HOPE题库的设计与实现

# 题目(question)

HOPE题库主要围绕question列表，外加一些附属结构，实现题目的录入，搜索，以及删除。题库支持四种题目：

* 单选题
* 多选题
* 填空题
* 解答题

其中单选题和多选题，答案存于question中的五列

* question.choice\_a
* question.choice\_b
* question.choice\_c
* question.choice\_d
* question.choice\_e

如果是填空题，以上五列为NULL，答案另存于answer列表中，通过answer.q\_id联系到题目，不限制填空答案的数目，其顺序由answer.sequence决定，而每个填空答案的具体内容则存于answer.answer中。解答题和填空题类似，答案也是存在answer列表中。在逻辑上，解答题只有一个答案，但题库设计本身并不强迫解答题只能是单答案，多答案的解答题亦可存。

题目有多个状态，状态由两个question.verified和question.available联合决定：

|  |  |  |
| --- | --- | --- |
|  | question.verified | question.available |
| 最初录入 | N | N |
| 校验提交 | S | N |
| 校验拒绝 | R | N |
| 校验通过 | Y | N |
| 题目发表 | Y | Y |
| 题目撤回 | Y | N |

题目搜索，限制非题库人员只能搜索到发表的题目，但不限制题库人员。题目的这几种状态只能按固定模式变迁：

verified: N

available: N

verified: S

available: N

录入员提交

verified: R

available: N

发现错误, 校验员拒绝

经过修正，录入员再次提交

verified: Y

available: N

通过校验员校验

verified: Y

available: Y

发表题目

题目撤回

题目被删除

# 题群 (q\_group)

作为相关题目的一种组织形式，多个题目可以组织为题群(或称题包)。题目和题群之间的关系，是N:N的关系，如前所述，题目存于question列表，题群存于q\_group列表，二者之间的关系则存于q\_q列表。题群起初建立时为空，所谓把题目加入到某个题群中，就是建立一个q\_q表项，在题目和题群之间建立联系。同一个题目可以放入多个题群中，适应不同的组群方式。

# 知识点 (tap\_point)

作为一种科学分类，知识点的组织，是个多层的树状结构。比如，中学数学作为一个根知识点，其下包含若干中间知识点，每个中间知识点，可以包含若干更细的中间知识点，如此反复，直至叶知识点。题库本身不限制层数，多少层都可以。在实践上，则极少超过三层。

题目在录入时，总是直接联系在一个或者几个叶知识点上，这些叶知识点之间，未必有直接的逻辑联系，但通过这个题目组合在一处。题目在存入题库时，系统反向搜寻每个叶知识点的各层中间知识点直至根知识点，所搜寻到的所有中间知识点连同叶知识点全部存入数据库。以后按知识点搜索题目时，则可以按叶知识点搜索，也支持按中间知识点搜索。

题目

# 题库人员

题库支持三种用户：

* 录入员 (builder)
* 校验员 (inspector)
* 管理员 (admin)

录入员的职责是题目录入。校验员接受录入员提交的题目，检查题目录入的正确性，决定是否需要修正。需要修正的题目，退还录入员。如果录入正确，则接受题目。经过校验员接受的题目，则系统不再允许修改。管理员主要负责用户的建立。

# 低层数据库功能

每个数据库操作，形成一个DbRequest对象，在其中定义反调函数exec()，实现SQL操作。DbRequest对象本身，则放入队列中，由DbRequest处理函数，从队列中取出DbRequest，然后一一处理。因此，每个SQL操作，至少有两条线索参与：一个是DbRequest的产生者，另一个是DbRequest的处理者。这样做的好处是可以预先定义很少几个DbRequest的处理者，即可处理大量产生者产生的DbRequest且不会由于数据库的处理速度和SQL数目的匹配问题造成大量连接，而引起更大的处理问题失控。理论上，DbRequest的产生者可以不等待SQL完成，可以和处理者异步并行。但在Hope实践上，并无异步处理的需要，因此产生者在产生DbRequest插入队列后，总是等待其处理结束，取回SQL结果，然后返回，比较简单直接。

MySQL

DbMan.\_requestPool\_

DbMan.dbRunner

DbRequest

DbRequest处理者实现于DbMan.java，其功能低层，不限于Hope应用。题库的核心功能，集中在HopeDbMan.java，其中函数，均为DbRequest的产生者。这些函数，直接和题库功能相关。

# 题库函数

只要理解前述题库数据库的设计，则围绕DB的题库函数，总体比较直接。题库的核心函数，均定义于HopeDbMan.java。HopeDbMan中的函数，在HopeLogic.java中提供接口。除了接口之外，虽然系统并不强制，但在实践上不宜有接口外的其他函数直接调用HopeDbMan中定义的函数。这样，就可以把数据库的具体实现独立出来，甚至可以做比较大的改动而不影响系统的其他部分，实现层次结构。比如，以后如果用NoSQL DB取代RDBMS，则可把改动大部限制在局部而不涉及更广泛的区域。

HopeDb.java

HopeService.java

HopeLogic.java

HopeDbMan.java

上图反映题库函数的层次：

* 最外层的HopeService.java，提供题库访问的RESTful函数。
* 中间的HopeLogic.java，提供题库接口函数。在实践上，因效率考量，也可以从外部绕过RESTful而直接调用接口函数。
* 更下层是HopeDbMan.java，产生含有SQL的DbRequest。
* 核心是HopeDb.java，是题库的影像。

# 单元测试的例子

前面的内容描述了数据模型，及其一些有关数据模型的操作。在修改或者扩张数据模型之后，为确保功能正确，单元测试是非常重要的，即不通过W3浏览器，直接用REST接口测试，控制接口的输入，观察接口的输出。下面列举几个单元测试的例子。

假定有两个题库，内容不同，我们需要通过API实现把题库151中的若干题目复制到题库156中，通过这个例子，就可以测试题库题目的读取和存储。按照题库函数的层次结构，在最外面的REST测试，如果成功，则内层功能正确。如果失败，则需按层逐次由外向内寻找问题的根源。这个测试的主要内容可以概括为4点：

1. 导入数据模型。题目的数据模型是Question.java，通过API复制题目，就需要导入题目的数据模型。
2. 登录到题库151和题库156两处。这里使用了一个名称为builder1录入员账户，其登录口令为trial。复制过程本身并不要求读取和写入使用同一用户名，不同用户名也可以。
3. 从题库151读取出题目。注意，第43行把题目的ID置空的目的是避免这个题目ID以及出现在题库156中。置空之后，在题目加入题库156时，题库156会自动分配一个新的ID给被复制的题目。当然，题目在题库151中的ID和在题库156中的ID会不同，但其他内容相同。
4. 把题目加入到题库156中。

注意，这个例子假定题目不含图片。在Hope题库中，图片的地址是直接嵌入在题干中，而图片本身存放于文档系统。含图片题目的复制不仅需要复制题目，而且需要复制文档。

A

|  |
| --- |
| 1 import com.lanking.hope.data.Question; |

2

3 import org.codehaus.jackson.map.ObjectMapper;

4 import org.apache.http.impl.client.DefaultHttpClient;

5 import org.apache.http.client.methods.HttpGet;

6 import org.apache.http.client.methods.HttpPost;

7 import org.apache.http.HttpResponse;

8 import org.apache.http.HttpEntity;

9 import org.apache.http.util.EntityUtils;

10 import org.apache.http.entity.StringEntity;

11

12 public class Mycopy {

13 public static void main(String[] args) throws Exception {

14 DefaultHttpClient httpClient151 = new DefaultHttpClient();

15 DefaultHttpClient httpClient156 = new DefaultHttpClient();

16 ObjectMapper mapper = new ObjectMapper();

17

B

|  |
| --- |
| 18 HttpPost logonPost151 = new HttpPost("http://192.168.1.151/rest/logon/builder1/trial");  19 HttpResponse logonResp151 = httpClient151.execute(logonPost151);  20 System.out.println(logonResp151.toString());  21 logonPost151.releaseConnection();  22 |
| 23 HttpPost logonPost156 = new HttpPost("http://192.168.1.156/rest/logon/builder1/trial");  24 HttpResponse logonResp156 = httpClient156.execute(logonPost156);  25 System.out.println(logonResp156.toString());  26 logonPost156.releaseConnection(); |

27

28 String[] qIds= {

29 "300846",

30 "300847",

31 "300852",

32 "300853"

33 };

34

35 for (int i = 0; i < qIds.length; i++) {

36 HttpGet get151 = new HttpGet("http://192.168.1.151/rest/question/" + qIds[i]);

37 HttpResponse resp151 = httpClient151.execute(get151);

38 System.out.println(resp151);

39 System.out.println(resp151.getStatusLine().getStatusCode());

40

|  |
| --- |
| 41 HttpEntity entity = resp151.getEntity();  C  42 Question question = mapper.readValue(EntityUtils.toString(entity), Question.class);  43 question.setId(null);  44 get151.releaseConnection(); |

45

46 String qStr = mapper.writeValueAsString(question);

47 System.out.println("qStr=[" + qStr + "]");

48

D

|  |
| --- |
| 49 HttpPost post156 = new HttpPost("http://192.168.1.156/rest/question");  50 post156.addHeader("Content-Type", "application/json; charset=UTF-8");  51 post156.setEntity(new StringEntity(qStr, "UTF-8"));  52 |

53 HttpResponse resp156 = httpClient156.execute(post156);

54 System.out.println(resp156.toString());

55 post156.releaseConnection();

56 }

57 }

58 }

HOPE作业系统的设计与实现

HOPE作业系统是一个比较典型的在线交易处理系统(OLTP)。作业系统的数据，除了出于运行速度的考虑，在作业成绩统计部分包含了数据冗余，其他部分基本上是个正交关系模型，尽力避免数据冗余，通过数据列表的若干组合，支持各种数据访问, 且保持数据的一致性。

作业系统的数据，和题库系统不混合。作业系统有其独立的数据库，通过外围接口函数，对外提供一个接口，但内部设计和实现，二者是相对独立的。如下图所示：

题库数据库

作业数据库

题库函数

作业函数

HOPE接口介面(HopeLogic & HopeService)

# 作业系统的层次结构

# 作业系统的层次结构和题库系统类似。在最外两层的模块，和题库系统相同。但这只是实现上的组织，在逻辑上，即使是最外两层，二者并无直接关联。只是所实现的功能可以类比。

SchoolDb.java

HopeService.java

HopeLogic.java

SchoolDbMan.java

# 时间表

课程时间表和学校之间，在一个给定的时间段里，是1:N的关系。在不同的学期，一个学校可以有不同的时间表，但在一个学期里，一个学校只有一个时间表。同时，不同学校，也可以使用同一个时间表。这种关系，反映在学校列表(school)里，都包含一个s\_t\_id，指向其当前使用的时间表。这个时间表的使用时间段，则由两个日期决定，开始于start\_ymd，结束于end\_ymd。

* s\_t -- 在逻辑上，s\_t和学期的概念对应，有起始日期和结束日期。
* s\_t\_slot -- 在一个学期当中，学校每天的时间安排基本上是天天在重复的。比如，每天上午第一节课开始于8:00，结束于8:50，第二节课开始于9:00，结束于9:50，下午第一节课开始于2:00，结束于2:50，等等。s\_t\_slot就是记录这个信息，对应每节课(slot\_num)，定义其起始时间(start\_t)和结束时间(end\_t)。很明显，s\_t 和s\_t\_slot之间，是1:N的关系。
* v\_schedule\_template -- 这数据视窗反映学校和时间表的组合。对于某个学校，如果要列出其当前时间表，通过这个视窗查询比较方便。
* schedule -- 前述时间表，还只是一个学期中每天的作息安排，没有和具体的课程挂钩。一个课程在某个学期的每堂课的安排，则保存于schedule中。这个列表把课程(c\_t\_c)，具体上课日期(ymd)，和该日期中的课程时间段(slot\_num)联系起来。通过这个表，某个老师给某个班级在某个学期上课的时间安排就一目了然了。
* v\_schedule -- 和schedule内容等价。在schedule中没有直接存储课程时间段的起始时间和结束时间，而是存储了s\_t\_slot的时间段号码(slot\_num)。虽然这样作避免了数据冗余，但在最终时间显示上，还是需要显示课程的具体时间，而不仅仅是时间段的号码。比如，罗老师给高一二班代数课安排在9月1日上午第二节课，schedule就是提供这个信息，v\_schedule则联系s\_t\_slot列表，额外指出9月1日上午第二节课开始于9:00，结束于9:50。这样，在数据的具体使用上比较方便，同时又避免了反复存储9:00-9:50这样的时间段数据，假如s\_t\_slot时间变更，则与其相关联的所有课程表项全部会自动变更。
* v\_student\_schedule & v\_teacher\_schedule --学校安排时间表，对象是课程。但Hope系统的用户，是教师和学生。教师和学生的时间表，内容和学校给课程安排的时间表相关，但需要把各个相关课程的时间表结合起来。比如，高一二班有语文，代数，物理，化学四门课，把这四个时间表结合起来，就是该班每个学生的时间表。再以教师为例，罗老师给高一一班和二班上代数课，那么，她的时间表，就是把两个代数课的时间结合起来。以学生和教师为核心的时间表，为了避免数据冗余，同样不再另外存储，而是通过数据视窗实现。

下图示意各种时间表的关系

school

s\_t

s\_t\_slot

课程

schedule

时间表实存列表

v\_schedule

v\_student\_schedule

v\_teacher\_schedule

# 课程

在Hope系统中，课程是作业(或者测验)的分派对象之一(另一个分派对象是单个学生)，因此课程的概念不仅仅是一门课，比如“高一代数”这么简单。作为作业的分配对象，课程必须联系到某个具体的班级，同时，还需要联系到作业的制造者-- 某个教师。因此，在Hope系统中，有t\_c和c\_t\_c列表。

* catalog - 其中每一表项代表某一门课，比较笼统。比如，“高一代数”。
* t\_c –反映的是教师和某门课之间的关系，这个教师是某门课的任课教师，则在t\_c列表中有一相关表项，是由某个教师任教的某一门课。比如，某校高一代数这门课，由罗老师和李老师开课，那么在t\_c中，就有两个表项。
* c\_t\_c –反映的是班级和课程的关系。比如，由罗老师任教的高一代数，授课对象是一班和二班，由李老师任教的同一门课(但由于教师不同，如前所述，是不同的t\_c)，授课对象是三班和四班。那么就有四个c\_t\_c表项：由罗老师任教的给一班上的高一代数，由罗老师任教的给二班上的高一代数，由李老师任教的给三班上的高一代数，以及由李老师任教的给四班上的高一代数。
* v\_course - 和时间表类似，课程的查询和搜索，通过数据视窗v\_course比较方便。虽然不用视窗而直接查询也可以实现同样的功能，但会出现大量查询代码重复，为避免这种情况，课程常用查询模式被总结在v\_course视窗中，不仅节省了大量冗余代码，且相关列表中的数据改动，自动反映在视窗中，有利于保持数据的一致性。

“课程”的概念本身比较笼统，在Hope中的课程，除非单独说明，则课程单指某个c\_t\_c表项，这样，知道了课程，就知其任课教师以及授课班级，同时，因为学生总是有个隶属班级，通过c\_t\_c，我们也就知道了学生和课程的关系。当然，学生和任课教师的关系，以及学生和课程的关系，就不需单独另存，以求避免数据冗余。

如前所述，c\_t\_c也是作业分派对象。作业由某个教师分派到某个班级。在Hope系统中，这种分派并不是任意的。作业只能由任课教师分派给由她授课的某个班级，因此，c\_t\_c成为作业分派对象。同时，也给任课教师分派作业的灵活性，比如，罗老师任教两个班，她可以选择给不同的班级分派不同的作业。

下图示意和课程相关的列表和视窗。

teacher

catalog

class

v\_course

# 教案

教案(plan)，讲座(topic)，题组(q\_set)，文档(document)属于静态对象，但和动态的作业(或者测验)之间，有紧密的关系。在数据组织上，讲座是存放题组链接和文档链接的载体(注意，存放的是链接)，而教案则是讲座的载体。由于讲座里保存的是题组链接和文档链接，因此删除一个讲座，原先保存在讲座中的题组链接和和文档链接也被删除，但不会删除题组和文档本身。讲座直接保存在教案中，一旦教案被删除，讲座也随之被删除。

如前所述，题组和文档可以不依赖讲座而存在。一个很自然的问题是在无讲座或者讲座被删除的情况下，题组和文档是如何组织和存放的。在作业系统中，有大量“我的XX“概念，例如，”我的教案“，”我的讲座“，”我的题组“，以及”我的文档“。教案，讲座，题组和文档，均含有所有人属性，一般来说，其所有人也就是其创建者。题组和讲座之间是软链接，讲座删除时，二者的软链接被断开，题组依旧存在于”我的题组“之中。只有当题组从”我的题组“中删除时，题组才真正被删除。讲座和教案之间的链接，设计上为硬链接，教案从”我的教案“中删除时，讲座被连带从”我的讲座“中删除。

很显然，一个题组或者文档，可以存在于零个或者多个讲座中。一个讲座，只能存在于一个教案之中，且必须存在于一个教案之中。而一个教案，则可包含零个或者多个讲座。几者之间的逻辑关系如下图所示：

讲座 I

讲座 II

讲座 III

题组1

题组2

题组3

文档A

文档B

题组4

教案甲

值得注意的是，在Hope系统中，作业的分派源头和分派单位是题组，且这个题组必须存在于某个讲座之中。数据模型本身并不要求题组必须出现在讲座中才能分派，但目前的用户介面只搜寻和链接到讲座中的题组。从外在表现上看，作业(或者测验)可以被看作是题组的动态部分。

# 题组

前面在题库部分介绍了“题群”(q\_group)。在学校部分，和题群对应的题目集合，就是题组(q\_set)。和题群类似，题组是题目的一种组织形式。但题库里的题群是题库录入和管理人员定义的，而题组是由教师通过搜寻题库选出题目，然后由教师产生的。两种题目的集合在逻辑上无关。

题组，是作业(或者测验)的分派单位。换而言之，教师分派给某个班级或者学生的作业(或者测验)，不是单个题目，而是由她预先定义好的一组题目，这就是题组。当然，系统也允许题组只包含一个题目，甚至允许定义空题组(但不允许分派空题组)。题库的题目，在题组建立过程中，并不复制到学校的数据库，只是在学校的数据库里保存题库题目的ID (question.id)。在同一题组中，设计上不允许出现重复的题目ID，但同一个题目ID，可以出现在不同题组中。题组本身是个静态结构：教师可以修改或者删除题组，但系统不会自动去修改题组。

结合前述教案和讲座，再加上题组，几者的层次关系如下：

讲座 I

讲座 II

讲座 III

题组1

题组2

题组3

文档A

文档B

题组4

教案甲

题群A

题群B

题库

# 任务

作业(或者测验)，在HOPE实现中，统称为任务(assignment)。任务是有时效的，从时间上看，任务可以出于几个不同的状态：

1. 任务被创立之后，进入”等待“状态(PENDING)。这时，任务数据已经保存：按照任务定义的题组和分派对象，相关题组从题库中取出题目，进行变题操作(目前题目变题部分为空)，把任务分派到每个学生的书包(s\_a)中去。但此时任务还只对其创立教师可见，对学生还不可见。在这个状态，创立教师可以取消任务，如果任务取消，作业系统会把自动地把处于等待态的任务从学生书包中删除。当然，这种等待任务的删除操作，对学生是不可见的。
2. 任务开始时间到，HOPE系统的”任务开启器“ (在Scheduler.java中定义) 打开任务，任务进入打开状态(OPEN)。当任务进入此态，任务对学生可见，学生可以回答题目，但不能查询标准正确答案。
3. 任务结束时间到，HOPE系统的”任务关闭器“(在Scheduler.java中定义) 关闭任务。任务进入”关闭“状态(CLOSED)。在任务被关闭后，学生不能再回答任务中的题目。但可以查询题目的标准正确答案。
4. 任务关闭后教师审阅学生答案，下发结果。任务进入”下放“状态(RELEASED)。此时，学生可以查看成绩。

PENDING

OPEN

CLOSED

RELEASED

任务被删除

开始

时间到

结束

时间到

审阅

完毕



在整个HOPE系统中，大部分操作是由用户引起的，比如，创立题目，修改题组，建立教案讲座等等。但有若干例外，任务的开启和关闭，就是例外之一。其发生时间虽然是任务建立者选定的，但发生在将来的某个时间，具体执行是由时钟控制的，属于自运行子系统。另外一个例外，是批改接口，会在后面介绍细节。

# 任务的分派和跟踪

任务有两种分派方式。一种由是教师分派给一个c\_t\_c (见前述，c\_t\_c是班级和课程的组合)。另一种是由教师分派给某个学生。

任务被分派之后，有个跟踪和统计的问题，教师分派任务给一个班级之后，HOPE作业系统需要

* 跟踪每个学生对任务的完成情况。
* 跟踪每个学生对每个题目的完成情况。
* 跟踪每个学生对每个题目的每个答案的回答情况。

所有和任务相关的数据，在任务分派之后，都是在不定时间动态变化的，直到任务关闭。在HOPE作业系统中，以下的列表用于任务的跟踪和记录：

|  |  |
| --- | --- |
| assignment | 教师端的任务表项 |
| s\_a | 学生端的任务表项 (书包) |
| as\_q\_set\_c | 任务中包含的题目 |
| var\_q | 变体题 |
| var\_a | 变体题答案 |
| s\_q | 分派到学生的题目 |
| s\_w | 学生回答的题目答案 (作业本) |

# 以上这些与作业直接相关的列表，其间的逻辑关系示意如下：

任务

assignment

as\_q\_set\_c

学生X任务

s\_a

学生Y任务

s\_a

学生X题目1 (s\_q)

学生X题目2 (s\_q)

学生Y题目1 (s\_q)

学生Y题目2 (s\_q)

var\_q

var\_a

题库

question

变题复制

在产生任务时，教师需要挑选一个已经定义好的题组(q\_set)，比如，这个题组可以是来自她的某个讲座。题组里包含的是指向题库题目的链接。在任务(assignment) 实际产生时，题组里的题目链接均被复制到”任务题组“ (as\_q\_set\_c)。这样作的目的是在任务产生之后，任务里面包含的题目就和题组的定义脱离直接联系，能够保持”任务题组“的稳定性，任务的内容不会因题组修改而改变。

学生和教师看见的任务映象有区别。教师关心的是她分派出去的任务，任务可以分派给不同的班级和学生，但教师甲分派出去的任务和教师乙分派出去的任务并无直接关联。学生关心的则是他收到的任务，任务可以来自多个教师，但A班的学生收到的任务和B班学生收到的任务并无直接关联。这种关系，体现在两个任务数据视窗上：

* v\_teacher\_assignment - 为方便从教师角度查询任务而设立。
* v\_student\_assignment - 为方便从学生角度查询任务而设立。

学生在任务处于OPEN状态回答问题之后，其答案被保存在”作业本“(s\_w)。对于单选题和多选题，在答案存入作业本时，系统简单比较学生答案(student\_answer) 和标准答案(official\_answer)，立刻就知道对错。对于填空题和解答题，除非是学生答案和标准答案完全吻合，否则无法简单地判断对错，需要人工或者机器进行分析和批改。注意，对错是针对每个答案的，其值存储在s\_w.correct表列。有四种可能：

* Y -- 答案正确。
* N -- 答案错误。
* U -- 无法判断对错。需提交人工或者智能机器批改。
* P -- 部分正确但不完全正确。此值由教师人工批改时赋予。

任务起始时间到，进入OPEN状态之后，学生必须被通知。这种通知，有两种模式：

* 查询模式 -- 学生查看书包，如果有OPEN态的任务，学生会看见。
* 推送模式 -- 系统自动推送任务消息给学生。

很显然，查询模式比较简单，但效率较低且实时性比较差。推送模式效率较高，且实时性能较好，但需要浏览器方面支持这种推送。在作业系统实现中，两种模式都支持。查询模式通过REST函数实现。推送模式则通过HornetQ实现WebSocket。目前，推送模式的代码因无用而暂时被屏蔽，但这是以后的发展方向。

在作业系统中，提示信息(AssignmentAlert)不仅仅是告诉学生任务到达，而且实时跟踪学生回答题目的状态：哪个题已答，哪个题目未答。对于教师的提示信息，则实时跟踪学生的哪个答案处于“未知对错”态，可能会需要人工批改。

# 成绩统计

成绩统计数据的来源，是数据视窗v\_assignment\_result。由于任务跟踪的需要，作业系统中包含了多个列表，虽然原始数据存于这些列表，但如果要找出某个学生对于某个特定任务里面某个题目的答案，则必须把相应列表链接起来。这种查询，在成绩统计时大量使用，为方便起见，定义了数据视窗。这样作，虽然并不能提高统计数据产出的速度(与直接查询速度相当)，但有了这个视窗，不仅简化了统计代码的长度，且系统可以在任何时间产出任何任务的统计数据，即使是还处于OPEN状态的任务，也可以产生统计报表，且随学生回答问题的对错，实时变更统计数据。这在要求实时统计的场合(比如测验) 特别有用。

任务

(assignment)

学生任务

(s\_a)

学生题目

(s\_q)

用户信息

(account)

学生作业本

(s\_w)

变题及答案

(var\_a & var\_q)

v\_assignment\_result

统计信息的数据模型是AssignmentStat.java。”任务统计“(AssignmentStat) 对应任务(assignment)，其中包含几个子结构：

* TapPointStat -- 存储每个知识点掌握情况的统计数据。
* AnswerStat -- 存储每个答案对错的统计数据。
* StudentStat -- 存储每个学生的成绩。

另外，统计数据还包括最高成绩，最低成绩，平均成绩和标准方差。

批改完毕任务进入RELEASED状态之后，分数不会再改变。由于统计数据的产生，属于高成本运算，因此，最终统计数据存入固表中。随后的统计查询则直接读取固表内容：

* as\_stat
* as\_tap\_stat
* as\_answer\_stat
* as\_student\_stat

# 批改接口

对于无法简单地判断对错的学生答案，需要传送到智能机器进行批改。这种传递具体的实现是种间接的方式：

对于无法简单判断对错的答案，系统将题号，标准答案和学生答案保存于HornetQ中。在REST接口，则有”getUnknownAnswer()"函数，从HornetQ中读取待判断对错的题号，标准答案和学生答案。接口中另有函数“resolveUnknownAnswer()”，设定答案的对错。

未知答案读写器，通过REST接口，不断地取出待判断数据，然后传递给智能系统，智能系统完成后，再由未知答案读写器透过前述REST接口将结果保存进作业系统。

# 单元测试的例子

下面这个例子测试学生回答作业的接口。模拟两个学生完成一组含有选择题的任务。

1 import org.codehaus.jackson.map.ObjectMapper;

2

3 import com.lanking.hope.data.\*;

4

5 import org.apache.http.impl.client.DefaultHttpClient;

6 import org.apache.http.client.methods.HttpGet;

7 import org.apache.http.client.methods.HttpPost;

8 import org.apache.http.client.methods.HttpPut;

9 import org.apache.http.HttpResponse;

10 import org.apache.http.HttpEntity;

11 import org.apache.http.entity.StringEntity;

12 import org.apache.http.util.EntityUtils;

13

14 public class Myqa {

15 public static String[] students = {

16 "student1",

17 "student2"

18 };

19

20 public static void main(String[] args) throws Exception {

21 if (args.length == 0) {

22 System.out.println("Usage: myqa <asnId>");

23 System.exit(0);

24 }

25

26 String asnId = args[0];

27

28 DefaultHttpClient httpClient = new DefaultHttpClient();

29 ObjectMapper mapper = new ObjectMapper();

30

31 for (int i = 0; i < students.length; i++) {

32 String name = students[i];

33 String password = "trial";

34

35 // logon

36 HttpPost logonPost = new HttpPost("http://192.168.1.156/rest/logon/" + name + "/" + password);

37 HttpResponse logonResp = httpClient.execute(logonPost);

38 logonPost.releaseConnection();

39

40 // get assignment questions

41 HttpGet saGet = new HttpGet("http://192.168.1.156/rest/student\_assignment/" + asnId);

42 HttpResponse saResp = httpClient.execute(saGet);

43

44 HttpEntity saEntity = saResp.getEntity();

45 StudentAssignment sa = mapper.readValue(EntityUtils.toString(saEntity), StudentAssignment.class);

46 saGet.releaseConnection();

47

48 String[] questions = sa.getStudentQuestions();

49 for (int j = 0; j < questions.length; j++) {

50 double r = Math.random();

51 int n = (int)(r \* 100);

52 if (n < 20) {

53 continue;

54 }

55

56 String sqId = questions[j];

57 String[] answers = new String[1];

58 answers[0] = "D";

59

60 if (n > 95) {

61 answers[0] = "A";

62 } else if (n > 90) {

63 answers[0] = "B";

64 } else if (n > 85) {

65 answers[0] = "C";

66 } else if (n > 80) {

67 answers[0] = "E";

68 }

69

70 String ansStr = mapper.writeValueAsString(answers);

71 System.out.println(name + " " + sqId + " " + ansStr);

72

73 HttpPut sqPut = new HttpPut("http://192.168.1.156/rest/student\_question/" + sqId);

74 sqPut.addHeader("Content-Type", "application/json; charset=UTF-8");

75 sqPut.setEntity(new StringEntity(ansStr, "UTF-8"));

76

77 HttpResponse sqResp = httpClient.execute(sqPut);

78 sqPut.releaseConnection();

79

80 }

81 }

82 }

83 }