## Java学习目标

Spring，JVM，MYSQL优化，dubbo，springcloud，并发编程，netty

## 综合

### CAP理论：

CAP原则又称CAP定理，指的是在一个分布式系统中， Consistency（一致性）、 Availability（可用性）、Partition tolerance（分区容忍性），三者不可得兼。

一致性（C）：在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）

可用性（A）：在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）

分区容忍性（P）：以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择。

**PS：**一个分布式系统里面，节点组成的网络本来应该是连通的。然而可能因为一些故障，使得有些节点之间不连通了，整个网络就分成了几块区域。数据就散布在了这些不连通的区域中。这就叫分区。

当你一个数据项只在一个节点中保存，那么分区出现后，和这个节点不连通的部分就访问不到这个数据了。这时分区就是无法容忍的。

提高分区容忍性的办法就是一个数据项复制到多个节点上，那么出现分区之后，这一数据项就可能分布到各个区里。容忍性就提高了。

然而，要把数据复制到多个节点，就会带来一致性的问题，就是多个节点上面的数据可能是不一致的。要保证一致，每次写操作就都要等待全部节点写成功，而这等待又会带来可用性的问题。

总的来说就是，数据存在的节点越多，分区容忍性越高，但要复制更新的数据就越多，一致性就越难保证。为了保证一致性，更新所有节点数据所需要的时间就越长，可用性就会降低。

### 进程和线程的区别

操作系统资源+n个线程=进程

Nginx中，操作系统资源+1线程=进程

### epoll模式

惊群模式：新连接来了，大家一哄而上，谁抢到算谁的（没抢到的无用功）

poll和select模式：被惊醒的线程，需要轮训所有的http连接，定位触发事件的那个连接

epoll模式：线程被惊醒之后，（Linux操作系统在触发事件时把触发的连接保存进了map，供线程获取）

event+poll=epoll

### Nginx的多进程模型

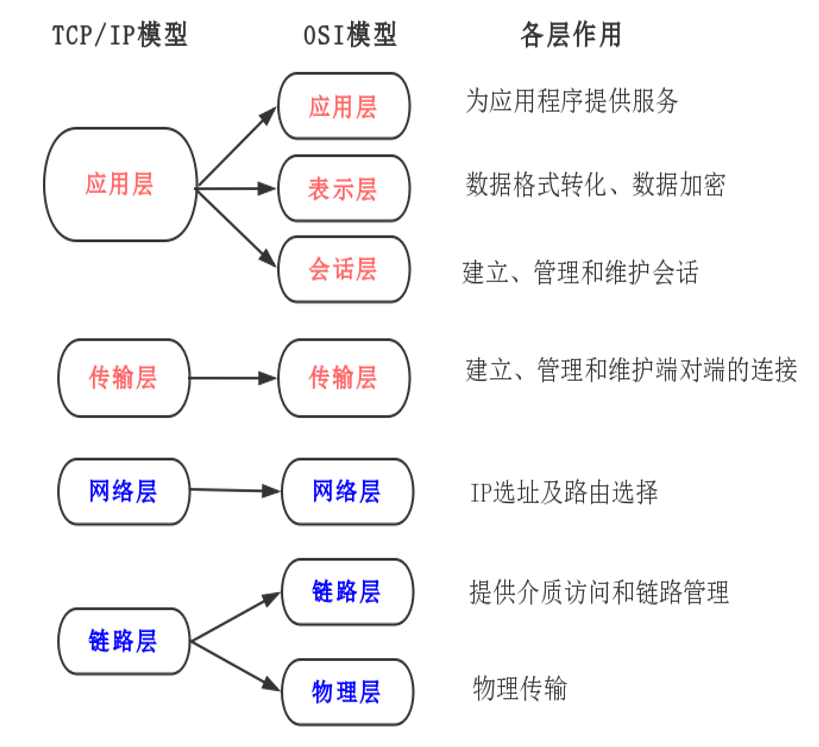


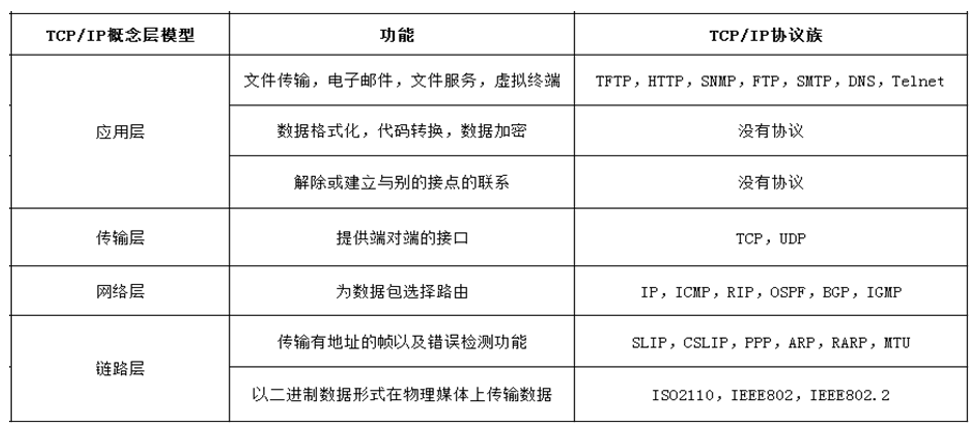
### 单例模式



## 网络协议与Netty

### OSI七层模型和TCP/IP模型

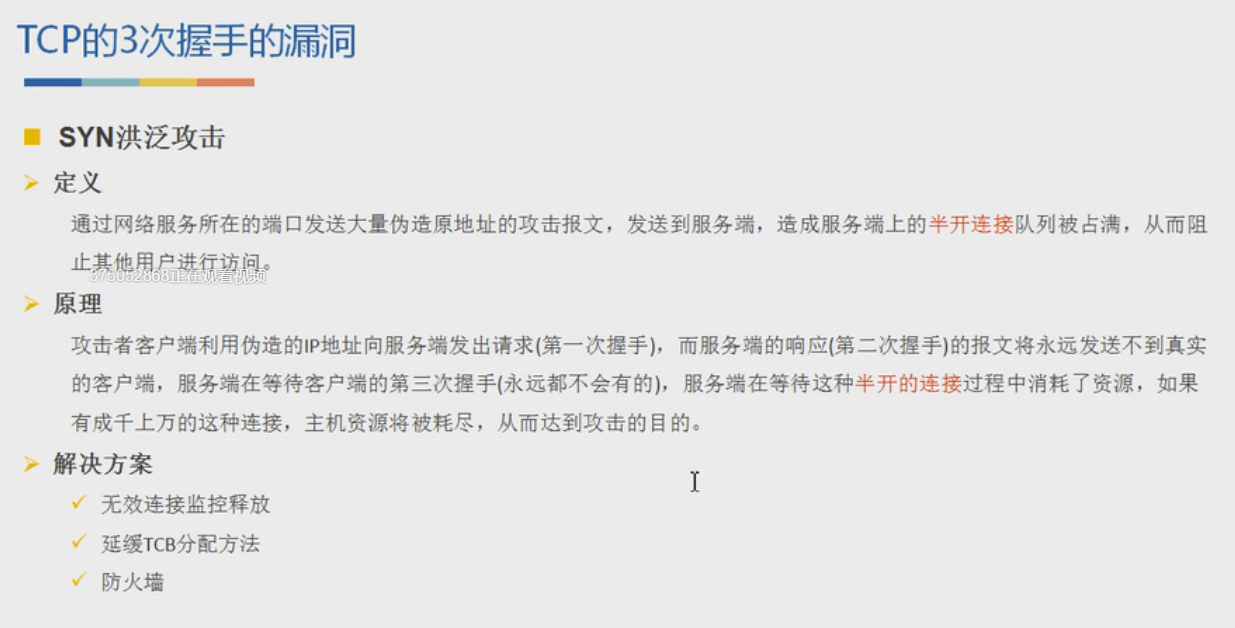




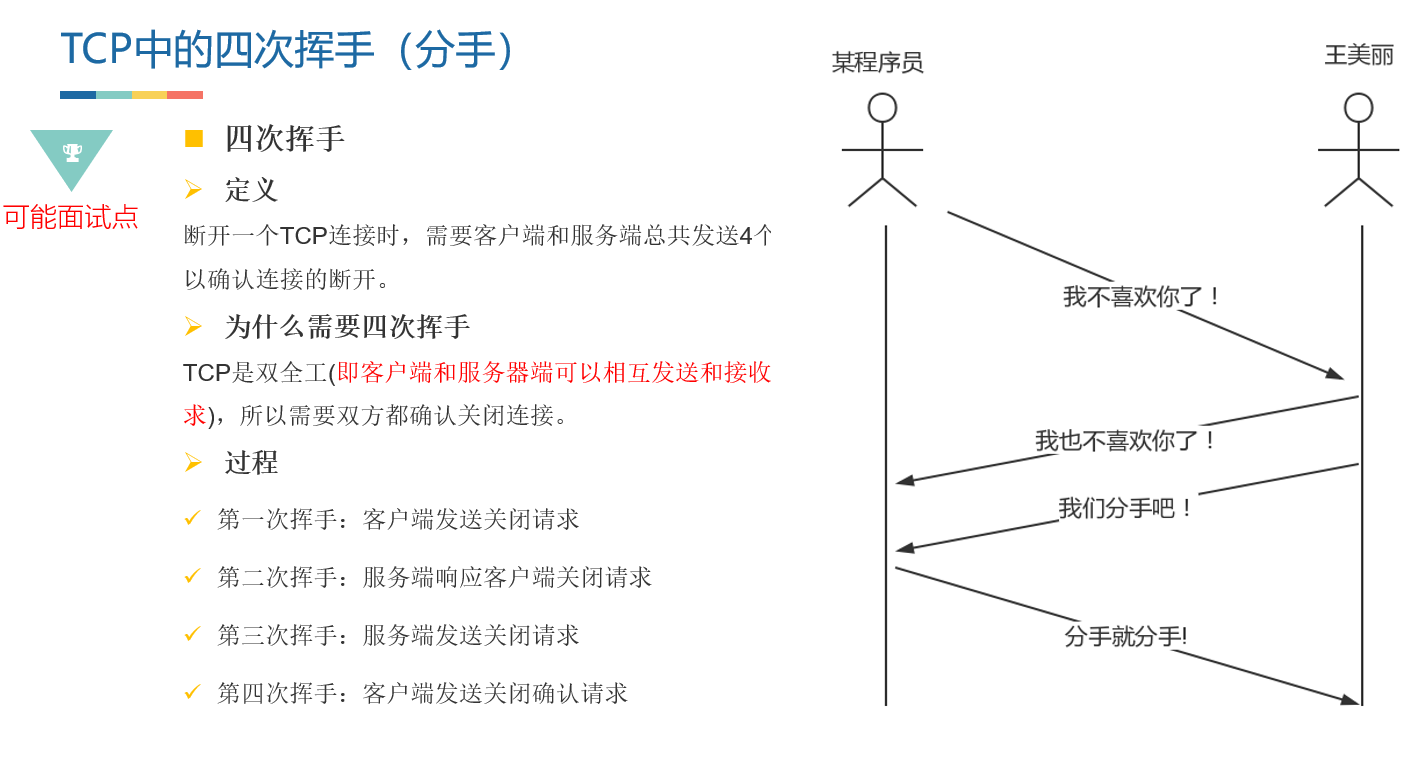
### TCP建立连接的三次握手



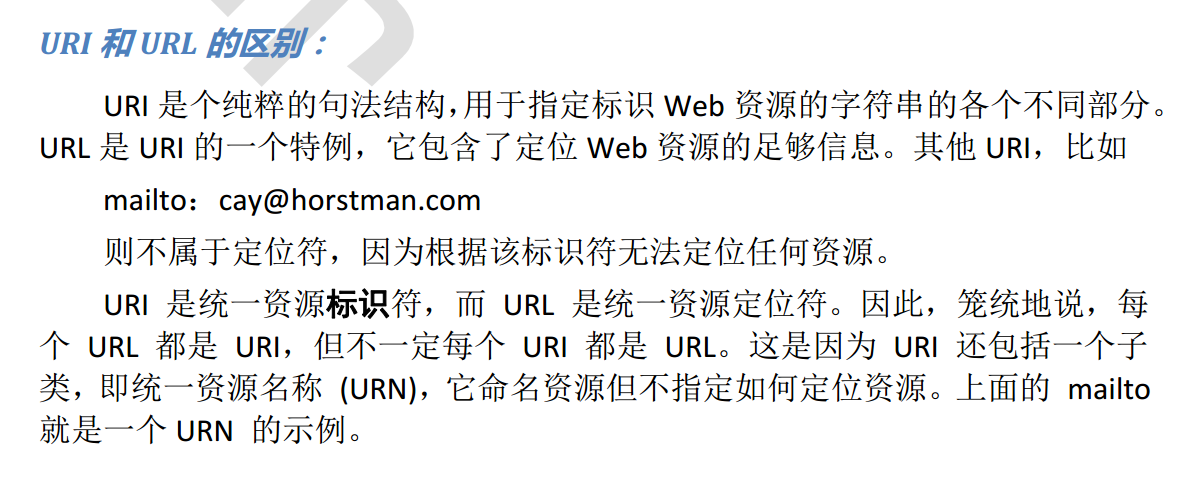
### \*TCP三次握手的漏洞和解决方案



### TCP断开连接的四次分手



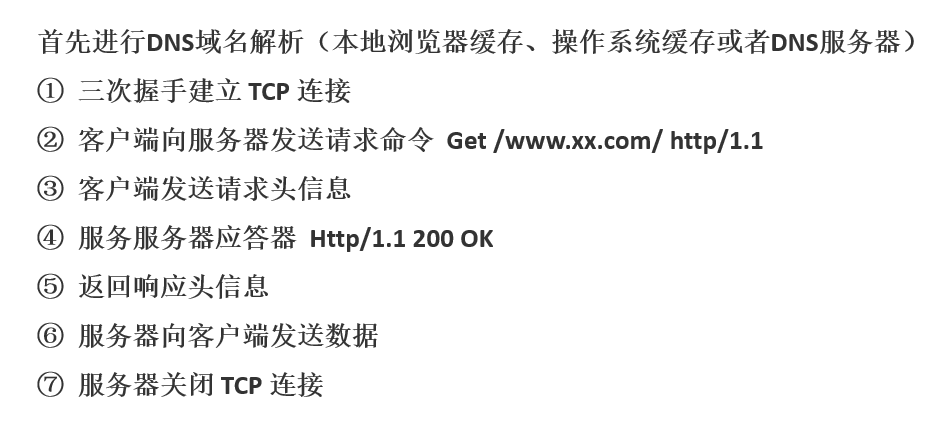
### URI和URL的区别



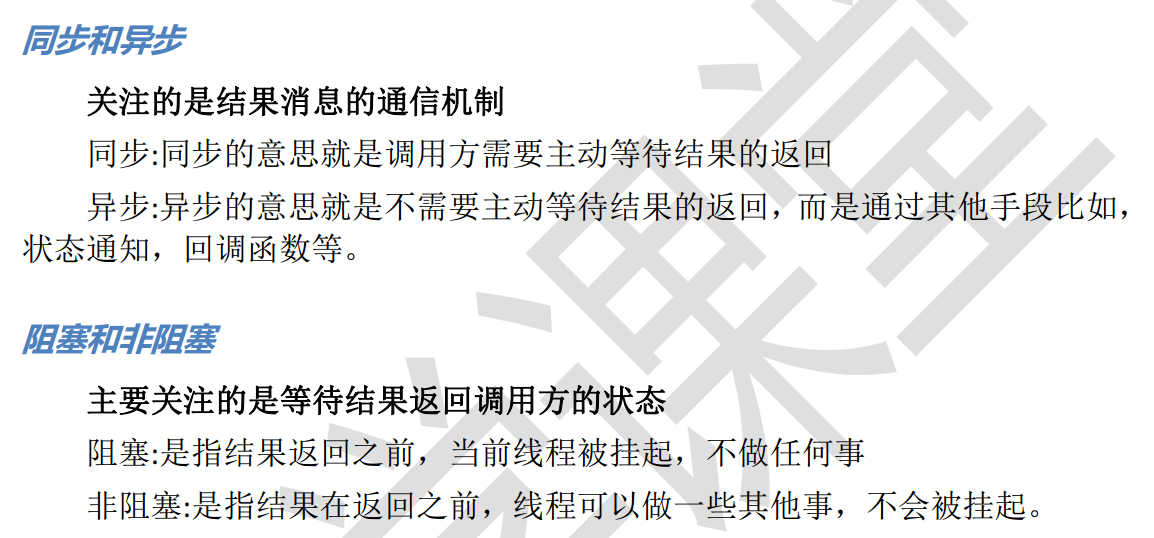
### http协议媒体类型

Content-Type：text/html、 image/jpeg，等。表示请求的资源的类型，与URL路径无关。即使路径为<Http://www.123.com/abc/jpg> 也不代表资源类型为jpeg。

### 一次完整http请求的7个过程



### 同步和异步，阻塞和非阻塞



## 并发编程

### 进程和线程

进程：程序运行资源分配（资源包括CPU，内存，IO等等）的最小单位。

线程：CPU调度的最小单位，必须依赖于进程而存在。多个线程之间可以共享资源。

PS：Windows一个进程所能开的最大线程数为2000；Linux为1000。由OS内核写死了。

每次new出一个线程，JVM要给该线程分配一个栈空间，缺省值大小为1M。如果开了1000个线程，则必须分配1G的内存空间。

### 时间片轮转

RR调度（Round-Robin，轮询调度），把CPU时间进行切片，每个进程分配一个时间段

### 上下文切换

一次上下文切换需要20000个CPU时间周期

### 并行和并发

并行：同时进行多个任务。

并发：交替进行多个任务。并发不能脱离时间。时间片轮转本质上就是并发机制

### Thread和Runnable的区别

Thread：java语言对线程的抽象

Runnable：对任务和业务逻辑的抽象

### 中断线程不用stop()而用interrupt()

因为会导致线程占用的资源无法正常释放。

interrupt()：中断线程

isInterrupted()：判断线程是否中断

interrupted()：当调用interrupt中断后，把中断标志位改回false，而isInterrupted不会改回false依然是true（不常用）

### JDK里线程是协作式，而不是抢占式

## JVM性能调优