网络:

1. **三次握手和四次挥手**
2. **https和http:**

https=http+ssl协议

ssL加密：

对称加密：客户端-服务端用同一套秘匙；

非对称加密：发给客户端的公匙—服务端私钥；

对称+非对称：

使用非对称加密来加密 对称加密的秘匙—>发给服务端获取对称加密的秘匙后，使用对称秘匙传输数据；

客服端请求公匙—>服务端；

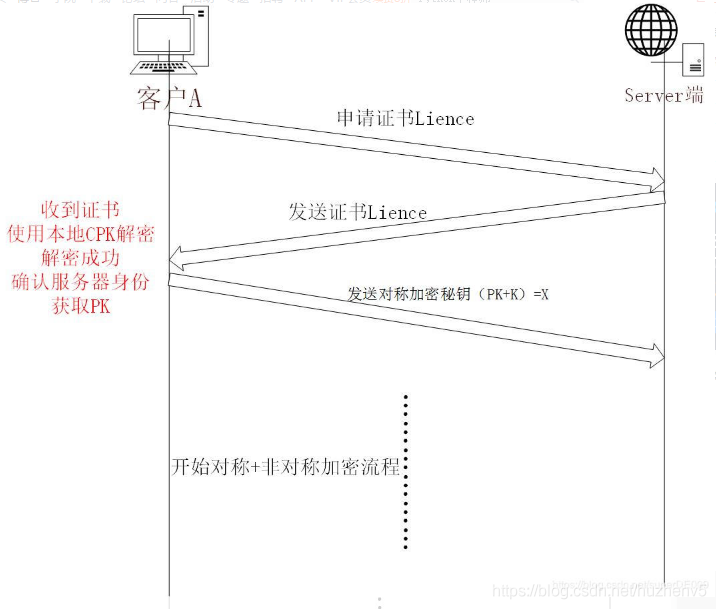
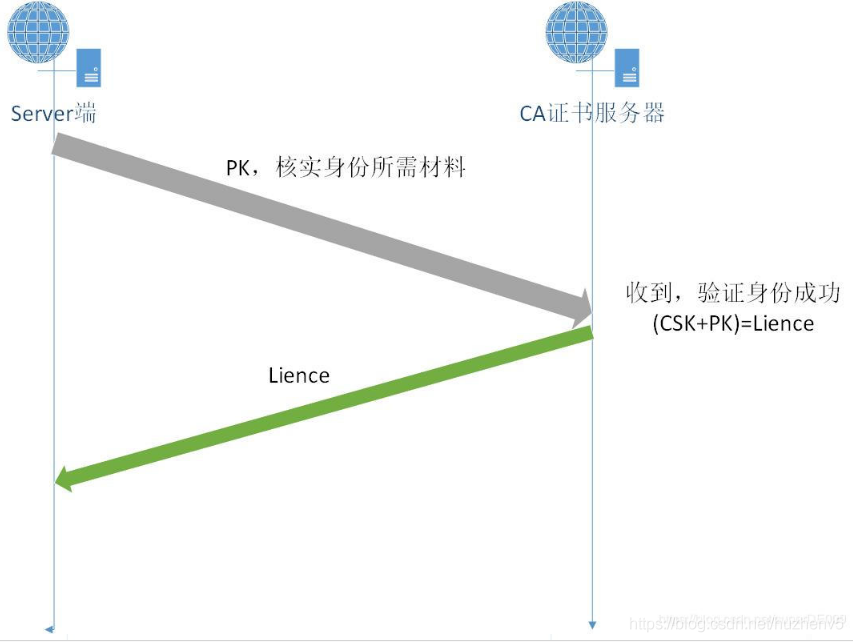
客户端收到服务端下发的公匙，使用该公匙加密自己生成的秘匙发送—>服务端；

服务端用私钥解密公匙加密的秘匙后，用该秘匙传输数据；

但是：如果黑客在客户访问访问服务器之前，对客户进行DNS劫持，使得

客户访问<—>黑客自己建立的伪装服务器<—>真正服务器;

CA证书方式：



**http和https的区别：**

https 加了ssl层，需要CA证书，是加密传输；http明文传输。

HTTPS=HTTP+加密+认证+完整性保护，比HTTP安全

https默认端口443，http默认端口80

1. **get和post区别：**<https://www.cnblogs.com/logsharing/p/8448446.html>

GET在浏览器回退时是无害的，而POST会再次提交请求。(回退重新请求接口，get请求走缓存，post则每次调用会创建新的资源）

GET产生的URL地址可以被Bookmark，而POST不可以。（由于书签仅包括URL，但post请求的参数在请求body中，Web服务器不知道该如何处理请求）

GET请求会被浏览器主动cache，而POST不会，除非手动设置。

GET请求只能进行url编码，而POST支持多种编码方式。

GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。

GET请求在URL中传送的参数是有长度限制的，而POST么有。

对参数的数据类型，GET只接受ASCII字符，而POST没有限制。

GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。

GET参数通过URL传递，POST放在Request body中。

**应用服务器处理bai业务逻辑，web服务器是用于处理HTML文件的**

**http header相关：**

请求：请求是由浏览器生成的，包括：

<request-line>请求类型（get/post）、访问的资源、http版本

<head>

<blank-line>

<body>

四个部分组成。其中头部信息包括：host，userAgent,connection,content-type(请求的编码方式),content-length，accept（接受的MIME类型），accept-charset,accept-Encoding,cookie,If-Modified-Since,pragma,referer(是从哪个页面跳转过来的)

响应是由服务器生成的，包括：

<status-line>

<head>

<blank-line>

<body>

其中head规定了catch-control,Date(响应的时间)，LastModified，Etag

**缓存机制（浏览器缓存就是http缓存）：**

浏览器想请求一个资源-》查看缓存，发现资源过期了（Cache-Control）-》发现资源有lastModified和Etag->请求头部带上If-modified-since和If-None-Match-》服务器判定资源是不是修改过-》决定200重新发送还是304用缓存

Cache-Control：(**控制强制缓存，**主要用于控制网页缓存)

在服务器端设置:response.setHeader('Cache-Control', 'max-age=30')，服务器返回带有cache-control的响应. 当浏览器再次请求同一URL资源时, 若没有超过30秒, 则会加载缓存中的资源. 若超过30秒则会向服务器重新请求.

通常将首页不设置缓存：不设置cache-control时, 当服务器更新资源时, 资源对应的URL也会改变. 此时客户端发送请求, 其请求资源的URL也会改变. 即客户端不会从缓存中获取资源, 而是从服务器端获取资源.

Expires和cache-coontrol的用处一样expires是老的API, cache-control是新的API.

ETag和If-None-Match

将资源的md5值放在ETag属性中, 这样当我们请求服务器的资源时, (server)比较上次请求的md5值和本次请求的md5值. 注意请求的资源的URL必须是一致的.

若相等则说明资源未更新, 返回部分响应, 包含304状态码, 客户端从缓存中获取资源.

反之说明资源更新了, 服务器返回完整的响应, 即客户端从服务器端获取资源.

**四、进程和线程：**

进程是操作系统分配资源最小单位（是程序执行时的一个实例。程序运行时系统就会创建一个进程，并为它分配资源）；

线程是cpu调度任务和执行的最小单位；

区别：

* 地址空间和资源：进程间地址空间 和 资源独立。[资源：如内存、i/o等]

线程共享本进程的地址空间和资源。

* 执行过程：每个进程有一个程序运行入口，顺序执行序列，执行开销大；

线程不能独立运行，由进程提供多个线程执行控制，执行开销小

* 健壮性： 一个进程崩溃后，在保护模式下不会对其他进程产生影响，但是一个线程崩溃整个进程都死掉。
* 包含关系：没有线程的进程可以看做是单线程的，如果一个进程内有多个线程，则执行过程不是一条线的，而是多条线（线程）共同完成的；线程是进程的一部分，所以线程也被称为轻权进程或者轻量级进程。
* 并发和并行：进程之间相互独立可以实现并行；

线程不可以：多线程只能并发执行，而且是顺序执行，只是在同一时间片段，类似同时执行，CPU可以按时间切片执行，单核CPU同一时刻只支持一个线程执行任务，多线程并发事实上就是多个线程排队申请调用CPU，CPU执行速度很多，所以看上去是多个线程任务说是并发处理。

session和cookie

1，session 在服务器端，cookie 在客户端（浏览器）  
2，session 默认被存在在服务器的一个文件里（不是内存）  
3，session 的运行依赖 session id，而 session id 是存在 cookie 中的，也就是说，如果浏览器禁用了 cookie ，同时 session 也会失效（但是可以通过其它方式实现，比如在 url 中传递 session\_id）  
4，session 可以放在 文件、数据库、或内存中都可以。  
5，用户验证这种场合一般会用 session