App架构设计经验谈:接口的设计

2016-03-10 安卓应用频道

(点击上方公众号，可快速关注)

> 来源：Keegan小钢
> 链接：http://keeganlee.me/post/architecture/20160107

App与服务器的通信接口如何设计得好，需要考虑的地方挺多的，在此根据我的一些经验做一些总结分享，旨在抛砖引玉。

安全机制的设计

现在，大部分App的接口都采用RESTful架构，RESTFul最重要的一个设计原则就是，客户端与服务器的交互在请求之间是无状态的，也就是说，当涉及到用户状态时，每次请求都要带上身份验证信息。实现上，大部分都采用token的认证方式，一般流程是：

1. 用户用密码登录成功后，服务器返回token给客户端；
2. 客户端将token保存在本地，发起后续的相关请求时，将token发回给服务器；
3. 服务器检查token的有效性，有效则返回数据，若无效，分两种情况：
    - token错误，这时需要用户重新登录，获取正确的token
    - token过期，这时客户端需要再发起一次认证请求，获取新的token

然而，此种验证方式存在一个安全性问题：当登录接口被劫持时，黑客就获取到了用户密码和token，后续则可以对该用户做任何事情了。用户只有修改密码才能夺回控制权。

如何优化呢？第一种解决方案是采用HTTPS。HTTPS在HTTP的基础上添加了SSL安全协议，自动对数据进行了压缩加密，在一定程序可以防止监听、防止劫持、防止重发，安全性可以提高很多。不过，SSL也不是绝对安全的，也存在被劫持的可能。另外，服务器对HTTPS的配置相对有点复杂，还需要到CA申请证书，而且一般还是收费的。而且，HTTPS效率也比较低。一般，只有安全要求比较高的系统才会采用HTTPS，比如银行。而大部分对安全要求没那么高的App还是采用HTTP的方式。

我们目前的做法是给每个接口都添加签名。给客户端分配一个密钥，每次请求接口时，将密钥和所有参数组合成源串，根据签名算法生成签名值，发送请求时将签名一起发送给服务器验证。类似的实现可参考OAuth1.0的签名算法。这样，黑客不知道密钥，不知道签名算法，就算拦截到登录接口，后续请求也无法成功操作。不过，因为签名算法比较麻烦，而且

容易出错，只适合对内的接口。如果你们的接口属于开放的API，则不太适合这种签名认证的方式了，建议还是使用OAuth2.0的认证机制。

我们也给每个端分配一个appKey，比如Android、iOS、微信三端，每个端分别分配一个appKey和一个密钥。没有传appKey的请求将报错，传错了appKey的请求也将报错。这样，安全性方面又加多了一层防御，同时也方便对不同端做一些不同的处理策略。

另外，现在越来越多App取消了密码登录，而采用手机号+短信验证码的登录方式，我在当前的项目中也采用了这种登录方式。这种登录方式有几种好处：

1. 不需要注册，不需要修改密码，也不需要因为忘记密码而重置密码的操作了；
2. 用户不再需要记住密码了，也不怕密码泄露的问题了；
3. 相对于密码登录其安全性明显提高了。

接口数据的设计

接口的数据一般都采用JSON格式进行传输，不过，需要注意的是，JSON的值只有六种数据类型：

- Number：整数或浮点数
- String：字符串
- Boolean：true 或 false
- Array：数组包含在方括号[]中
- Object：对象包含在大括号{}中
- Null：空类型

所以，传输的数据类型不能超过这六种数据类型。以前，我们曾经试过传输Date类型，它会转为类似于”2016年1月7日 09时17分42秒 GMT+08:00″这样的字符串，这在转换时会产生问题，不同的解析库解析方式可能不同，有的可能会转乱，有的可能直接异常了。要避免出错，必须做特殊处理，自己手动去做解析。为了根除这种问题，最好的解决方案是用毫秒数表示日期。

另外，以前的项目中还出现过字符串的”true”和”false”，或者字符串的数字，甚至还出现过字符串的”null”，导致解析错误，尤其是”null”，导致App奔溃，后来查了好久才查出来是该问题导致的。这都是因为服务端对数据没处理好，导致有些数据转为了字符串。所以，在客户端，也不能完全信任服务端传回的数据都是对的，需要对所有异常情况都做相应处理。

服务器返回的数据结构，一般为：

```
{
  code：0
  message: "success"
  data: { key1: value1, key2: value2, ... }
}
```

- code: 状态码，0表示成功，非0表示各种不同的错误
- message: 描述信息，成功时为”success”，错误时则是错误信息
- data: 成功时返回的数据，类型为对象或数据

不同错误需要定义不同的状态码，属于客户端的错误和服务端的错误也要区分，比如1XX表示客户端的错误，2XX表示服务端的错误。这里举几个例子：

- 0：成功
- 100：请求错误
- 101：缺少appKey
- 102：缺少签名
- 103：缺少参数
- 200：服务器出错
- 201：服务不可用
- 202：服务器正在重启

错误信息一般有两种用途：一是客户端开发人员调试时看具体是什么错误；二是作为App错误提示直接展示给用户看。主要还是作为App错误提示，直接展示给用户看的。所以，大部分都是简短的提示信息。

data字段只在请求成功时才会有数据返回的。数据类型限定为对象或数组，当请求需要的数据为单个对象时则传回对象，当请求需要的数据是列表时，则为某个对象的数组。这里需要注意的就是，不要将data传入字符串或数字，即使请求需要的数据只有一个，比如token，那返回的data应该为：

```
// 正确
data: { token: 123456 }

// 错误
data: 123456
```

接口版本的设计

接口不可能一成不变，在不停迭代中，总会发生变化。接口的变化一般会有几种：

- 数据的变化，比如增加了旧版本不支持的数据类型
- 参数的变化，比如新增了参数
- 接口的废弃，不再使用该接口了

为了适应这些变化，必须得做接口版本的设计。实现上，一般有两种做法：

1. 每个接口有各自的版本，一般为接口添加个version的参数。
2. 整个接口系统有统一的版本，一般在URL中添加版本号，比如 http://api.domain.com/v2。

大部分情况下会采用第一种方式，当某一个接口有变动时，在这个接口上叠加版本号，并兼容旧版本。App的新版本开发传参时则将传入新版本的version。

如果整个接口系统的根基都发生变动的话，比如微博API，从OAuth1.0升级到OAuth2.0，整个API都进行了升级。

有时候，一个接口的变动还会影响到其他接口，但做的时候不一定能发现。因此，最好还要有一套完善的测试机制保证每次接口变更都能测试到所有相关层面。

写在最后

关于接口设计，暂时想到的就这么多了。各位看官看完觉得有遗漏或有哪些需要优化的欢迎提出一起讨论。