

到云概要设计



2020-3-13

工程实践小组：27组

目录

[1 产品概述 2](#_Toc34993483)

[2 技术栈 2](#_Toc34993484)

[2.1 前端框架 2](#_Toc34993485)

[2.2 后端框架 2](#_Toc34993486)

[2.3 数据库 2](#_Toc34993487)

[2.4 Web引擎 2](#_Toc34993488)

[2.5 云环境 2](#_Toc34993489)

[3 宏观架构 3](#_Toc34993490)

[3.1 宏观架构设计图 3](#_Toc34993491)

[3.2 宏观技术架构浅析 3](#_Toc34993492)

[4 接口规格 4](#_Toc34993493)

[5 分析 4](#_Toc34993494)

[5.1 分析与模块化 4](#_Toc34993495)

[5.2 海量数据适应性 4](#_Toc34993496)

[5.3 信息封装和隐藏 5](#_Toc34993497)

[5.4 前后端分离 6](#_Toc34993498)

[5.5 灵活性 6](#_Toc34993499)

[6 错误处理 7](#_Toc34993500)

[6.1 前端 7](#_Toc34993501)

[6.2 后端 7](#_Toc34993502)

# 1 产品概述

我们团队的目标是实现一个实时互动的教学平台，涵盖手机端移动应用程序APP和PC端的管理应用程序。在确定需求和功能后，我们通过学习讨论，决定采用java作为前端主要开发框架，Django作为后端服务器开发框架。数据库暂时选用MYSQL。

# 2 技术栈

## 2.1 前端框架

我们将考虑使用vue.js框架对于前端进行一次完整的重构，使得架构更加清晰，可维护性更强，未来的开发效率和潜力提高。使用vue.js框架有以下技术特点：

1、vue两大特点：响应式编程、组件化。

2、vue的优势：轻量级框架、简单易学、双向数据绑定、组件化、数据和结构的分离、虚拟DOM、运行速度快。

3、vue是单页面应用，使页面局部刷新，不用每次跳转页面都要请求所有数据和dom，这样大大加快了访问速度和提升用户体验。而且他的第三方ui库很多节省开发时间。

市场优势：

1、vue是目前最热门的前端框架之一，社区活跃，方便易用，可避免重复造轮子；

2、vue已有非常丰富的生产实践检验，可以避免在开发过程中产生重大隐患

## 2.2 后端框架

cloud sign后端框架使用Java语言开发，使用目前业界最为流行的Spring Boot框架，系统采用MySlq数据库存储数据，持久层使用MyBatis框架，API全部支持Restful风格，项目工程基于Maven构建部署。

技术选型：

1、系统环境

Java EE 8

MySql 5.5

Apache Maven 3

2、主框架

Spring Boot 2.1

Spring Framework 5.1

Spring Security 5.1

3、持久层

Apache MyBatis 3.4

Hibernate Validation 6.0

Alibaba Druid 1.1

4、数据缓存

Redis 3.2

## 2.3 数据库

数据库使用的是Mysql。数据库使用的是Mysql，具体是5.6还是5.7不太清楚，不过应该问题不是很大。

后端代码里面，并没有较多的明文sql拼接，所以正常来说，只要是mysql系数据库，都不会存在兼容性问题（mariadb应该也没大问题）。

## 2.4 Web引擎

web引擎的话，可以有很多选择，Apache或者Nginx。

我们更倾向于使用Nginx，进行一次反向代理访问后端，以及前端也可以部署在同一个nginx下。

## 2.5 云环境

云环境一个比较理想的配置如下（腾讯云）：

Ubuntu环境云服务器，一个实例（理想配置2core4G，最低不得低于1core1G）

MYSQL云数据库，一个实例（理想配置2G200Qps，最低不得低于1G100Qps）

Redis缓存数据库，一个实例（理想配置1G400+Qps，最低不得低于1G100Qps）

对象存储服务，一个Bucket（常备模式即可，未来上了规模再考虑冷备Bucket，用于用户资源的存储）

内容分发网络服务（也就是CDN），两个域名

一个用于存储web前端的静态内容（js、css、图片等）

一个用于用户资源（比如文件等）存储

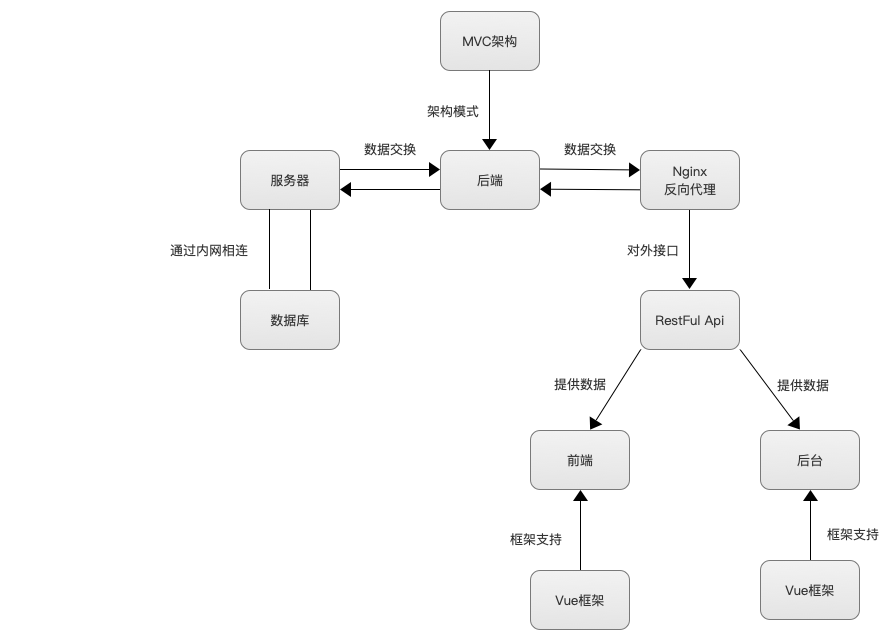
当然，最紧缩的状态下，那就只能：

一台微型服务器数据库和缓存库部署在本机

存储内容也一概存储于本机路径下

# 3 宏观架构

## 3.1 宏观架构设计图



## 3.2 宏观技术架构浅析

* 从整体功能上看，我们的项目分为以下三个大的模块：
* 服务端（包括Server、Datebase及Nginx）
* 后端
* 用户交互端（前端+小程序）
* 服务端与后端的数据交换

考虑到软件后期用户量的上升及特殊时期（如新型冠状病毒疫情期间）的用户访问需求，我们在后端与服务器之间加入了一层Nginx反向代理。当出现大量的用户请求时，服务器可以配置成集群的形式，用户在访问前端或小程序时，请求会先发送到Nginx服务器，Nginx再在服务器集群中选择一个压力较小的服务器，将该访问引入被选择的服务器，以达到负载均衡的效果。对于本项目而言，进行这样的准备无疑很有必要。

* 服务器与数据库的部署

本项目服务于广大师生，有着大量的教学资料及视频存储需求，把这些静态资源放在主服务器上显然不是一个明智的做法。我们采取了服务器和数据库分离的架构，静态资源将会通过OSS对象存储存放在一系列单独的云硬盘中，减轻服务器的空间压力。

* RESTful API

为了使社团管理人员和普通用户具有不同的操作权限，我们在本项目中设置了用于管理班课、人员信息的后台界面，和供普通用户操作的h5端页面。由于前端开发和后台开发分别由不同人员开发，因此传统的后端+模板引擎生成前端界面的方法就不适用了。为了解决这个问题，我们的后端将提供一系列RESTful API接口，这些接口对前端和后台而言都有着相同的调用形式，用户交互端不需要考虑后端的内部细节，只需要按照约定调用这些API，就可以获得自己需要的数据。这样的框架设计将前端与后端完全解耦，也让前后端并行开发成为了可能。

# 4 接口规格（暂定）

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **请求方法** | **请求路径** | **用途** |
| 1 | get | /api/background/classmanager/getClassList | 获取所有班课列表 |
| 2 | put | /api/background/classmanager/createClass | 创建新的班课 |
| 3 | delete | /api/background/classmanager/deleteClassById | 根据班课id删除班课 |
| 4 | post | /api/background/classmanager/updateClassInfoById | 根据班课id更新班课信息 |

# 5 分析

## 5.1 分析与模块化

抽象与模块化的话，主要分为几层：

* 底层数据抽象化
  + 即将底层的数据库访问，封装成面向对象的形态（orm）
  + 具体来说，分为三个点
    - 将SQL的各类操作语句，封装成给予OOP的操作形态，甚至支持链式操作
    - 将数据库内的数据信息，封装成OOP对象形态，方便进行读写操作
    - 将数据库结构的变动，维护成数据库迁移对象（migration），方便日后的不断后续开发维护
* 数据行为模块化
  + 将数据的调度行为，基于需求域进行分类
  + 将不同类型的行为划分到不同的区域内（controller）
  + 将同一controller内的不同行为，划分为不同的method
  + 对于不同的method，定义各类前后缀操作，极大方便有些共性操作的拼装（这里需要用到一些AOP程序设计模式）
* 行为对外接口化
  + 通过一个映射（route，路由表），将controller内的各个method映射至对外访问的api
  + 通过路由表，将api传入的数据请求，进行基本的数据处理与封装之后，交给对应controller对应method执行
  + method执行完毕后，返回请求response数据

## 5.2 海量数据适应性

* 数据库读写分离
  + 这一机制适用于数据io很大，且存在读性能瓶颈的情况（实际上，这种情况对于webapp来说，相当常见）
  + 设置主库和多个从库，主库主要负责写操作，从库负责读操作，且分担掉读压力
  + 不过值得注意的是，需要有可靠的手段保证数据一致性（这也是分布式系统都必须面对的难题）
* 数据库冷热备份机制
  + 这一机制适用于数据库存储数据量巨大，且真正活跃的数据大部分集中在近期的情况
  + 使用一个热备份数据库，和一个（也可能是分布式）冷备份数据库
  + 大部分操作均在热备份库进行，极少数读操作可能涉及到冷备份
  + 同样的，对于table单位，也可以做出类似的机制
  + 当然，在后端orm部分，可以考虑通过进一步的封装，以实现和普通单体数据库无差异的编程体验
* 服务端分布式化
  + 将服务端做成分布式的形态（无数据冒险的单体处理架构，或者有充分的冒险控制机制）
  + 由nginx或者其他解决方案进行负载均衡管理
* 数据上云，主要体现在以下几个方面：
  + 对于前端，一些静态资源可以直接使用cdn进行分流，减轻原服务器压力
  + 对于后端的静态资源，可以直接上传至公有云，读取时进行接口请求即可，减轻本地硬盘存储以及IO压力

## 5.3 信息封装和隐藏

**后端Spring boot**

Spring Boot是由Pivotal团队提供的全新框架，被很多业内资深人士认为是可能改变游戏规则的新项目。早期我们搭建一个SSH或者Spring Web应用，需要非常繁琐的步骤，比如配置web.xml，配置数据库连接，配置事务，配置日志，配置Tomcat，装配Bean，声明和配置切面等等等等，如果项目过大多人协作各种冗长啰嗦的配置让人烦不胜烦，这么多年下来，给人一种Java就是大型配置文件的感觉。

Spring Boot的设计目的是用来简化新Spring应用的初始搭建以及开发过程，吸引更多开发者的最大亮点之一是集成了自动配置的魔力。Spring Boot的四个主要新特性如下：

1、Spring Boot Starter：它将常用的依赖分组进行了整合， 将其合并到一个依赖中， 这样就可以一次性添加到项目的Maven或Gradle构建中；Spring Boot通过提供众多起步依赖降低项目依赖的复杂度。起步依赖本质上是一个Maven项目对象模型（Project Object Model， POM），定义了对其他库的传递依赖，这些东西加在一起即支持某项功能。很多起步依赖的命名都暗示了它们提供的某种或某类功能。Spring Boot经过了足够的测试，确保引入的全部依赖都能相互兼容。这是一种解脱，只需指定起步依赖，不用担心自己需要维护哪些库，也不必担心它们的版本。

2、自动配置：Spring Boot的自动配置特性利用了Spring 4对条件化配置的支持， 合理地推测应用所需的bean并自动化配置它们；最后， Spring Boot没有引入任何形式的代码生成，而是利用了Spring 4的条件化配置特性，以及Maven和Gradle提供的传递依赖解析，以此实现Spring应用程序上下文里的自动配置。简而言之， Spring Boot的自动配置是一个运行时（更准确地说，是应用程序启动时）的过程，考虑了众多因素，才决定Spring配置应该用哪个，不该用哪个。每当应用程序启动的时候， Spring Boot的自动配置都要做将近200个这样的决定，涵盖安全、集成、持久化、 Web开发等诸多方面。所有这些自动配置就是为了尽量不让你自己写配置。

3、命令行接口（Command-line interface， CLI）：Spring Boot的CLI发挥了Groovy编程语言的优势， 并结合自动配置进一步简化Spring应用的开发；

4、Actuator： 它为Spring Boot应用添加了一定的管理特性。

常见的搭建一个Spring Boot应用只要如下几步：

1、打开http://start.spring.io/

2、点击“Switch to the full version，选择Java版本，如1.8，选择好构建工具，如Maven或Gradle

4、点击Generate Project下载项目压缩包

5、解压后，使用eclipse或者intellij idea，导入项目即可

**前端Vue.js**

在我们的设计中，前端将会使用Vue.js进行开发。Vue.js推崇的是“单文件组件”的组织结构，即一个.vue文件中同时包含了页面、样式和脚本代码，并且像其他编程语言的import一样，可以被其它组件所包含。组件对外提供一些数据访问接口，外界可以通过这些接口向组件传递参数或从组件中获取返回值，但无法获得组件内部的运行信息。通过使用组件，Vue做到了前端开发的面向对象化，即信息的封装与隐藏。采用Vue.js作为前端开发框架，也正是为了体现这一点。

**RESTful API**

传统的Web App是后端与前端写在一起，后端使用类似Java Servlet的技术操控http请求，而前端则使用JSP这样的模板引擎生成。这样做的缺点是，前端和后端互相透明，能够通过模板引擎中的参数随意修改对方的值。显然，这样的开发方式是不利于信息的封装的。

使用了RESTful API，前端只需要在特定的时候调用后端提供的API，就可以获得其想要的数据，而无需也不能知道后端具体是如何工作的。同理，后端只需要为前端提供API，无法干涉前端的运作方式。如此这般，前后端都被各自封装起来，内部信息不会对外界暴露，只通过API来交换信息，做到了信息的封装和隐藏。

**总结**

总而言之，通过使用以下技术手段，我们的项目体现了很好的信息封装与隐藏：

* 面向对象语言与面向对象化的代码设计方法
* 前端将功能模块封装为单文件组件
* 前端与后端之间通过API交流，互相不暴露内部细节

## 5.4 前后端分离

项目的用户展示层有面向老师和学生的两个版本。采用的前后端分离方案，就是由后端提供统一的RESTful API接口，前端去调用以获得数据。虽然前端Vue.js的运行机制和代码结构都大相径庭，但由于接口是统一的，所以共用一套后端就可以。我们组的前端开发人员负责编写APP和前端页面的代码，后端开发人员负责编写后端代码，两组人员之间没有交叠，可以并行开发，靠的就是这一套设计得当的API。RESTful API传递的是与语言环境无关的json格式数据，借此我们做到了完全的前后端分离。

## 5.5 灵活性

由于前后端是分离开发的，因此在遇到新的需求或需求变更时，只需要新增或修改一下API接口，便可以前端后端同步进行开发，以实现新的需求。由于后端和前端都有良好的模块化特性，许多新需求变更只需要编写一个简单的模块，再将其嵌入整个系统便可以解决。

# 6 错误处理

## 6.1 前端

应对用户的错误输入是提高软件产品体验的重要一环。Vue.js和APP对于表单验证提供了非常好的支持，可以实时检测用户的输入是否满足特定的格式，甚至可以通过调用后端提供的API连接数据库进行验证。错误的输入将不会作为一个请求发送。

当服务器出现意外，返回错误的信息时，前端和小程序会及时根据HTTP Status Code判断发生了什么异常，并给予用户弹窗提醒。对于最常见的没有找到目标对象的错误，前端会根据后端提供的信息，给用户以详细的提示和操作指导。

## 6.2 后端

对与后端存在的错误可能是业务上的逻辑错误，也可能是框架层面的异常堆栈（空指针等）无论何种错误同一封装成

|  |  |  |
| --- | --- | --- |
| 变量名 | 含义 | 类型 |
| ret | 错误吗 | Int |
| msg | 错误消息 | String |
| 示例：  {  “ret”:101,  “msg”:”餐数studentid为空”  } | | |

错误码表（暂定）：

|  |  |
| --- | --- |
| 错误码 | 含义 |
| 101 | 餐数错误 |
| 102 | 服务器异常 |
| 103 | 数据不存在 |