# 接口规范定义

## 协议规范

为了确保不同系统/模块间的数据交互，需要事先约定好通讯协议，如：TCP、HTTP、HTTPS协议。为了确保数据交互安全，建议使用HTTPS协议。

## 接口路径规范

作为接口路径，为了方便清晰的区分来自不同的系统，可以采用不同系统/模块名作为接口路径前缀。

格式如下：

支付模块 /pay/xx

订单模块 /order/xx

## 版本控制规范

为了便于后期接口的升级和维护，建议在接口路径中加入版本号，便于管理，实现接口多版本的可维护性。如果你细心留意过的话，你会发现好多框架对外提供的API接口中(如：Eureka)，都带有版本号的。如：接口路径中添加类似"v1"、"v2"等版本号。

格式规范如下：

/xx/v1/xx

更新版本后可以使用v2、v3等、依次递加

## 接口命名规范

和Java命名规范一样，好的、统一的接口命名规范，不仅可以增强其可读性，而且还会减少很多不必要的口头/书面上的解释。

可结合【接口路径规范】、【版本控制规范】，外加**具体接口命名**(路径中可包含请求数据，如：id等)，建议具体接口命名也要规范些，可使用"**驼峰命名法**"按照实现接口的业务类型、业务场景等命名，有必要时可采取多级目录命名，但目录不宜过长，两级目录较为适宜。

格式规范如下：

/user/v1/sys/login 用户服务/模块的系统登录接口

/zoo/v1/zoos/{ID} 动物园服务/模块中，获取id为ID的动物

具体接口命名，通常有以下两种方式：

（1）接口名称动词前/后缀化

接口名称以接口数据操作的动词为前/后缀，常见动词有：add、delete、update、query、get、send、save、detail、list等，如：新建用户addUser、查询订单详情queryOrderDetail。

（2）接口名称动词+请求方式

接口路径中包含具体接口名称的名词，接口数据操作动作以HTTP请求方式来区分。常用的HTTP请求方式有：

**GET**：从服务器取出资源（一项或多项）。

**POST**：在服务器新建一个资源。

**PUT**：在服务器更新资源（客户端提供改变后的完整资源）。

**PATCH**：在服务器更新资源（客户端提供改变的属性）。

**DELETE**：从服务器删除资源。

如：

GET /zoo/v1/zoos：列出所有动物园

POST /zoo/v1/zoos：新建一个动物园

GET /zoo/v1/zoos/{ID}：获取某个指定动物园的信息

PUT /zoo/v1/zoos/{ID}：更新某个指定动物园的信息（提供该动物园的全部信息）

PATCH /zoo/v1/zoos/{ID}：更新某个指定动物园的信息（提供该动物园的部分信息）

DELETE /zoo/v1/zoos/{ID}：删除某个动物园

GET /zoo/v1/zoos/{ID}/animals：列出某个指定动物园的所有动物

DELETE /zoo/v1/zoos/ID/animals/ID：删除某个指定动物园的指定动物

## 请求参数规范

（1）请求方式

（2）按照GET、POST、PUT等含义定义，避免出现不一致现象，对人造成误解、歧义。

（3）请求头

（4）请求头根据项目需求添加配置参数。如：请求数据格式，accept=‘application/json’等。如有需要，请求头可根据项目需求要求传入用户token、唯一验签码等加密数据。

（5）请求参数/请求体

请求参数字段，尽可能与数据库表字段、对象属性名等保持一致，因为保持一致最省事，最舒服的一件事。

## 返回数据规范

统一规范返回数据的格式，对己对彼都有好处，此处以json格式为例。返回数据应包含：**返回状态码**、**返回状态信息**、**具体数据。**

格式规范如下：

{  
 "status":"000000",  
 "msg":"success",  
 "data": {  
 //json格式的具体数据  
 }

}

# 接口管理工具

接口开发完后，最终的目的是提供给其他系统/模块来使用的，因此，接口的管理是必不可少的。

## 接口管理的痛点

（1）系统/模块太多、接口太多，没有系统统一管理所有接口。

（2）代码修改后，接口文档没有及时更新，造成接口文档和实际接口不一致的现象。

（3）接口管理系统自主研发成本高。

（4）接口管理缺少接口mock功能。

## 接口管理工具推荐

### word

开发人员将系统的接口维护在word文档里，不管是组内沟通还是和其他团队的接口沟通都离不开这些接口文档，每次修改文档和代码都要同步修改。相信使用word的缺点大家应该也很清楚，就是维护和管理很麻烦，我们经常会遇到**文档和代码不一致**的情况，大部分不一致都是因为接口因为种种原因修改了，开发人员大部分都是只改了代码里的接口实现，而没有去修改接口文档。而且word文档搜索接口也很麻烦，没办法建全局索引，只能一个个文档点开查看，想想就很痛苦。但不可否认的是，word对于一些小团队用起来还是挺方便的，不用搭建系统，给谁一看就明白。

### wiki

wiki特别适合于小团队高速线性迭代开发，在wiki上看到的就是最新的接口，团队内所有成员看到的都是一样的，如果接口有变化，相关开发人员修改后立即生效，保证了顺畅的接口沟通。但是wiki的缺点也很多，接口文档只是静态页面，**无法实现一些动态效果**，无法实现追溯等等缺点。

### swagger

上面说的那些接口管理工具，其实都有一个很大的问题就是修改代码后需要同步维护接口文档，但是让程序员去修改文档是很难的，大部分程序员都比较讨厌维护各类文档。当我第一次了解到swagger的时候，发现这简直就是为程序员定制的接口管理工具，swagger定义了很多注解，在对接口加上swagger相关的注解，当接口代码修改后，swagger在工程启动后会根据代码自动生成最新的接口html文档，同时swagger提供了mock接口模拟的功能，也能够更加方便的模拟接口，并且还能够在swagger界面上直接发起接口调用，可以方便调用方在还没写代码的时候就能够尝试下接口调用后的结果。

看了那么多swagger的优点，下面也说说swagger的缺点，那就是swagger是跟随着每个工程一起启动的，这就导致每个工程都有一个swagger的访问地址，如果公司系统很多的话，那就会导致查看不同系统的接口都要到不同的地址去查看，每个开发都要自己收藏好各个系统的swagger地址。有些公司也自己开发了统一网关，将所有swagger的接口地址聚合起来，但是多少还是涉及到一些开发工作的，而且做的还不一定很完善。

### Easy Mock



官网的这张图基本上介绍清楚了easymock的核心功能，这其中我最看重的功能有两块，一个是能够集成swagger接口并集中管理所有接口，另一个就是响应式数据。

EasyMock能够根据swagger接口的地址自动导入所有swagger接口，非常方便，对于非swagger的接口也可以手工维护进去，这样可以很方便的做到全公司接口统一维护，而且也有比较完善的接口权限管理，方便分组管理。但缺点就是过于庞大，可能适合小一点项目或团队。