一 、访问地址设计

在访问地址中包括的内容有：协议头、域、设备标识、实体类、版本号、动作。

列举如下：

协议头 https://

域 do.myweb.com/

设备标识 [mobile/]

实体类 user/

版本号 [v{n}]/

动作 list\_all

对于地址的设计有三点需要注意：

1、版本号在第一版时无需添加，从第二版开始添加。

2、设备标识是可选的，对于不同终端功能差异较大，需要用到多套接口的情况，可添加此字段以便区分，并且按照习惯可进行如下区分：移动端加mobile、桌面端加PC、web端不添加。

举例：

web端：https://do.myweb.com/user/get\_by\_id

桌面端：https://do.myweb.com/PC/user/v2/get\_by\_id

移动端：https://do.myweb.com/mobile/user/get\_by\_id

3、对实体类发出的动作一般都包含‘谓词’，用于描述对实体类的操作或操作过程。

下面就以user作为实体对常见的动作进行举例：

1）获取id=X的用户

/user/get\_by\_id

2）获取id= X的部门下的用户

/user/list\_by\_depart\_id

3）将角色id 属于XX列表的角色，赋给id=X的用户

/user/assign\_role\_list\_to\_user (用role\_list取代roles表示多个)

4）删除id=X的用户

/user/del\_by\_id

5）修改id=X的用户的基本信息

/user/edt\_base\_info\_by\_id

6）修改id=X的用户的密码

/user/edt\_pwd\_by\_id

7）批量修改体重x=X，身高y=Y，年龄z=Z的用户的状态

/user/edt\_status\_by\_xyz (如果条件在3个以上，请用连续的一段英文字母表示，如a,b,c,d,e 5个条件参数，动作中可以简写为a\_to\_e)

8）将id=X的角色，赋给 id 属于 XX列表的用户

/user/assign\_role\_to\_user\_list

在service层，给用户分角色只用一个接口即可，但在controller层，分成了两个接口以避免权限混淆。

二 、输入格式设计

一般接口收到的参数由前端通过body或header传到后台，可以有form、json、xml、普通文本、二进制等多种方式。不管以哪种方式传参，尽量避免混用，且后台接收参数的方式要统一。

功能完整的接口，必须对前端传来的数据做验证，即便前端做了验证，后台也要做验证。针对Java的MVC模式，可以用表单类做验证，通过Hibernate 或 Spring注解对参数进行验证，为了对这些类进行区分，可以将form作为后缀结尾。根据对实体的操作，命名中可添加相应的谓词，如：‘add’、‘edt’、‘get’、‘list’、‘del’等，重复的情况下加V{n}做区别，格式如下 ：

谓词 + 实体 + Form

以user做实体类举例，提交的表单可以有：addUserForm，edtUserForm，getUserForm，listUserForm，listUserFormV2，listUserFormV3等。

如果参数比较少也可以使用Map<String,Object>来接收参数，通过代码对参数做验证。例如前端只传递一个id的情况就适合用这种方式接收。

表单类尽量避免复用，除非确定两个表单含义一样，错误示例：

IdForm，里边只有一个id属性，验证的提示为id不能为空，这种验证是含糊不清的，使用接口的人需要阅读更多的代码才知道id到底是哪个实体类的id。

好的输入格式让接口阅读起来更舒适。

三 、输出格式设计

首先确定接口返回的关联实体类，例如，查询用户信息的接口，关联的实体类是UserEntity，返回视图应命名为UserInfo，如果此命名被占用，先查看UserInfo视图是否满足该接口需要，如不满足，则创建新视图并命名为UserInfoV2，依此方法，用户视图可能为：UserInfo，UserInfoV2，UserInfoV3，UserInfoV4 …。如果视图需要完全包含UserEntity的信息，则让视图继承UserEntity，并补充UserEntity缺乏的信息。

在接口开发完成后，返回的视图应该有很多个版本，为了减少冗余，我们要从业务逻辑、安全性、性能三个方面考虑，着手合并一部分视图。

四 、接口设计原则

1）通用性原则

第一，同样的信息量，在手机端，web端，桌面端都满足需要。第二，接口低耦合，如果返回结果存在多个视图且从业务角度分析这些视图必须是密不可分的，才能在一个接口中处理，否则，请将这样的接口拆分掉。第三，返回结果的数据结构尽可能简单。

2）贴合业务逻辑原则

举例：修改用户基本信息和修改用户密码是两个独立的接口，尽管都是对用户信息的修改，但根据实际的业务情况，普通用户修改基本信息不需要二次认证，而修改密码需要二次认证。即便硬将这两项业务设计成一个接口（符合程序员CURD的习惯）命名为updateUserInfo，却还是要通过传值来区分用户到底update的是什么。更不可以将修改权交给用户让用户随意update，这样做可能产生严重的安全漏洞。用户修改实体类的能力其实受业务逻辑本身的限制，所以在设计接口时一定要注意贴合业务逻辑，分清楚哪些能力是通过权限控制的，哪些能力是通过接口控制的。

3）最小输入、输出原则

在满足通用性和贴合逻辑原则的前提下，尽可能减少接口往返的信息量，信息量越少就越安全。对于后端而言，信息能从session中捕获就不要让用户从前台传入，对前台多传的信息要适当进行屏蔽，以避免接口漏洞。返回给前台的字段如果非必须则不返回。