中山大学南方学院本科生毕业论文（设计）

对象模型高并发数据存取BaaS服务

（SwApi）

学院名：电气与计算机工程学院

专业：计算机科学与技术

学号：142011042

姓名：梁斯宇

指导教师：苑俊英

二○一七年十一月

摘 要

本项目采用RESTful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

目录

第一章 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。

1.3论文结构

论文总体分三部分，系统搭载环境、系统开发设计需求、系统使用说明

1.4基础术语解析

第二章 开发、运行环境

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**:

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**:

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

**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。

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。

第三章 系统需求分析

一、总体需求

下发侧操作：

1. 类的增删查
2. 类对象的增删查改
3. 列表的增删查
4. 列表元素的增删查
5. 查询缓存

管理侧操作：

1. 登陆鉴权类操作
2. 数据模型操作
3. API使用目录

安全：

1. 防SQL注入与XSS攻击
2. 单台服务器本地压测3K+（双核）
3. Nodejs进程守护放崩溃
4. Nginx均衡负载

二、需求拆分

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

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、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

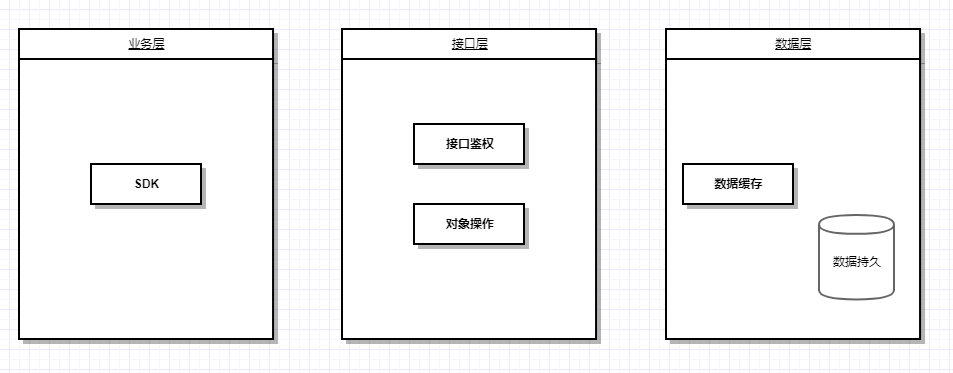
·删除列表中数据：删除列表中的数据

三、接口文档

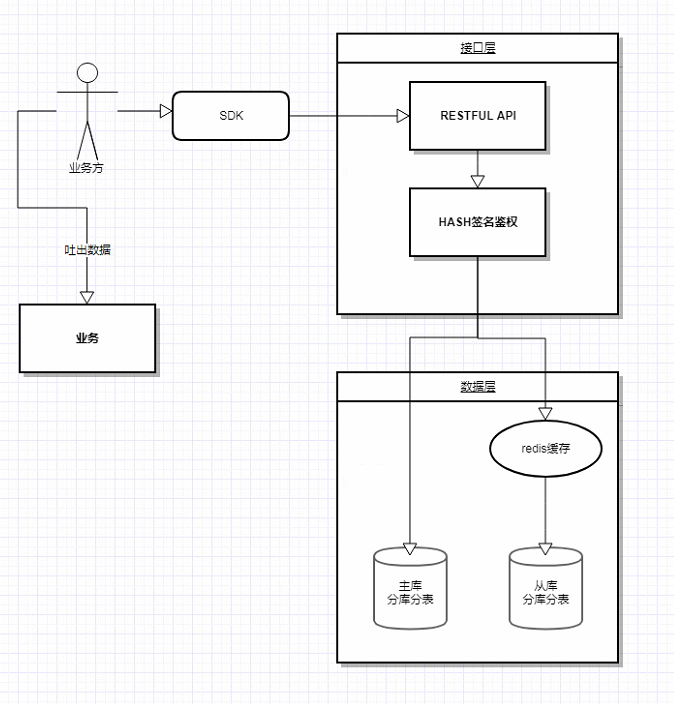
第四章 系统设计

4.1 系统结构设计

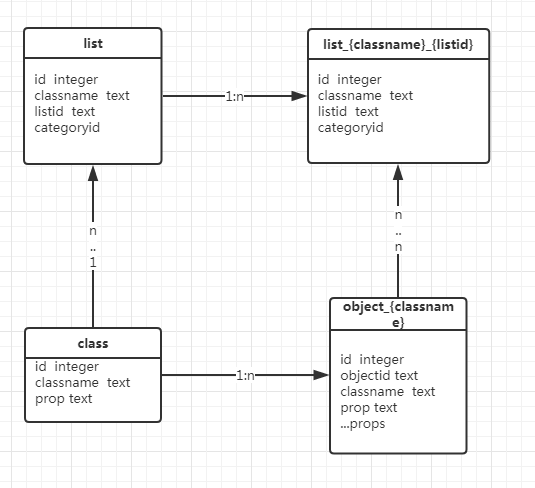
系统概要设计：



系统详细设计（业务层）



4.2 数据库设计



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。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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、对象表

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。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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

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

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

|  |  |  |
| --- | --- | --- |
| 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 |

1、类操作

**1、获取类列表**

**GET /class/read**

Paramters：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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

|  |  |
| --- | --- |
| 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

|  |  |
| --- | --- |
| 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

|  |  |
| --- | --- |
| 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

|  |  |
| --- | --- |
| 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 |

第五章 系统功能实现

5.1 环境配置准备

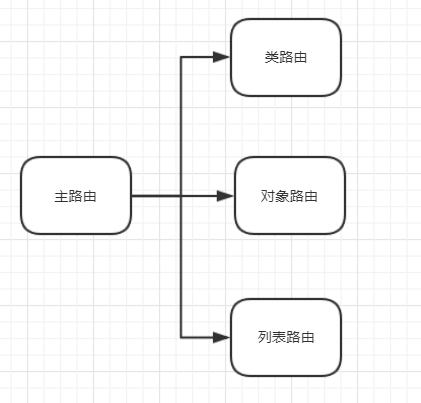
5.1.1 nginx配置

|  |
| --- |
| 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. } |

5.1.2package.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



模块实现：

|  |
| --- |
| 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

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

5.3.3创建方式

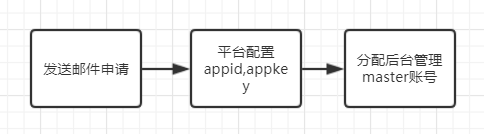
创建类的同时，创建对应的对象表即可。

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



第七章 总结

参考文献