设计模式研讨报告

在线测试子系统

组员：赵无瑕 何淇丹 万信逸 赵冰骞 李飞

2012-4-21

目录

1 引言

1.1 编写目的

从本阶段开始，项目进入正式开发阶段。本设计说明书的编写目的，是以本项目的需求分析说明书为依据，从总体设计的角度，明确在线支付系统的总体架构，流程，数据结构，数据库设计。

目的在于：

1. 为编码人员提供依据
2. 为修改，维护提供条件
3. 明确各模块外部接口，内部接口，用户接口
4. 项目负责人将按计划说明书的要求布置和控制开发工作全过程

本说明书的预期读者包括：

1. 软件客户
2. 项目经理
3. 项目开发人员
4. 软件质量分析员
5. 软件维护人员

1.2 背景

1. 软件系统名称

   教务系统之在线测试子系统
2. 任务提出者

   浙江大学软件工程基础课程任课老师——陈越
3. 开发者

   浙江大学 2012-2013 学年春夏学期软件工程基础课程红五组
4. 用户群

任课教师、在读学生、教务中心职工、系统管理员

5. 实现该软件的计算机网络

   由若干台 PC 机组成局域网

6. 该软件系统同其他子系统的相互来往关系

   与选课子系统及信息子系统，从中获取数据信息。

1.3 定义

1. **MYSQL:**一个小型关系型数据库管理系统。
2. **APACHE:**比较流行的 轻量级 WEB 服务器端软件之一。
3. **PHP:**是英文超级文本预处理语言 HYPERTEXT PREPROCESSOR 的缩写。PHP 是一种 HTML 内嵌式的语言，是一种在服务器端执行的嵌入 HTML 文档的脚本语言，语言的风格有类似于 C 语言，被广泛的运用。
4. **JAVASCRIPT:**JAVASCRIPT 是一种面向对象的动态类型的区分大小写的客户端脚本语言。
5. **YII：** Yii 是一个基于组件、用于开发大型 Web 应用的高性能 PHP 框架。Yii 提供了今日 Web 2.0 应用开发所需要的几乎一切功能。Yii 是最有效率的 PHP 框架之一。
6. **SQL 注入：**通过把 SQL 命令插入到 WEB 表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令。
7. **数据库连接池：**数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，这项技术能明显提高对数据库操作的性能。
8. **安全证书：**安全证书是在进行登陆时的身份证，或者说是私人钥匙，安全证书是唯一的，与任何其他人的证书都不相同。

2 论文研讨

2.1 MVC 模式

2.1.1 简介

MVC（Model/View/Controller）模式是一种非常有用的交互软件系统框架。它的主要思想是将用户界面从数据分离出来。在 MVC 设计模式中，视图（view）将信息展示给用户，控制器（controller）处理用户请求，模型（model）用来表示数据及逻辑关系。使用 MVC 模式后，应用程序可以方便的更改视图而不用修改控制器和模型。

2.1.2 MVC 在 WEB 中的应用

Web 应用可以像其他应用程序一样从 MVC 模式中受益。但有一个问题是，Web 应用是分客户端和服务器端的。显然，视图在客户端中展示，模型和控制器可以被多种方式放在客户端和服务器端。开发者需要在设计阶段预先将其 Web 应用分好。但 MVC 模式是与划分无关的，因为模型、视图、控制器属于同一地址空间。

当然，如果一个 Web 应用被正确地划分，并有相应的技术支持，MVC 可以很好地适用。Fwap（Flexible Web-Application Partitioning）可以让 Web 应用更自然地应用 MVC 模式。

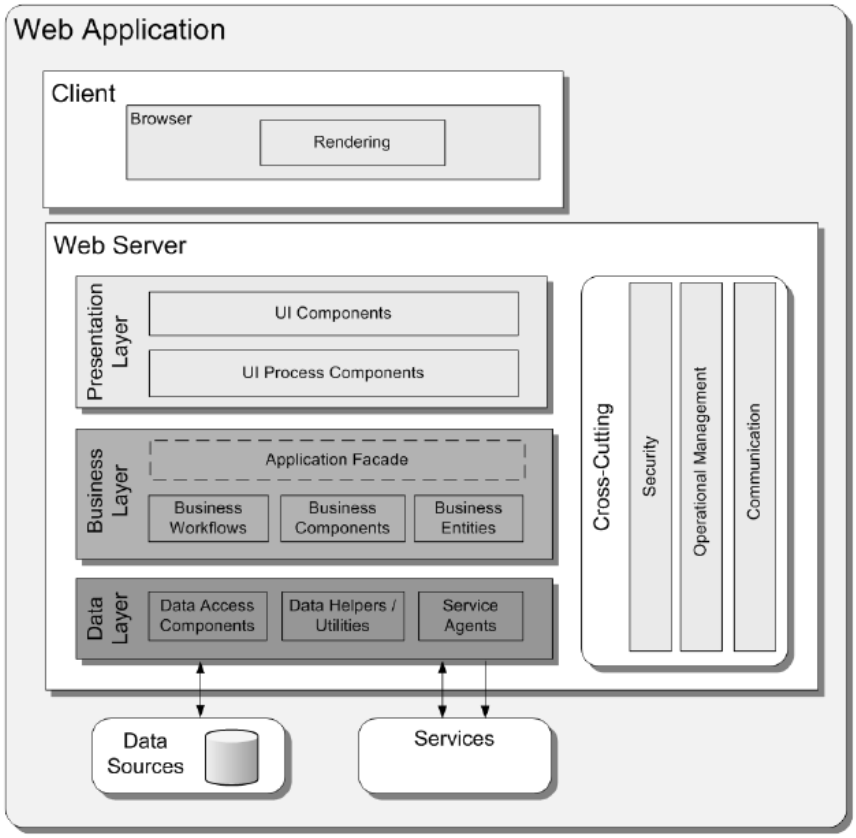

FIGURE 1 经典 MVC 在 WEB 中的应用

2.2 FWAP 设计模式

2.2.1 概述

Fwap 支持以下 3 种架构：

1) Single-mvc: 此架构和传统的 MVC 设计模式一样，模型、控制器、视图都属于同一地址空间，而不存在客户端与服务器端划分的问题。
2) Thin-client: 此架构的模型和控制器在服务器端的同一地址空间内，视图位于客户端上。
3) Dual-mvc: 此架构的模型和控制器同时处于客户端和服务器上。客户端和服务器都可以根据需求产生视图并在客户端上显示。

2.2.2 编程模型

Fwap 的视图是有一系列的 GUI 组件构成。这些组件通过接口创建、删除、访问等。因为组件只能通过 API 访问，服务器端的视图可以明显地和客户端视图同步。这样控制器可以实时地重新划分在两端运行的部分。

模型有和视图一样的要求，同样通过接口来实现。

控制器是一个可以在客户端或服务器端运行的逻辑单元。控制器的实际执行位置是由控制器部署描述符来决定。这与传统的在一个地址空间执行的代码不同。

可调 Web 应用部署的第一步是设定哪些方法是可调划分的。如果一个方法是这个特定应用接口的一部分，那它就是可调的。根据 MVC 设计模式，程序员根据逻辑关系访问模型生成需要的视图，并适时地更新模型。

2.3 FWAP 应用

2.3.1 SMVC

Smvc 的视图组件在 Java Swing 组件中被使用。Fwap 在 Swing 应用中开发、测试。

2.3.2 THIN-CLIENT

Web 客户端通过 url 的 GET 请求来获取结果：

1) 实例化一个应用来处理客户端的请求；
2) 根据请求响应，并生成视图；
3) 返回 HTML 到客户端。

因为所以的 Web 应用都使用相同的一系列方法，所以一个控制器实例可以处理所有的请求。

2.3.3 DUAL-MVC

客户端和服务器端各有一个应用实例在运行。客户端重载了 smvc 的实现：

1) 运行时，访问控制器的部署描述符来决定该控制器是在客户端还是服务器端执行。
2) 若是客户端调用，则和 smvc 模式类似，调用 super.Method()。
3) 如是服务器端调用，先序列化客户端模型；序列化客户端视图；序列化传给控制器的参数；通过 HTTP 的 Post 请求将以上信息传给服务器端；控制器将服务器端的模型、视图与客户端的同步；在服务器端像 smvc 模式一样调用方法；通过 Post 传回信息给客户端。
4) 客户端的控制器根据服务器的模型根系视图上的 GUI 组件。

3 架构设计分析

3.1 架构设计目标

3.1.1 关键功能

1. 题库管理：

账户条件：任教该课程的教师、管理员

可行操作：

1） 增加题目

2） 删除题目

3） 修改题目

4） 查询题库：输入题号查询或者输入关键字模糊查询

2. 试卷管理：

账户条件：任教该课的教师、管理员

可行操作：

1） 查看试卷

2） 生成试卷

3） 修改未发布试卷

4） 删除试卷

5） 自动生成试卷

3. 学生在线测试：

账户条件：选上该课程的学生

可行操作：

1） 计时在线做试卷

2） 保存试卷(以防临时有事暂时保存已做题目)

4. 分数统计分析：

账户条件：任教该课程的教师或正在上课的学生或管理员

可行操作：

1） 教师或管理员查看某份试卷各题难度分布

2） 教师或管理员查看题库各题难度分布

3） 学生查看历史成绩

附加功能

1. 提醒功能：提醒学生有未完成的试卷
2. 试卷发布功能：教师可选择是否发布某份试卷

3.1.2 系统性能

本系统的性能将由服务器端数据库，网络数据传输延时，以及并发访问该系统的用户数量决定。本系统将为用户提供良好的界面体验。

3.1.3 输入输出要求

客户端通过网页展现给用户一个友好的界面，用户可以通过提交表单或者点击超链接向服务器提供数据与命令。

服务器后台处理后，将结果显示到用户的网页界面上。

3.1.4 数据管理能力要求

1. 安全：服务器将予以数据库最高等级的保护，防止黑客从后台下载数据库，防止通过网页 SQL 注入的方式从数据库中获取信息或者破坏数据库。
2. 性能：对于频繁访问数据库的操作，后台需要建立持久的数据库连接，以避免重复连接数据库耗费资源。

3.2 架构设计原则

3.2.1 关键原则

1. “开闭”原则(OCP)

   一个软件实体应当对扩展开放，对修改关闭。“抽象化”是 OCP 的关键。
2. 里氏代换原则(LSP)

在一个软件系统中，子类应该可以替换任何基类能够出现的地方，并且经过替换以后，代码还能正常工作。“继承”是 LSP 的关键。

3. 依赖倒转原则(DIP)

   要依赖于抽象,不要依赖于具体。或者说是：要针对接口编程，不要对实现编程。“规范抽象”是 DIP 的关键。

4. 接口隔离原则(ISP)

   使用多个专门的接口比使用单一的总接口要好。也就是说，一个类对另外一个类的依赖性应当是建立在最小的接口上的。“多重继承”是 ISP 的关键。

5. 组合[1]/聚合[2]复用原则(CARP)

   在一个新的对象里面使用一些已有的对象，使之成为新对象的一部分：新的对象通过向这些对象的委派达到复用已有功能的目的。“组合/聚合”是 CARP 的关键。

6. 迪米特法则(LoD)

   又叫作最少知识原则（Least Knowledge Principle 简写 LKP），就是说一个对象应当对其他对象有尽可能少的了解,不和陌生人说话。“传递间接的调用”是 LoD 的关键。

3.3 数据设计

3.3.1 EER 图

[1]组合：也表示类之间整体和部分的关系，但是组合关系中部分和整体具有统一的生存期。一旦整体对象不存在，部分对象也将不存在。部分对象与整体对象之间具有共生死的关系。

[2]聚合：指的是整体与部分的关系。通常在定义一个整体类后，再去分析这个整体类的组成结构。从而找出一些组成类，该整体类和组成类之间就形成了聚合关系。例如一个航母编队包括海空母舰、驱护舰艇、舰载飞机及核动力攻击潜艇等。

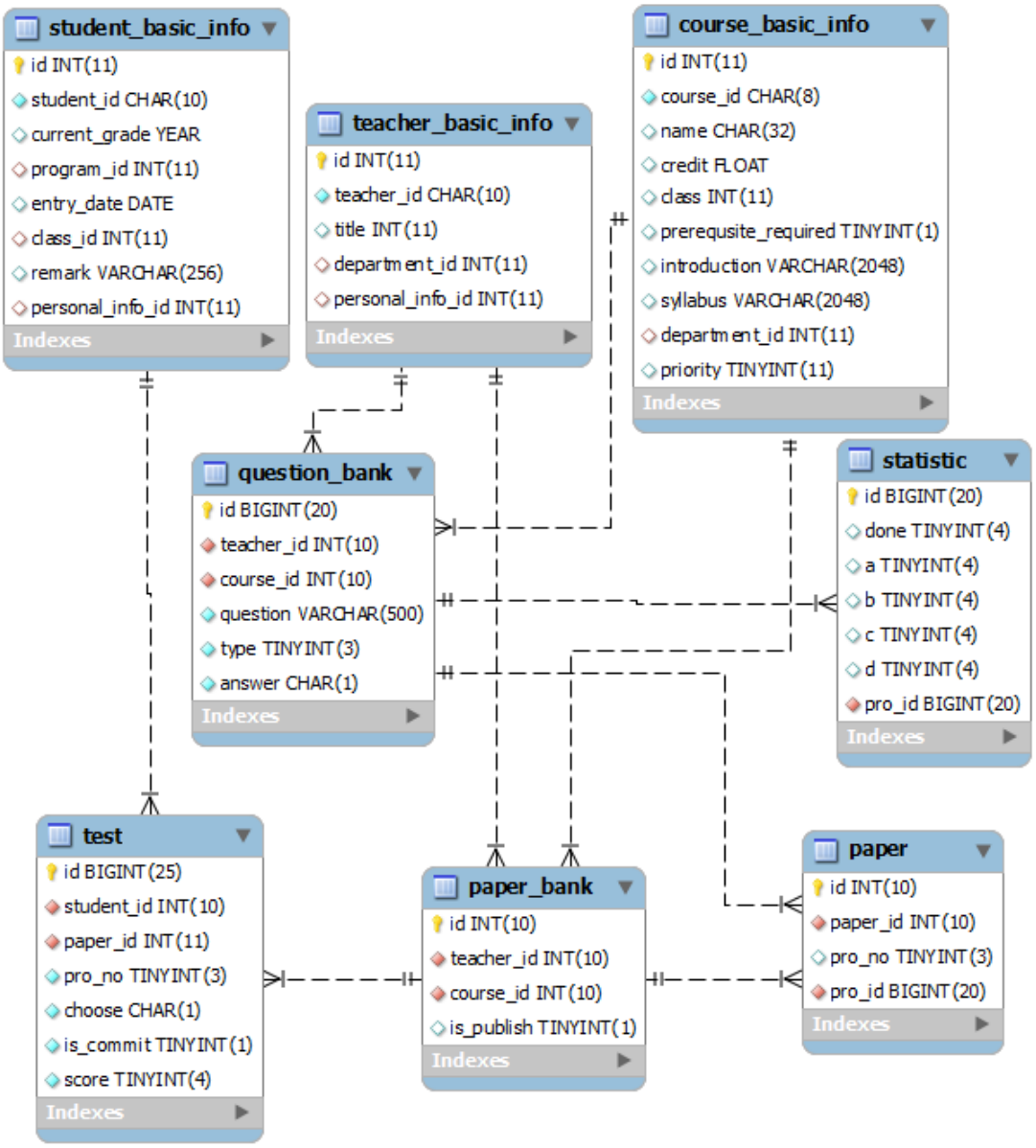

FIGURE 2 在线测试子系统数据库设计 EER 图

3.3.2 物理结构设计[3]

1. 依赖表单（来自其他子系统创建并维护的表单）

   1) 学生信息表单 student_basic_info(由信息子系统运维)[4]

字段名	字段含义	字段类型	引用自
id	无意义主键	INTEGER	无
student_id	学号(unique)	CHAR(10)	无

   2) 教师信息表单 teacher_basic_info (由信息子系统运维)

字段名	字段含义	字段类型	引用自
id	无意义主键	INTEGER	无
teacher_id	教师号(unique)	CHAR(10)	无

   3) 课程表单 course_basic_info (由信息子系统运维)

字段名	字段含义	字段类型	引用自
id	无意义主键	INTEGER	无
course_id	课程号(unique)	CHAR(8)	无

2. 本系统表单(由本子系统运维)

   1) 题库表单 question_bank

字段名	字段含义	字段类型	引用自
id	题号	BIGINT(20)	无
teacher_id	任课教师代号	INTEGER	引用自教师信息表
course_id	课程代号	INTEGER	引用自课程信息表
question	问题内容	varchar(500)	无
type	题型	tinyint	无

[3] 其中蓝色的表示该表的 primary key

[4] 对于其他系统提供的表单，只列出我们关心的字段

answer	答案	char	无

正确率由选项情况和正确答案可以计算出.

2) 试卷库 paper_bank

字段名	字段含义	字段类型	引用自
id	试卷号	INTEGER	无
teacher_id	任课教师代号	INTEGER	引用自学生信息表
course_id	课程代号	INTEGER	引用自课程信息表
is_publish	是否发布	BOOLEAN	无

3) 试卷 paper

字段名	字段含义	字段类型	引用自
id	主键	BIGINT(15)	无
paper_id	试卷号	INTEGER	无
pro_no	试卷题目序列号	TINYINT(3)	无
pro_id	试卷题号	BIGINT(20)	引用自题库

4) 学生测试情况 test

字段名	字段含义	字段类型	引用自
id	无意义主键	BIGINT(25)	无
student_id	学生学号	INTEGER	引用自学生信息表
paper_id	试卷号	INT(11)	无
pro_no	试卷题目序列号	TINYINT(3)	无
choose	学生答案	char	无
is_commit	是否提交	boolean	无
score	得分	TINYINT(4)	无

5) 统计表 statistic

字段名	字段含义	字段类型	引用自
id	无意义主键	BIGINT(20)	无
pro_id	题号	BIGINT(20)	question_bank
done	做该题的总人数	TINYINT(4)	无
a	选 A 的/选 T 的	TINYINT(4)	无
b	选 B 的/选 F 的	TINYINT(4)	无
c	选 C 的	TINYINT(4)	无
d	选 D 的	TINYINT(4)	无

3.4 逻辑架构视图

3.4.1 职责划分与职责确定

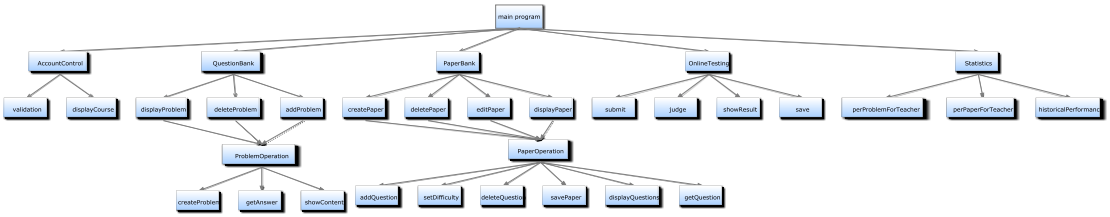

FIGURE 3 主程序/子程序体系结构

3.4.2 系统的环境表示

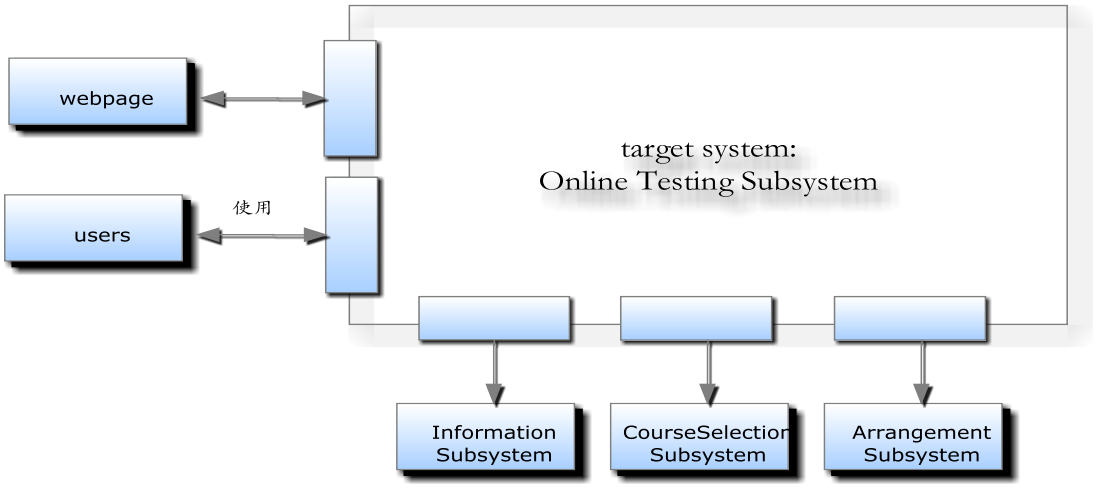

FIGURE 4 在线测试子系统的体系结构环境图

4 选定设计模式说明

4.1 设计模式介绍

4.1.1 MVC 模式的问题

MVC 模式本身要求所有程序运行于单一位置，或者通俗地说，运行于一台计算机上。但对于 web 应用而言这是不可能的，因此整个应用程序被划分为客户端/服务端两部分。大多数情况下，V(view)被分配在客户端，Model 被分配于服务端，而 Control 则既可以分配在服务端又可以在客户端。而这样的分配往往与最后部署时的状态有关，比如如果客户端较差那么就只应该在客户端分配 View 而把 Control 和 Model 完全放置在服务端，这样就构成了 thin-client；如果网络环境比较差，则应该一次性将数据保存在客户端上，即将 V,C 和一部分 M 放置于客户端上，以减少网络请求数量，这样就构成了 fat-client。但部署时的状态往往是开发时无所预知的，于是这样的分割就成了一个很大的问题。

4.1.2 FWAP 模式

fwap (Flexsible Web-Application Partitioning)设计模式主要对 MVC 模式进行了改造，使得它能够克服 MVC 模式本身必需在单一地点运行的这个缺点，从而能够支持必需分开在服务端和客户端运行的网络程序。对于 web 应用而言，应用必需被分割为服务端/客户端两部分。由于开发过程中往往不能确定部署应用时的客户端质量、网络聚集程度等不确定信息，从而对在设计过程中服务端/客户端的分离方式造成了很大的麻烦。fwap 一定程度上克服了这样一个困难，让服务端/客户端的分离能够被延迟到部署时决定。

fwap 推荐了一种介于 thin-client 和 fat-client 之间的 dual-mvc 方法以及一些分配模块的相应的技术，使得这样的分割问题可以推迟到部署时决定但又不用修改程序代码。

4.2 模式应用

4.2.1 FWAP 模式应用

Fwap 模式可以很大程度上应用于我们的网页部署中。

Fwap 模式是专门为设计网页应用这样客户端与服务端分离的应用提出的，在我们的应用中可以帮助我们决策究竟哪一部分应该放到服务器端而哪一部分应该放到客户端。

4.2.2 YII 框架

Yii 框架使用了 Web 开发中广泛采用的模型-视图-控制器(MVC)设计模式。 MVC 的目标是将业务逻辑从用户界面的考虑中分离，这样开发者就可以更容易地改变每一部分而不会影响其他。 在 MVC 中，模型代表信息(数据)和业务规则；视图包含了用户界面元素，例如文本，表单等； 控制器则管理模型和视图中的通信。

除了 MVC, Yii 还引入了一个前端控制器，叫做 应用，它表示请求处理的执行上下文。应用处理用户的请求并将其分派到一个合适的控制器以继续处理。

4.2.3 MVC@YII 框架

下面的示意图展示了 Yii 应用的静态结构：

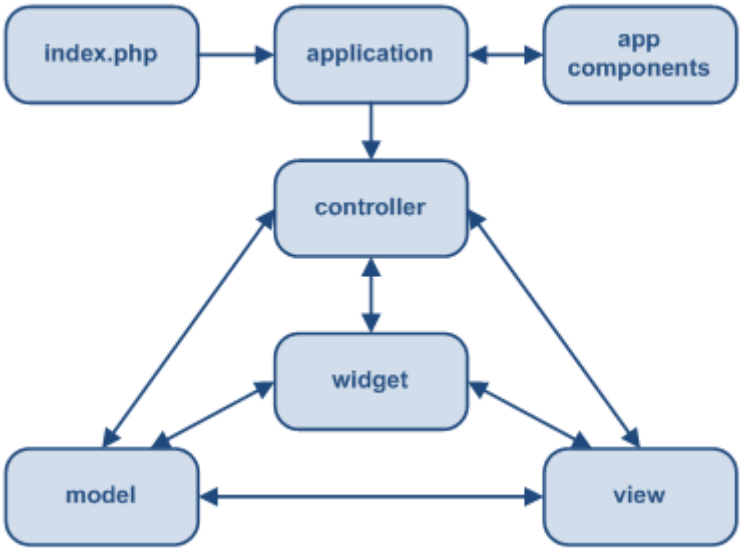

FIGURE 5 YII 应用静态结构

Yii 应用中，执行如下流程：

1. 用户发出了访问 URL http://www.example.com/index.php?r=post/show&id=1 的请求，Web 服务器通过执行入口脚本 index.php 处理此请求。
2. 入口脚本创建了一个 应用 实例并执行。
3. 应用从一个叫做 request 的 应用组件 中获得了用户请求的详细信息。
4. 应用在一个名叫 urlManager 的应用组件的帮助下，决定请求的 控制器 和 动作 。在这个例子中，控制器是 post，它代表 PostController 类；动作是 show ，其实际含义由控制器决定。
5. 应用创建了一个所请求控制器的实例以进一步处理用户请求。控制器决定了动作 show 指向控制器类中的一个名为 actionShow 的方法。然后它创建并持行了与动作关联的过滤器（例如访问控制，基准测试）。 如果过滤器允许，动作将被执行。
6. 动作从数据库中读取一个 ID 为 1 的 Post 模型。
7. 动作通过 Post 模型渲染一个名为 show 的 视图。

8. 视图读取并显示 Post 模型的属性。

9. 视图执行一些 小物件。

10. 视图的渲染结果被插入一个 布局。

11. 动作完成视图渲染并将其呈现给用户。

如下图所示：

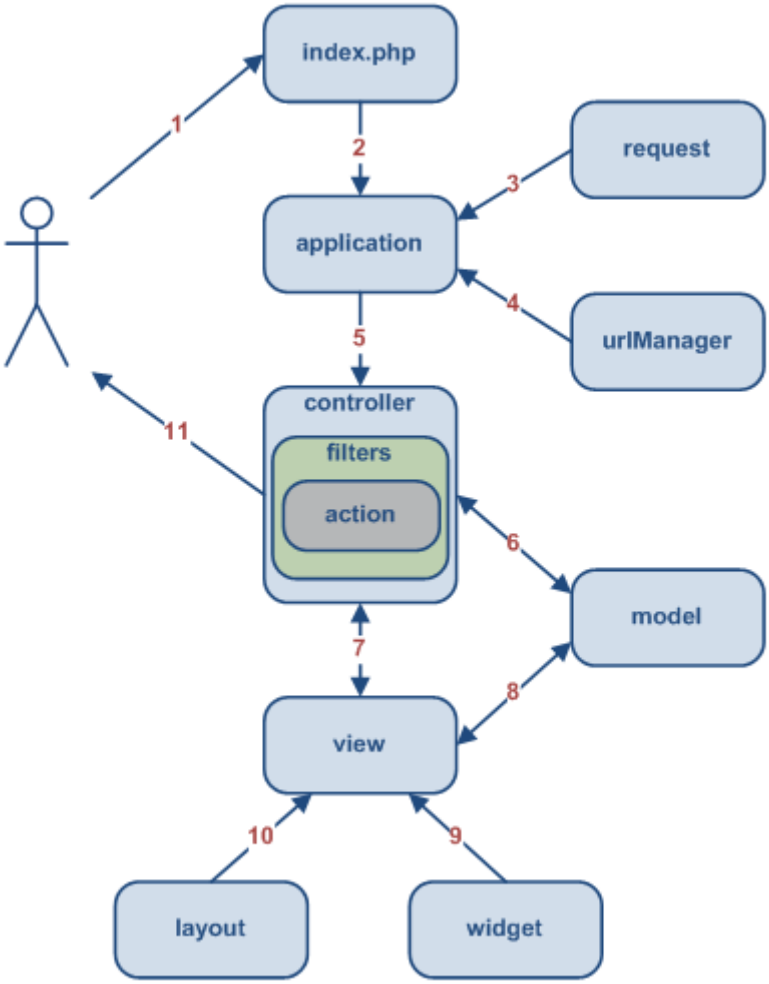

FIGURE 6 YII 经典工作流

4.3 模式优劣势分析

4.3.1 优势分析

能够极大程度上的适应多变的网络程序运行环境，理性的进行 MVC 模式对于客户端服务端的分割。在整个应用的初始设计过程中不需要考虑多种运行环境带来的分割问题。

将 FWAP 模式应用于已经应用了 MVC 模式的 Yii 框架，有如下优势：

1. 性能

   Yii 可能市场上主流 PHP 框架里面相对性能最好的一个，Yii 对于类装载的机制充分利用了 PHP 5.3 的优势，每一个类只有当第一次被用到才被载入，并不是所有框架都“按需载入”从而获得性能上的优势。

2. widget

   widget 机制为开发提供了自然灵活的代码封装和重用，使其在 Yii 里面的实现很自然清晰。

3. 恰到好处的 Model 层设计

   Yii 的 Model 相对 Symfony 这样的框架来说被设计得比较薄，没有使用 ORM，保证了框架整体的小巧，model 层调用产生的 overhead 比如 Symfony 这样的框架要小，Yii 使用直接的 DAO 和 Active Record 来和数据层交互，感觉更加实用，没有特别的学习曲线。

4. 平滑扩展

   要扩展 Yii 或者引入第三方库比较容易，YII 里的扩展没有给开发人员预设很多约定，有一些框架若需扩展就需要遵守“框架”本身的一些个约定，而 YII 的扩展基本就是纯 php 的扩展方式，本质上就是 include 一下，无需太多配置，组建（component）机制为扩展提供了很好的全局支持，一个扩展可以作为一个组建被引入到需要的地方，当然这不是唯一的方式

5. 功能设计

相对更加轻量的 php 框架如 CodeIgniter，Yii 所提供的一般性功能的设计水准比较高，考虑比较全面，工作在 CodeIgniter 上，在某些情况下你可能碰到框架过于简单，对部分功能缺少深入设计。

4.3.2 劣势分析

对于 FWAP 的引入，因为 Fwap 的根本原理在于维护客户端对于服务端数据和控制的缓存，从而调配客户端和服务端任务的分割。但当服务端动态程度超过一定程度时，这样的缓存会造成客户端的延时或者错误，有时会适得其反。

因此 fwap 必需谨慎的处理缓存问题，这给设计带来了一些不便。

5 备选设计模式

5.1 OBSERVER 模式

5.1.1 设计模式介绍

观察者<Observer>模式是软件设计模式的一种。在此模式中，一个目标物件管理所有相依于它的观察者物件，并且在它本身的状态改变时主动发出通知。这通常透过呼叫各观察者所提供的方法来实现。此种模式通常被用来制作事件处理系统及降低信息发送者和信息订阅者之间的耦合。

5.1.2 模式应用

Observer 模式适用于如下三种情况：

1. 当抽象个体有两个互相依赖的层面时。封装这些层面在单独的物件内将可允许单独地去变更与重复使用这些物件，而不会产生两者之间耦合过深的问题。
2. 当其中一个物件的变更会影响其他物件，却又不知道多少物件必须被同时变更时。
3. 当物件应该有能力通知其他物件，又不应该知道其他物件的调用细节时。

在本测试子系统中，Observer 模式可用于某模型更新时通知所有已注册的物件。

例如，可以将学生注册为成绩、课程对象的 Observee，当这些对象有更新时，已注册的对象即可立即获取通知，从而避免轮询所带来的开销及降低耦合。

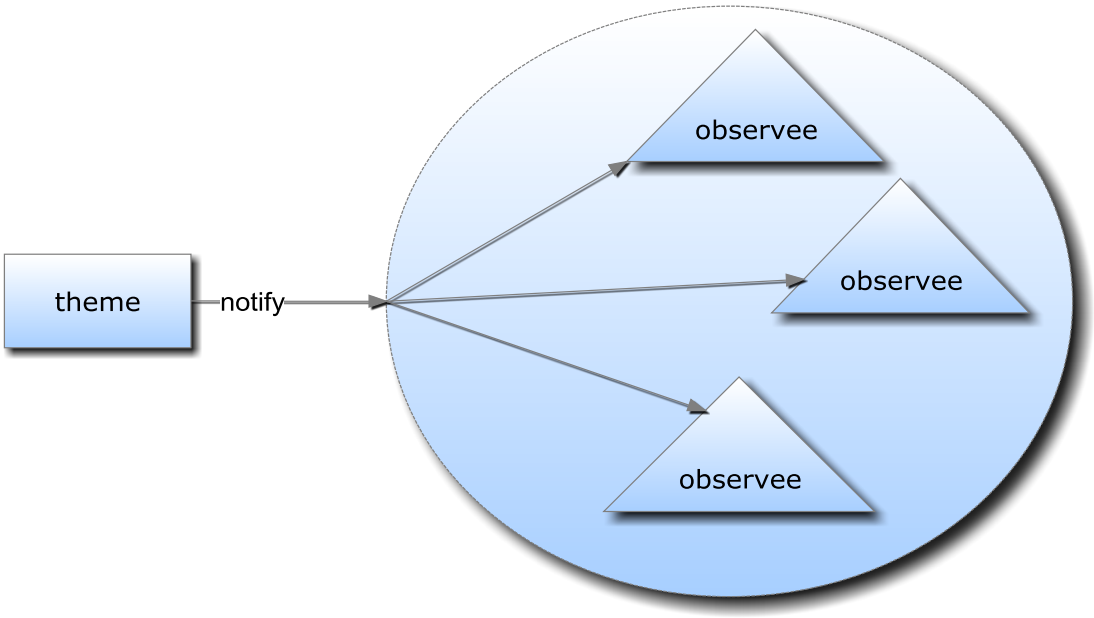

Figure 7 Observer 模式模型

Observer 模式被广泛用于 GUI 模型甚至于 MVC 中。MVC 的 view 和 controller 模式事实上就是观察者和被观察者的关系。在具体功能模块中也可以采用观察者模式，以实现通知推送等功能。

5.2 FACADE 模式

5.2.1 设计模式介绍

外观(Façade)模式提供一个高层次的接口，使得子系统更易于使用。从客户程序的角度来看，Facade 模式不仅简化了整个组件系统的接口，同时对于组件内部与外部客户程序来说，从某种程度上也达到了一种“解耦”的效果——内部子系统的任何变化不会影响到 Facade 接口的变化。

Facade 设计模式更注重从架构的层次去看整个系统，而不是单个类的层次。Facade 很多时候更是一种架构设计模式。需要注意的是 Facade 设计模式并非一个集装箱，可以任意地放进任何多个对象。Facade 模式中组件的内部应该是“相互耦合关系比较大的一系列组件”，而不是一个简单的功能集合。

5.2.2 模式应用

Facade 模式可以隐藏组件背后复杂的流程，而提供给客户简单的流程。例如对于本子系统中自动生成试卷业务，需要有如下流程：

1. 选择试卷生成参数
2. 选择题库
3. 设置生成题数和难度
4. 注册试卷到试卷库

等较长流程。Facade 模式可以将这些步骤包装起来，只想外界提供一个单独的接口，并在内部指定默认参数：

“生成默认试卷”

即可。当然 Facade 模式并不会将低层接口设为不可见，当用户想自行定制时可以越过高层接口调用低层接口。

5.3 职责链模式

5.3.1 设计模式介绍

职责链模式（China of Responsibility）使多个对象都有机会处理请求，从而避免请求的送发者和接收者之间的耦合关系 。Chain of Responsibility(CoR) 是用一系列类(classes)试图处理一个请求 request,这些类之间是一个松散的耦合,唯一共同点是在他们之间传递 request. 也就是说，来了一个请求，A 类先处理，如果没有处理，就传递到 B 类处理，如果没有处理，就传递到 C 类处理，就这样象一个链条(chain)一样传递下去。职责链模式是灵活的，因为每一个链节点都能控制是否将请求传递下去。

在一个请求只有一个职责的时候，职责对象可以判断请求条件是否符合自身条件，不符合的话则交给下一个链处理。

该模式有以下优点：

1. 降低耦合度

职责链可以简化相互之间的连接，他们仅需要保持一个指向其后继者的引用，而不需保持它所有候选接受者的引用。使得一个对象无需知道是其它哪一个对象处理其请求，对象仅需知道该请求会被正确的处理，接收者和发送者都没有对方的明确的信息，且链中的对象不需要知道链的结构。

2. 增强了给对象指派职责的灵活性

当在对象中分配职责时，职责链给你更多的灵活性，你可以通过在运行时可对该链进行动态的增加或者修改来增加或者改变处

理请求的那些职责，你可以将这种机制与静态的特例化处理对象的继承机制结合起来使用。

5.3.2 模式应用

如试卷生成业务：在试卷生成中，可以由老师来挑选题目组成试卷，也可以由电脑从题库中随机抽取题目组成试卷，甚至可以部分试题由老师挑选，部分电脑生成。通过职责链模式，大大增强了整个系统的应用范围。

让整个子系统操作如一条流水线。但是，对于本系统，存在一些因素会使该模式应用较牵强：

1. 链式结构不明显：

对于我们的子系统，共有四大业务，每个业务又有多个子业务，业务内部的链式结构较明显，但业务与业务之间的链式结构不够明显，没有必须采取职责链模式的需求。

2. 效率低：

一个请求的完成可能要遍历到最后才可能完成。因为无法预知来自外界（客户端）的请求是属于哪种类型，如果碰到它不能处理的请求需要遍历到最后才会放弃。

3. 扩展性差：

在该模式中，一定要有一个统一的接口 Handler，这是局限性所在。

6 运行设计

6.1 运行模块组合

本子系统按照功能划分模块，客户端为浏览器。功能模块之间相互不会共享界面，后台程序只共享建立数据库连接的方法。

6.2 运行控制

教务处选课中心工作人员、师生以及系统管理员根据不同的身份，具有不同的登录权限来登录教务系统平台。不同的权限限制了用户的控制范围。

对于在线测试子系统而言，在教师操作题库、试卷或者查看统计时，其控制信号主要来源于教师的对网页的鼠标点击和相关输入；对于学生测试或查看历史成绩而言，其控制信号主要为学生对相关网页的功能按钮的点击；对系统管理员而言，其控制信号和老师相同。

6.3 运行时间

1. 响应时间：

   要求响应较快，一般 2s 以内。

2. 更新处理时间：

   鉴于本系统并不像金融交易系统一样(瞬息万变的金融数据，即使相差甚微也会对决策作出重大影响从而导致巨大的盈亏)，及时更新功能作为可选项。教师可以自行设置是否每隔一定时间更新一次统计查看页面以保证看到的数据尽量是最新的，且教师勾选该项时会提醒这会对数据库造成压力从而导致操作流畅性有一定程度的下降。

7 系统出错处理设计

7.1 出错信息

突发情况	对用户的影响	应对措施
突然断电或断网	若教师在生成试卷，则丢失了已经被选中的题目信息； 若学生正在测试，则丢失了已经答好的题，且有超时危险。	对生成实际操作中的每一次添加/删除题目操作即更新数据库；对学生测试中的每一次答题更新数据库。
数据库连接不上	在用户量很大时导致 数据库访问读写效率降低。导致用户数据库连接不上，无法更新自己的操作。	修改数据库配置，限制同一用户两次执行搜索的时间间隔。
SQL 语句执行错误	用户恶意破坏	过滤恶意 SQL 语句
服务器崩溃	登不上该网站	定期维护服务器
磁盘损坏	数据丢失	周期性备份

7.2 补救措施

1）后备技术

按照一定周期，备份数据库，并且将其储存在更加稳定的介质上，比如磁带。将系统所需的不同数据库部署到不同的计算机上，减小因硬件问题而导致数据全部丢失的可能性。

2）恢复及再启动技术

系统崩溃后，通过系统运行日志记录恢复数据。

7.3 系统维护设计

连接数据库方面，需要在创建数据库连接，销毁数据库连接，执行 sql 语句的模块使用 try catch 语句捕获异常。于是发生此类错误便能很快得知。

维护数据库方面，需要定期检查数据库内部数据结构和内容是否合法，如果遇到异常，直接人工干预。

网络方面需要检测是否有特定的 IP 地址频繁访问系统，不能排除其攻击服务器的可能性。

代码方面，定期统计系统异常事件，当异常频率过高，检查并修正代码。

内部人员如（开户人员，系统维护人员）操作留下操作痕迹，使用权管理层可以定期或不定期地稽核系统。

8 参考文献

[1] Roger S.Pressma. 软件工程—实践者的研究方法. 机械工业出版社. 2007.1

[2]陈娴. 中型在线系统开发实践. 中国铁道出版社. 2007.4

[3] CSAI 架构设计专家. ADMEMS 架构设计. http://www.csai.cn/admems/.

[4] Linda F. Ettinger. Adopting Software Design Patterns in an IT Organizati on: An Enterprise Approach to Add Operational Efficiencies and Strategic Benefits.

[5] Ackerman, L., & Gonzalez, C. (2007). The value of pattern implementations. Dr. Dobb's Journal: The World of Software Development, 28 -32. Retrieved Mar 30, 2011 from Computer Source.

[6] Bhatt, G. D., & Grover, V. (2005). Types of informational technology capabilities and their role in competitive advantage: an empirical study. Journal of Management Information System s, 22(2).

[7] QingSheng Wang, Wen Gao,YueQin Zhang. Application Research of Design Patterns in Virtual Trade Platform.

[8] Liu YuFei, Li Jian. 2008. “Design of online contribution and peer review system based on MVC pattern” .Computer Engineering & Design. No. 2, pp.504-506.

[9] Hemangi Gawand, R.S.Mundada, P.Swaminathan. Design Patterns to Implement Safety and Fault Tolerance. International Journal of Computer Applications (0975 – 8887) Volume 18– No.2, March 2011.

[10] M. Jazayeri, R. Loos, D. R. Musser. Lecture notes in computer science 1776 : Generic programming.