接口安全性

#积累/常用

接口身份验证
——session机制
——token机制
——JWT
数据校验
唯一性校验
频率限制

1.接口身份验证

对于有用户概念的产品，数据一般是根据用户隔离的，数据的请求要对身份进行验证。为了防止用户信息被随意获取，身份验证要做安全性的考虑。

1. session机制
   登陆请求第一次，服务器验证通过之后，开启session，并产生一个sessionid与之对应，对应关系会保存下来。对应关系的保存方式有很多，用表保存、redis保存。
   在响应的报文里，返回一个sessionid的cookie。为了更安全可以添加http-only的属性，cookie就不能通过document.cookie等js方式读取到并改写了。
   cookie未过期的情况下，同域下的其他接口请求，都会带上这个cookie，服务器收到sessionid之后，就会查询对应表，找出session。
   如果存在则不用再查询数据库做身份验证，可以直接读取session中的信息。

- sessionid一般比较复杂，可以防止他人伪造
  express-session中生成sid的规则：
  Node 的crypto.randomBytes方法随机生成一个24位的二进制，然后编码为base64的字符，得到一个32个字符的随机串。
- session可以共享变量，即使是不同的接口。例如，后台对用户做频率限制，可以在session中保存用户接口请求的次数，超过一定次数的时候，返回超出频率限制的错误；可以把常用信息，例如用户名、手机号保存在session中，避免对数据库做频繁访问。
- session一般都是和cookie一起使用的，通过cookie来传输sessionid，也因此有所限制。cookie机制只是浏览器具有的功能，app是没有的，所以用session机制的接口不能提供给app用。
- 还经常会有cookie跨域相关的问题：
  1.在小手机宝中，登陆态cookie都是app写入的；小手机宝的H5根域有两个lianxiangcloud.com和onethingpcs.com；之前，app退出登陆的时候清除cookie，只清除了

lianxiangcloud.com，没有清除onethingpcs.com，导致未登陆情况下，onethingpcs.com下的页面，后台依然判断为已登录状态。

2.在玩客云H5中，也存在要维护两个根域cookie的问题，早期花了很多时间和精力去处理相关问题。

- 后台存在多台服务器时，需要做session分布管理的额外处理。session创建在服务器A，但请求被分发到服务器B，服务器B上是没有session的，必须重新登录。

2. token机制

另外一种身份验证的机制是token机制。

- token机制的流程，投票H5的身份验证方案

用户第一次登陆，获取到统一登录平台的ticket之后请求后台，后台查询数据库验证用户，生成token，并在用户表里记录对应的token，同时后台把token作为body的属性返回。

约定，在http的头部属性中添加auth-token属性，传递token信息。之后的请求中，后台获取到token信息，就不会再做身份认证了。

- 这种token机制，和session机制的区别：

1.后台不会创建session，容易做分布式部署；但是也不能在session共享变量，得用其他方式来保存和传递共同变量。

2.身份表示不用cookie，通过自定义http的头部属性来传递，不会受cookie的局限。

- token泄漏了，如何防治他人冒充?
- 服务器端生成token的时候，加入userAgent或者ip等客户端信息；服务端解析token的时候，如果UA不同会导致解析失败。一定程度上可以防止冒充。

3. JSON Web Token(JWT)

- JWT的组成

Header（头部）：一个描述 JWT 的元数据的对象，包括签名算法、token类型，最后使用 Base64URL 转换为字符串。

Payload（负载）：用来存放实际需要传递的数据，也包括过期时间、签发时间、签发人等信息，使用Base64URL 转换为字符串。

Signature（签名）：对前两部分进行签名，防止数据篡改，例如：

```
HMACSHA256(
  base64UrlEncode(header) + "." +
```

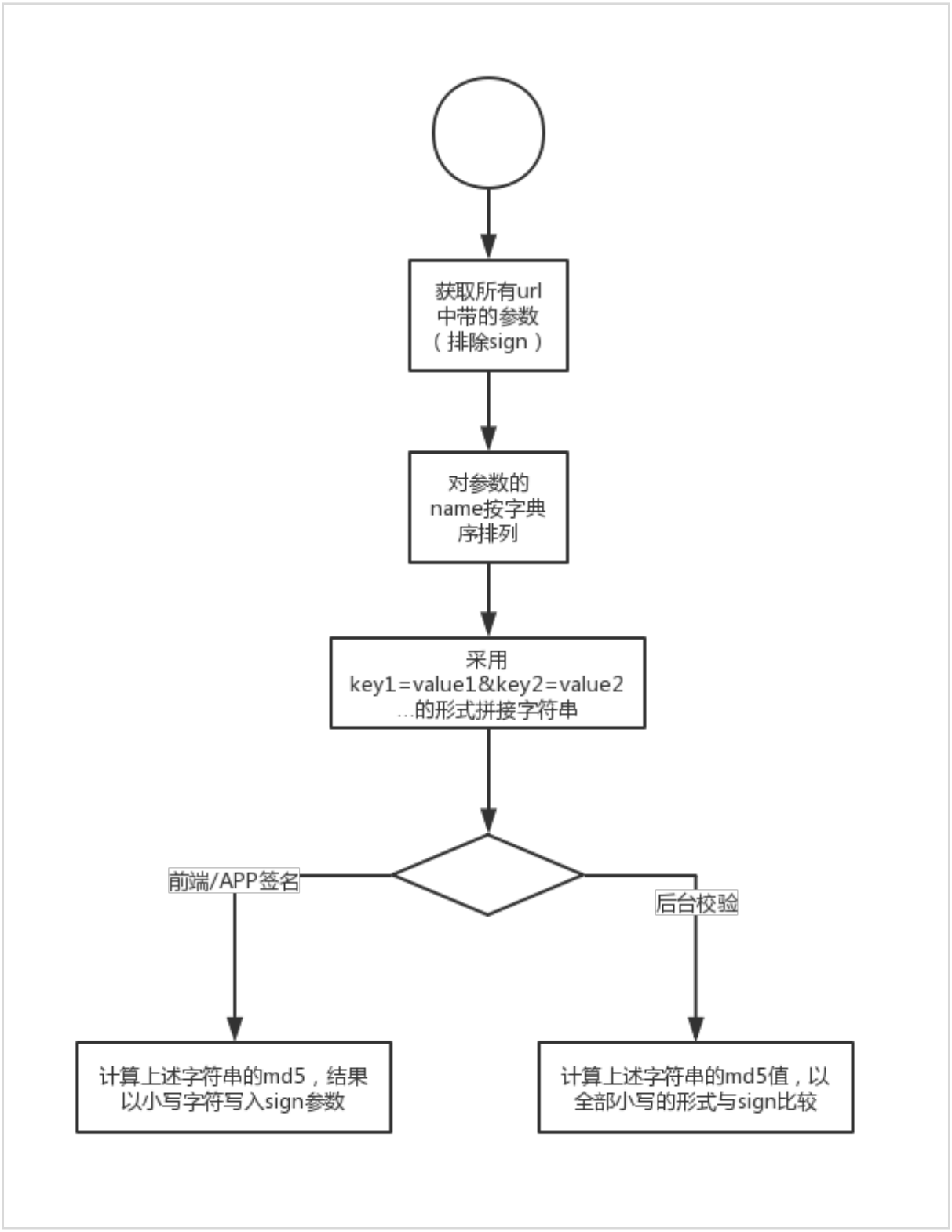
```
    base64UrlEncode(payload),
    secret)
```

- 身份验证流程是：
  用户发送用户名和密码给后台，后台验证通过之后，按**JSON Web Token**的规则生成一个token，这个token本身就包含用户信息，像uid之类的信息。
  下次用户发送请求的时候，在头属性中带上这个属性，按jwt规范，后台会检查签名是否正确，检查Token是否过期，然后解码自定义信息uid，通过uid识别指定的用户。
- 优势跟token一样，有利于分布式部署，能提供给前端或app使用。
  特别的是规范化保存自定义变量，可以把更多用户信息也放入token中，可以相应减少查询数据库频率、减少后台服务器内存压力。

2.数据校验

要验证请求数据的合法性，防止他人篡改请求参数。通过签名机制来保证安全。

- sign机制
  客户端对请求参数按照一定规则计算sign；服务端收到请求，也做同样规则计算sign，对比值是否一致，一致则参数没有被篡改，否则返回请求参数不合法的错误。
  计算sign的逻辑是实现接口校验的关键，可根据业务自行制定计算sign的规则，可以自由规定计算sign使用的参数(ua、ip等)、变量之间的排序方式、要不要加密钥、密钥是多少。
- 例如，小手机宝的签名机制



- 因为这是提供给H5的接口，前端不能隐藏密钥，所以规则就没有把密钥加进去。规则也相对简单，多一层防护而已。

3.唯一性校验

- 重放攻击问题

  黑客拦截了一个请求，获取到了token，能够通过身份验证；它也获取到了正常请求的请求参数，没有对参数做修改，也能够通过数据验证。

黑客通过脚本大量重复同一个请求，如果这个请求要对数据库进行耗时操作，很快就会导致数据库服务器瘫痪。为了安全，要保证合法请求只被执行一次。

- 上次抽奖的线上事故，原因就是接口被刷，导致数据库查询太长，数据库服务器挂掉了，也因此影响到了该服务器上的玩客云官网
- 阿里云的重放攻击防御机制

  请求API接口的时候，在header中加入属性X-Ca-Timestamp，为当前客户端请求的时间戳；

  在header中加入属性X-Ca-Nonce，客户端生成的UUID，通用惟一标识符可以标示这次请求的唯一性。

  服务端会校验X-Ca-Timestamp，获取当前服务端时间，如果间隔大于15分钟，则请求无效；

  验证X-Ca-Nonce，同样的值，15分内只能被使用一次。

  把X-Ca-Timestamp和X-Ca-Nonce加入签名的计算中，防止两个属性被篡改。

4.频率限制

像短信验证码这种频繁使用的接口，为了服务器稳定运行，也应该设置频率限制。

- nginx层可以ip进行频率限制
- 后台逻辑层可以对具体用户进行频率限制

  session、redis或表来保存接口请求次数数据。

5.后台接口拦截规则

0. nginx层把请求分发到后台，后台开始做安全性校验；
1. 接口是否包含sessionid、timestamp、sign等必要性参数；没有则抛出请求参数不合法；
2. 根据时间戳和唯一标识符，以及服务器当前时间，校验请求唯一性；校验失败，则抛出请求不合法；
3. 签名校验；校验失败，则抛出签名错误；
4. 为了维持系统稳定，可能会抛出超出频率限制错误；
5. 最终进入业务逻辑