# 

# Code Dream

## 一个编程自学网站

# 架构设计

|  |  |
| --- | --- |
| **成员** | **学号** |
| 卢彦谚 | 201630364818 |
| 吴潘安 | 201630665908 |
| 吴志祥 | 201630665939 |
| 谢旻珊 | 201630665984 |
| 谢珮爽 | 201630665991 |
| 谢哲威 | 201630666035 |
| 李林键 | 201630406969 |

目录

[Code Dream 1](#_Toc25904)

[一个编程自学网站 1](#_Toc17102)

[架构设计 1](#_Toc7790)

[1引言 4](#_Toc23700)

[1.1目的 4](#_Toc25538)

[1.2范围 4](#_Toc11082)

[1.3术语表 4](#_Toc6063)

[1.4参考资料 5](#_Toc15094)

[2软件系统架构设计概述 5](#_Toc17260)

[2.1软件系统架构设计策略与原则 5](#_Toc4122)

[2.2关键功能性需求 5](#_Toc271)

[2.3非功能性需求及解决方案 9](#_Toc19875)

[2.3.1QAW产物-生成情景 9](#_Toc23609)

[2.3.2效用树 10](#_Toc7029)

[2.3.3性能需求及解决方案（基于ATAM体系结构权衡分析方法） 11](#_Toc1274)

[2.3.4可测试性需求及解决方案 11](#_Toc23084)

[2.3.5安全性需求及解决方案 12](#_Toc20233)

[2.3.6可用性需求及解决方案 14](#_Toc22021)

[2.3.7易用性需求及解决方案 15](#_Toc1925)

[2.3.8可移动性需求及解决方案 20](#_Toc31258)

[2.3.9可修改性需求及解决方案 21](#_Toc15823)

[2.3.9互操作需求及解决方案 23](#_Toc20038)

[3软件系统架构设计 24](#_Toc16457)

[3.1系统分层架构视图 24](#_Toc7542)

[3.2用例视图 25](#_Toc21501)

[3.3逻辑视图 26](#_Toc20272)

[3.3.1逻辑架构图 26](#_Toc9842)

[3.3.2包图 27](#_Toc5775)

[3.3.3类图 28](#_Toc13117)

[3.3.4状态图 28](#_Toc27157)

[3.4部署视图 30](#_Toc25363)

[3.5进程视图 30](#_Toc4855)

[3.6开发视图 32](#_Toc6242)

[3.6.1构件图 32](#_Toc22476)

[3.6.2如何保持架构与代码的一致性 32](#_Toc15805)

[4关键技术设计 33](#_Toc9679)

[4.1公共构件设计 33](#_Toc19614)

[4.1.1构件描述 33](#_Toc29967)

[4.1.2抽取规则 33](#_Toc14159)

[4.2接口设计 34](#_Toc18244)

[4.2.1动作 34](#_Toc14069)

[4.2.2路径（接口命名） 34](#_Toc24098)

[4.2.3版本（Versioning） 34](#_Toc18191)

[4.2.4过滤信息（Filtering） 34](#_Toc27994)

[4.2.5状态码（Status Codes） 35](#_Toc8689)

[4.3数据架构设计 36](#_Toc28085)

[4.3.1主题数据库模型 36](#_Toc28154)

[4.3.2数据访问层设计 36](#_Toc8681)

[4.4UI架构设计 37](#_Toc2456)

# 1引言

## 1.1目的

本文档从架构方面对系统进行综合概述，记录了架构设计团队在确定质量属性场景、重要的功能需求和非功能需求后的产出。可用于后续开发团队对系统架构的了解及架构团队面对更改时能快速了解已有架构并对此做出针对性修改。本文档表述了已对系统的架构方面做出的重要决策。

## 1.2范围

本文档适用于Code dream小组开发的Code dream项目。Code dream项目是一个能够在线学习编程语言的网站，提供了编程语言知识学习、学习进度保存、帖子留言交流等功能。

本文档包括本项目组QAW生成的场景回顾、效用树，对系统架构进行结构权衡分析的ATAM记录表，系统的整体架构设计，系统架构设计的“4+1”视图，防止架构侵蚀的策略，公共构件设计，接口设计和系统的数据架构设计。

## 1.3术语表

|  |  |
| --- | --- |
| **术语** | **术语解释** |
| QAW | QAW 方法是一种用于在创建软件体系结构之前发现质量属性的方法。 |
| ATAM | ATAM 包括“权衡”，因为它不仅描述某个体系结构对特定质量目标的满足程度，而且还提供了对那些属性在体系结构质量中所具有的权衡的深入认识。 |
| “4+1”视图 | “4+1”视图模型从5个不同的视角包括逻辑视图、进程视图、物理视图、开发视图和场景视图来描述软件体系结构。 |
| 敏感点 | 敏感点是一个或多个构件（或之间的关系）的特性 |
| 权衡点 | 权衡点是影响多个质量属性的特性，是多个质量属性的敏感点。 |
| 风险 | 风险是可能导致架构出现问题的可能情况。 |
| 非风险 | 非风险是预测不会造成风险的战术。 |
| 性能（代号P） | 性能是关于时间的，是软件系统满足时序要求的能力。 |
| 可测试性（代号T） | 软件可测试性是指通过测试可以使软件易于演示其故障。 |
| 安全性（代号S） | 安全性是系统保护数据和信息免受未授权访问的能力的度量，同时仍然提供被授权的人和系统的访问。 |
| 可用性（代号A） | 可用性是软件的一种属性，它存在那里并且时刻准备着执行他的任务。当你需要它时。 |
| 易用性（代号U） | 易用性涉及用户完成期望的任务有多容易，还有系统提供的用户支持的种类。 |
| 可移动性（代号M） | 可移动性包括对移动设备的功能支持、提供功能的一致性以及对移动设备的适应性。 |
| 可修改性（代号Mo） | 可修改性是关于变化的，我们对它的兴趣在于变化的变化的成本和风险。 |
| 互操作性（代号I） | 互操作性是指两个或多个系统可以有效交换有意义信息的程度。 |

## 1.4参考资料

1. 《调研报告》 Code dream开发小组
2. 《需求分析》 Code dream开发小组
3. 《用户手册》 Code dream开发小组
4. 《软件构架实践》 Len Bass,Paul Clements,Rick Kazman著 清华大学出版社
5. 《数据库系统概念》 西尔伯沙茨 机械工业出版社
6. 《UML和模式应用》 （美）拉曼(Larman，C.)著 李洋等译 机械工业出版社

# 2软件系统架构设计概述

## 2.1软件系统架构设计策略与原则

为了达到用户方便使用、交互良好且可以跨平台的特性，该项目采用了基于web的MCV架构。同时，鉴于开发小组的开发经验不足，因此在架构设计过程中尽可能采用前后端分离的方式，通过使用Broker架构，在开发过程中使用mock平台作为中间件来规范接口和模拟数据，前后端应用JSON作为数据传输的规范。

前端方面为了达到与用户交互良好且美观的特性，采用了能与用户进行动态交互的JavaScript，并使用bootstrap作为前端框架，以此利用框架封装良好的前端组件。

后端方面，鉴于项目的轻量性，为了快速开发同时不失可维护性，采用了php语言与基于php7的Thinkphp框架，该框架较好地封装了控制器和模型的底层服务，提供对多种数据库的良好支持。此外，本项目以Mysql作为数据库以达到节省经济支出的目的。

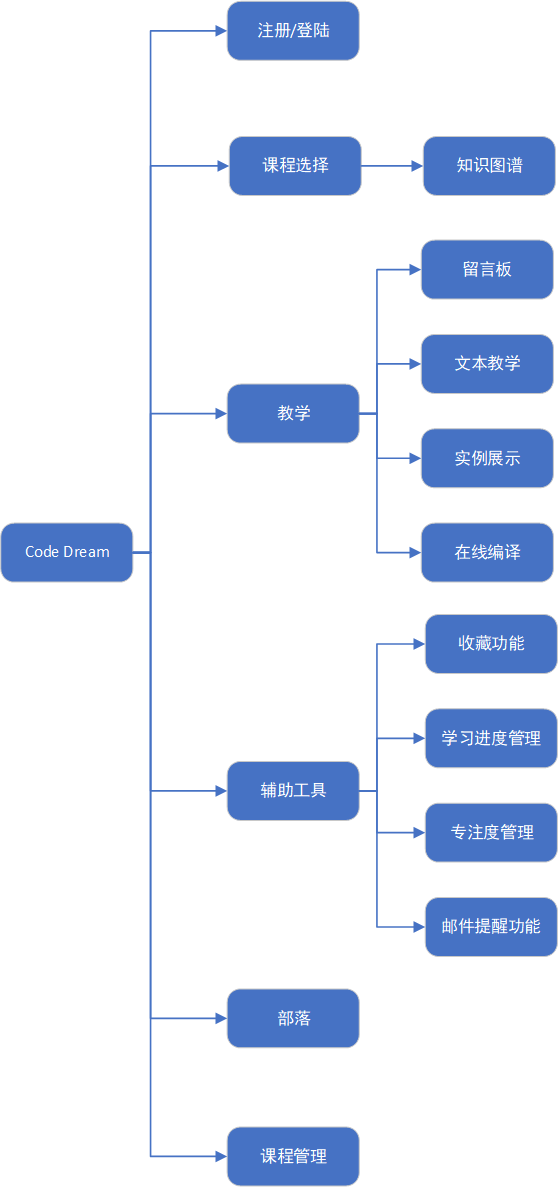
在架构设计的过程中，本架构设计小组以前后端分离的web MVC为基础，在设计时考虑项目的轻量性、开发小组的经验不足、项目工期短等因素，尽可能达到高效、经济的目的。

## 2.2关键功能性需求

本系统包括一个面向游客的子系统、一个面向用户的子系统以及一个面向业务管理员的子系统。游客可以进行课程浏览、在线编译以及可以浏览部分留言板留言。用户可以进行课程浏览、在线编译以及留言板所有留言；并可以对自己的学习情况进行个性化管理，包括可以查看个人学习专注度（专注“森林”），查看个人每日登陆时长，查看个人课程进度，并可以对留言板留言进行收藏等管理功能。业务管理员可以对系统课程进行管理，包括对课程进行增删改操作以及对课程章节进行增删改操作。

本系统的外延包括作为系统管理员的开发者，开发者可以根据用户反馈对系统进行更加具体、全面的更改。

系统功能模块划分如下：



关键功能记录如下：

1.注册/登录功能

1.1功能目标

记录用户个人信息，包括基本信息和系统使用信息，用于提供个性化服务。

1.2功能内容

用户可以进行注册，填写基本信息，系统将保存基本信息；用户注册后使用帐号登录系统，系统将从数据库中读取并同步该用户系统，包括进行网站数据更新和页面重绘；系统将定时及通过时间点（如章节学习完成、退出等）进行用户学习信息同步，更新数据库。

2.学习内容选择功能

2.1功能目标

给用户列举在网站内可以学习的内容，用户可以自由选择课程并进入相关页面进行学习。

2.2功能内容

在网站的侧边栏或顶栏显示主要课程分类或推荐给用户的课程，用户通过点击查看相应的课程简介页面。

3拓扑图功能

3.1功能目标

在网站首页上显示课程的拓扑图，用户可以直观了解课程学习的先后顺序，并且能够通过点击直接进入相关课程的页面。

3.2功能内容

设计课程学习先后顺序的拓扑图，结合国内学院对这些课程的学期安排，以拓扑图+大致学习阶段（如大一、大二等）的形式展示给用户，用户可以从拓扑图上了解到课程的先后学习顺序，并对该学科有整体的把握，对课程及知识间的关系有了解。用户可以通过点击选择拓扑图上相应课程并能够直接进入课程的页面。

4实例展示功能

4.1功能目标

为用户提供代码实例展示，使用户在学习过程中对所学知识有更深的理解，帮助用户更加有效地消化理论知识。

4.2功能内容

实例展示功能根据当前传授内容为用户提供实际操作的完整代码，并对代码进行注释。

5在线编译功能

5.1功能目标

实现一个网页版在线编译器，加深用户对示范代码的理解，方便用户即时实操所学的内容。

5.2功能内容

该在线编译器能够对教程示例中的示范代码进行编译，页面会显示编码结果；用户也可以在编译器中输入自己的代码进行试验。

6留言板功能

6.1功能目标

用户通过留言板的交流，能够更好更有效地进行学习，让别人解答或解答别人心中的疑问，并且所有用户都可以看得到，这样学习起来更有效率。

6.2功能内容

用户可以在留言板上提出自己的学习问题或疑问，其他用户同时也可以看得到并选择是否进行回复解答；用户也可以回答别人在留言板板块留下的问题。

7留言板点赞功能

7.1功能目标

通过留言板发言的用户点赞数，按发言价值权重对用户发言排序，让大多数人认可的发言最先给用户呈现，使教程学习起来更有效率。

7.2功能内容

用户可以在留言板点赞别人或自己的发言，让其他用户更容易注意到。

8用户使用时间统计功能

8.1功能目标

统计用户每日用于学习的时间，以折线图的形式展示，便于用户了解自己的每日学习时间投入及一段时间内学习投入波动变化，调整适宜的学习时间，从而提高学习效果。

8.2功能内容

通过统计获得用户的学习时长，以折线图的形式展示，体现用户每日学习时长及一段时间内的波动变化。

9专注度评估功能

9.1功能目标

评估用户的学习专注度，以种树形式呈现，吸引用户努力提高专注度，最终达到激励用户认真学习的效果。

9.2功能内容

通过统计获得用户的页面停留时长，评估用户的学习专注度，并以种树的形式表现（随着用户在线时间的增长，树苗也会成长）。

10学习进度管理功能

10.1功能目标

管理用户学习进度，以进度条的形式展示，便于用户了解自己在该门课程中的学习进度。

10.2功能内容

通过学习内容章节划分进度，通过用户浏览/掌握章节情况，为课程学习进度加以评估，并以进度条的形式展示。

11学习提醒邮件功能

11.1功能目标

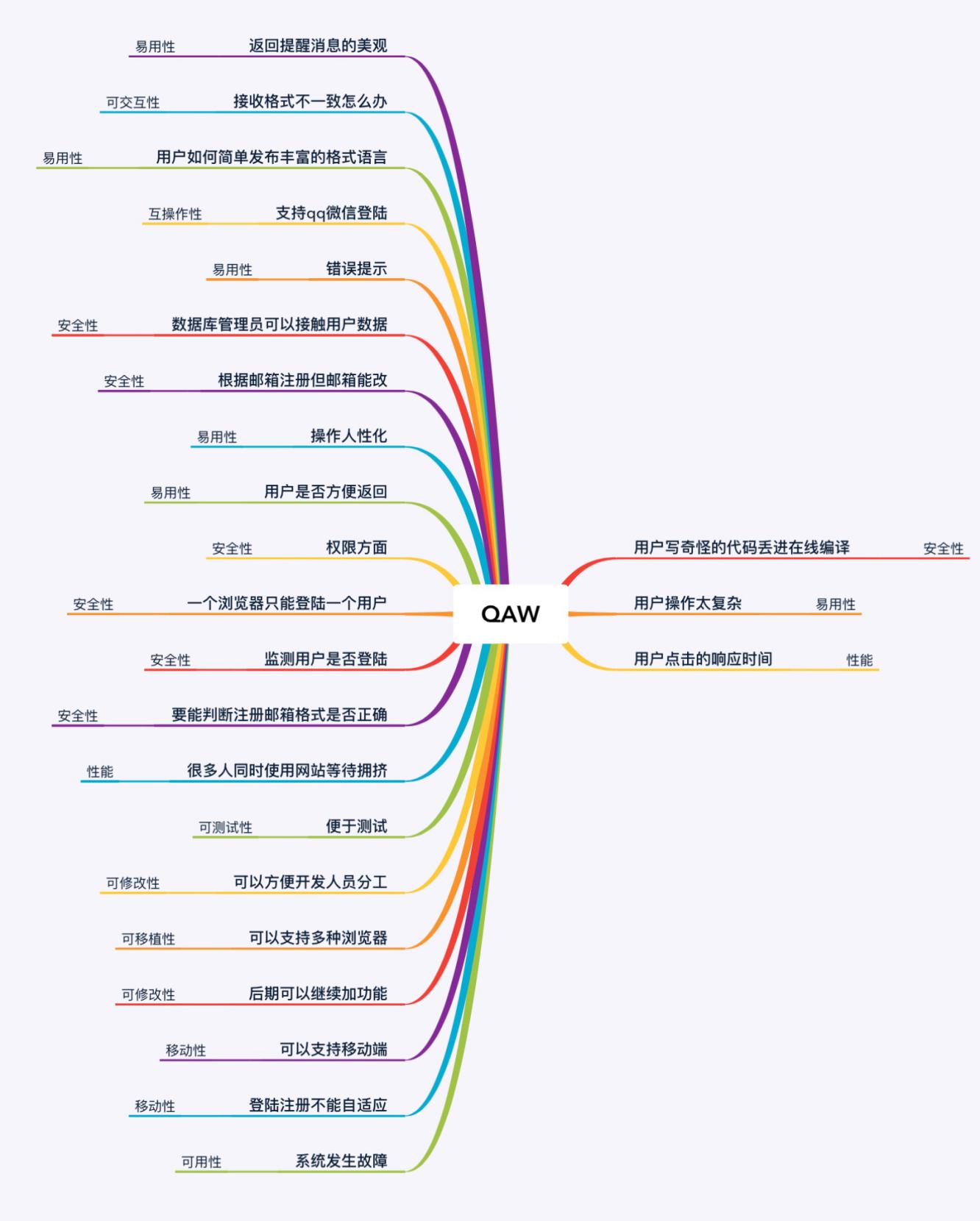
通过系统向用户个人邮箱发送邮件的方式提醒用户保持学习。

11.2功能内容

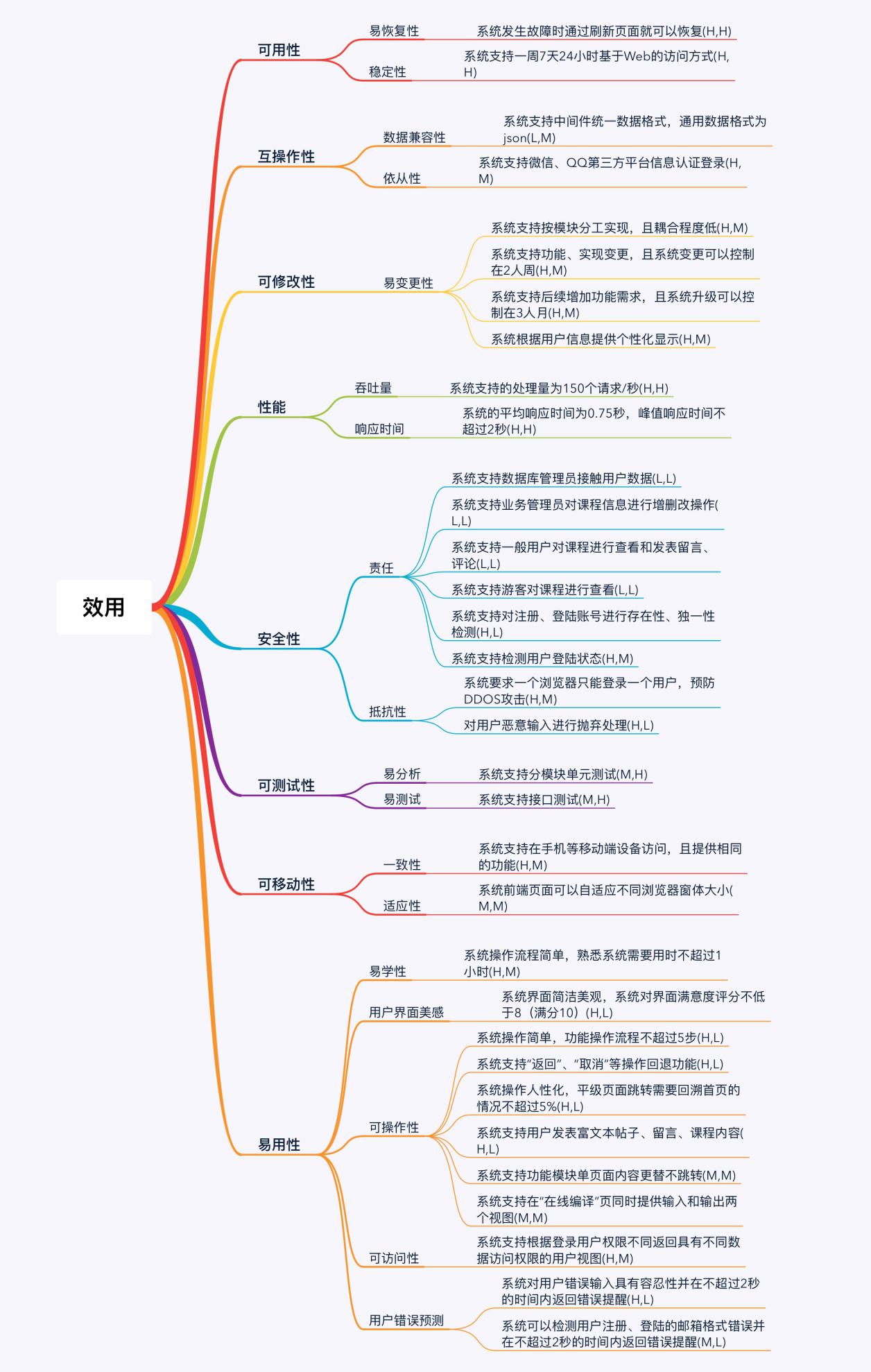
可以在设置中设置学习进度提醒功能，每日定时发送邮件提醒用户学习。

## 2.3非功能性需求及解决方案

### 2.3.1QAW产物-生成情景



### 2.3.2效用树



### 2.3.3性能需求及解决方案（基于ATAM体系结构权衡分析方法）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | P1 | **描述** | 很多人使用网站等待拥挤 | |
| **质量属性** | 性能 | | | |
| **环境** | 一般使用 | | | |
| **刺激** | 用户请求系统数据 | | | |
| **响应** | 系统正常情况下1s返回响应数据,超出响应时间则返回404 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 并行 | S1 | T1 |  |  |
| 增加资源 |  |  |  | N1 |
| 限制执行时间 |  | T2 |  |  |
| 减少采样率 | S2 |  |  |  |
| **解释** | 1. 为了提高网站的响应性能，提高吞吐量，采用并行程序设计思想设计网站，例如并行加载JS，但是执行的时候却按顺序执行，提高网站速度。（S1） 2. 可以通过提高应用服务器性能，主要包括CPU、存储、I/O访问、内存、TCP/IP连接数等等来解决网站的拥堵的问题，使用更好的硬件资源。（N1） 3. 为了网站系统的稳定性，我们需要限制执行时间控制流量，避免高峰期导致的网站崩溃。（T2） | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | P2 | **描述** | 降低用户点击的响应时间 | |
| **质量属性** | 性能 | | | |
| **环境** | 正常操作运行状态 | | | |
| **刺激** | 用户点击请求等待系统响应 | | | |
| **响应** | 系统按照用户的请求正常时间内响应数据 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 增加资源 |  |  |  | N2 |
| 减少响应执行时间 |  | T3 |  |  |
| **解释** | 1.为了让用户有更好的体验效果，可以通过减少网站的响应时间，具体可以减少HTTP的响应请求，可以合并JavaScript脚本文件，合并CSS样式文件，或者利用浏览器的Cache功能减少重复文件的请求。（T1） | | | |
| **架构图** |  | | | |

### 2.3.4可测试性需求及解决方案

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | T1 | **描述** | 开发人员对已完成的单个或多个组件能进行快速或完整的测试 | |
| **质量属性** | 可测试性 | | | |
| **环境** | 开发过程中，有已经完成的一个或多个可用组件与连接件 | | | |
| **刺激** | 对部分组件进行常规操作 | | | |
| **响应** | 组件输出预期结果或调用其它设定中的组件 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 特定接口 |  |  |  |  |
| 限制接口复杂性 |  | T1 |  |  |
| 使用本地数据库 |  |  | R1 |  |
| **原因** | 1. 为了达到前后端分离的策略，我们使用前后端利用接口进行通信的策略，由此使用了特定接口的战术。 2. 为了降低整体的复杂性以达到可测试性的目的，我们使用了限制接口复杂性的战术。而权衡点在于单个接口复杂性小，虽然提高了内聚性，但为了完成较为复杂的功能时需要一次调用多个接口，降低了系统的性能，也提高了耦合度（T1）。 3. 在开发时使用本地的数据库进行开发，优点在于开发时便捷程度高，缺点在于最终部署时可能会有额外的环境配置问题。（R1） | | | |
| **架构图** |  | | | |

### 2.3.5安全性需求及解决方案

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | S1 | **描述** | 数据库管理员尝试访问用户数据并进行操作 | |
| **质量属性** | 安全性 | | | |
| **环境** | 常规操作 | | | |
| **刺激** | 用户数据访问申请 | | | |
| **响应** | 已认证的管理员的申请被批准 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 限制数据访问 |  | T1 |  |  |
| 限制数据暴露 |  | T2 |  |  |
| 识别 |  |  |  | N1 |
| 授权 |  |  |  | N2 |
| 验证 |  |  |  | N3 |
| **原因** | 1.限制数据访问与数据暴露能增加安全性，但这可能会增加合法的数据访问的操作复杂度。 (T1, T2) | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | S2 | **描述** | 一个游客想要进行用户登录操作 | |
| **质量属性** | 安全性 | | | |
| **环境** | 常规操作 | | | |
| **刺激** | 登录申请 | | | |
| **响应** | 合法用户的申请被批准，其他被拒绝。 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| Identify |  |  |  | N4 |
| Authorization |  |  |  | N5 |
| Authentication |  |  |  | N6 |
| **原因** |  | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | S3 | **描述** | 系统尝试检测用户登录状态 | |
| **质量属性** | 安全性 | | | |
| **环境** | 常规操作 | | | |
| **刺激** | 系统尝试检测用户登录状态 | | | |
| **响应** | 系统从用户端得到返回结果，判断是否登录以及所登录的用户 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| Ping/echo |  | T3 |  |  |
| **原因** | 1.系统不能频发发起检测请求，否则会浪费资源（T3） | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | S4 | **描述** | 一个用户想要在一台设备上登录多个账号发起DDoS攻击 | |
| **质量属性** | 安全性 | | | |
| **环境** | 常规操作 | | | |
| **刺激** | 一个用户想要在一台设备上登录多个账号 | | | |
| **响应** | 系统拒绝多账号登录请求，预防DDoS攻击 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 授权 | S1 |  |  |  |
| **原因** | 1.同一时间系统对一台设备只授权登录一个账号，如果不做限制，恶意用户就可以在一台设备上登录大量账号发起攻击。（S1） | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | S5 | **描述** | 用户输入恶意内容，利用系统的在线编译运行功能攻击系统 | |
| **质量属性** | 安全性 | | | |
| **环境** | 常规操作 | | | |
| **刺激** | 输入恶意内容 | | | |
| **响应** | 系统拒绝编译执行，抛弃恶意输入，通知用户与管理员 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 检测用户输入 | S2 | T4 |  |  |
| 通知管理员 |  | T5 |  | N7 |
| **原因** | 1. 若不进行输入检测，系统就有可能会被恶意输入所攻击。（S2） 2. 检测功能如果实现得不高效，会拖累性能。（T4） 3. 若通知管理员太频繁，管理员可能会疲于管理大量消息。（T5） | | | |
| **架构图** |  | | | |

### 2.3.6可用性需求及解决方案

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | A1 | **描述** | 检测并从软件错误中恢复 | |
| **质量属性** | 可用性 | | | |
| **环境** | 一般使用 | | | |
| **刺激** | 用户正常使用系统 | | | |
| **响应** | 网站能够正常检测错误并触发响应的处理机制。 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 重试刷新 |  |  |  |  |
| 回滚 |  | T1 |  |  |
| 热备份 |  |  |  |  |
| 故障恢复 |  |  |  |  |
| 检测有无故障 |  | T2 |  |  |
| **原因** | 1. 为了在运行时捕获并检测故障，需要在开发中增加检测故障的机制，因此采用了检测有无故障的战术，但检测故障会带来性能的下降，因为需要额外的时间用于检测可能的错误。（T2） 2. 发生错误后使用回滚战术和重试刷新战术，但这会使得用户输入而未保存的数据丢失，带来部分易用性的下降。（T1） 3. 使用热备份战术对数据进行备份，防止主数据库上的数据丢失。 | | | |
| **架构图** |  | | | |

### 2.3.7易用性需求及解决方案

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | U1 | **描述** | 系统容易学习与熟悉 | |
| **质量属性** | 易用性 | | | |
| **环境** | 初次使用 | | | |
| **刺激** | 新用户首次接触本系统 | | | |
| **响应** | 新用户熟悉系统需要用时不超过1小时 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 用户模型 | S1 | T1 |  | N1 |
| 任务模型 | S2 | T2 |  | N2 |
| **原因** | 1.为了预测并拟合目标用户可能的行为模式，可以设计3~6个用户模型代表本产品的目标用户群体。  2.建立用户模型有利于分析目标用户的行为模式，同时引入的行为模式的变化会影响可修改性（T1）。  3.为了进行功能的合理设置，采用任务模型预测目标用户任务的上下文，从而确定一些系统功能。  4.建立任务模型引入的任务上下文的变化会影响可修改性（T2）。 | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | U2 | **描述** | 系统界面简洁美观 | |
| **质量属性** | 易用性 | | | |
| **环境** | 一般使用 | | | |
| **刺激** | 用户正常使用系统 | | | |
| **响应** | 对系统用户进行回访，在样本数大于50的情况下，评分不低于8（总分10） | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 用户模型 | S3 |  |  | N3 |
| **原因** | 1.为了评估目标用户的审美需求，可以设计3~6个用户模型代表本产品的目标用户群体。 | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | U3 | **描述** | 系统功能方便使用 | |
| **质量属性** | 易用性 | | | |
| **环境** | 一般使用 | | | |
| **刺激** | 用户需要使用某个功能时 | | | |
| **响应** | 用户从当前页面到目标使用功能功能使用的操作流程不超过5步 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 任务模型 | S4 | T3 |  | N4 |
| 系统模型 | S5 | T4 |  | N5 |
| **原因** | 1.为了预测用户需要使用的功能的上下文以及设置合理的UI架构支持用户任务上下文，采用任务模型。  2.使用任务模型可以确定一些用户任务的上下文，同时引入的上下文变化会影响可修改性（T3）。  3.为了制定系统在用户指令下给出适当的反馈，采用系统模型。  4.系统模型规定了用户需要使用某个功能后将得到的反馈，包括跳转、执行、刷新等，同时引入的系统响应的变化会影响可修改性（T4）。 | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | U4 | **描述** | 系统支持部分用户操作回退 | |
| **质量属性** | 易用性 | | | |
| **环境** | 一般使用 | | | |
| **刺激** | 用户希望取消当前操作 | | | |
| **响应** | 用户点击“取消”或“返回”按钮，系统就会取消用户当前操作并回退到上一步状态 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 用户主动性 | S6 |  | R1 |  |
| 任务模型 | S7 |  |  | N6 |
| **原因** | 1.系统应该支持用户从其他系统带来的惯性操作，并被视为一般要求，采用支持用户主动性，允许用户取消当前操作。  2.取消当前操作应该是有限度的，用户如果可以随意取消当前操作，可能会引起系统错误（R1）。  3.系统应该设置用户取消当前操作的回退路径作为用户任务上下文，可以采用任务模型。  4.用户取消当前操作的回退路径通常已经包含在已确定的系统状态里，不构成风险（N6）。 | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | U5 | **描述** | 系统页面间跳转人性化，操作方便 | |
| **质量属性** | 易用性 | | | |
| **环境** | 一般使用 | | | |
| **刺激** | 用户希望从当前页面跳转到平级的其他页面 | | | |
| **响应** | 平级页面跳转需要回溯首页的情况不超过5% | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 任务模型 | S8 | T5 |  | N7 |
| 用户主动性 | S9 |  | R2 |  |
| 系统模型 | S10 | T6 |  | N8 |
| **原因** | 1.系统页面间跳转应该考虑系统页面跳转状态，可以采用任务模型制定跳转任务的上下文，包括跳转前页面状态、跳转任务指令发出模式、跳转后页面状态。  2.任务模型确定了用户页面跳转的操作以及对应的系统上下文，同时引入的操作和上下文变化会影响可修改性（T5）。  3.系统应该支持用户主动的进行页面间跳转作为页面自动跳转的补充，采用支持用户主动性。  4.用户主动性的引入可能导致用户在系统未能达到预期响应，如在1秒内完成页面跳转的情况时进行其他操作，可能引起系统崩溃（R2）。  5.系统页面间跳转需要确定用户发出页面跳转指令后的系统响应，包括跳转的动作提醒和确定跳转时间，采用系统模型。  6.系统模型可以确定系统应该在用户给出跳转指令的1秒内完成页面跳转，同时引入的响应要求会影响系统的性能。 | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | U6 | **描述** | 系统模块页面内容更替 | |
| **质量属性** | 易用性 | | | |
| **环境** | 一般使用 | | | |
| **刺激** | 用户希望在同一个页面看到尽量多的内容而不需要页面跳转 | | | |
| **响应** | 页面内容更替，内容更替时间不超过1秒 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 任务模型 | S8 |  |  | N9 |
| 系统模型 | S10 | T7 |  | N10 |
| 数据独立于视图 | S11 | T8 | R3 |  |
| **原因** | 1.模块页面内容替换作为单页面应用场景，系统应该考虑进行页面内容替换的任务上下文，采用任务模式。  2.任务模式确定了内容更换并不发生页面间跳转的状态变化，而发生页面本身状态变化。  3.模块内容更替对系统的响应要求是在保持页面主体不变的同时对页面的内容部分进行更替，同时更替时间应该在1秒内，以免用户误以为系统错误或崩溃，这对系统性能做出了要求，此处采用的是系统模型（T7）。  4.模块内容更替作为单页面应用场景，系统可以采用数据独立于视图，根据用户选择，在客户端维持一个模块视图不变的同时，为视图注入不同的数据提供页面内容。  5.采用数据独立于视图引入了数据和视图的分离，会影响可修改性，同时数据在后台和前端间的传输对系统性能提出了考验（T8）。  6.数据独立于视图引入了数据和视图的分离，当视图崩溃或数据丢失时，系统都会不能提供它应该有的功能（R3）。 | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | U7 | **描述** | 面向不同权限的用户提供不同的内容权限 | |
| **质量属性** | 易用性 | | | |
| **环境** | 一般使用 | | | |
| **刺激** | 管理员希望可以对课程内容进行编辑，希望一般用户只能查看课程 | | | |
| **响应** | 管理员可以对课程内容进行更改，一般用户只能查看课程，两者的课程内容是基于同一份数据的，管理员更改课程内容的同时不会影响用户查看课程 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 用户模型 | S8 | T9 |  | N11 |
| 任务模型 | S9 | T10 |  | N12 |
| 数据独立于视图 | S10 | T11 | R4 |  |
| 分发管理 | S11 | T12 | R5 |  |
| **原因** | 1.系统需要考虑目标用户群体划分，可以设计3~6个用户模型用于理解不同用户群体对系统的需求差异，并分析不同用户群体的行为模式差异。  2.采用用户模型有利于确定不同用户群体的行为模式差异，尤其是对于管理员用户希望的对系统进行的更高交互级别的操作行为模式、习惯进行适配。  3.用户模型进行群体划分引入了更多的功能设计要求，会影响可修改性（T9）。  4.系统应该考虑不同权限用户的操作引入的上下文差异，包括进行登录操作后返回的用户视图差异，采用任务模型。  5.任务模型引入的任务上下文差异变化会影响可修改性（T10）。  6.管理员和一般用户面对的课程内容应该是基于同一数据的，但课程内容等可管理修改内容应该因用户权限不同及用户任务不同呈现在不同的用户视图，如管理员希望得到可修改内容的视图，而一般用户不需要，系统可以采用数据独立于视图。  7.采用数据独立于视图引入了数据和视图的分离，会影响可修改性，同时数据在后台和前端间的传输对系统性能提出了考验（T11）。  8.数据独立于视图引入了数据和视图的分离，当视图崩溃或数据丢失时，系统都会不能提供它应该有的功能（R4）。  9.系统应该在管理员对课程内容进行编辑的同时保证系统对已查看课程内容的用户继续使用而不至于崩溃，系统可以采用分发管理。  10.系统采用分发管理可以确保课程内容生产和消费在一定程度上解耦，但同时引入的中间构建对系统的可修改性以及性能提出了要求（T12）。  11.如果中间件崩溃将导致系统无法正常提供应该提供的功能（R5）。 | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | U8 | **描述** | 用户输入格式错误的信息时，在两秒内返回错误提醒。 | |
| **质量属性** | 易用性 | | | |
| **环境** | 运行时 | | | |
| **刺激** | 用户输入格式错误的邮箱 | | | |
| **响应** | 系统显示错误提示 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 维持任务模型 | S12 |  |  | N13 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | U9 | **描述** | 当用户输入了不存在的账号时，系统给予对应的“账号不存在”的反馈。 | |
| **质量属性** | 易用性 | | | |
| **环境** | 运行时 | | | |
| **刺激** | 用户输入了一个不存在的账号进行登录 | | | |
| **响应** | 系统对用户的错误输入进行判断，并返回具体的错误提示。 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 维持任务模型 | S13 |  |  | N14 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | U10 | **描述** | 用户发表留言、管理员增添修改课程内容时，可以使用丰富的文本、图片添加等易用性操作，并且迅速掌握使用方法。 | |
| **质量属性** | 易用性 | | | |
| **环境** | 运行时 | | | |
| **刺激** | 用户发表留言、管理员增添修改课程内容 | | | |
| **响应** | 用户可以很容易地使用丰富的文本编辑、图片添加的操作 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 提供用户支持 | S14 | T13 | R6 |  |
| 维持任务模型 | S15 |  |  | N15 |
| **原因** | T：实现丰富的用户操作涉及引入解析包，会增加网页的加载时间，导致性能下降。  R：实现用户支持的技术版本过多，不统一，兼容性差；引入解析包，导致系统性能下降 | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | U11 | **描述** | 当用户查看编译结果时，可以在同一页面中查看代码。 | |
| **质量属性** | 易用性 | | | |
| **环境** | 运行时 | | | |
| **刺激** | 用户查看编译结果 | | | |
| **响应** | 在同一页面中同时显示代码实例视图和编译结果视图 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 提供用户支持 | S16 |  |  | N16 |

### 2.3.8可移动性需求及解决方案

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | M1 | **描述** | 系统支持移动设备访问 | |
| **质量属性** | 移动性 | | | |
| **环境** | 正常操作 | | | |
| **刺激** | 用户使用移动设备访问系统 | | | |
| **响应** | 用户顺利进入系统，且无差错 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| Web | S1 |  | R1 |  |
| **原因** | 1. Web支持许多浏览器和移动设备，使用Web是最方便有效的。（S1） 2. 在移动端的性能很大程度受Web的限制。（R1） | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | M2 | **描述** | 登陆注册不能自适应 | |
| **质量属性** | 移动性 | | | |
| **环境** | 正常操作 | | | |
| **刺激** | 用户在登录/注册时，改变网站页面大小 | | | |
| **响应** | 页面不能自适应，登录/注册框位置不居中 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 基于流体网格的布局 |  | T1 | R2 |  |
| **原因** | 基于流体网格的布局可以解决登录/注册的自适应问题，但对比直接用HTML5写的页面而言，可能会使网页刷新变慢。（T1，R2） | | | |
| **架构图** |  | | | |

### 2.3.9可修改性需求及解决方案

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | Mo1 | **描述** | 系统支持按模块分工实现，且耦合程度低 | |
| **质量属性** | 可修改性 | | | |
| **环境** | 设计时 | | | |
| **刺激** | 开发人员要分前后台开发和分模块开发 | | | |
| **响应** | 从架构中查找要修改的功能修改和测试，不会影响其他模块 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 划分模块 | S1 | T1 | R1 |  |
| 封装 | S2 |  |  | N1 |
| 增加语义内聚性 | S3 |  |  |  |
| **原因** | 1. 划分模块是指将不同的模块分开，分别开发，有利于提高系统的开发速度，且易于不同的开发人员理解系统代码。（S1） 2. 划分模块增加了系统的访问路径，使系统响应时间变长。（T1） 3. 封装隐藏了系统的具体实现，为后续增加功能而不影响原有功能提供了保证。（S2） 4. 增加语义内聚性要求将不同的职责放在不同的模块中，尤其在设计创建新模块时，要将不受模块内部变更影响的职责移出模块，避免不必要的负担。（S3） 5. 开发时，模块如果划分不合理，在后续的变更中可能会造成混乱。（R1） | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | Mo2 | **描述** | 系统支持功能、实现变更 | |
| **质量属性** | 可修改性 | | | |
| **环境** | 开发时 | | | |
| **刺激** | 开发人员想要修改某一功能 | | | |
| **响应** | 在模块中查找特定功能进行修改，且系统变更可以控制在2人周 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 划分模块 | S4 | T2 |  |  |
| 封装 | S5 |  |  | N2 |
| 抽象公共服务 |  | T3 |  |  |
|  |  |  |  |  |
| **原因** | 1. 划分模块是指将不同的模块分开，方便开发人员查找特定的功能的实现位置进行修改，而不会影响其他模块的性能。（S4） 2. 划分模块增加了系统的访问路径，使系统响应时间变长。（T2） 3. 封装隐藏了系统的具体实现，使得在修改其他功能时，其他原有功能不会受到影响。（S5） 4. 在开发时，将具有相似功能的函数放在一起，抽象公共服务，方便了代码的后续修改，同时也减少了系统的响应时间。（T3） | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | Mo3 | **描述** | 系统支持后续增加功能需求 | |
| **质量属性** | 可修改性 | | | |
| **环境** | 运行时 | | | |
| **刺激** | 用户希望增加新的功能 | | | |
| **响应** | 开发人员查找模块中的功能进行增加，测试和部署，并在3个月内完成系统升级 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 划分模块 | S6 | T4 |  |  |
| 封装 | S7 |  |  | N3 |
|  |  |  |  |  |
|  |  |  |  |  |
| **原因** | 1. 划分模块是指将不同的模块分开，方便开发人员查找特定的功能的实现位置进行增加/删除。同时，在新功能不属于原有设计的模块时，要创建新的模块，以实现新功能。（S6） 2. 划分模块增加了系统的访问路径，使系统响应时间变长。增加的模块路径太长可能会影响其他页面的刷新。（T4） 3. 封装隐藏了系统的具体实现，使得在增加其他功能时，由于对原有功能的不可见，并不会影响到原有的功能。（S7） | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | Mo4 | **描述** | 系统根据用户信息提供个性化显示 | |
| **质量属性** | 可修改性 | | | |
| **环境** | 运行时 | | | |
| **刺激** | 用户修改个人信息 | | | |
| **响应** | 修改成功并在用户登录后显示 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 划分模块 | S7 |  |  |  |
| **原因** | 用户修改个人信息时，只会影响到用户个人信息的模块，而其他模块不受影响。（S7） | | | |
| **架构图** |  | | | |

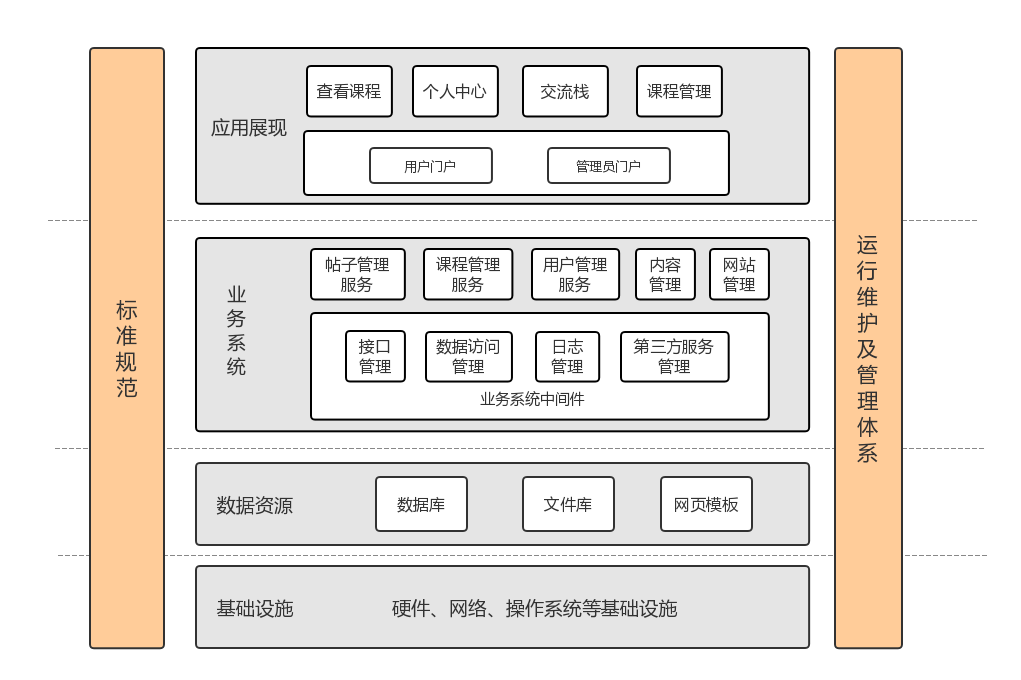
### 2.3.9互操作需求及解决方案

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | I1 | **描述** | 同一接收数据格式 | |
| **质量属性** | 互操作性 | | | |
| **环境** | 正常操作运行状态 | | | |
| **刺激** | 系统功能模块之间进行交流沟通 | | | |
| **响应** | 系统使用中间件统一交流，使系统模块正常运行 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 中间件pipe | S1 | T1 |  |  |
| **解释** | 通过统一接收格式可以规格化模块之间，前后端之间的交流，我们可以使用ajax在后台与服务器进行少量数据交换,统一使用json数据格式等。(S1,T1) | | | |
| **架构图** |  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **场景编号** | I2 | **描述** | 系统支持QQ微信等第三方接口登录 | |
| **质量属性** | 互操作性 | | | |
| **环境** | 用户正常登录过程中 | | | |
| **刺激** | 用户其他使用第三方接口登录,并授权登录 | | | |
| **响应** | 系统调用第三方接口允许用户登录，记录用户登录状态 | | | |
| **架构策略** | **敏感点** | **权衡点** | **风险** | **非风险** |
| 接口管理 |  |  | T1 |  |
| **解释** | 使用第三方接口方便用户登录，减少登录操作流程。 | | | |
| **架构图** |  | | | |

# 3软件系统架构设计

## 3.1系统分层架构视图



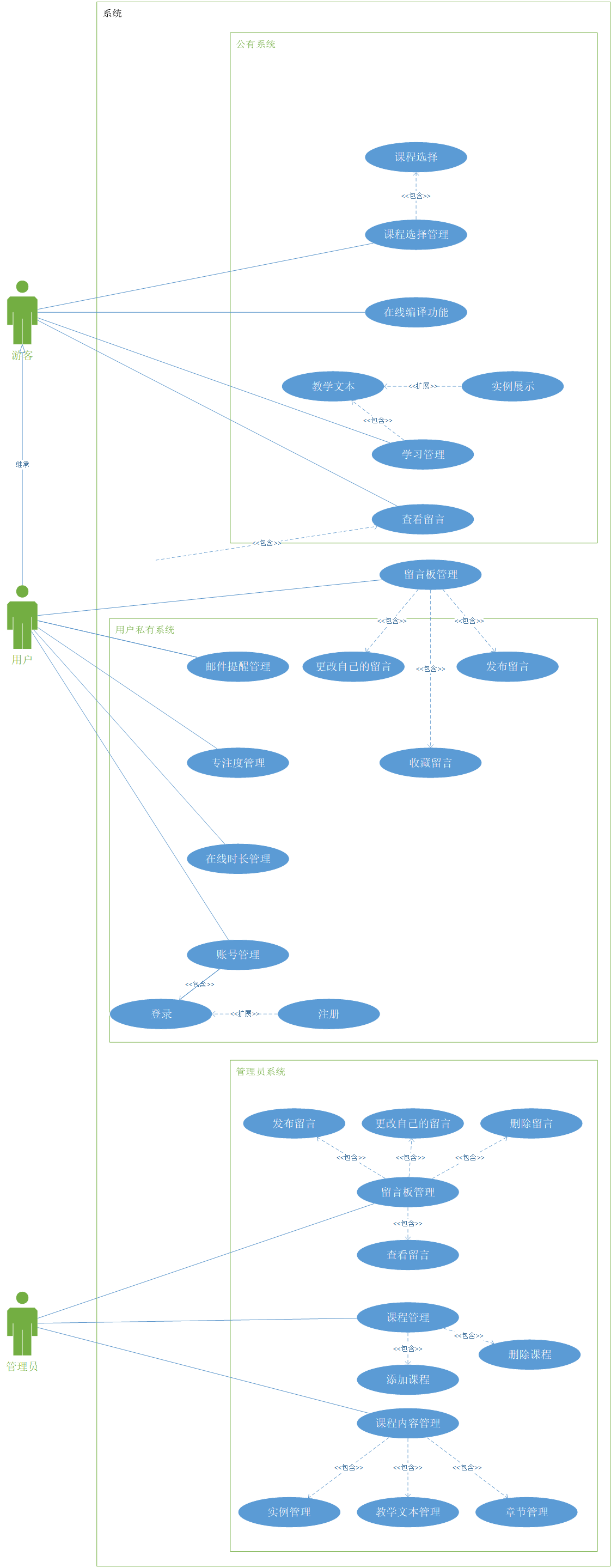
层级架构方面分为四层，应用展现层包括了展现给客户的功能，其中根据用户角色的不同分为了用户门户和管理员门户。

业务系统为后台的主要部分，实现了网站支持的各类服务和功能，包含帖子管理、课程管理等，并且通过利用框架集成或第三方集成的软件或工具包引入中间件，包括前后端用于约束接口和模拟数据的mock平台，后台框架集成的数据访问、日志及第三方服务管理。

数据资源层存储了网站所使用的数据、静态文件如图片、CSS、JavaScript函数库等，以及用于支持动态填充内容的网页模板。

基础设施层包含了服务器硬件、网络及操作系统等基础设施，考虑到方便性和易用性使用了第三方提供的云服务器平台。

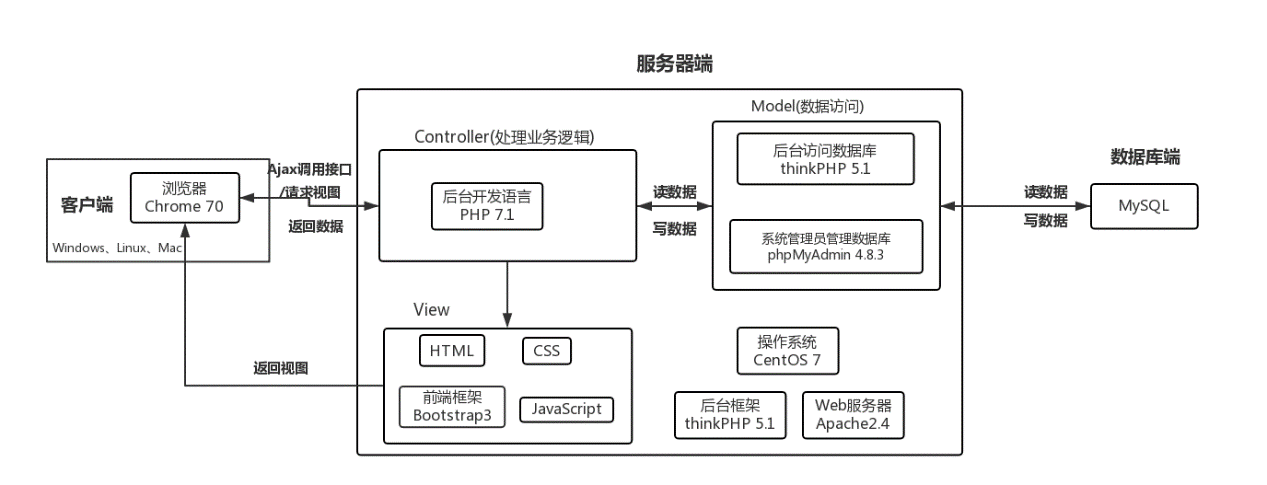
## 3.2用例视图



## 3.3逻辑视图

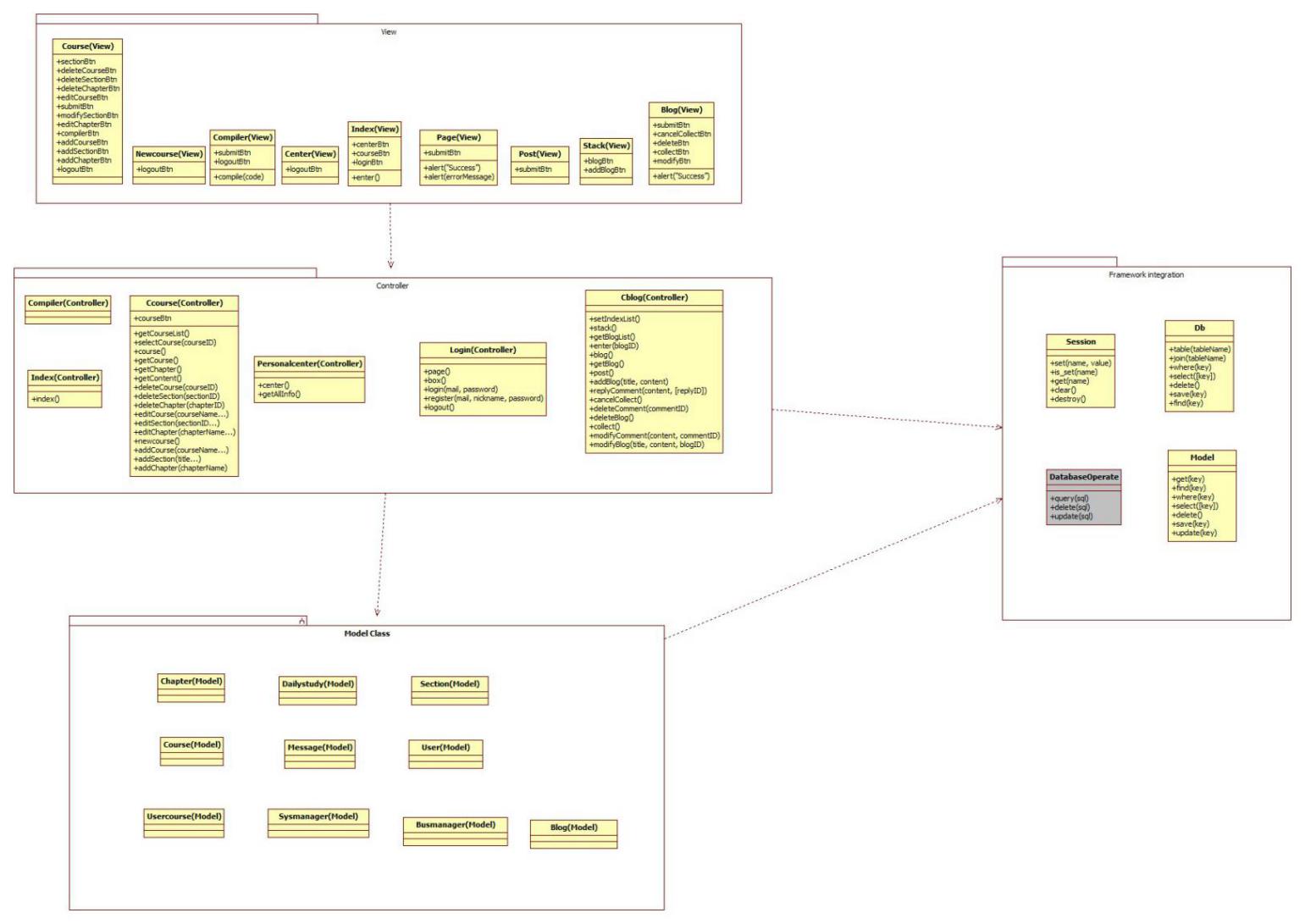
逻辑视图是架构设计中的核心视图。它展示在构架方面具有重要意义的设计元素。逻辑视图描述最重要的类、和这些类到包和子系统，再到层的组织形式。它还要描述最重要的用例实现，例如构架的动态方面。

### 3.3.1逻辑架构图



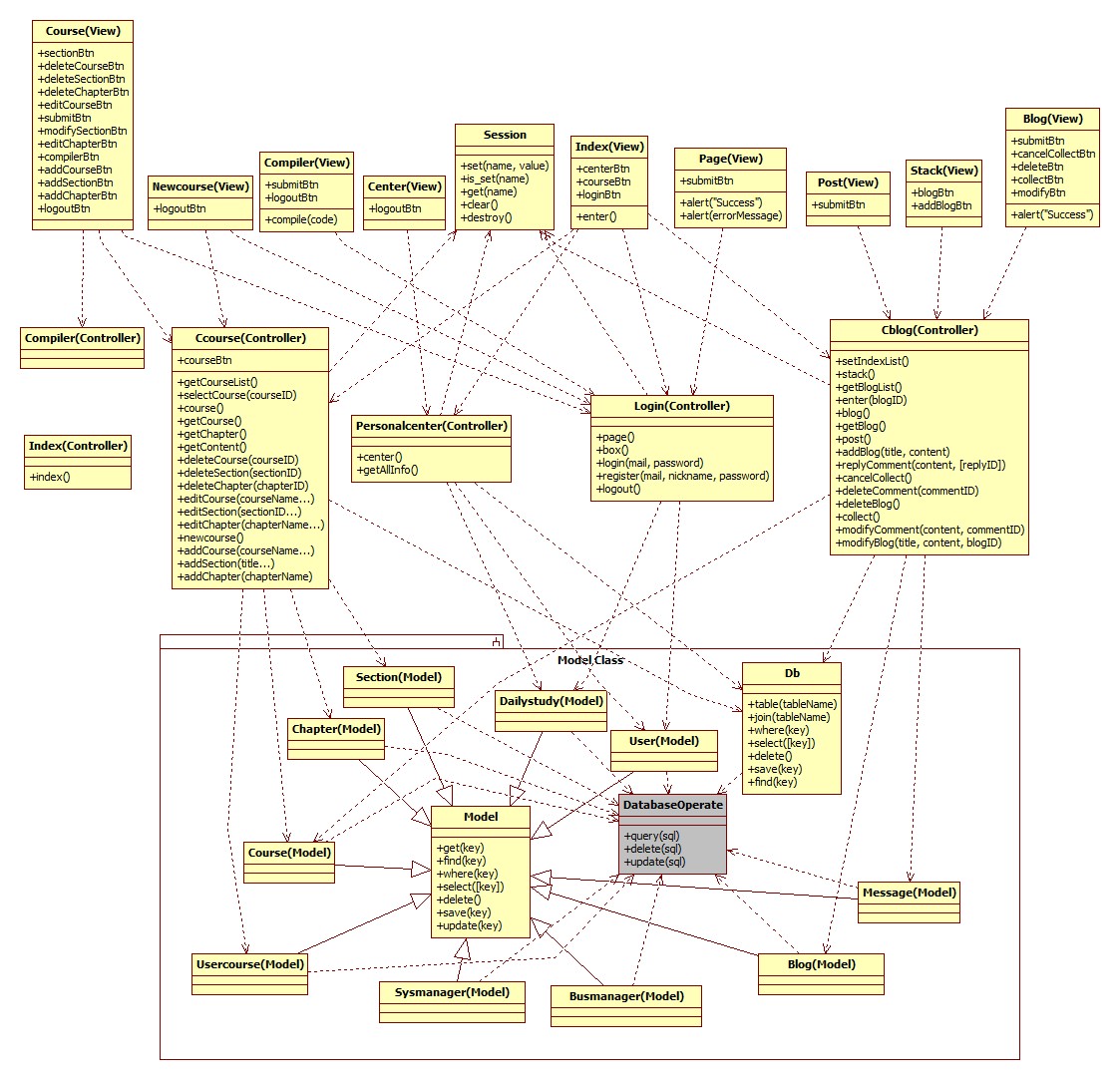
系统的运行逻辑如图所示，客户端方面，用户从浏览器访问网站，控制器接收请求，根据请求是页面请求还是数据请求返回视图或对应的数据，若是请求数据，则通过框架集成的用于数据访问的模型来访问数据库。其中网页与控制器进行数据动态交互的方式为Ajax，页面动态响应的方式为JavaScript。后台方面利用thinkPHP框架集成的模型及多种功能类，后台基于apache网页服务器及centos7操作系统，数据库使用Mysql，框架本身已集成与MySQL进行交互，进行访问与读写数据操作的功能。

### 3.3.2包图



如图所示，系统分为了四个包，分别是前端网页模板类、后台控制器类、模型类以及框架集成功能类。其中前端网页模板类包含各个网页模板，模板内部包含html、css以及JavaScript，可根据用户点击动态填充数据。后台控制器类接受页面请求并进行实际的业务处理，作为前端视图和模型的中间件。模型类为连接数据库对应表的类，每个类对应了数据库的一个表，以此达到方便快速进行数据读写的目的。框架集成功能类包含了后台框架thinkPHP本身集成的功能类，如记录用户会话的Session类，数据库整体访问的Db类，模型类的抽象父类Model类，以及框架内部实现的用于数据库操作的DatabaseOperate类。

### 3.3.3类图



如图所示，类图详细记录了包含的类以及类与类之间的交互。鉴于该项目是基于MVC的网站，因此类之间的关系以依赖为主，辅以少量的继承关系。

### 3.3.4状态图

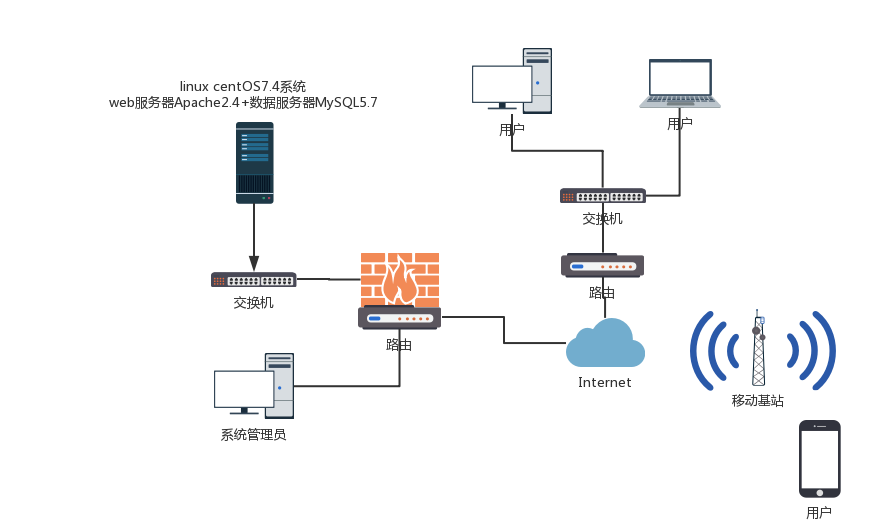
系统状态图：



课程管理状态图：



## 3.4部署视图



## 3.5进程视图

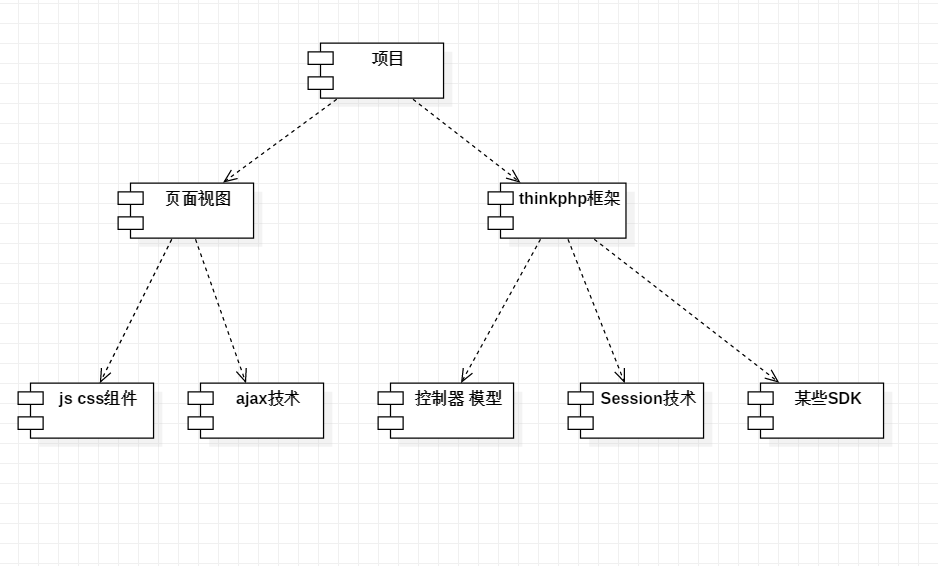




本系统前后端数据交换应用Ajax开发。Ajax 是一种用于创建快速动态网页的技术。Ajax 是一种在无需重新加载整个网页的情况下，能够更新部分网页的技术。通过在后台与服务器进行少量数据交换，Ajax 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。Ajax 即“Asynchronous Javascript And XML”（异步 JavaScript 和 XML），是指一种创建交互式[网页](https://baike.baidu.com/item/%E7%BD%91%E9%A1%B5/99347" \t "https://baike.baidu.com/item/ajax/_blank)应用的网页开发技术。Ajax的原理简单来说通过XmlHttpRequest对象来向服务器发异步请求，从服务器获得数据，然后用javascript来操作DOM而更新页面。这其中最关键的一步就是从服务器获得请求数据。XMLHttpRequest是ajax的核心机制，它是在IE5中首先引入的，是一种支持异步请求的技术。简单的说，也就是javascript可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。

## 3.6开发视图

### 3.6.1构件图



### 3.6.2如何保持架构与代码的一致性

**一、将设计嵌入代码**

将架构设计中产生的约束以注释形式写入代码中，使后续开发者能够轻易获取、理解架构设计。

**二、框架**

框架是围绕特定主题组织的可重用的一组类。

后台使用ThinkPHP框架，详见后端框架复用计划（详细设计文档）。

前端使用Bootstrap框架，详见前端框架复用计划（详细设计文档）。

**三、使用代码模板**

Ajax模板具有统一的代码格式，给view组件提供了访问Controller组件的接 口。

使用Ajax模板给我们带来了一下便利：

·具有类似属性的组件以类似的方式运行。

·模板只需要调试一次。

·复杂的部分可以由技术人员完成并交给技术水平较低的人员。

**四、保持代码和架构一致**

1、发生架构侵蚀时，将相应文档标志为过期，增加剩余部分文档的可信度。

2、定期修改文档，使架构与代码保持同步、一致。

# 4关键技术设计

## 4.1公共构件设计

### 4.1.1构件描述

**1、Db构件**

Db构件位于MVC架构模式的Model组件，用于为多个视图提供数据。由于应用于Model的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

**2、课程视图模板**

课程视图模板位于MVC架构模式中的View组件。该视图模板使用推迟绑定策略，在运行时动态加载课程内容，提高了代码的可重用性。

**3、留言板视图模板**

留言板视图模板位于MVC架构模式中的View组件。该视图模板使用推迟绑定策略，在运行时动态加载留言信息，提高了代码的可重用性、可修改性。

**4、Ajax接口模板**

Ajax接口模板可视为View组件和Controller组件之间的连接器，隐藏了controller的细节，为View提供统一的调用接口。

### 4.1.2抽取规则

**1、增加语义内聚性**

增加语义连贯性——如果模块中的职责A和B不是出于相同的目的，则应将它们放在不同的模块中。 这可能涉及创建新模块，或者可能涉及将责任移交给现有的其他模块。 识别要移动的职责的一种方法是假设影响模块的可能变化。 如果某些职责不受这些变更的影响，那么可能应该删除这些职责。

**2、运行时绑定**

因为人们的工作几乎总是比计算机的工作更昂贵，所以让计算机尽可能地处理变更几乎总能降低进行变更的成本。如果我们设计具有内置灵活性的工件，那么实现这种灵活性通常比手动编码特定更改便宜。

通常，在生命周期中越晚绑定值是越好的。然而，建立机制以促进后期绑定往往更昂贵，这是一个权衡点。

**3、使用中间件**

使用中间件打破依赖关系。 给定责任A和责任B之间的依赖关系（例如，执行A首先要求执行B），可以通过使用中介来打破依赖性。 中介的类型取决于依赖的类型。

**4、封装**

封装引入了模块的显式接口。该接口包括应用程序编程接口（API）及其相关职责。旨在增加可修改性的接口是关于未来可能要进行修改的模块细节的抽象，它们应该隐藏这些细节，从而降低修改的复杂度。

## 4.2接口设计

### 4.2.1动作

GET （SELECT）：从服务器获取内容。

POST （CREATE）：在服务器上创建一个新的资源或更新旧资源。

### 4.2.2路径（接口命名）

第一个为名词，在接口中被用于表示与属于某一具体模块的接口。由于以及接口网站中命名的不可重复性，我们用动词表示具体操作。方便寻找接口以及理解接口的具体操作。下面是具体的例子：

/blog：交流栈模块

GET：/blog/getBlogList：获取交流栈模块中的全部帖子

GET：/blog/enter：进入交流栈的某一具体帖子

POST：/blog/addBolg：在交流栈添加一个新帖子

POST：/blog/modifyBlog：在交流栈某一编辑帖子

POST：/blog/deleteBlog：在交流栈删除某一帖子

### 4.2.3版本（Versioning）

应该将API的版本号放入URL。如：

https://api.example.com/v1/

另一种做法是，将版本号放在HTTP头信息中，但不如放入URL方便和直观。Github采用这种做法。

### 4.2.4过滤信息（Filtering）

如果记录数量很多，服务器不可能都将它们返回给用户。API应该提供参数，过滤返回结果。

下面是一些常见的参数。

?limit=10：指定返回记录的数量

?offset=10：指定返回记录的开始位置。

?page\_number=2&page\_size=100：指定第几页，以及每页的记录数。

?sortby=name&order=asc：指定返回结果按照哪个属性排序，以及排序顺序。

?animal\_type\_id=1：指定筛选条件

### 4.2.5状态码（Status Codes）

状态码范围

1xx 信息，请求收到，继续处理。范围保留用于底层HTTP的东西，你很可能永远也用不到。

2xx 成功，行为被成功地接受、理解和采纳

3xx 重定向，为了完成请求，必须进一步执行的动作

4xx 客户端错误，请求包含语法错误或者请求无法实现。范围保留用于响应客户端做出的错误，例如。他们提供不良数据或要求不存在的东西。这些请求应该是幂等的，而不是更改服务器的状态。

5xx 范围的状态码是保留给服务器端错误用的。这些错误常常是从底层的函数抛出来的，甚至

开发人员也通常没法处理，发送这类状态码的目的以确保客户端获得某种响应。

当收到5xx响应时，客户端不可能知道服务器的状态，所以这类状态码是要尽可能的避免。

服务器向用户返回的状态码和提示信息，常见的有以下一些（方括号中是该状态码对应的HTTP动词）。

200 OK - [GET]：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。

201 CREATED - [POST/PUT/PATCH]：用户新建或修改数据成功。

202 Accepted - [\*]：表示一个请求已经进入后台排队（异步任务）

204 NO CONTENT - [DELETE]：用户删除数据成功。

400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。

401 Unauthorized - [\*]：表示用户没有权限（令牌、用户名、密码错误）。

403 Forbidden - [\*] 表示用户得到授权（与401错误相对），但是访问是被禁止的。

404 NOT FOUND - [\*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的。

406 Not Acceptable - [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。

410 Gone -[GET]：用户请求的资源被永久删除，且不会再得到的。

422 Unprocesable entity - [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误。

500 INTERNAL SERVER ERROR - [\*]：服务器发生错误，用户将无法判断发出的请求是否成功。

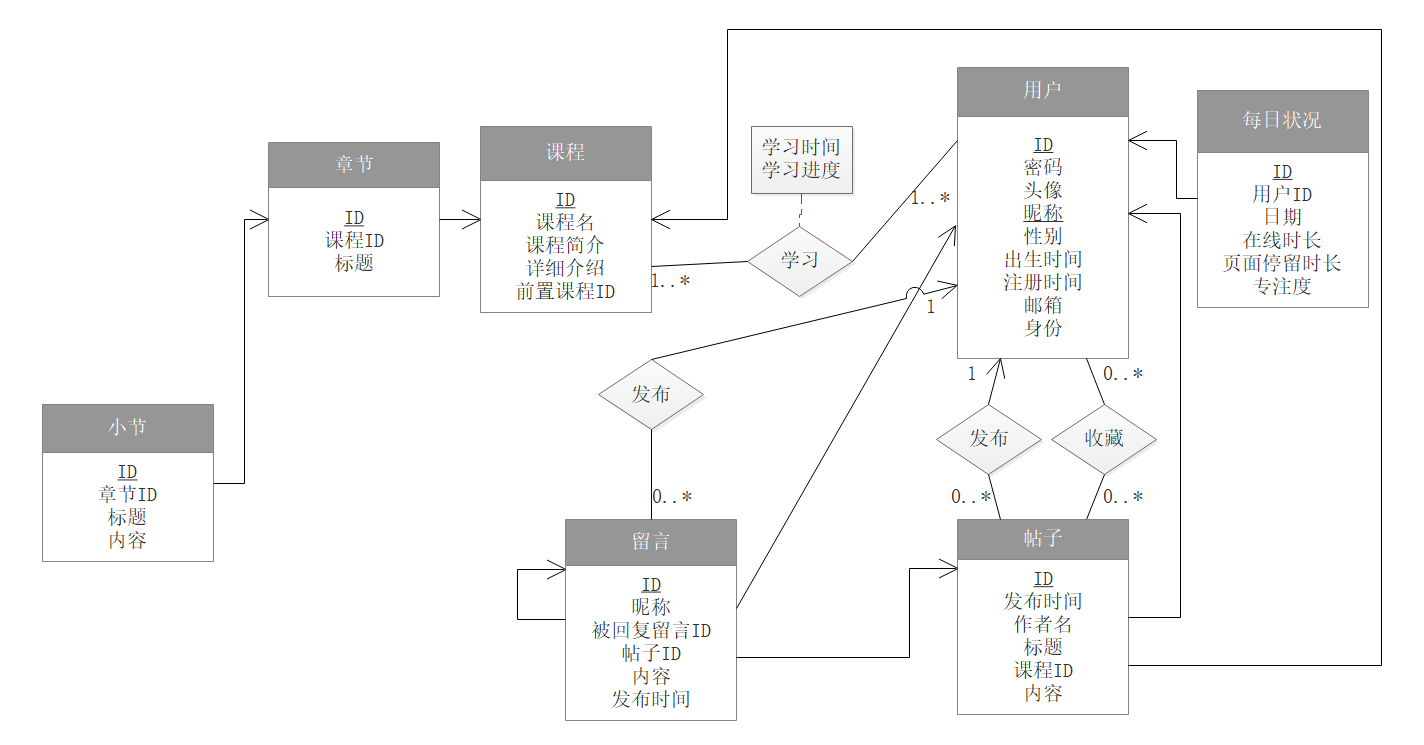
502 网关错误

503 Service Unavailable

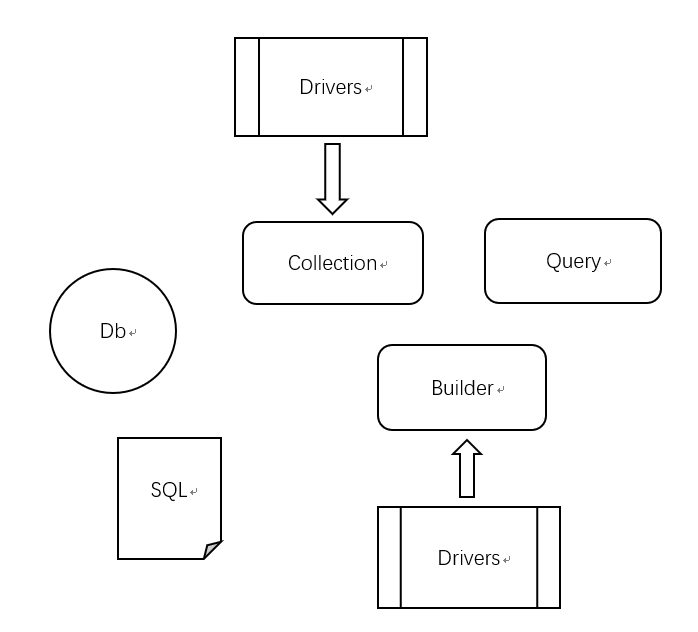
504 网关超时

## 4.3数据架构设计

### 4.3.1主题数据库模型



### 4.3.2数据访问层设计



对数据库的操作是通过TP5的数据访问中间层。

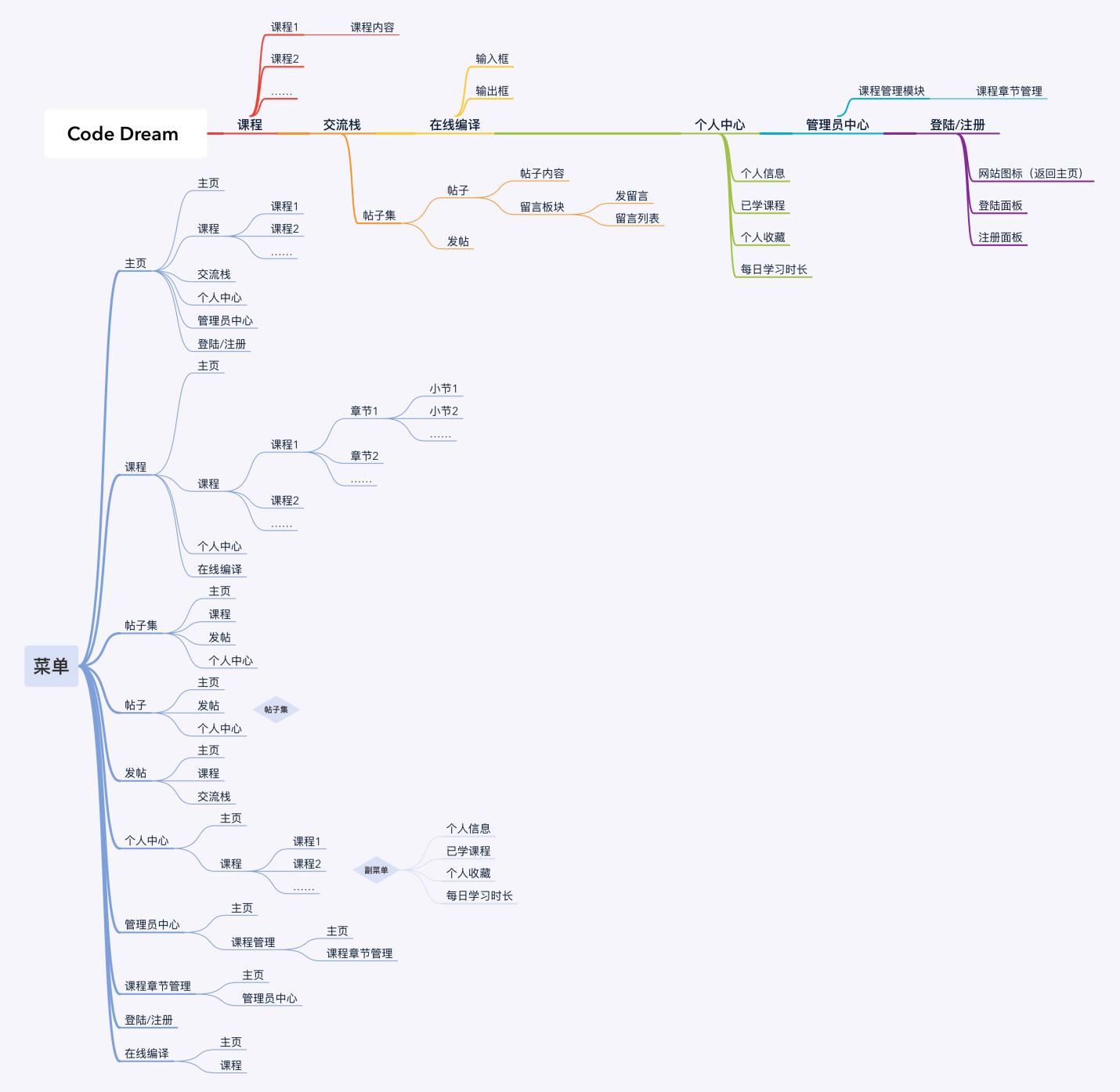
database.php文件配置数据库的配置文件。TP5默认支持4种数据库，其中包括MySQL，在type设置数据库类型。hostname配置数据库IP，如果外网需要外网IP。database改成数据库名。根据配置文件选择不同的驱动Driver，目录thinkphp\library\think\db\connector是连接数据库的文件夹，就是连接器的驱动。

TP5查询数据库方式有3种，第一种是通过原生sql语句查询。第二种是使用构造器操作数据库，第三种是使用模型和关联模型操作数据库。

TP5提供Db类用于数据库操作，Db是数据库操作的入口对象。Db实例化了一个Collection对象，通过实例化Collection对象来连接数据库。首先，Collection是通过PHP的PDO来实现连接。另外它并不是真正的连接数据库，而是出于待命状态，当执行SQL语句才会真正连接数据库。TP5还提供了另一种方式，叫做查询器Query。通过这样一种方式也能操作数据库。Query是对数据库常用CURD的封装，它可以让我们很优雅编写sql语句，同时它也很方便链式操作。

TP5的3种操作数据库方式都是先翻译成原生sql语句从而执行的。只不过Query查询器对原生sql做了封装。Builder把相关的Query查询器翻译成原生sql之后，把原生sql语句传给Collection，再由连接器类连接数据库执行sql语句。

## 4.4UI架构设计



**UI架构设计图**

上图是本产品的UI架构设计图，本项目产品共有10个用户视图，包括主页、课程页、帖子集（交流栈）页、帖子页、发帖页、个人中心页、管理员中心页（课程管理页）、课程章节管理页、登录/注册页、在线编译页。

基于“系统界面简洁美观”、“网站样式风格鲜明统一”、支持可移动性的原则，UI架构采用bootstrap框架，采取使用并修改bootstrap框架提供的样式以及网站骨架结构设计作为UI架构设计和实现路线。

每一个用户视图具备有差异性的菜单，菜单差异性基于“平级页面跳转需要回溯首页的情况不超过5%”、“功能模块划分明确”、“单个页面菜单深度不超过2级”、“单个页面菜单数量不多于2个”、“单个页面过长则需要位置跳转容易”的原则权衡得到。

菜单在“课程”这一条目以及课程页面下的“章节”条目上基于课程数量及名称或章节数量及名称的后续可能变动的可扩展支持原则，采用数据独立于视图、数据在创建时绑定的战术，课程信息独立于用户视图、在用户视图创建时利用ajax获取课程名绑定视图填充菜单的“课程”条目。菜单在“课程”这一条目以及课程页面下的“章节”条目上基于“系统界面简洁美观”的原则，不宜将所有课程或章节名同时列出在菜单中，采用了折叠菜单的设计。个人中心页因为单个页面过长并被划分为多个功能模块，所以使用了副菜单作为（主）菜单的补充。菜单基于需要支持可移动性的原则，需要可以根据浏览器窗体大小改变样式，实现上可以参考bootstrap框架提供的可伸缩菜单样式。

除了发帖页、在线编译页、登录/注册页，剩余用户视图都具备采用静态骨架和动态内容绑定结合的页面主体部分。静态骨架和动态内容绑定结合的主体部分基于“部分加载优于全部加载，避免用户误以为系统崩溃而丧失信心”、“用户希望在同一个页面看到尽量多的内容而不需要页面跳转”、节约带宽而提高性能的原则，采用数据和视图分离、数据在创建时绑定的战术。用户视图内容作为数据单独存储于数据库，在页面加载时根据页面或用户任务通过ajax获取数据填充页面主体，此外当用户给出系统模块页面内容更替任务时，在该模块页面（与当前页面是同一个页面）通过ajax获取新数据更替原有数据作为新的页面内容，从而避免了过多的页面跳转。

基于“面向不同权限的用户提供不同的内容权限”的原则，采用数据独立于视图、分发管理的战术。课程内容信息作为数据独立于视图单独存储于数据库，同时数据库作为分发管理的中间件存在。作为数据生产者的管理员（管理员也可以作为数据消费者）和作为数据消费者的一般用户都是在用户视图通过ajax从后台发送请求、获取数据。作为数据生产者的管理员还可以在用户视图（课程管理页和课程章节管理页）通过ajax向后台发送数据更改请求、更改数据。后台进行对数据库的操作。因为管理员和一般用户都是从数据库获取数据，且管理员修改数据需要在用户视图（课程管理页和课程章节管理页）确定再传送回后台，同时访问课程内容的一般用户的课程内容数据是已经在用户视图（课程）加载，所以管理员的更改操作不会影响到已访问课程内容的一般用户。

基于“在同一页面中同时显示代码实例视图和编译结果视图”的设计目的，在线编译页采用提供用户支持的战术。在线编译页布局设计了两个文本框分别作为数据输入和数据输出，降低了用户记忆难度，布局设计更加人性化。

UI架构的视图布局可以参考《用户手册（GB8567——88）V1.1》。