## HTTPS：

Htpps相对http是安全的，s指的是SSL/TLS协议

http是明文传输，第三方拦截可见

https认证过程：

1. 客户端发起请求，服务器端返回服务器公钥，服务器信息等
2. 客户端通过CA校验服务器返回的公钥(公钥直接内置在操作系统或浏览器内的)是否是指定服务器的(客户端通过公钥向CA获取服务器的信息，从而比较公钥是否是指定服务器的)
3. 客户端发送自己支持的加密方法给服务器端
4. 服务器端选择加密度高的加密方式，然后以明文的方式将加密方式返回给客户端
5. 客户端收到加密方式后，生成随机码作为密钥，使用服务器的公钥进行加密，发送给服务端
6. 服务端使用私钥解密获取密钥，这样服务器和客户端通信的就可以使用对称加密

校验公钥是否被修改：校验服务器的信息是否与传输的对应，hash校码

## URL地址解析过程：

域名解析IP地址过程：

本地hosts文件查找

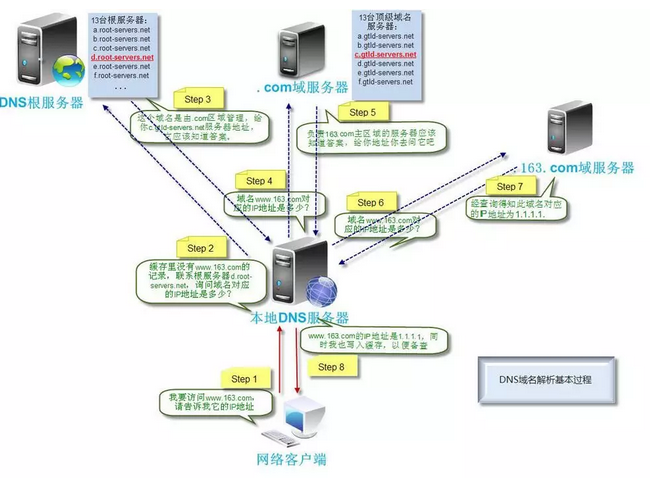
本地DNS(网络接入服务提供商，如电信)查找，本地DNS在缓存中查询，没有则向DNS根服务器查询

根DNS服务器没有记录具体的域名和IP地址的对应关系，而是告诉本地DNS服务器，到域服务器上去继续查询，并给出域服务器的地址。这种过程是迭代的过程。

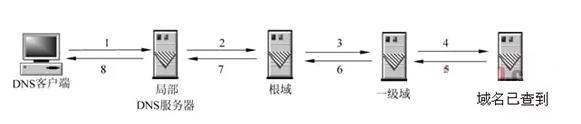
本地DNS服务器继续向域服务器发出请求，域服务器收到请求之后，也不会直接返回域名和IP地址的对应关系，而是告诉本地DNS服务器，你的域名的解析服务器的地址。

本地DNS服务器向域名的解析服务器发出请求

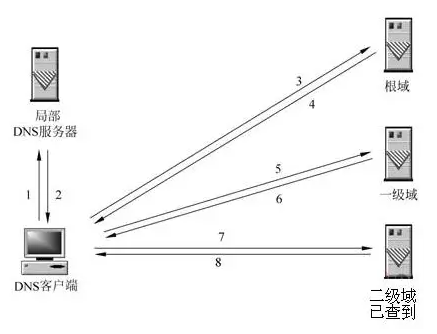
比如请求[www.163.com先在hosts](http://www.163.com先在hosts)中查询，再到本地DNS查询，再到根DNS中获取.com域服务器的地址，本地DNS再向.com域服务器发请求告诉本地DNS163.com域服务器的地址，本地DNS再请求163.com域服务器的地址，然后保存到本地DNS



DNS查询方式：递归和迭代



递归



迭代

## HashMap：

hashMap的key是对象时，为什么需要重写equals和hashcode方法

key(1)对象作为hashMap的key，在获取的时候，new key(1)获取key(1)对应的值，因为未重写hashcode方法，hashcode是对象的地址，从而获取不到对应key(1)的值，需要重写equals方法，保证new key(1)和new key(1)的hashcode一致，但是获取值任然为null，hashcode一致后会比较equals方法(hash冲突)，所以需要重写equals方法

equals和hashcode在插入和获取都会使用

## 注解：

注解：代码的元数据

@Override注解：可有可无，@Override编译时，如果父类无对应的方法会报错

XML vs annotation

XML使得配置文件和代码解耦

annotation紧紧的耦合到代码中

根注解：

@Document：是否将注解包含在JavaDoc中

@Retention：什么时候使用该注解(定义注解的声明周期)

RetentionPolicy.SOURCE：编译时使用(编译时校验)，不会写入字节码，如@Override

RetentionPolicy.CLASS：类加载时使用，使用完丢弃(默认)

RetentionPolicy.RUNTIME：始终不丢弃，运行时也保留该注解，运行可以利用反射读取(一般自定义使用此方式)

@Target：注解用于什么地方(如果不明确指出，该注解可以放在任何地方)

ElementType.TYPE:用于描述类、接口或enum声明

ElementType.FIELD:用于描述实例变量

ElementType.METHOD

ElementType.PARAMETER

ElementType.CONSTRUCTOR

ElementType.LOCAL\_VARIABLE

ElementType.ANNOTATION\_TYPE 另一个注释

ElementType.PACKAGE 用于记录java文件的package信息

@Inherited：是否允许子类继承该注解

@A注解(@Inherited允许继承)作用于Class A，Class B extends A，那么Class B也有@A注解的功能

注解的属性：

只支持基本数据类型，String，枚举

定义注解：

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface Todo{

public enum Status {STARTED,NOT\_STARTED}

String auth() default "josiah";

Status status() default Status.STARTED;

}

使用注解：

@Todo(auth = "lukas", status = Todo.Status.STARTED)

public void test(){

}

然后通过反射获取注解

## Token设计：

## 数据库：

MySQL的InnoDB判断死锁：

事务等待锁的时间超过指定时间(可通过innodb\_lock\_wait\_timeout设置)，就会回滚事务，释放当前锁，其他事务就可以获取当前资源

wait-for graph算法：

将事务视为节点，事务A需要事务B的资源时，事务A就指向事务B，如果事务之间的有向图成环的视为存在死锁，当事务加锁请求无法完成时，就会触发wait-for graph算法，这样不用等待超过指定时间

## 类加载：

ClassLoader与class.forname

两者都是对类进行加载，ClassLoader遵循双亲委派原则，class.forName底层也是调用的ClassLoader

未完，看源码