让你在 API 设计中少踩坑的实战分享

在项目开发中，实际的编码只占用了整个项目不到 30% 的时间，更多的时间都耗在了需求分析与接口设计上。每一次的需求变更就可能涉及到多个 API 的修改，那么 API设计书作为项目开发的核心文档，如何才能让前端和移动端开发人员更快的理解参数的意义和作用、适应版本更新，以及保障数据的安全性呢？

本场chat将根据之前项目中踩坑和总结对API设计进行分析。通常来说，软件的生命周期可以归纳为：

1. 可行性分析
2. 需求分析
3. 概要设计
4. 详细设计
5. 编码开发
6. 测试
7. 交付客户
8. 维护

在整个项目周期中，需求分析和设计（概要设计和详细设计）占据了60%甚至更多的时间，需求总是在不断变化的，客户有的时候都不知道自己想要什么，很多时候只是说了一个模糊的概念，更多的时候还需要我们根据客户一个模糊的概念进行拓展说明。基本上在客户确认好70%左右的需求的时候，这个时候设计就开始了。

概要设计

最先进行的是概要设计，根据需求可以整理出以下内容：

项目模块划分

最核心的需求确认好就可以开始划分项目的模块了，根据项目的特点进行模块划分。比如常用的用户模块，企业模块等，这个是根据具体项目来的，不过通常来说，用户体系是最基本的。

DB表设计

根据划分出的模块，可以进行对应的模型设计了。可能有哪些表，表中有哪些字段。这个时候字段都只是大概定义，不会进行实际的字段类型定义，而是先设置好逻辑名称。

版本控制

这个应该是项目确认要开始做的时候就已经开始弄了，比如在SVN上建立工程目录，存放客户的需求和整理的文档。新建git工程用于代码库的管理。

模块分配

如果是小项目的话，可能只需要1-2个人就可以完成了，那么就不需要特别划分模块给设计人员。如果是大项目的话，模块可能就会比较多，比如最近做的电商系统，就划分了企业模块，企业用户模块，会员模块，支付模块，内容模块，商品等几大模块。这个时候就需要分工处理了，每个人负责对应的一个或几个模块，专人专职，对模块根据用户需求进行分析设计以及后面的详细设计。

页面设计

设计人员根据客户的需求，进行简单的线图设计，可以让客户可以更直观的看到页面上的元素信息

详细设计

根据概要设计整理出的东西，就可以对系统进行详细设计了。这里的详细设计包括但不限于API接口的设计，比如还有系统之间的设计，系统交互设计，UI设计，对DB中字段进行类型设置，索引配置和物理名称设定，API设计书编写等。作为开发人员来说，最关注的莫过于API设计和DB设计了，首先看下API设计书。

API设计书

作为开发人员来说，只需要粗略的了解业务，根据设计书进行编码即可，不需要对整个系统非常了解，重要的是编码。

而对于API设计人员来说，如何设计出直观方便的API设计书还是需要根据项目的需要和特性好好构思的。

进行API设计时，对整个系统已经有考究了，比如共通参数，API版本设定，API参数设定等。

本文这里选用的是excel作为设计书工具，当然也可以使用其他工具，例如markdown。

如果不对外提供API的话，直接可以使用API设计书。如果需要对外提供API的话，那么需要将请求和返回需要单独抽取出来，对外提供只有参数和返回说明的文档，而不会将详细设计的内容对外提供的。那么先说说设计书的要点：

设计书目录规范

设计书根据模块划分目录，每个设计书根据每个模块所在的位置划分到不同的文件目录下，结构为：*projectName*/{moduleName}。比如用户相关的就存放到user目录下。则user目录为${projectName}/user

设计书命名规范

为了方便设计书的查找，同一个模块下的设计书编号递增，且以固定模块名开头，比如用户模块编号为user，假设用户登录为第一个API，那么用户登录的文件名为：user-001-用户登录(user_login).xls。这里的编号可以根据项目大小设置位数，三位数的话从000-999可以有1000个API。

共通设置

对于API来说，很多地方是共通的，比如共通的请求参数和返回。这时候需要对共通进行抽取，使用时只需要引用即可，否则如果共通的地方有修改的话，那么每个API设计书都需要修改，想想就可怕。比如：用户token校验的设计，只要涉及到用户权限相关的API都需要校验，这时候在每个API设计书中只需要引用下共通API设计书中token校验文档。而不是在每个API设计书中都写token校验的逻辑。

简要流程图

在每个API设计书中都要有对应的流程图，这样审查的时候可以非常直观的看出这个设计书涉及到了哪些业务，而不需要重头到尾再看一次设计。

更新履历

每个API设计书在开始开发后，所有的修改都必须要记录履历，标明修改的原因和修改的地方。这样开发人员只需要根据变动点进行对应的代码修改。

作者

每本设计书都需要标注作者，这样开发人员有问题时，可以直接询问作者而不是啥事都需要找项目经理或技术经理。

请求方式

如果整个系统的请求方式的都统一为POST的话，可以不需要这个说明，但是如果有不同的请求方式，那么API设计书中就需要标注出这个API接受哪些请求方式。并且在API一览中添加请求方式说明。

API一览

所有的API都应该统计到一个文档中，根据这个文档就可以看出系统中有多少接口，每个接口是做什么使用的。同时也可以在这个文档中对每个API添加访问需要的权限，这样就可以非常直观的看出哪些接口对应的权限了。

DB设计

在详细设计过程中，表中的字段需要根据概要设计中定义的逻辑名称来定义对应的物理名称。ER图设计工具也有很多，我们使用的是ermaster。表设计应该遵从第三范式规范

（参考https://www.cnblogs.com/0515offer/articles/4132171.html说明），这是最基础的要求，其他根据项目需要添加。

共通字段抽取

每张表中应该都有共通字段的，如创建日期，更新日期，创建者，更新者，删除标志等。这些共通的需要抽取出来，每张表中引用即可。

锁

对于部分模块来说，项目初期就能确定是需要进行并发处理的，这里就需要添加乐观锁进行处理，通常使用UUID。在表中添加version字段，每次更新的时候将查询到的version和db中的version进行对比，如果相同则更新成新生成的version，此时更新成功。如果version不一致，则更新失败，进行事务回滚或者进行重试。

sql生成

表设计完成后，需要导出对应数据库的DDL，导出时注意不要将外键约束导出。实际情况中很少会对表数据进行物理删除，基本上是逻辑删除的。为了搜索更加快速，可以在对应的外键上加上索引。

逻辑删除标志

每张表都应该有个逻辑删除标志位(delete_flg)，表示该数据已经在逻辑上删除。进行业务操作时，如果已删除的数据就不需要抽取出来了。

API请求设计

版本设置

对于APP和对外提供接口来说，api的版本还是比较需要的。版本信息可以定义到url中，格式可以为 `/${version}/module/operateName(/v1/user/login)`，也可将版本信息放到Header中。但是不推荐放到请求参数中。

因为如果放在参数中，使用rest方式请求时，参数是以json格式存放在输入流中，此时如果要获取对应的用户凭证，则需要将流中的信息读取出来，而且得要重置流下标，否则后面的处理将获取不到参数。

共通参数抽取

API中需要将所有共通参数抽取出来，比如用户凭证。建议将用户信息存放到Header中，而不是放在参数中。

原因同上。

请求方式

标明此API是需要哪种方式请求，POST还是GET，又或者是其他HTTP规范中的方式。

参数定义

参数定义指的是在API中请求和返回的字段，必须要标注出字段的类型和是否必须等。比如用户登录时用户名和密码是必须的，那么设计书中就必须要体现出来。而且返回的数据类型必须是json支持的数据类型。（Number，String，Boolean，Array，Object）

共通返回定义

所有的返回都应该按照一定的模板来，定义必定会返回的信息，比如请求的状态码，错误信息等。这里就可以根据实际设计来了，比如返回不同status表示不同的错误信息，那么errMsg就不需要了，这种设计比较适合用于需要处理国际化信息上。

缩进

所有的参数中，如果是对象类型数据，那么对象中的数据定义都应该基于此对象的定义进行一个tab的缩进，表示这些定义都是属于对象属性。比如下面设计书中的data字段。

返回示例：

> 为了方便开发人员理解，还需要根据返回的字段信息，添加对应的返回示例。

例如：

- status表示返回的状态码,200表示处理成功，400表示请求失败...
- errMsg返回失败的原因
- data返回对应的数据，可以是对象，可以是数组

```
{
"status": 200,
"errMsg": "",
"data": {}
}
```

根据以上的说明，整理出用户登录的设计书：

项目名	ChatDemo项目	资料名	详细设计书	制作时间/制作者	2017/11/27 developer1
模块	用户	模块	用户	更新时间/更新者	
名称	用户登录				

概要

编号	API名	说明
1	user_login	根据用户输入的用户名和密码，执行用户登录，登录成功后返回用户信息和token。

URL,请求方式

编号	URL	I/F方式	方式	文字编码	超时等待时间	备注
1	/user/login	request:json responce:json	POST	UTF-8	30秒	

请求参数

编号	项目名	必须	数组	类型	项目名称	备注
1	usernme	○	—	string	用户名	
2	password	○	—	string	登录密码	

返回结果

编号	字段名	必须	数组	类型	字段说明	数据来源	备注
1	status	○	—	number	处理结果		见共通设定
2	data	○	—	对象			
3	id	○	—	number	用户ID	处理2.用户表.ID	
5	username	○	—	string	用户名	处理2.用户表.用户名	
5	name	○	—	string	姓名	处理2.用户表.姓名	
7	token	○	—	string	访问用令牌	处理3生成的访问令牌	
10	errorMsg	—	—	string	错误信息		

返回Json值

编号	返回json例子	备注
1	{ "status": 200, "data": { "id": 1000, "username": "admin", "name": "管理员张三", "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG91IiwiYWRtaW4iOnRydWV9.TJVA950rM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ" }, "errMsg": "" }	

1.验证参数的正确性

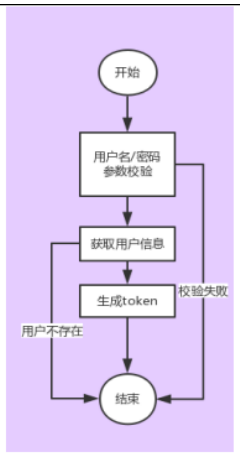
1.1 用户名为空 处理结束，状态码：400 错误信息：用户名为空

1.2 密码为空 处理结束，状态码：400 错误信息：密码为空

2.验证用户登录的正确性

从用户表检索。

数据库查询	
获取字段：	
编号	字段
1	id
2	用户名
3	姓名
表：	
编号	表名
1	用户表
条件：	
编号	条件
1	用户表.用户名 = 参数.用户名
2	用户表.密码 = 参数.密码（需MD5加密）



根据查询结果判断

a.count = 0的场合，设定错误消息，处理结束。错误消息：用户名或密码错误

b.count > 0的场合，处理继续。

2.生成token，根据步骤2获取的用户id和用户昵称生成JWT-token

3.设定返回值，处理结束。

用户凭证

涉及到权限相关的请求都需要对用户身份进行认证，只有已授权的用户才能访问指定的API。这里只讨论前后端分离的情况，非前后端分离的项目直接在session中就可以存储用户权限了，这里不加讨论。

前后端分离时

在请求头中添加标识用户身份信息的字段token。在用户登录时在后端生成一个全局唯一的token返回。在每次API请求时将token放入Header中，如果此API需要特定的权限， 那么需要根据token从数据库或者缓存中获取用户的权限。

普通网站

直接在session存储用户相关的权限信息。

用户凭证生成

用户凭证token，代表着一个用户。所以这个token需要是全局唯一的，且不可伪造的。token生成方式可以分为两种

UUID

UUID 是通用唯一识别码（Universally Unique Identifier）的缩写，能够确保在系统中是唯一存在的。用户每次访问时通过token到DB（缓存）中查询此用户的权限信息，根据用户的权限进行访问控制。
缺点也很明显了，判断是否有权限访问和token是否合法都需要从DB（缓存）中获取。

JWT

JWT由三部分组成，第一部分我们称它为头部（header),第二部分我们称其为载荷（payload,类似于飞机上承载的物品)，第三部分是签证（signature).这里可以参考此博客：

http://www.jianshu.com/p/576dbf44b2ae

JWT适合存储用户非敏感信息，因为第二部分payload存放的数据是原始数据通过base64编码得到的数据，这里可以直接获取到信息。比如存放用户的名称，昵称，id等信息。

使用JWT的优点是可以在token中设置超时时间，可以防止token被伪造和修改，直接通过验证JWT信息的完整性即可以判断用户的权限，比使用uuid的方式可以存储更多的信息，而且可以减少用户查询用户信息的次数。

权限校验

通过以上步骤可以确认出使用哪种方式分配用户token。在每次请求需要权限的API时，系统需要根据用户的权限进行访问控制。常用的框架有SpringSecurity与Shiro。这里推荐

使用Shiro，因为Shiro相对于SpringSecurity来说更加轻便。

上面说的将用户的token存放在请求Header中，是因为在Filter层中就可以直接获取出token信息，更加方便。使用Shiro可以自定义Filter进行权限的拦截。这里添加token拦截器：

```
        @Bean("shiroFilterFactoryBean")
public ShiroFilterFactoryBean
shiroFilterFactoryBean(DefaultWebSecurityManager securityManager,
ShiroConfigProperties properties,
        JwtManager jwt) {
    ShiroFilterFactoryBean shiroFilter = new ShiroFilterFactoryBean();
    shiroFilter.setSecurityManager(securityManager);
    shiroFilter.setLoginUrl(properties.getLoginUrl());
    shiroFilter.setSuccessUrl(properties.getLoginSuccess());
    shiroFilter.setUnauthorizedUrl(properties.getUnauthorized());
    shiroFilter.setFilterChainDefinitionMap(properties.getFilters());

    TokenFilter tokenFilter = new TokenFilter(jwt);
    Map<String, javax.servlet.Filter> filters = new HashMap<>();
    tokenFilter.setEnabled(true);
    tokenFilter.setUnauthorizedUrl(properties.getUnauthorized());
    filters.put("token", tokenFilter);
    shiroFilter.setFilters(filters);

    return shiroFilter;
}
```


为什么推荐在Filter层中做权限校验？

> 权限校验宜早不宜迟，在最外层就做好了拦截了，就不需要在经过其他Filter一系列的处理了，可以提高效率。

权限拦截策略

这里推荐使用白名单规则，默认都是需要权限校验的，不需要权限校验的API都需要手动配置。这样可以防止某些重要的API忘记加权限而导致系统不安全。

踩坑总结

目前我们的API设计书都是按照以上要求进行的，之前的项目因为没有在设计书中添加简要流程图，导致一段时间过后再看设计书时有点晕，要了解一本设计书涉及到的业务还需要从头到尾再看一遍，非常的浪费时间。所以简要的流程图还是非常必要的。由于一开始项目中没有把共通字段抽取出来，导致后面变更字段时需要将所有的设计书都更新一遍这个就非常浪费时间了。

以上规范只是项目中总结出来觉得非常有必要的信息，这个读者可以根据自己项目要求进行取舍。如果觉得上文中有什么不对的地方，这里也欢迎各位读者留言指正。

至于如何使用JWT，已经将如何使用token的代码提交至：https://github.com/cmlbeliever/ProjectForChat 下的ChatForApi工程。

项目中共有登录和获取用户信息两个接口，使用postman调用登录接口后获取到用户token，根据token通过user/getUserInfo接口获取到用户信息。这里的token是作为Header传入的。