**服务端怎么保证客户端的合法性（UID与Token被劫持）**

HTTP 是一种没有状态的协议，也就是它并不知道是谁是访问应用。这里我们把用户看成是客户端，客户端使用用户名还有密码通过了身份验证，不过下回这个客户端再发送请求时候，还得再验证一下。

解决的方法就是，当用户请求登录的时候，如果没有问题，我们在服务端生成一条记录，这个记录里可以说明一下登录的用户是谁，然后把这条记录的 ID 号发送给客户端，客户端收到以后把这个 ID 号存储在 Cookie 里，下次这个用户再向服务端发送请求的时候，可以带着这个 Cookie ，这样服务端会验证一个这个 Cookie 里的信息，看看能不能在服务端这里找到对应的记录，如果可以，说明用户已经通过了身份验证，就把用户请求的数据返回给客户端。

上面说的就是 Session，我们需要在服务端存储为登录的用户生成的 Session ，这些 Session 可能会存储在内存，磁盘，或者数据库里。我们可能需要在服务端定期的去清理过期的 Session 。

Token用户登录

客户端使用用户名跟密码请求登录

服务端收到请求，去验证用户名与密码

验证成功后，服务端会签发一个 Token，再把这个 Token 发送给客户端

客户端收到 Token 以后可以把它存储起来，比如放在 Cookie 里或者 Local Storage 里

客户端每次向服务端请求资源的时候需要带着服务端签发的 Token

服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，就向客户端返回请求的数据

**账号密码的缓存如何保证安全**

对称加密：MD5，本地DES

非对称加密：

客户端用公钥加密用户密码，保存到本地；

用户要登陆时，发送加密串到服务器；

服务器用私钥解密，得到用户的密码，再验证。

如果是常见的数据持久化，比如文件存储，SharePreference、数据库是安全的，默认存储在内部存储中，因为他们只能被该程序访问。

<http://blog.csdn.net/itluochen/article/details/52183920>

**代码混淆后如何保证可读性，代码混淆后不一定可用。**

有问题吧这题？代码混淆就是为了降低混淆的可读性。

代码混淆后不一定可以安全运行，特别是第三方库。

**mvc，mvp，mvvc的差别与优点**

mvc与mvp的差别：mvc中view与model会进行交互，而mvp中m与v不会进行交互

mvc的缺点：activity中代码会十分臃肿，需要负责与用户的交互，界面的展示，还有大量的逻辑代码。

mvp的优点：将大量逻辑操作放在了presenter中，activity只用做简单的初始化与逻辑操作。

v包含了view与activity

model负责提供数据

p负责逻辑操作，view与model的纽带

一般来说view为接口，activity实现接口，相应的逻辑中实现方法由presenter来完成。

mvc与mvp的差别：<https://www.tianmaying.com/tutorial/AndroidMVC>

mvc、mvp、mvvm的区别：<http://www.ruanyifeng.com/blog/2015/02/mvcmvp_mvvm.html>

**activity生命周期，从A到B生命周期的变化**

打开A:onCreate()-onStart()-onResume()

A跳转B：A:onPause()-B:onCreate()-B:onStart()-B:onResume-A:onStop()

B跳转回A：B:onPause()-A:onRestart()-A:onStart()-A:onResume()-B:onStop()- B:onDestory()

**从A跳到B，系统内存不够要杀死A Activity，至少等到它执行完哪个生命周期方法**

当一个Activity在Resumed状态下，它是不会因内存不够而被系统直接杀死（在极端的情况下也有可能被杀死，但是一般不会考虑这种情况）。只有进入Paused或者Stopped状态才会，而且可能根本就不会去调用onStop()和onDestory()方法，所以onPause()方法是我们最大程度上保证Activity在销毁之前能够执行到的方法。因此，如果你的某个Activity需要保存某些数据到数据库，您应该在onPause()里编写持久化数据的代码。但要注意，你应该选择哪些信息必须保留在onPause()

**谈谈IPC通信**

进程间通信的方式：

Bundle加Intent（隐式）：基本数据类型、字符串、序列化，当我们在一个进程中启动了另外一个进程的Activity、Service、Receiver，我们就可以在Bundle中附加我们需要传输给远程进程的信息（前提是能够被序列化）并通过Intent发送出去

文件共享（SharePreference）：不适合高并发。

Messenger：底层为Binder，数据通过Message传输，只能传输Bundle支持的类型

AIDL：底层为Binder，支持实时通信，功能强大。

ContentProvider：底层为Binder，使用比较受限，通讯录访问用的比较多。

Socket：网络通信

参考资料：<http://blog.csdn.net/jdsjlzx/article/details/51973813>

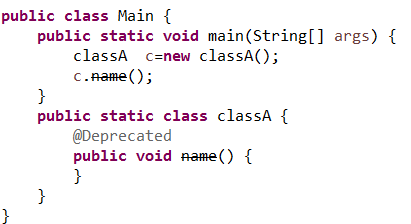
**java注解**

作用：注解也被称为元数据，为我们在代码中添加信息提供了一种形式化的方法，使我们在稍后的某个时刻非常方便地使用这些数据。

java内置的几种注解

@override，表示当前方法覆盖超类中的方法

@Deprrecated，表示当前方法即将废弃，不推荐使用



@SuppressWarnings，表示忽略编译器的警告信息。

java的元注解（meta-annotation）

@Target:修饰对象的范围，@Target(ElementType.TYPE)

@Retention：注解的生命周期，@Retention(RetentionPolicy.RUNTIME)

@Documented：被javadoc此类的工具文档化

@Inherited：某个被标注的类型是被继承的，annotation将被用于该class的子类

<http://www.cnblogs.com/peida/archive/2013/04/24/3036689.html>

**设计一个线程安全的hashmap**

Map m = Collections.synchronizedMap(new HashMap(...));

HashMap采用链地址法解决哈希冲突，多线程访问哈希表的位置并修改映射关系的时候，后执行的线程会覆盖先执行线程的修改，所以不是线程安全的

Hashtable采用synchronized关键字解决了并发访问的安全性问题但是效率较低

ConcurrentHashMap使用了线程锁分段技术，每次访问只允许一个线程修改哈希表的映射关系，所以是线程安全的

<http://blog.csdn.net/sbq63683210/article/details/51679790>

**手写单例模式**

01 public class StaticSingleton {

02     private StaticSingleton(){

03         System.out.println("StaticSingleton is create”);

04     }

05     private static class SingletonHolder {

06         private static StaticSingleton instance = new StaticSingleton();

07     }

08     public static StaticSingleton getInstance() {

09         return SingletonHolder.instance;

10     }

11 }