

中山大学南方学院

装

订

线

本科生毕业论文（设计）

题目：对象模型数据存取BaaS

系 名：电气与计算机工程学院

专 业：计算机科学与技术

学 号：1420110422

姓 名：梁斯宇

指导教师：苑俊英

二○一八 年 四 月

# 摘 要

本项目采用RESTful服务架构，为用户提供简单的，高效的，规范的数据存取服务。

该服务主要面向学生团队、初创团队，完成版可定制面向小中型企业。致力解决各种服务数据存储的规范、效率问题。让大部分入门开发者都能轻易处理高并发场景、亿级数据场景。

项目设计使用nginx处理均衡负载，nodejs开发核心服务层、分库分表逻辑层，mysql担任持久层存储。

关键词：nodejs、高并发、亿级数据

# ABSTRACT

This project USES a RESTful service architecture to provide users with simple, efficient and standardized data access services.

The service is mainly aimed at the student team and the start-up team. The completed version can be customized to small and medium-sized enterprises. Strive to solve the specification and efficiency of various service data storage. Make it easy for most entry-level developers to deal with high concurrency scenarios and data scenarios.

The project design USES nginx to handle the balanced load, and nodejs develops the core service layer, the sorting table logic layer, and mysql as the persistence layer storage.

**Key words:** nodejs, High concurrency, Million level data

# 目 录

[摘 要 I](#_Toc369529349)

[ABSTRACT II](#_Toc369529350)

[目 录 III](#_Toc369529351)

[第1章 绪论 1](#_Toc369529352)

[1.1课题的背景和意义 1](#_Toc369529353)

[1.2国内市场现况 1](#_Toc369529354)

[第2章 开发运行环境 1](#_Toc369529363)

[2.1技术选型思路 2](#_Toc369529364)

[2.1.1 nodejs 2](#_Toc369529365)

[2.1.2 nginx 2](#_Toc369529366)

[2.1.3 egg.js 2](#_Toc369529367)

[2.1.4摘要格式 3](#_Toc369529368)

[2.1.5 redis 3](#_Toc369529368)

[2.1.6 mysql 3](#_Toc369529368)

[2.1.7 alinode 3](#_Toc369529368)

[2.2 服务端环境 3](#_Toc369529369)

[2.2.1操作系统 3](#_Toc369529370)

[2.2.2服务工具 4](#_Toc369529371)

[2.2.3存储工具 4](#_Toc369529372)

[2.3系统计算模式 4](#_Toc369529374)

[2.4开发环境 5](#_Toc369529378)

[第3章 系统需求分析 1](#_Toc369529363)

3[.1总体需求 5](#_Toc369529378)

[3.1.1 生产、下发侧需求 2](#_Toc369529365)

[3.1.2 管理侧、安全性能需求 2](#_Toc369529365)

3.2[需求拆分 5](#_Toc369529378)

[3.1.1类操作 5](#_Toc369529365)

[3.1.2 对象操作 5](#_Toc369529365)

[3.1.3 列表操作 6](#_Toc369529365)

[第4章 系统设计 6](#_Toc369529363)

[4.1系统结构设计 6](#_Toc369529378)

[4.1.1 系统概要设计 7](#_Toc369529365)

[4.1.2 系统业务详细设计 8](#_Toc369529365)

[4.2数据库设计 8](#_Toc369529378)

[4.3接口设计 10](#_Toc369529378)

[4.4系统用例设计 17](#_Toc369529378)

[4.4.1用例描述 18](#_Toc369529381)

[第5章 系统功能实现 19](#_Toc369529363)

[5.1环境配置准备 19](#_Toc369529378)

5[.1.1 nginx配置 20](#_Toc369529381)

[5.1.2 package.json 20](#_Toc369529381)

5[.2路由实现 22](#_Toc369529378)

5[.3 对象库表实现 24](#_Toc369529378)

5[.3.1分库分表规则 25](#_Toc369529381)

5[.3.1查询方式 26](#_Toc369529382)

5[.4 业务接入方式 26](#_Toc369529378)

5[.5 代码结构 28](#_Toc369529378)

[第5章 系统功能实现 29](#_Toc369529363)

[致 谢 30](#_Toc369529389)

[参考文献 31](#_Toc369529390)

[附 录 32](#_Toc369529391)

# 

# 第1章 绪论

## 1.1课题的背景和意义

课题目的设计一种利用restful api结构的系统，代替WEB应用对持久层的操作。

考虑到市面上大部分web应用开发中，对持久层的使用存在部署繁琐，大量数据处理方式不统一，数据结构不统一等问题，导致造成大量开发、运维成本，应用之间数据无法共用等问题。

Swapi完全不需要业务方部署任何服务环境，降低开发成本；部分字段规范命名，加强数据共通性；自带分库分表，读写分离。意义在于让高并发web应用开发重回简单的CURD操作。

## 1.2国内市场现况

中小型企业基本不使用统一的对象模型存储系统，系统本身存在开发成本与维护成本，但像阿里巴巴UC，腾讯微信，中国联通等稍有规模的企业都有一套自己的restful api对象模型存储系统。

但中小型企业，甚至初创团队，学生团队并不存在这类系统。对他们而言，虽然有一定的接入成本，但在未来企业规模变大时，将会遇到数据。这些问题很需要抽象为统一由一套存储系统解决，或者由一个存储平台解决。

# 第2章 开发、运行还环境

## 2.1技术选型思路

考虑到该系统平台使用场景有高并发、数据量中等多、多业务等特点，因此选用nginx做http代理服务器，使用nodejs作为系统核心开发工具，持久层服务使用mysql，缓存服务使用redis。Linux和windows都可使用，但推荐使用linux系列操作系统。

在此基础上借助阿里巴巴开源框架egg.js搭建服务，借助微软开源框架napa.js实现node多线程操作以实现分库分表的增删查改操作。

### 2.1.1 Nodejs:

Node.js作为基于chrome V8引擎的一个js虚拟机（运行环境），拥有事件驱动，非阻塞I/O模型，轻量高效等属性。这里主要看重nodejs的单线程异步非阻塞属性，能够高效、精准、低成本地处理高并发场景。

### 2.1.2 Nginx:

Nginx (engine x) 是一个高性能的[HTTP](https://baike.baidu.com/item/HTTP" \t "_blank)和[反向代理](https://baike.baidu.com/item/%E5%8F%8D%E5%90%91%E4%BB%A3%E7%90%86" \t "_blank)服务器，也是一个IMAP/POP3/SMTP[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8)。

Nginx 可以在大多数 [Unix](https://baike.baidu.com/item/Unix" \t "_blank)Linux OS 上编译运行，并有 [Windows](https://baike.baidu.com/item/Windows) 移植版。 Nginx 的1.4.0稳定版已经于2013年4月24日发布，一般情况下，对于新建站点，建议使用最新稳定版作为生产版本，已有站点的升级急迫性不高。

Nginx 的[源代码](https://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81" \t "_blank)使用 2-clause BSD-like license。

Nginx 是一个很强大的高性能[Web](https://baike.baidu.com/item/Web/150564" \t "_blank)和[反向代理](https://baike.baidu.com/item/%E5%8F%8D%E5%90%91%E4%BB%A3%E7%90%86" \t "_blank)服务器，它具有很多非常优越的特性：

在连接高并发的情况下，Nginx是[Apache](https://baike.baidu.com/item/Apache/6265" \t "_blank)服务器不错的替代品：Nginx在美国是做虚拟主机生意的老板们经常选择的软件平台之一。能够支持高达 50,000 个并发连接数的响应，感谢Nginx为我们选择了 epoll and kqueue作为开发模型。

Nginx作为一个高性能http代理和反向代理服务器，拥有占用内存少，并发能力强等属性，能够胜任解决均衡负载的问题。在nodejs服务的基础上，用nginx反向代理http请求，能够有效应付高并发场景

### 2.1.3 Egg.js:

我们深知企业级应用在追求规范和共建的同时，还需要考虑如何平衡不同团队之间的差异，求同存异。所以我们没有选择社区常见框架的大集市模式（集成如数据库、模板引擎、前端框架等功能），而是专注于提供 Web 开发的核心功能和一套灵活可扩展的插件机制。我们不会做出技术选型，因为固定的技术选型会使框架的扩展性变差，无法满足各种定制需求。通过 Egg，团队的架构师和技术负责人可以非常容易地基于自身的技术架构在 Egg 基础上扩展出适合自身业务场景的框架。

Egg.js作为nodejs社区的一个为企业级项目而生的框架，能解决函数式开发系统规范性问题，有效降低系统后续维护成本。并且社区活跃，提供大量开源支撑。

### 2.1.4 Napa.js:

Napa.js是microsoft提供的开源nodejs模块。该项目可解决nodejs单线程的短板。在本系统里面，Napajs主要为nodejs的分库分表的并发操作进行支撑。

### 2.1.5 Redis:

redis是一个key-value[存储系统](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E7%B3%BB%E7%BB%9F)。和Memcached类似，它支持存储的value类型相对更多，包括string(字符串)、list([链表](https://baike.baidu.com/item/%E9%93%BE%E8%A1%A8))、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。这些[数据类型](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B" \t "_blank)都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis支持各种不同方式的排序。与memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。

Redis是一个基于内存的key-value数据库，并提供网络操作接口。拥有高速读写数据的特色，在本系统中负责存储缓存数据。

### 2.1.6 Mysql

MySQL是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

老牌开源关系型数据库。本系统中负责存储持久层数据，

### 2.1.7 alinode

Alinode是阿里巴巴集团一款收费产品。是基于nodejs的nodejs运行环境性能监控平台，致力于监控nodejs服务在服务过程中的CPU、GC、内存使用量，并提供日志追溯、服务报警等功能。

## 2.2 服务端环境

### 2.2.1 操作系统

Ubuntu 16.04：

Ubuntu基于Debian发行版和[GNOME](https://baike.baidu.com/item/GNOME)桌面环境，而从11.04版起，Ubuntu发行版放弃了Gnome桌面环境，改为[Unity](https://baike.baidu.com/item/Unity/5779064)，与Debian的不同在于它每6个月会发布一个新版本。Ubuntu的目标在于为一般用户提供一个最新的、同时又相当稳定的主要由[自由软件](https://baike.baidu.com/item/%E8%87%AA%E7%94%B1%E8%BD%AF%E4%BB%B6)构建而成的操作系统。Ubuntu具有庞大的社区力量，用户可以方便地从社区获得帮助。Ubuntu对[GNU/Linux](https://baike.baidu.com/item/GNU%2FLinux" \t "_blank)的普及特别是桌面普及作出了巨大贡献，由此使更多人共享开源的成果与精彩。

### 2.2.2 服务工具

Nodejs：alinode2.1（对应node6.10）

Nginx：stable

Egg.js：1.10

Napa.js：stable

### 2.2.3 存储工具

Redis: stable

Mysql: 5.5

## 2.3 系统计算模式

**BaaS(backend as a service）**

解释：BaaS 是一种新型的云服务，旨在为移动和 Web 应用提供后端云服务，作为应用开发的新模型，能有效降低开发者成本，让开发者只需专注于具体的业务开发之上。

BaaS是[移动中间件](http://baike.baidu.com/view/5578531.htm" \t "_blank)的替代品，它使用统一的API和SDK来连接移动应用到后端云存储，传统的移动中间件通过本地的物理服务把后端服务集成到应用中。而BaaS通过云来集成后端服务

本系统应用模式主要是platform to bussiness的结构，平台信赖业务，业务开发应用，鉴权使用平台。

**BaaS后端即服务**为应用开发提供后台的云服务，在当前的移动互联网+云计算热潮中，尤其为这2类应用提供了大量的技术支持。包括提供存储、托管环境，也包括提供推送等通行后端技术能力。有单独的**mBAAS**的说法，但大多数人习惯将BAAS等同于为移动互联网应用开发而提供的云服务。目前以后云计算巨头也在进入这个领域，比如亚马逊AWS，微软的Windows Azure、Google App Engine和Rackspace。  
  **BaaS的主要功能**：自定义的数据对象和存储客户数据、支持Facebook和Twitter账号登陆、本地数据、分析和度量、自定义的数据对象和存储客户数据、支持推送通知。

## 2.4 开发环境

Windows或macos或ubuntu系统开发，环境安装编辑器、mysql5.5（可远程）、nodejs6.10、redis（可远程）、cnpm包管理工具即可。

# 第3章 系统需求分析

## 3.1 总体需求

本平台主要面向业务，服务业务的数据存储、读取压力等问题，提供业务在数据存储方面的帮助。

平台主要面向业务的三个角度进行服务，包括数据生产侧、数据下发侧、数据管理侧。

需要完成的功能包括数据生产的写入、更新、删除操作，数据下发的读取、索引功能，平台对高并发场景内的容量，数据整体的容量需要达到一个可兼容绝大部分业务需求的程度。

### 3.1.1 生产、下发侧需求

1、类的增删查

对类的创建，查询，修改等操作

2、类对象的增删查改

对对象的CURD操作，包括建立索引、ES查询

3、列表的增删查

对列表的创建、查询等操作

4、列表元素的增删查

对列表数据的增删操作，类似建立索引

5、查询缓存

所有查询操作都建立缓存，在更新操作中进行清空

### 3.1.2 管理侧、安全性能需求

1、登陆鉴权类操作

管理侧需要业务app鉴权，鉴权力度细化到所有接口

2、数据模型操作

对生产、下发侧接口的直接操作

3、API使用目录

具体操作的Api使用简要说明

4、防注入与XSS

系统接口的安全性问题，对于简单的注入攻击、XSS攻击做好防御。

5、单台服务器本地压测3K+（双核）

对于单个worker的模式，要求压测要到3K qps

6、Nodejs进程守护放

使用mastter/worker模式对nodejs进程进行崩溃维护

7、Nginx均衡负载

用nginx反向代理转发HTTP请求，减轻个别node服务的压力。

8、运行性能监控

## 3.2 需求拆分

### 3.2.1类操作：

·创建类：传入属性字段，主键。创建对应的对象表（分库分表，建立索引主键）

·查询类列表：查询类表中所有类

### 3.2.2对象操作：

·创建对象：传入字段内容，主键，创建对应类对象表中的记录。

·查询对象：通过类名查询对象列表

·修改对象：通过对象ID修改对象非主字段

·删除对象：逻辑删除类表中的对象

·上线对象：修改对象status为上线状态

·下线对象：修改对象status为下线状态，不可被搜索。聚合场景使用

### 3.2.3列表操作：

·创建列表：自定义category\_id（分类）和list\_id（分类内唯一ID）和classname（类名）

·查询列表：传入category\_id、list\_id和classname，查询列表中对应的对象ID列表。

·关联查询：在查询的基础上，关联查询出该列表中对象详细属性

·新增列表中数据：插入对象到列表中

·删除列表中数据：删除列表中的数据

# 第4章 系统设计

## 4.1 系统结构设计

### 4.1.1系统概要设计：

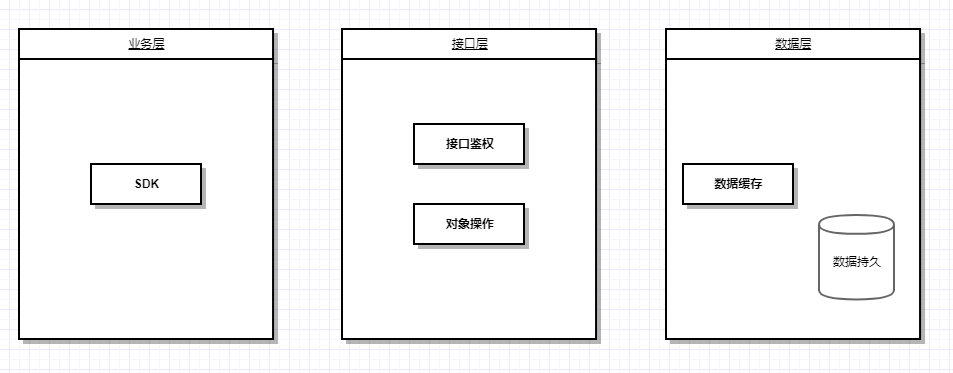


图4-1 系统概要设计图

### 业务层

关注点主要集中在业务规则的制定、业务流程的实现等与业务需求有关的[系统设计](https://baike.baidu.com/item/%E7%B3%BB%E7%BB%9F%E8%AE%BE%E8%AE%A1)，也即是说它是与系统所应对的领域（Domain）逻辑有关，很多时候，也将业务逻辑层称为领域层。主要从业务的角度思考，平台不做过多考虑

SDK：平台提供业务方多语言版本的SDK，方便业务对平台的调用。

### 接口层

接口层关注点主要在业务与平台之间的交互通道。解决请求是否安全，请求量是否可承载等问题

鉴权：为确保平台接口都是被信赖的来源请求，需要在接口保证业务数据的安全性，用实时生成的签名做鉴权。

接口：平台提供类、对象、列表的增删改查接口，

### 持久层

持久层主要是解决平台数据的存取方面的问题，关注点在主从分离、容灾、分库分表、索引查询等问题上

缓存服务：为了减少硬盘IO，需要在某些场景设置把部分请求缓存起来，方便下次直接从内存中获取结果。

### 4.1.2系统详细设计（业务层）

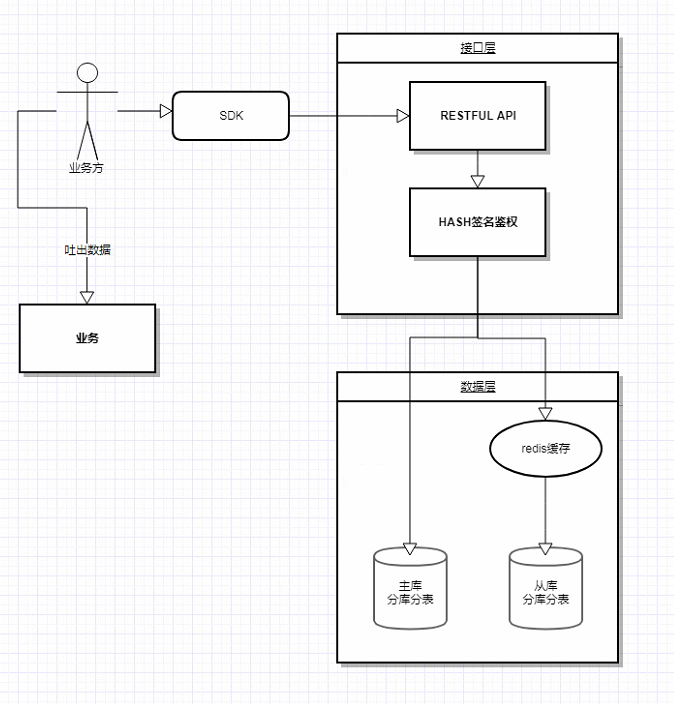


图4-2 业务概要设计图

业务方使用SDK或原生http请求操作使用平台接口，然后业务自身实现业务逻辑，平台不关注业务方业务逻辑。平台接口层接受业务方请求后，首先走鉴权逻辑，确定业务方是否可信赖。然后若是设置了缓存的操作，则再缓存中查询是否需要使用缓存数据。最后才到数据持久层的操作，分库分表查询后聚合

## 4.2 数据库设计

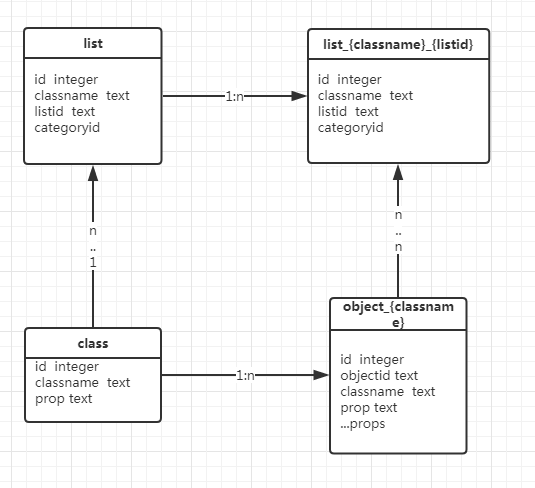


图4-3 数据关系图

平台数据主要由2张表+2堆表实现。

类表存放业务下所有类的名称、属性、ID等初始化信息，主要用于查询，对管理侧，聚合下发侧着重开放，但由于单表实现，所以不建议对下发侧开放

列表表主要用于存放系统建立的列表。列表可以理解为对象与对象之间的映射表。不同的对象可以说是有不同的类型，这一类对象可以是另一类对象的聚合体，通过列表的方式进行关联即可。

对象表群是用于存放所有的对象信息，用objectid做分库分表索引。在ES查询中另外建立专用的查询映射索引关系。

列表群则用于存放对象之间的索引关系而存在。

### 1、Class表

作用：存放业务下的所有类

解释：该表只存在一个，主要用于存放业务建立了的类的列表，方便后续管理，查询。

使用场景：该表一般情况下不对下发方开放，只对管理测开放使用。

表4-1 class表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 类型 | 默认值 | 是否可空 | 说明 |
| id | Long |  | 否 | 自增ID |
| classname | Text |  | 否 | 类名，唯一 |
| props | text |  | 否 | 对象属性字段 |

### 2、List表

作用：存放业务建立的所有列表信息

解释：LIST表一般用objectid作为listid，用于建立一个对象与一堆对象之间的关联关系。Classname作为列表数据项的类，listid为业务方任意配置。

使用场景：该表一般不对下发侧开放，只对管理侧开放使用。

表4-2 List表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 是否可空 | 说明 |
| id | Long |  | 否 | 自增ID |
| listid | text |  | 否 | 列表唯一ID |
| classname | Text |  | 否 | 列表父级类名 |
| categoryid | Text |  | 分类ID | 代表列表类型（业务指定） |

### 3、Object表

作用：此表为动态生成的一组表，用于存放类创建的对象内容。

解释：业务创建类的同时会创建一个对象表，往后用这个类创建数据对象时，则在对象的对象表中插入数据。此表用于存放其继承类的所有实例化数据对象。

使用场景：内容池的管理，对象内容的存放处，聚合下发的搜索来源。

表4-3 Object表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 是否可空 | 说明 |
| id | Long |  | 否 | 自增ID |
| objectid | text |  | 否 | 列表唯一ID |
| classname | Text |  | 否 | 对象类名 |
| …props | Text |  |  | 对象的其他属性字段 |

### 4、列表对象关联表

作用：此表为动态生成的一组表，用于存放列表内容

解释：LIST表一般用objectid作为listid，用于建立一个对象与一堆对象之间的关联关系。Classname作为列表数据项的类，listid为业务方任意配置。该表主要用于建立对象与对象之间的映射关系。

使用场景：该表主要作用于下发侧，用于聚合内容产出。以及建立对之间的映射关系。

表4-4 列表对象关系表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 默认值 | 是否可空 | 说明 |
| id | Long |  | 否 | 自增ID |
| listid | text | 是 | 否 | 列表唯一ID |
| classname | Text | 是 | 否 | 列表父级类名 |
| categoryid | Text | 是 | 分类ID | 代表列表类型（业务指定） |

## 4.3 接口设计

### 公共query参数说明

表4-5 公共query参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| appid | String | 是 | 业务id |  |
| sign | String | 是 | 签名  md5(appid+appkey+tm) |  |
| tm | long | 是 | 毫秒级时间戳 |  |

### 公共请求头说明

表4-6 公共请求头

|  |  |
| --- | --- |
| 参数名 | 值 |
| Content-Type | Application/json |

### 错误码定义

表4-7 错误码定义

|  |  |  |
| --- | --- | --- |
| 错误码 | 含义 | 备注 |
| 0 | 正确 |  |
| -4xx | 请求错误 |  |
| -400 | 鉴权失败 |  |
| -401 | 请求参数缺失或类型错误 |  |
| -403 | 无权请求数据 |  |
| -404 | 请求数据不存在 |  |
| -5xx | 服务错误 |  |
| -500 | 系统异常报错 |  |
| -501 | 数据库异常异常 |  |

### 公共对象字段说明

表4-8 公共字段说明

|  |  |  |
| --- | --- | --- |
| 字段名 | 含义 | 备注 |
| status | 代表对象可用性 |  |
| create\_at | 对象创建日期 | Mysql处理 |
| update\_at | 对象更新日期 | Mysql处理 |
| objected | 对象唯一id | 平台生成 |

### 类操作

1、获取类列表

GET /class/read

Paramters：

表4-9类操作参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| page | integer | 否 | 页数 默认1 |  |
| size | integer | 否 | 页容量 默认10 |  |

Response:

{

“code” : 状态码

“data”: [], //类列表内容

“total” : 总数

}

2、新增类

POST /class/create

Paramter

表4-10 新增类操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
|  |  |  |  |  |

Body:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| classname | String | 是 | 类名 | 全局唯一类名 |
| prop | String | 是 | Key1`key2  类的属性字段 |  |

Response:

{

“code” : 状态码

“message”：状态信息

“classname” : 创建的类名

}

### 2、对象操作

1、获取对象列表

GET /object/read

Paramters：

表4-11 获取对象参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| classname | string | 是 | 类名 |  |
| page | integer | 否 | 页数 默认1 |  |
| size | integer | 否 | 页容量 默认15 |  |
| fetch | integer | 否 | 是否拉去关联对象内容 |  |

2、创建对象

POST /object/create

body：

表4-12 创建对象参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| classname | String | 是 | 代表类对应的对象名 |  |
| prop | String | 是 | Kv结构字段  Key1=value1`key2=value2 |  |

Response:

{

“code”:状态码，

“message”：状态信息

}

3、修改对象

PATCH /object/update

body：

表4-13 修改对象参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| classname | String | 是 | 代表类对应的对象名 |  |
| prop | String | 是 | Kv结构字段  Key1=value1`key2=value2 |  |

Response:

{

“code”:状态码，

“message”：状态信息

}

4、删除对象

DELETE /object/delete

Paramters：

表4-14 删除对象操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| classname | String | 是 | 代表类对应的对象名 |  |
| objectid | String | 是 | 对象唯一ID |  |

Response:

{

“code”:状态码，

“message”：状态信息

}

5、上线对象

POST /object/activite

Paramters：

表4-15 上线对象操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| objectid | String | 是 | 对象唯一ID |  |

Response:

{

“code”:状态码，

“message”：状态信息

}

6、下线对象（修改对象status值）

POST /object/inactivite

Paramters：

表4-14 下线对象操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| objectid | String | 是 | 对象唯一ID |  |

Response:

{

“code”:状态码，

“message”：状态信息

}

### 3、列表操作

1、获取列表信息

GET /list/read

Paramters：

表4-15获取列表操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| classname | string | 是 | 类名 |  |
| categoryid | string | 是 | 列表分类 |  |
| listid | string | 是 | 列表唯一ID |  |
| fetch | integer | 否 | 是否获取列表关联数据 | 1代表获取关联数据 |
| fetch\_total | integer | 否 | 是否获取总数 | 1代表获取总数 |
| size | integer | 否 | 页容量 默认15 |  |
| fetch | integer | 否 | 是否拉去关联对象内容 |  |

Response:

{

“code”:状态码，

“message”：状态信息

}

2、创建列表

POST /list/create

body：

表4-16 创建列表操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| listid | String | 是 | 列表唯一ID |  |
| classname | string | 是 | 列表对应的类 |  |
| categoryid | String | 是 | 列表分类 |  |

Response:

{

“code”:状态码，

“message”：状态信息

}

3、添加列表数据项

PUT /list/add

body：

表4-17 填充列表操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| listid | String | 是 | 列表唯一ID |  |
| classname | string | 是 | 列表对应的类 |  |
| categoryid | String | 是 | 列表分类 |  |
| objected | string | 是 | 对象ID | 必须同个类 |

Response:

{

“code”:状态码，

“message”：状态信息

}

4、删除列表数据项

DELETE /list/delete

body：

表4-18 删除列表对象操作

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数名 | 类型 | 是否必传 | 说明 | 备注 |
| listid | String | 是 | 列表唯一ID |  |
| classname | String | 是 | 列表对应的类 |  |
| categoryid | String | 是 | 列表分类 |  |
| objected | String | 是 | 对象ID |  |

Response:

{

“code”:状态码，

“message”：状态信息

}

## 4.4 系统用例设计

表4-19 系统用例图

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 对象操作 | 类操作 | 列表操作 | 业务配置 |
| 业务下发侧 | 1 |  | 1 |  |
| 业务管理侧 | 1 | 1 | 1 |  |
| 业务生产侧 | 1 |  | 1 |  |
| 系统管理侧 | 1 | 1 | 1 | 1 |

### 4.4.1 用例描述

### 对象操作

表4-20 对象操作用例描述

|  |  |
| --- | --- |
| 用例名称 | 对象操作 |
| 描述 | 业务方对对象的CURD操作 |
| 参与者 | 业务方（全），平台管理方 |
| 前置条件 | 操作鉴权、平台登陆 |
| 后置条件 | 操作成功；操作失败、平台申请appid |
| 基本操作 | CURD对象  搜索对象（ES） |

### 类操作

表4-21 类操作用例描述

|  |  |
| --- | --- |
| 用例名称 | 类操作 |
| 描述 | 业务方对类的CURD操作 |
| 参与者 | 业务方管理侧，平台管理方 |
| 前置条件 | 操作鉴权、平台登陆 |
| 后置条件 | 操作成功；操作失败、平台申请appid |
| 基本操作 | 查询业务类表  创建类  删除类 |

### 列表操作

表4-22 列表操作用例描述

|  |  |
| --- | --- |
| 用例名称 | 列表操作 |
| 描述 | 业务方对列表创建，添加元素操作 |
| 参与者 | 业务方（全），平台管理方 |
| 前置条件 | 操作鉴权、平台登陆 |
| 后置条件 | 操作成功；操作失败、平台申请appid |
| 基本操作 | 创建列表  修改列表属性  添加列表元素  删除列表元素  排序列表元素  查询列表 |

### 业务配置

表4-14 业务配置用例描述

|  |  |
| --- | --- |
| 用例名称 | 业务配置 |
| 描述 | 平台对接入业务的配置管理 |
| 参与者 | 平台管理方 |
| 前置条件 | 平台登陆 |
| 后置条件 | 操作成功；操作失败、申请管理员权限 |
| 基本操作 | 查询业务列表、详情  创建业务  修改业务  上下线业务 |

# 第5章 系统功能实现

## 5.1 环境配置准备

### 5.1.1 nginx配置

|  |
| --- |
| 1. user nobody; 2. worker\_processes  8; 3. events { 4. use   epoll; 5. worker\_connections  4096; 6. } 8. http { 9. include    mime.types; 10. default\_type  application/octet-stream; 11. log\_format  main  '$remote\_addr - $remote\_user [$time\_local] "$request" ' 12. '$status $body\_bytes\_sent "$http\_referer" ' 13. '"$http\_user\_agent" "$http\_x\_forwarded\_for"'; 15. access\_log  logs/access.log  main; 16. sendfile     on; 17. keepalive\_timeout  65; 18. tcp\_nodelay     on; 20. gzip  on; 21. gzip\_disable "MSIE [1-6]."; 23. client\_header\_buffer\_size    128k; 24. large\_client\_header\_buffers  4 128k; 26. server { 27. listen    80; 28. server\_name  www.cybergear-cn.com; 29. root html; 30. access\_log  logs/nginx.access.log  main; 31. location / { 32. index index.php index.html index.htm; 34. } 36. error\_page   500 502 503 504 /50x.html; 37. location = /50x.html { 38. } 40. location ~ ^/(images|javascript|js|css|flash|media|static)/ { 42. expires 30d; 43. } 44. } 45. } |

### 5.1.2 package.json

|  |
| --- |
| 1. { 2. "name": "swapi", 3. "version": "1.0.0", 4. "description": "rebuild sapi", 5. "main": "server/startup.js", 6. "scripts": { 7. "test": "node server/startup.js" 8. }, 9. "repository": { 10. "type": "git", 11. "url": "git+https://github.com/SteveWooo/swapi.git" 12. }, 13. "keywords": [ 14. "BaaS" 15. ], 16. "author": "SteveWoo", 17. "license": "Apache-2.0", 18. "bugs": { 19. "url": "https://github.com/SteveWooo/swapi/issues" 20. }, 21. "homepage": "https://github.com/SteveWooo/swapi#readme", 22. "dependencies": { 23. "crypto": "^1.0.1", 24. "express": "^4.15.3", 25. "mysql": "^2.14.0" 26. } 27. } |

## 5.2 路由实现

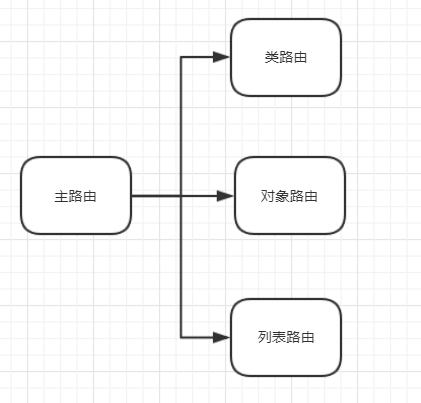


图5-1 路由概要图

### 模块代码实现：

|  |
| --- |
| 1. let classCtrl = require('./class/classHandle').handle, 2. objCtrl = require('./obj/objHandle').handle, 3. listCtrl = require('./list/listHandle').handle; 5. let router = { 6. 'class' : classCtrl, 7. 'obj' : objCtrl, 8. 'list' : listCtrl 9. } 11. // 封装 请求path 12. // req.type;req.option; 13. let getReq = (req, callback)=>{ 14. let path = req.path.split('/').slice(1); 15. **if**(path[0] === 'favicon.ico'){ 16. callback("200"); 17. **return** undefined; 18. } 20. //赋值方法 21. req.type = path[0]; 22. req.option = path[1]; 24. //校验路由是否存在 25. **if**(!router[req.type]){ 26. callback(403); 27. **return** undefined; 28. } 30. **return** req; 31. } 33. /\* 34. \* 入口，处理全局回掉以及路由入口 35. \*/ 36. let handle = (req, res, next)=>{ 37. //回调出口 38. let callback = (data, message)=>{ 39. **var** result; 40. req.mysql.end(); 41. //直接返回状态码 42. **if**(**typeof** data == 'number'){ 43. res.sendStatus(data); 44. **return** ; 45. } 47. //返回错误码 48. **if**(**typeof** data == 'string'){ 49. result = { 50. data : data, 51. message : message 52. } 54. res.send(JSON.stringify(result)); 55. **return** ; 56. } 58. //正常回调 59. **if**(**typeof** data == 'object'){ 61. } 62. res.send(data); 63. **return** ; 64. } 65. //封装request 66. req = getReq(req, callback); 67. **if**(!req) { 68. **return** undefined; 69. } 71. //路由 72. router[req.type](req, callback); 73. } 75. exports.handle = handle; |

## 5.3 对象库表实现

随着一个网站的业务不断扩展，数据不断增加，数据库的压力也会越来越大，对数据库或者SQL的基本优化可能达不到最终的效果，我们可以采用读写分离的策略来改变现状

数据切分可以是物理上的，对数据通过一系列的切分规则将数据分布到不同的DB服务器上，通过路由规则路由访问特定的数据库，这样一来每次访问面对的就不是单台服务器了，而是N台服务器，这样就可以降低单台机器的负载压力。

数据切分也可以是数据库内的，对数据通过一系列的切分规则，将数据分布到一个数据库的不同表中，比如将article分为article\_001，article\_002等子表，若干个子表水平拼合有组成了逻辑上一个完整的article表，这样做的目的其实也是很简单的。举个例子说明，比如article表中现在有5000w条数据，此时我们需要在这个表中增加（insert）一条新的数据，insert完毕后，数据库会针对这张表重新建立索引，5000w行数据建立索引的系统开销还是不容忽视的。但是反过来，假如我们将这个表分成100 个table呢，从article\_001一直到article\_100，5000w行数据平均下来，每个子表里边就只有50万行数据，这时候我们向一张 只有50w行数据的table中insert数据后建立索引的时间就会呈数量级的下降，极大了提高了DB的运行时效率，提高了DB的并发量。当然分表的好处还不知这些，还有诸如写操作的锁操作等，都会带来很多显然的好处。

### 5.3.1分库分表规则：

|  |
| --- |
| **总库目：M**  **总表数：N**  **中间变量middle = {hash(primary id) % (M\*N)}**  **库名：{业务名\_swapi}+{取整（middle/N）}**  **表名：{表名}+{取整（middle%N）}** |

### 5.3.2查询方式

接口传入查询条件，利用napa.js对全库表实现并发查询，然后聚合内容进行排序、游标截取后返回业务即可

设计表的时候需要确定此表按照什么样的规则进行分库分表。例如，当有新用户时，程序得确定将此用户信息添加到哪个表中；同理，当登录的时候我们得通过用户的账号找到数据库中对应的记录，所有的这些都需要按照某一规则进行。

通过分库分表规则查找到对应的表和库的过程。如分库分表的规则是user\_id mod 4的方式，当用户新注册了一个账号，账号id的123,我们可以通过id mod 4的方式确定此账号应该保存到User\_0003表中。当用户123登录的时候，我们通过123 mod 4后确定记录在User\_0003中。

a.通过扫表的方式解决，此方法基本不可能，效率太低了。   
  b.记录两份数据，一份按照用户纬度分表，一份按照商品维度分表。   
  c.通过搜索引擎解决，但如果实时性要求很高，又得关系到实时搜索。

### 5.3.3创建方式

创建类的同时，创建对应的对象表即可。

## 5.4 业务接入

### 5.4.1鉴权规则

|  |
| --- |
| **业务标识：appid**  **业务密钥：appkey**  **时间戳（毫秒）：ts**  **签名规则：sign = md5\_32(appid+ts+appkey)** |

### 5.4.2接入流程

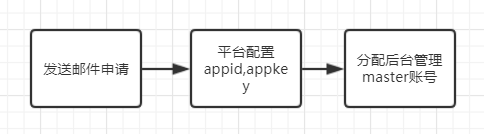


图5-2 业务接入

## 5.5代码结构

Root

-Controller

-class

Handle.js

-object

Handle.js

-list

Handle.js

-Model

-class

Create.js

Delete.js

Update.js

Get.js

-object

Create.js

Delete.js

Update.js

Get.js

Search.js

Operate.js

-list

Create.js

Put.js

Get.js

Delete.js

-Server

Server.js

Startup.js

Config.json

Package.json

-node\_modules

# 第6章 总结

该平台在开发前期的想法是单纯想给前端同学实现一套公用的数据村存储系统，解决前端同学对于服务端不熟悉、不擅长数据操作等问题。但后来在实习单位中看到两套类似的数据存储系统，老的一套比较成熟稳定，新的一套解决更多痛点难点，受到极大的启发，遂重新设计该平台，争取实现更多功能，能够解决更多的业务需求。

系统在开发过程中经历了4次重新设计，2次代码重构。从一开始的纯原生nodejs+内存缓存，到express框架+redis，再到后来添加nginx做均衡负载，cluster做进程守护，多进程任务处理等等，到现在完整的一套系统设计。

平台的目的是跟业务尽可能低耦合的同时，适应其他各个业务中去。因此再一开始的版本中，平台是设计得比较臃肿的，对各个业务都有耦合的关系。后来一削再削，把业务的独立性质一点一点削弱，从平台的角度思考系统的设计问题。

# 致 谢

# 参考文献

[1] 朴灵．《深入浅出nodejs》 人民邮电出版社，2016．3

[2] 徐涛 《javascript设计模式》人民邮电出版社 2016.3

# 附 录

代码仓库：Github:https://www.github.com/SteveWooo/swapi.git

**附表一 毕业论文（设计）开题报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 课题 |  | | |
| 学号 | 142011042 | 姓名 | 梁斯宇 |
| 专业 | 计算机科学与技术 | 指导教师 | 苑俊英 |
| 一、课题来源及意义  随着web前端技术的发展，近几年开发者在前端的开发、维护上相比过去几年有了较高水平的效率提升，进而越来越多的程序员扎根前端领域的开发。这种情况下遇到的问题是，前端开发对于后端数据的使用，环境搭建方面并不熟悉，对于大数据的处理，高并发场景，都无法通过简单的mysql实现。因此有该课题的来源，目的把web应用地持久数据在统一、规范的地方存储、下发。以平台业务接入的方式，代替一个应用部署一套存储系统的思想，给应用多一个数据存储的选择  二、国内外发展状况  类似的存储平台，目前在各大公司中都会根据该公司的业务进行抽象设计开发，一般来说这类平台都对该公司的业务友好，但不存在以普遍适用性为目的平台设计思路。  三、研究目标、内容  目标是研发一个能在一定范围数据规范约束之下的，代替普通数据库的服务平台。  四、方法、手段  以平台的思路来解决数据存储问题  五、进度安排  已完成  六、实验方案的可行性分析  一定规模的公司内部有现成的系统，并有两到三年的迭代，且在此之上还有第二套类似的升级版系统开发。  七、实验条件  开发demo条件：一台阿里云学生机  八、具体参考文献 | | | |
| 选题是否合适： 是 否  课题能否实现： 能 不能  指导教师（签字）  年 月 日 | | | |

**附表二 毕业论文（设计）过程检查情况记录表**

|  |  |
| --- | --- |
| 指导教师分阶段检查论文的进展情况（要求过程检查记录不少于3次）  **第1次检查**  学生总结：  进行调查，确定系统所需要的功能，对整个系统的功能行详细了解，并  总结分析，确定需求，确定系统的整个操作流程，并且进行模拟的检验，看看  会不会漏掉什么或者哪方面出问题  指导教师意见：  需求确定后即可进入系统设计与实现，设计需尽可能详细，在后续开发中  按照设计进行，才能做到有条不紊  **第2次检查**  学生总结：  初步完成系统整体功能开发，并根据不同业务场景考虑兼容情况，在实际使用操作的情况下重构两次系统功能，并达到满意状态  指导教师意见：  系统配套界面操作更佳  **第3次检查**  学生总结：  对管理后台进行界面开发，但由于对于该平他核心功能作用实在不会不大，因此把精力耗费在分库分表、均衡负载等问题研究上  指导教师意见：  毕业论文内容充实，结构合理，但是需要增加界面设计部分。并对论文进  行格式化设置。  学生签名： 年 月 日  指导教师签名： 年 月 日 | |
| **总**  **体**  **完**  **成**  **情**  **况** | 指导教师意见：  1、按计划完成，完成情况优秀（ ）  2、按计划完成，完成情况良好（ ）  3、按计划完成，完成情况中等（ ）  4、基本按计划完成，完成情况及格（ ）  5、完成情况不及格（ ）  指导教师签名： 年 月 日 |

**附表三 毕业论文（设计）答辩情况表**

|  |  |  |  |
| --- | --- | --- | --- |
| 答辩人 |  | 专 业 |  |
| 论文（设计）题目 |  | | |
| 答辩小组成员 |  | | |
| 答辩记录：  记录人签名： 年 月 日 | | | |

封底 **毕业论文（设计）成绩评定记录**

|  |
| --- |
| 指导老师评语：  成绩评定：  指导老师签名： 年 月 日 |
| 答辩小组或专业负责人意见：  成绩评定：  签名（章）： 年 月 日 |
| 系负责人意见：  成绩评定：  签名（章）： 年 月 日 |

注：本表不装订在论文内