09388302)
陈跃群(09388325)
刘文辉(09388294)

目录：

1 简介
2 系统架构设计
    2.1 技术选择
    2.2 系统架构
3 系统模块设计
    3.1 用户管理
        3.1.1 权限控制
        3.1.2 注册登录
        3.1.3 组权限
    3.2 文章管理
        3.2.1 文章
        3.2.2 标签与类别
        3.2.3 文章权限
    3.3 作业管理
        3.3.1 作业定位
        3.3.2 作业
        3.3.3 作业查看
        3.3.4 任务
        3.3.5 组内评分
        3.3.6 最终评分
4 系统界面设计
    4.1 评论
    4.2 在线查看源代码
    4.3 动态链接
    4.4 评分
5 系统开发环境
    5.1 基本开发环境
    5.2 Gems

1 简介

Taskium 主要是一个针对于 Web2.0 课程的作业提交及评审系统。老师可以通过该系统进行课件以及作业的发布，对学生上交的作业进行评审，给分并且发布评分结果。学生则可以通过该系统查看老师所使用的课件，知道需要完成的作业，并可以对完成的作业进行提交。并且学生还可以在允许的时间范围内进行自由组队，同组的队员还可以互相查看作业，并对作业进行小组互评。系统本身则可以自动部署学生上交的作业，并提供在线源代码查看以及作业打包下载等功能。

2 系统架构设计

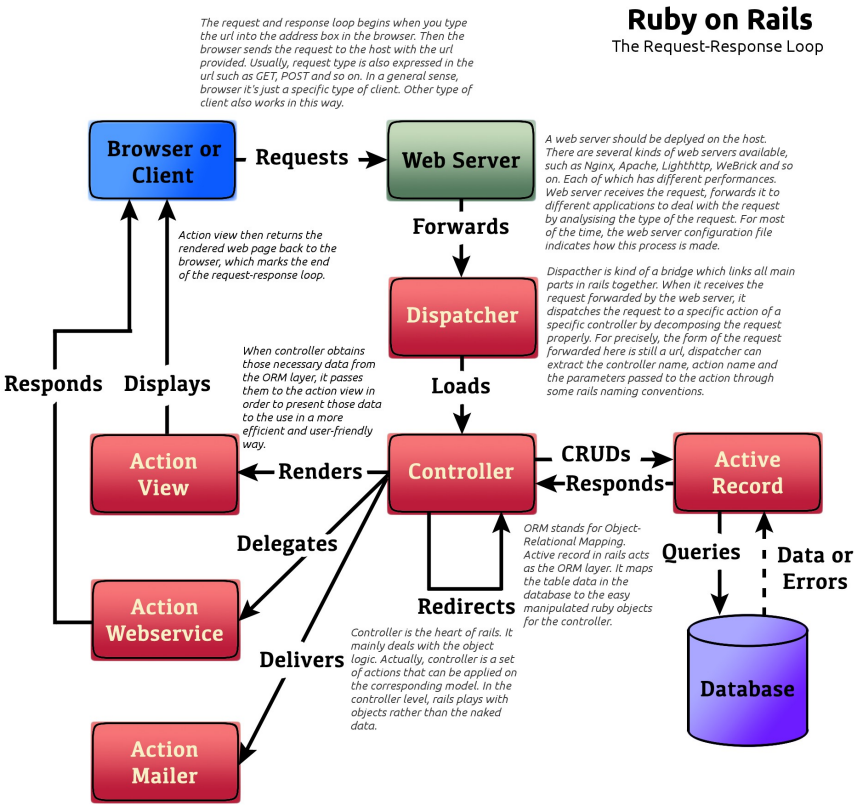
2.1 技术选择

我们在系统设计之初就对技术作出了选择。因为 Taskium 是一个网络应用，有众多的技术都可以对其进行实现，但是不同的技术会导致不同的系统设计方式，并且采取不同的技术也会对开发时间以及难度有很大的影响，所以我们感觉技术的选择对后续的设计过程影响非常的大，于是就先对技术作出了选择。

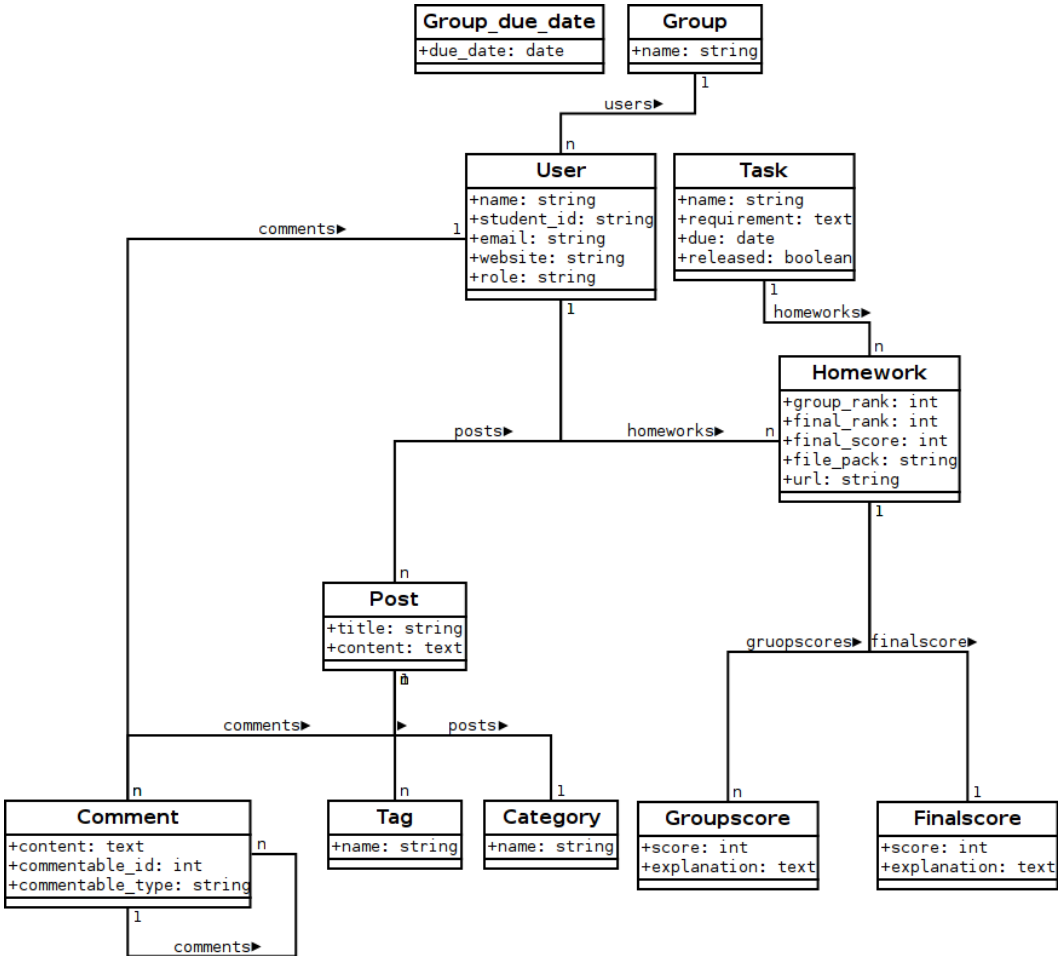
众观现在多种多样的 Web 开发技术，我们毫无犹豫地选择了适合于 Agile 开发的 Ruby on Rails。事实也证明，我们在后续的开发过程中因为使用了 Ruby on Rails 而感到非常的愉悦。后台使用 ruby 语言来进行开发无疑是最令人愉悦的事情，Rails 本身还原生支持 coffeescript，这就使得前端的开发也变得无比的便捷。同时，gems 提供的 haml 与 sass 模板语言更是使得在对网页结构及样式的设计中连繁琐的分号与括号也不复存在了。当然，在后面还会详细地列出我们所使用的技术及其版本。

2.2 系统架构

既然这是基于 Ruby on Rails 的应用，那么系统的整体架构毫无疑问采用 MVC 模式。这样，系统的总体架构实际上也就是 Rails 的总体架构，如下图所示：



在 Rails 中，Model 实际上是一个个体的概念，而其对应的 Controller 则是对这些同类个体集合的操作。比如对于用户来说，Model 中的 User 类实际上是个体用户的概念，其具体的实例就是单独的用户个体，而 UsersController 则是对这些用户进行各种增删改查的操作。根据这样的概念以及需求分析中得到的 Domain Model，我们得到了如下的 Class Diagram：



这里的 Class Diagram 并没有详细列出各个类的方法，主要原因在于为了保持类图的整洁以及方便从一个宏观的角度来对类图进行分析，从下面的图可以看出，Taskium 系统可以大致分为三个模块：**用户管理**，**作业管理**以及**文章管理**。这里的文章管理并不是作业的要求，但是在 Taskium 中，作业是以文章的形式发布的，并且同学可以通过评论与老师交流，所以在 Taskium 中也实现了这个功能。

还有一部分的需求并不能体现在下面的类图中，那就是系统的职责了。这是有关作业上交， 作业部署，以及作业打包下载的内容，会在稍后的地方进行讨论。下面先按照上面所划分的模块对系统的实现进行讨论。

3 系统模块设计

根据上面给出的类图对模块所进行的划分，下面将对用户管理，文章管理，以及作业管理的设计方案进行阐述。

3.1 用户管理

3.1.1 权限控制

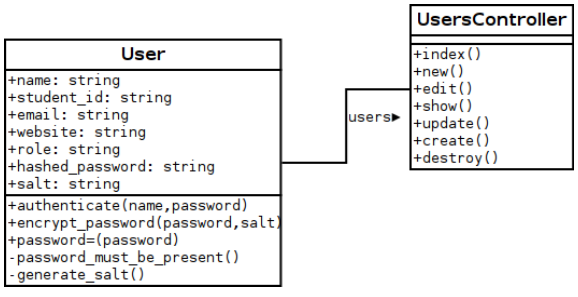
用户管理主要是处理用户权限控制，用户注册登录以及组权限等问题。在本系统里面，用户的角色可以分为几种：老师，学生，组长。之所以没有把 TA 分出来作为一种角色，完全是因为在需求里面，老师与 TA 的职责基本上是完全一样的，所以并没有必要拆分为两类。

再有，在作业要求里面提到：只有选定了本课程的学生、本课程的 TA 和老师才能够访问使用本系统，成为本系统的用户。我感觉这一点比较模糊。如果本系统是与教务系统进行绑定的话，Taskium 应该能从教务系统得到学生的选课结果，这样的话在学生注册系统的时候，就可以根据学生的学号姓名来判断学生是否选了该门课，有没有资格使用该系统。但是如果这个系统并不是与教务系统挂钩的话，那么教师的帐号应该是内定的。然后教室通过教务系统的选课结果来进行学生名单的导入，这样学生注册的时候就根据该名单进行判定。但我觉得两种理解都不是那么好实现，并且本次作业的宗旨也在于锻炼我们系统分析与设计的能力，所以我并没有对这样的功能进行实现。

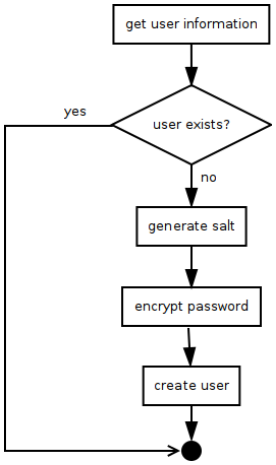
另外一个权限控制在于页面的访问上。例如学生与组长就无法访问所有用户的增删改查页面，而老师所能访问的页面中也没必要出现提交作业的按钮。这样的访问控制由于在很多地方都会被使用到，所以 Rails 自然也有 gem 来处理这样的情况，比如 cancan，但我并没有使用，而是自己去实现。

3.1.2 注册登录

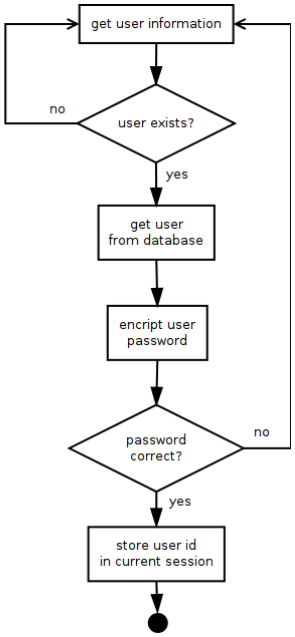
注册登录的主要任务在于用户的识别，当用户注册的时候，系统记录其学号以及姓名，其中学号是学生的标识，而其它的信息则不是必须的。系统进而根据该用户对象的运行时 id，随机盐值以及加密后的用户密码来生成一个跟用户一一对应的加密密码存放在数据库中。这样，当用户下次输入密码登录的时候，系统可以根据用户提供的密码，对应盐值以及对象 id 重新对密码进行加密，然后跟数据库对应用户的密码进行配对，如果配对成功则登录成功。对于注销，那仅仅是将用户的 id 从当前的 session 中删除即可，下面是 User 的类图：



为了更清晰地展示注册以及登录的流程，请参看下面的流程图：



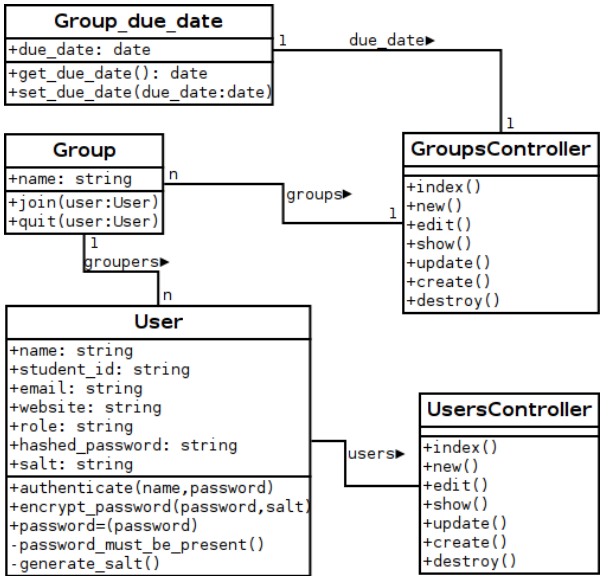
pic 3.1.2-1 register



pic 3.1.2-2 login

3.1.3 组权限

学生拥有部分的组的功能。在每个学期初，老师都会设定一个允许学生新建小组，自由加入退出小组的时间段。一旦这个时间段结束，学生就不可以随便加入和退出小组了。由此可以将组权限分为两个小部分，一个是老师的组权限时限控制，另一个是学生的组权限控制。其相关的类图如下：



由图可以看出，在 Group 里面有两个方法，分别是 join 和 quit，这提供了用户自由加入以及退出小组的功能。而组权限为学生开放的时间由一个单例对象 Group_due_date 来控制，这个类只有 getter 以及 setter，而并没有其它的添加，删除等方法，保证其唯一性。它与 GroupsController 是一一对应的关系，主要是因为有关于组的所有操

作都是根据这个时间来定的，如果该时间比当前时间要早，那么说明对于学生的组操作已经截止，应该被禁止；反之，则可以让学生们自由进行组操作。这个 Group_due_date 就相当于是一个控制变量。

3.2 文章管理

如上所述，这里的文章(post)其实有几个含义：

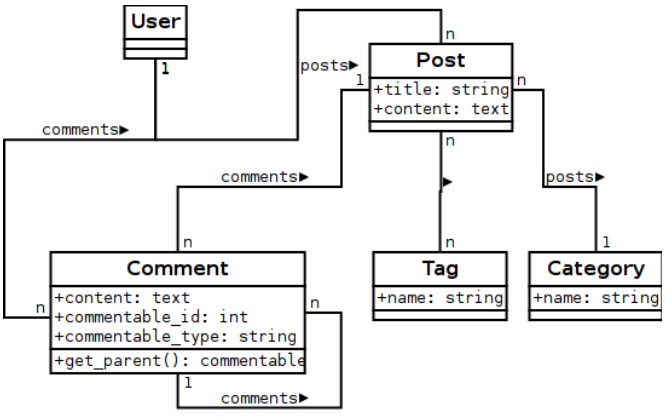
(1). 它可以是老师为了发布课堂笔记而撰写的文章，学生可以对其进行评论，提问，老师可以回复评论，从而达到师生互动的功能。
(2). 它也可以是学生们自己发表对课程的某些问题的看法，老师对其进行评核，其他学生对其进行评论。
(3). 它又可以是学生们对课堂所做的笔记，这只是普通纸质版笔记的电子形式，学生可以在组内共享这样的笔记，也可以选择将其在班上共享。

其功能还可以有很多，但是由于时间关系，这次的作业并不打算实现上述的所有功能，但此模型会予以实现，其中包括两个部分，分别是文章和评论。

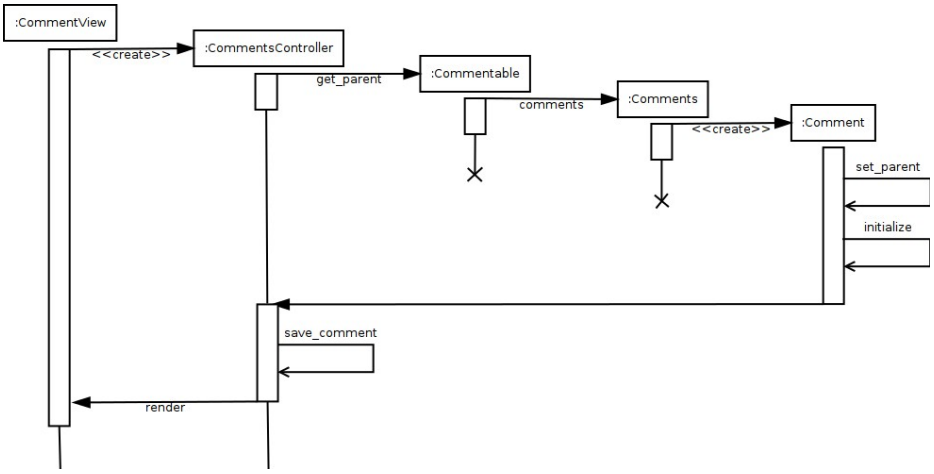
参考 wiki 的功能，发现评论可以嵌套，所以在本次作业中也打算实现这样的功能。本来评论完全可以托管给 disqus 来实现，这样既安全又省事，但是既然不是一个真正要投入使用的系统，我打算自己来实现这样的嵌套功能。

3.2.1 文章(Post)

文章最主要的部分只有几个，即标题，正文，作者，日期，有时候为了方便检索文章以及对文章进行归类，会为文章设置标签(tag)以及类别(category)，其实这两者并没有本质的区别，实际上对于一篇文章来说，它可以属于多个类别，但是为了突出两者的差异性，在本实现中会假定一篇文章只有一个类别，但可以有多个标签。其类图如下：



为了突出其逻辑性，Controller 就没有画在类图上了。为了实现评论的嵌套功能，评论自己关联到自己，并通过一个 commentable 对象的 id 来指明自己的 parent。顾名思义，commentable 就是可以评论的对象，它可以是 post，comment，或是后面说到的 task。如果评论对象是 post，那么其 commentable_id 就是该 post 的 id，如果其评论对象是评论，那么其 commentable_id 就是该评论的 id，这样实际上形成的是一个树状的评论结构，而树的根节点必然是一个 post，因为首先只能对 post 进行评论。评论的时序图如下：



3.2.2 标签与类别

标签与类别的概念都比较简单，其中标签与文章是多对多的关系，而类别与文章则是一对多的关系。这两者的设定都是为了方便检索文章。需要注意的一点是，在更新文章的时候，都需要对原来的标签进行清楚，这样保证更新以后文章的标签没有重复。

3.2.3 文章权限

如本部分开始所说的，文章的功能多种多样，但是由于本系统不打算实现所有的功能，所以就仅仅以最基本的权限设定来控制，具体的规则如下：

(1). 文章的作者有权对文章进行增删改查操作。
(2). 老师对所有的文章都有增删改查操作。
(3). 注册用户可以对任何文章进行评论。
(4). 游客可以浏览所有的文章，但是并不能进行任何的评论。

对于第一第二点，可以用过判断用户的 role 属性来进行控制，对于第三第四点则可以通过查看用户是否登录来实现。由于这些功能的入口都是视图中的某些按钮，所以以上的判断都是在视图模板中进行。

3.3 作业管理

作业概念比较复杂，这里先阐述一下我对作业的理解。作业实际上是完成一项“特殊”的工作，而这个工作的主要流程如下：

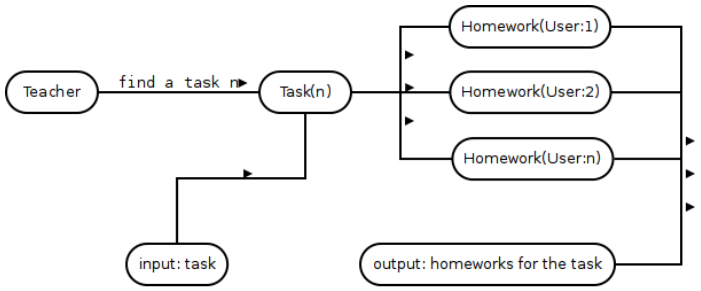
(1). 管理员发布工作。
(2). 学生完成工作。
(3). 学生提交工作。
(4). 管理员审核并评价工作。
(5). 学生获得工作的反馈。

以上提到的是一个粗略的流程，针对于 Taskium 系统来说，可以进一步把上述的过程进行扩展，并且替换掉某些术语，由此得到了下面的一个更为详细的流程：

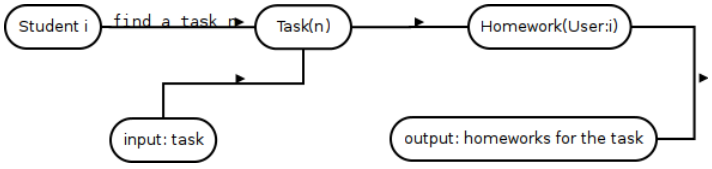
(1). 老师发布作业：
   i. 详细给出作业要求。
   ii. 清晰指明作业提交时间。
   iii. 为将要提交的作业开放一个空间。
(2). 学生完成作业：
   i. 学生阅读作业要求，如果有疑问，可以通过评论的方式与老师进行互动问答。
   ii. 学生按照自己的理解在本地完成作业。
(3). 学生提交作业：
   i. 学生通过一定的方式组织作业的文件结构，并对其进行打包。
   ii. 学生通过系统进行作业打包上传。
   iii. 作业上传后，学生可以用某种方式通过系统查看到自己上交的作业。
(4). 老师对作业进行批改并且给定分数：
   i. 老师可以通过系统以某种方式对作业进行查看，从而给出分数。
   ii. 当老师对所有作业都批改完毕以后，老师可以对作业评分进行发布。
(5). 学生获得作业的评分反馈：
   i. 只要老师对该作业的评分进行了发布，学生就可以查看到其分数。

由上面的讨论可以看出，“作业”实际上并不是一个实体，而是一个流程，一个比较复杂的流程，其中牵涉到多个角色的多个动作。当然，上面的流程并没有列出一些可能的额外路径，这是为了保持主流程的清晰。流程的复杂性主要体现在整个流程结束之后的那个状态，因为在那个状态对象是最多的。比如，在老师发布作业之前，根本就没有任务这个概念，在学生交作业之前，根本就没有作业这个概念，所以当一切都结束之后，所有概念都出现了，所以下面会用一些形象的图来指明在以上流程结束之际，老师，任务，作业与学生的关系。

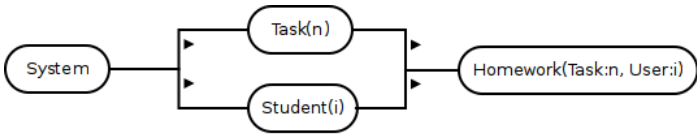
这张图表示了老师在批改作业的时候的需求。老师需要在众多任务中选择一个需要批改的任务，而在这个任务里面包含了学生们上交的所有作业，也就是说，老师的需求是：



在学生看来，他们并没有必要获取全部的作业，他们只关注于自己的作业而已，所以有以下的关系：



再有，对于系统来说，由于在老师发布作业的时候，系统曾经分配过空间来存放作业，并且留意到上面的关系输出都是作业，所以在以上请求发生的时候，系统必须能够定位到某一个学生的某一次作业，于是系统与任务，学生，作业的关系如下：



由以上的关系分析我们可以得到不少的信息，首先需要说的是作业的定位问题。

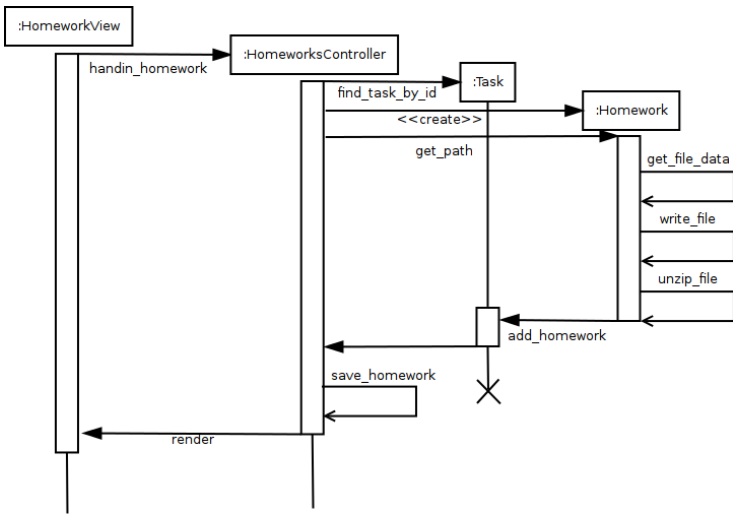
3.3.1 作业定位

根据以上分析，可以用一种简单的方法来对作业进行定位，那就是通过(Task, Student)这样的关系对来确定一个作业，既然如此，那么在服务器端的作业存放结构也可以直接使用这样的规则。比如现在的 web 根目录为 public，那么学号为 09388327 同学的第 9 次作业的存放路径就可以为：/public/9/09388327，而这个路径本身也已经可以作为该次作业部署的 url 路径的前缀了。

另外的一个严重的定位问题是，如何定位到该学生该次作业的主页呢？这里涉及到一个目录组织规则的问题。如果学生随便地组织目录结构，并且用随便的命名方式来对主页进行命名的话，那么无论查找算法多么精妙，也总会有找不到的时候，所以在 Taskium 中，作出了如下的假设：

(1). 主页的名称叫做 index.html
(2). 主页存放在上交.zip 包中最外层目录里面。

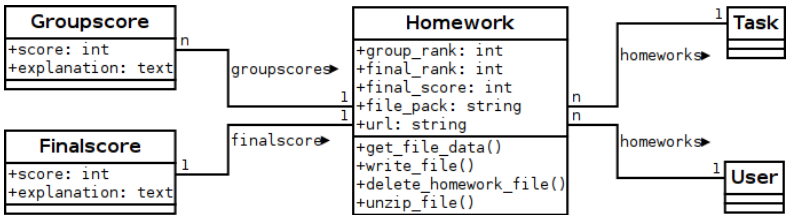
这样的话，用上面的例子来说，就是 09388327 同学的第 9 次作业的 url 就是/public/9/09388327/index.html。这样也比较符合常理。并且，对于打包下载的问题，由于学生本身上传作业的时候就是以打包上传的方式来提交的，所以该压缩包实际上并没有必要删掉，这样需要下载该作业的时候就可以直接下载该包了。有了这样的定位方式，交作业时要做的事情就明确多了，以下是交作业的时序图：



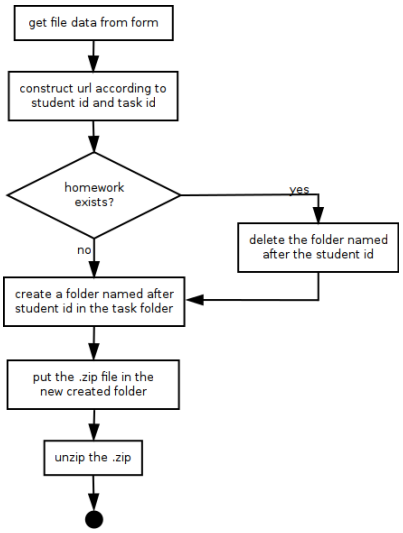
3.3.2 作业

上面提到，作业是靠学生以及任务来确定的，所以作业必须隶属于这两个类，即学生有多次作业(各次任务的作业)，一个任务也对应有多个作业(全班同学该次作业)，并且，由于作业需要评分，所以还需要两个辅助的类，一个是组分数，另外一个是最终分数。

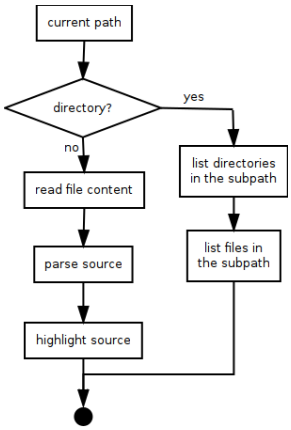
在要求里面说到，在作业上交截至之前，还需要附上小组的评审结果，而同组的成员会相互对对方的作业进行评分，一旦提交这样的小组互评结果，系统应该对评分作出一个排名，并记录到数据库中。这样等老师进行评分的时候，就可以从中看到小组互评的结果。以下是作业的类图：



可以留意到在 Homework 里面有一个 get_file_data 的方法，这个方法的作用在于从作业提交表单中拿到作业打包文件，然后先缓存起来。另外的 write_file 方法则是完成如下的操作：



pic 3.3.2-1



pic 3.3.3-1

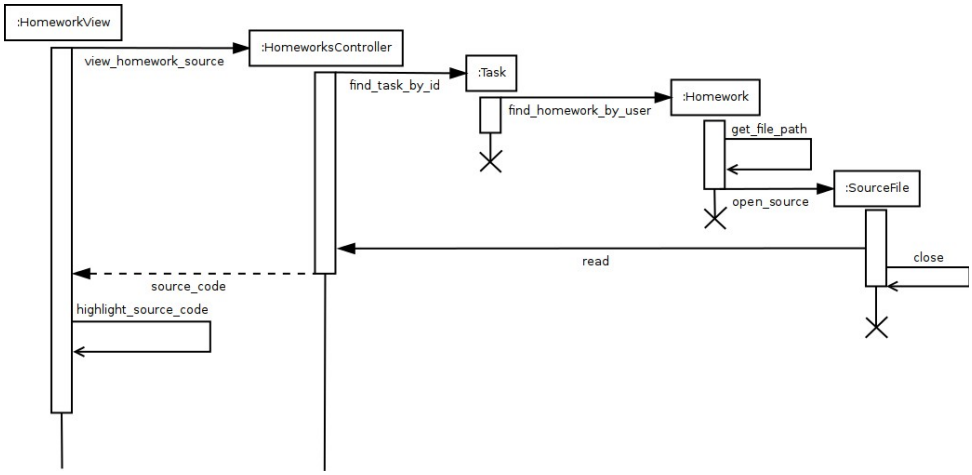
如果作业已经存在，那么说明当前提交的作业是更新的版本，所以就应该把原来的作业删掉，然后重新吧当前的作业作为最新的作业处理。

3.3.3 作业查看

根据上面的流程分析，学生应该可以在作业上传以后看到作业，虽然要求中并没有说，但是我感觉在线查看源代码是非常好的查看代码方式，所以在 Taskium 中实现了这个功能。

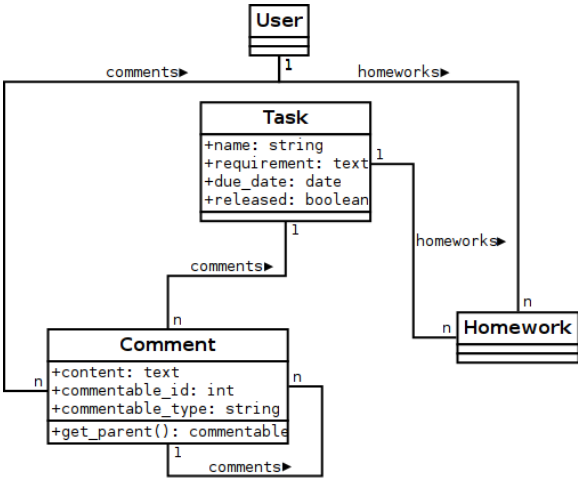
虽然网上也有很多在线文件目录树的实现，但是我还是打算自己实现一下。首先考虑到的就是，实际上每次列出的结构都只是文件结构中的某一层，而并不需要把整个目录树结构都列出来。所以，记录当前目录非常重要。再有，当用户点击的是文件，那么就应该把文件的内容(其实是源代码)读出来，然后对源代码进行语法分析，然后对分析结果进行适当的代码高亮。图 3.3.3-1 给出了这个过程的流程图。

从实现的层面上来说，这主要是使用到了 ajax。Ruby on Rails 对 ajax 支持很好。在查看源代码的页面主要分成两个主要的区域，分别为源码区域以及文件树区域。当用户请求的路径是文件的话，Rails 读取了该源码，经过转义以后刷新源码显示的区域，然后调用 highlight.js 来对代码进行语法分析以及高亮处理。如果用户请求的路径是目录的话，那么 Rails 则读取改目录的子目录结构，将新的该层目录返回并刷新文件目录树区域，从而达到了这样的效果。以下也给出了给出了查看作业源代码的时序图：



3.3.4 任务(Task)

任务只能由老师进行发布，老师对其有增删改查的权限，但学生只有对其评论的权限。从流程图可以看出，发布一次任务涉及到发布任务要求以及系统开放任务空间两个方面的工作。其实发布任务要求的过程跟发布文章的过程是类似的，下面给出任务的类图：



3.3.5 组内评分

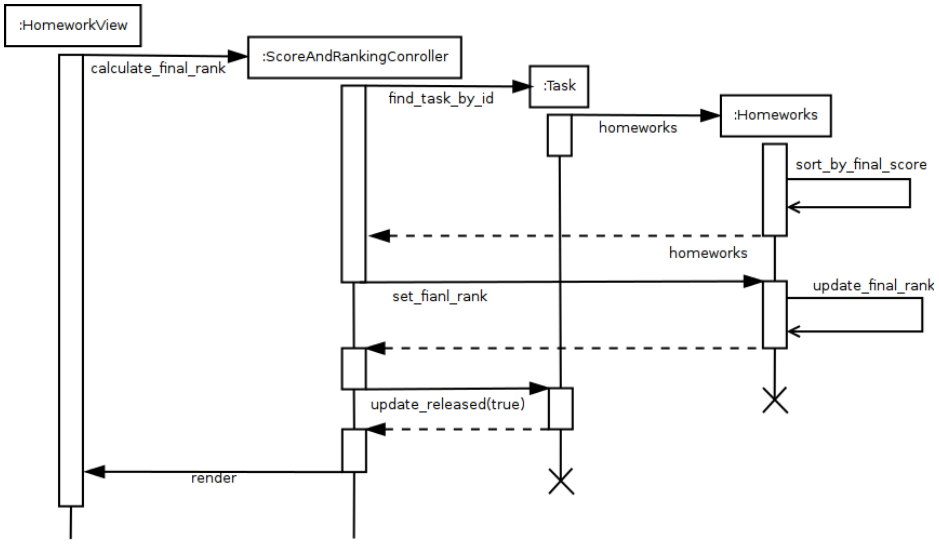
其实按照要求以及 web2.0 课程的小组评分方式，小组只要给出排名即可。但是我发现在实际操作当中，组员一般都不是同时评分，并且组长也不可能立刻给出一个确切的排名，而必定是首先让每个组员对其余的组员作业评分，然后对每个作业求组内平均分，然后根据平均分进行排名，最后再在作业提交截至之前把组内评审结果上交。而这个组内评审的提交是组长的权限。

3.3.6 最终评分

最终评分是老师的权限，但是老师也必须等到作业提交截至了以后才能够评分。

其次，在作业要求中有一条说到：老师可以发布评分结果。对此我深表疑虑。因为最终成绩是作业里面的一个成员，如果老师对该作业评分了，那么该作业的最终成绩变量将持有老师的评分结果，那么该同学就立刻知道了自己的分数(如果他刚好在线)。也就是说，不同学生在不同的时候知道自己的成绩，这样就不存在发布这样的一个动作了。为了实现这个动作，我在 Task 里面加入了一个名为 released 的变量，这个变量标志着该任务是否被发布成绩，如果没有发布成绩，那么所有学生都看不到该次作业的成绩，反之，则可以看到。并且设定这样的一个动作很有好处，当老师发布成绩的时候，就保证了提交作业的学生都有成绩了，这样就可以对这些成绩进行排序，从而选出优秀的作业。

下面是老师发布成绩的时序图：



4 系统界面设计

自从 twitter 的 bootstrap 出来以后，大家纷纷投入使用，导致 twitter 风格遍布大街，所以我写 web 应用一直都没有使用 bootstrap，而是自己写 css，这样虽然写出来的样式在不同浏览器之间可能会有点差异，但是总体风格我觉得还是很简朴清新的。

虽然 Taskium 所实现的功能都是很普通的，但是我对这些普通的功能的界面设计下了一点功夫，下面将分点来阐述。

4.1 评论

对评论的界面我做了两点的改进。其一就是评论的可收缩性。网上很多评论都是不可折叠的，这使得多 Post 的页面很冗长。在 Taskium 的查看 Post 页面中，文章都是以缩略的形式并排列出，在文章的右下角列出了评论的数目，但是并没有把评论展开，这样无疑是大大节省了页面空间，也可以把用户的注意力集中在 Post 上面。如果用户发现某一篇文章的评论数特别多并对其感兴趣，它可以点击评论展开的按钮，这样评论就展开了。如果他阅读完毕，希望继续阅读下面的 Post，他可以把评论收缩回去。

再有，由于 Taskium 的评论是递归定义的，所以可以对评论进行评论。在界面设计上面，每个评论的下方都对应了一个回复评论的按钮。点击这个按钮之后，随即在该评论的下方出现评论表单。当用户希望对另外的评论进行回复时，他可以点击该评论的回复按钮，这样当前展开的评论表单又会收缩回去，这样保证了不会让用户展开多于一个的评论表单。

最后，整个评论系统都是基于 ajax 的互动，所以用户基本上看不到整个页面的刷新，这是很多网站都没有实现的。在其他很多网站中，点击添加评论以后都是跳转到新的页面，这样无疑是大大影响用户体验的。

4.2 在线查看源代码

作业要求中提到，在小组评审的时候，同组的可以相互下载源代码方便评审，在老师评审的时候，老师也可以把源代码都下载以方便评审。**其实我觉得，把那些源码都下载下来评审是一点都不方便的。**自从使用 github 以来，我非常喜欢在线查看源代码的方式，这样既方便，又漂亮。但是 github 的源代码查看有个缺点，那就是切换源文件以及目录很不方便，必须后退一层，然后再选择新的文件或者目录。所以在 Taskium 中，我进行了扬弃，就是一方面实现了在线查看源代码的功能，并且在源代码查看区域的旁边添加了一个固定的文件目录树，这样无论源代码有多么长，用户随时都可以通过目录树来跳转到别的源文件。

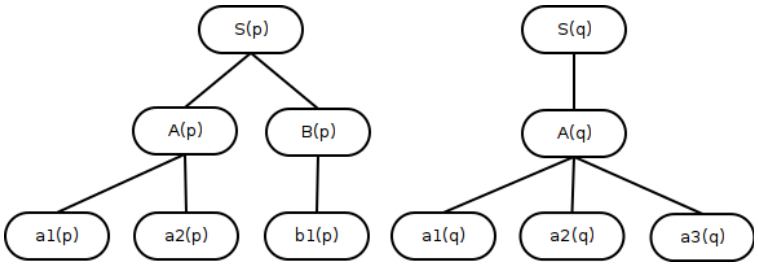
在代码高亮方面，我使用的是 highlight.js 库，这是在客户端渲染的，这样的好处是，在网速不佳的情况下，用户依然可以在页面渲染结束以后看到源代码，而高亮只是随后看到而已。另外有一款是 pygment，这是在服务端使用的代码分析及高亮工具，如果在服务端渲染，会直接导致页面速度变慢很多，而且用户也必须等到服务端渲染结束以后才能看到源代码。经过衡量，我最终还是选择了 highlight.js。

我觉得，有了在线查看源代码的功能，打包下载的功能估计就没有人会用了。

4.3 动态链接

毫无疑问，在查看源代码的时候，可能会想去部署页面看看最终效果，也可能会想把源码打包下载。这里涉及到一个细节问题。如果在每个人的文件目录树中标注这些“去部署页面”，“打包下载”等按钮，这样就会并排出现很多重复按钮，尽管它们的链接不一样，但是外观看上去很不和谐。所以我想到了一个办法，那就是整个页面只有一个“去部署页面”以及“打包下载”的按钮(当然，他们是放在页面合适的位置上的)，然后根据当前查看的源代码的主人，来动态确定链接。

下面举一个例子来理解这样的设计：当前查看的源代码是针对与作业 2 的，有两个学生 Sp 和 Sq，他们的文件结构如下：



动态改变链接指的是，当用户查看 S(p)目录树的任何节点，部署链接都是指向如下 url：

```
/public/2/#{S(p).student_id}/index.html
```

同时，下载链接指向的是如下 url：

```
/public/2/#{S(p).student_id}/#{S(p).homework.file_name}.zip
```

学生 S(q)的同理，这样设计的好处在交互中不言而喻。

4.4 评分

其实整个应用的功能基本上都集中于源代码查看页面了。因为在查看源代码的时候还有的一个重要操作就是评分，无论是小组评分还是老师评分都异常重要，所以在该页面设定了评分按钮，所以可以方便老师和同学能够在查看源代码的同时进行评分。

由于整个评分模块也都是用 ajax 来进行交互的，所以用户感觉不到页面的刷新，并且能够一边看源代码，一边在评分以及输入评语。

5 系统开发环境

5.1 基本开发环境

- 系统：Ubuntu 11.04
- 编辑器：Vim
- 框架：Rails 3.2.3
- Web 服务器：Webrick 1.3.1
- 数据库：Sqlite3

5.2Gems

对于 Rails 来说还使用了额外的 gems，下面一一列出：

- haml：模板语言
- sass：一种 css 的扩展
- rubyzip：提供服务端的 zip 文件解压及打包支持
- bluecloth：提供 markdown 支持
- highlight_js-rails：提供客户端的代码高亮