# Jvm

当出现了内存溢出如何定位排查

Jmap –dump命令生成堆内存快照 jmap-heap命令显示java堆详细信息。如使用哪种回收器，参数配置。分代状况等

Jhat 工具分析。或者使用ibm heapanalyzer 或eclipse memomry analyzer分析

Jvm调优

# Tomcat以及服务器

Tomcat如何调优

<https://blog.csdn.net/qq_28109171/article/details/84256783>

并发优化

Connector 优化

Connector是连接器，负责接收客户的请求，以及向客户端回送响应的消息。所以 Connector的优化是重要部分。默认情况下 Tomcat只支持200线程访问，超过这个数量的连接将被等待甚至超时放弃，所以我们需要提高这方面的处理能力。

参数说明

maxThreads 客户请求最大线程数

minSpareThreads Tomcat初始化时创建的 socket 线程数

maxSpareThreads Tomcat连接器的最大空闲 socket 线程数

enableLookups 若设为true, 则支持域名解析，可把 ip 地址解析为主机名

redirectPort 在需要基于安全通道的场合，把客户请求转发到基于SSL 的 redirectPort 端口

acceptAccount 监听端口队列最大数，满了之后客户请求会被拒绝（不能小于maxSpareThreads ）

connectionTimeout 连接超时

minProcessors 服务器创建时的最小处理线程数

maxProcessors 服务器同时最大处理线程数

URIEncoding URL统一编码

开启线程池

配置

参数说明

name：线程池名称，用于 Connector中指定。

namePrefix：所创建的每个线程的名称前缀，一个单独的线程名称为 namePrefix+threadNumber。

maxThreads：池中最大线程数。

minSpareThreads：活跃线程数，也就是核心池线程数，这些线程不会被销毁，会一直存在。

maxIdleTime：线程空闲时间，超过该时间后，空闲线程会被销毁，默认值为6000（1分钟），单位毫秒。

maxQueueSize：在被执行前最大线程排队数目，默认为Int的最大值，也就是广义的无限。除非特殊情况，这个值不需要更改，否则会有请求不会被处理的情况发生。

prestartminSpareThreads：启动线程池时是否启动 minSpareThreads部分线程。默认值为false，即不启动。

threadPriority：线程池中线程优先级，默认值为5，值从1到10。

className：线程池实现类，未指定情况下，默认实现类为org.apache.catalina.core.StandardThreadExecutor。如果想使用自定义线程池首先需要实现 org.apache.catalina.Executor接口。

tomcat各参数的意义

-server：启用 JDK的 server 版本；

-Xms：Java虚拟机初始化时堆的最小内存，一般与 Xmx配置为相同值，这样的好处是GC不必再为扩展内存空间而消耗性能；

-Xmx：Java虚拟机可使用堆的最大内存；

-XX:PermSize：Java虚拟机永久代大小；

-XX:MaxPermSize：Java虚拟机永久代大小最大值；

# 分布式

一致性hash

hash环

单调性(Monotonicity)，单调性是指如果已经有一些请求通过哈希分派到了相应的服务器进行处理，又有新的服务器加入到系统中时候，应保证原有的请求可以被映射到原有的或者新的服务器中去，而不会被映射到原来的其它服务器上去。 这个通过上面新增服务器ip5可以证明，新增ip5后，原来被ip1处理的user6现在还是被ip1处理，原来被ip1处理的user5现在被新增的ip5处理。

分散性(Spread)：分布式环境中，客户端请求时候可能不知道所有服务器的存在，可能只知道其中一部分服务器，在客户端看来他看到的部分服务器会形成一个完整的hash环。如果多个客户端都把部分服务器作为一个完整hash环，那么可能会导致，同一个用户的请求被路由到不同的服务器进行处理。这种情况显然是应该避免的，因为它不能保证同一个用户的请求落到同一个服务器。所谓分散性是指上述情况发生的严重程度。好的哈希算法应尽量避免尽量降低分散性。 一致性hash具有很低的分散性

平衡性(Balance)：平衡性也就是说负载均衡，是指客户端hash后的请求应该能够分散到不同的服务器上去。一致性hash可以做到每个服务器都进行处理请求，但是不能保证每个服务器处理的请求的数量大致相同，如下图

服务器ip1,ip2,ip3经过hash后落到了一致性hash环上，从图中hash值分布可知ip1会负责处理大概80%的请求，而ip2和ip3则只会负责处理大概20%的请求，虽然三个机器都在处理请求，但是明显每个机器的负载不均衡，这样称为一致性hash的倾斜，虚拟节点的出现就是为了解决这个问题。

当服务器节点比较少的时候会出现上节所说的一致性hash倾斜的问题，一个解决方法是多加机器，但是加机器是有成本的，那么就加虚拟节点，比如上面三个机器，每个机器引入1

常见的hash算法

Paxos算法

分布式一致性

Paxos将系统中的角色分为提议者 (Proposer)，决策者 (Acceptor)，和最终决策学习者 (Learner):

Proposer: 提出提案 (Proposal)。Proposal信息包括提案编号 (Proposal ID) 和提议的值 (Value)。

Acceptor：参与决策，回应Proposers的提案。收到Proposal后可以接受提案，若Proposal获得多数Acceptors的接受，则称该Proposal被批准。

Learner：不参与决策，从Proposers/Acceptors学习最新达成一致的提案（Value）。

<https://zhuanlan.zhihu.com/p/31780743>

https://zhuanlan.zhihu.com/paxos

# 消息队列

MQ如何保证消息不丢失

1. 突然宕机

通过消息持久化

1. ack确认机制

消费者消费过后要回复一个确认(超时没有响应别的消费者就回消费)。但是会可能产生重复消费

集群

一台宕机了换从机

# http相关

301 302区别

301永久重定向 302临时重定向

http和https的区别

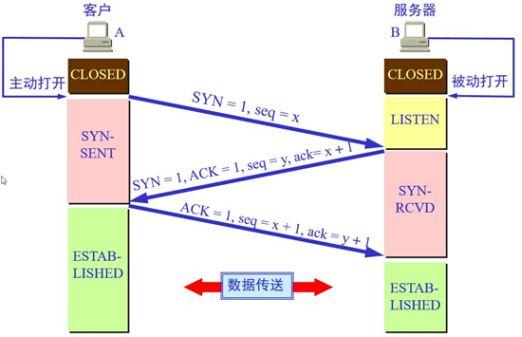
https加密ssl

http无状态

每次请求都是独立的

Tcp是有状态的通过包头的一些字段来表示前面发送的包和后面发送的包的关系，ip是无状态的是因为只负责把数据从一个ip传到另一个ip， udp是无状态的再拿udp举例子，它也是无状态传输协议的一种，它只负责发送数据并不需要知道上一次和谁连接过。而如果你每次通过udp传输的时候都带上某个固定字段，实际上这种行为就类似于cookie或session，是应用层的行为而不是协议层

Tcp三次握手四次挥手



）首先客户端向服务器端发送一段TCP报文，其中：

标记位为SYN，表示“请求建立新连接”;

序号为Seq=X（X一般为1）；

随后客户端进入SYN-SENT阶段。

（2）服务器端接收到来自客户端的TCP报文之后，结束LISTEN阶段。并返回一段TCP报文，其中：

标志位为SYN和ACK，表示“确认客户端的报文Seq序号有效，服务器能正常接收客户端发送的数据，并同意创建新连接”（即告诉客户端，服务器收到了你的数据）；

序号为Seq=y；

确认号为Ack=x+1，表示收到客户端的序号Seq并将其值加1作为自己确认号Ack的值；随后服务器端进入SYN-RCVD阶段。

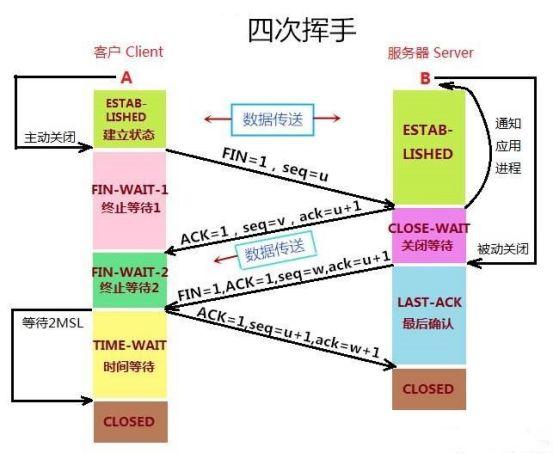
（3）客户端接收到来自服务器端的确认收到数据的TCP报文之后，明确了从客户端到服务器的数据传输是正常的，结束SYN-SENT阶段。并返回最后一段TCP报文。其中：

标志位为ACK，表示“确认收到服务器端同意连接的信号”（即告诉服务器，我知道你收到我发的数据了）；

序号为Seq=x+1，表示收到服务器端的确认号Ack，并将其值作为自己的序号值；

确认号为Ack=y+1，表示收到服务器端序号Seq，并将其值加1作为自己的确认号Ack的值；

随后客户端进入ESTABLISHED阶段。



首先客户端想要释放连接，向服务器端发送一段TCP报文，其中：

标记位为FIN，表示“请求释放连接“；

序号为Seq=U；

随后客户端进入FIN-WAIT-1阶段，即半关闭阶段。并且停止在客户端到服务器端方向上发送数据，但是客户端仍然能接收从服务器端传输过来的数据。

注意：这里不发送的是正常连接时传输的数据(非确认报文)，而不是一切数据，所以客户端仍然能发送ACK确认报文。

（2）服务器端接收到从客户端发出的TCP报文之后，确认了客户端想要释放连接，随后服务器端结束ESTABLISHED阶段，进入CLOSE-WAIT阶段（半关闭状态）并返回一段TCP报文，其中：

标记位为ACK，表示“接收到客户端发送的释放连接的请求”；

序号为Seq=V；

确认号为Ack=U+1，表示是在收到客户端报文的基础上，将其序号Seq值加1作为本段报文确认号Ack的值；

随后服务器端开始准备释放服务器端到客户端方向上的连接。

客户端收到从服务器端发出的TCP报文之后，确认了服务器收到了客户端发出的释放连接请求，随后客户端结束FIN-WAIT-1阶段，进入FIN-WAIT-2阶段

前"两次挥手"既让服务器端知道了客户端想要释放连接，也让客户端知道了服务器端了解了自己想要释放连接的请求。于是，可以确认关闭客户端到服务器端方向上的连接了

（3）服务器端自从发出ACK确认报文之后，经过CLOSED-WAIT阶段，做好了释放服务器端到客户端方向上的连接准备，再次向客户端发出一段TCP报文，其中：

标记位为FIN，ACK，表示“已经准备好释放连接了”。注意：这里的ACK并不是确认收到服务器端报文的确认报文。

序号为Seq=W；

确认号为Ack=U+1；表示是在收到客户端报文的基础上，将其序号Seq值加1作为本段报文确认号Ack的值。

随后服务器端结束CLOSE-WAIT阶段，进入LAST-ACK阶段。并且停止在服务器端到客户端的方向上发送数据，但是服务器端仍然能够接收从客户端传输过来的数据。

（4）客户端收到从服务器端发出的TCP报文，确认了服务器端已做好释放连接的准备，结束FIN-WAIT-2阶段，进入TIME-WAIT阶段，并向服务器端发送一段报文，其中：

标记位为ACK，表示“接收到服务器准备好释放连接的信号”。

序号为Seq=U+1；表示是在收到了服务器端报文的基础上，将其确认号Ack值作为本段报文序号的值。

确认号为Ack=W+1；表示是在收到了服务器端报文的基础上，将其序号Seq值作为本段报文确认号的值。

随后客户端开始在TIME-WAIT阶段等待2MSLRFC 793中规定MSL为2分钟，实际应用中常用的是30秒，1分钟和2分钟等

为什么tcp建立连接3次握手，关闭连接4次挥手

建立连接时，被动方服务器端结束CLOSED阶段进入“握手”阶段并不需要任何准备，可以直接返回SYN和ACK报文，开始建立连接。

释放连接时，被动方服务器，突然收到主动方客户端释放连接的请求时并不能立即释放连接，因为还有必要的数据需要处理，所以服务器先返回ACK确认收到报文，经过CLOSE-WAIT阶段准备好释放连接之后，才能返回FIN释放连接报文。

为什么客户端在TIME-WAIT阶段要等2MSL?

为的是确认服务器端是否收到客户端发出的ACK确认报文

当客户端发出最后的ACK确认报文时，并不能确定服务器端能够收到该段报文。所以客户端在发送完ACK确认报文之后，会设置一个时长为2MSL的计时器。MSL指的是Maximum Segment Lifetime：一段TCP报文在传输过程中的最大生命周期。2MSL即是服务器端发出为FIN报文和客户端发出的ACK确认报文所能保持有效的最大时长。

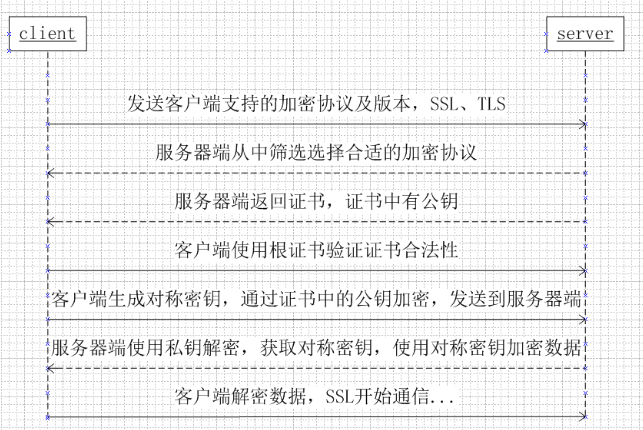
服务器端在1MSL内没有收到客户端发出的ACK确认报文，就会再次向客户端发出FIN报文；

如果客户端在2MSL内，再次收到了来自服务器端的FIN报文，说明服务器端由于各种原因没有接收到客户端发出的ACK确认报文。客户端再次向服务器端发出ACK确认报文，计时器重置，重新开始2MSL的计时；

否则客户端在2MSL内没有再次收到来自服务器端的FIN报文，说明服务器端正常接收了ACK确认报文，客户端可以进入CLOSED阶段，完成“四次挥手”。

所以，客户端要经历时长为2SML的TIME-WAIT阶段；这也是为什么客户端比服务器端晚进入CLOSED阶段的原因

https建立连接流程



当客户在浏览器中输入网址的并且按下回车,浏览器会在浏览器DNS缓存,本地DNS缓存,和Hosts中寻找对应的记录,如果没有获取到则会请求DNS服务来获取对应的ip

在使用HTTPS是需要保证服务端配置正确了对应的安全证书

客户端发送请求到服务端

服务端返回公钥和证书到客户端

客户端接收后会验证证书的安全性,如果通过则会随机生成一个随机数,用公钥对其加密,发送到服务端

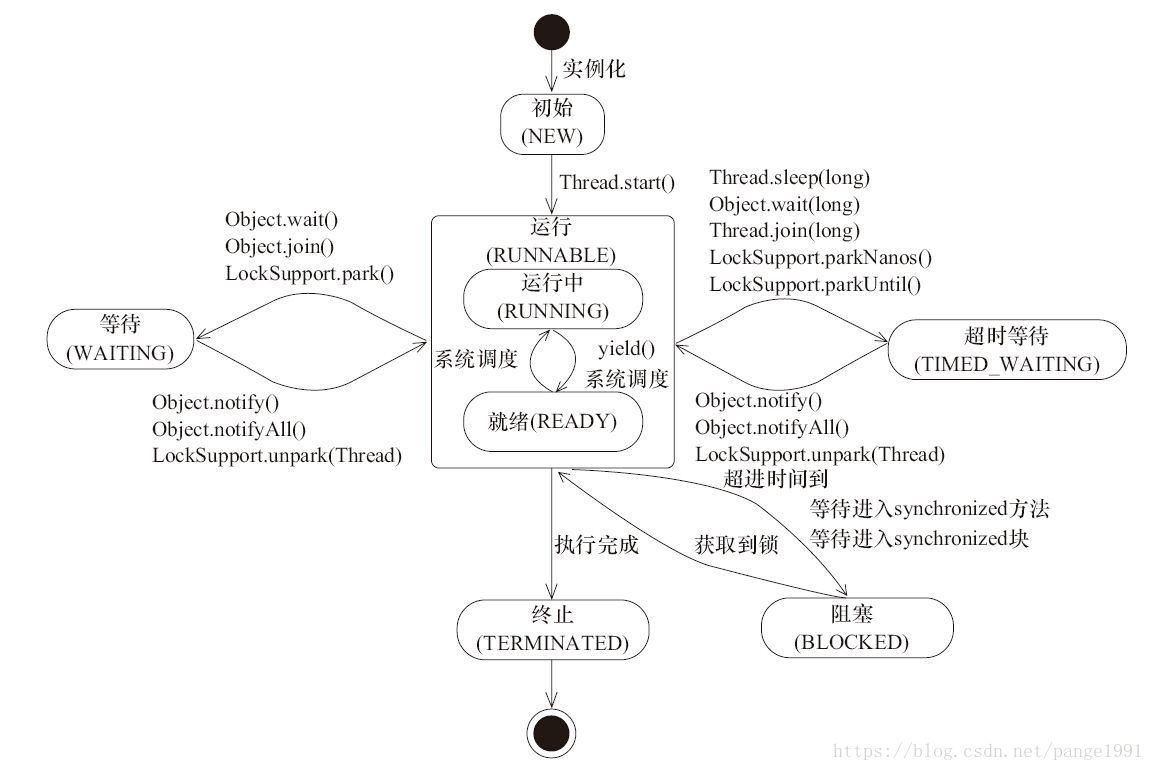
服务端接受到这个加密后的随机数后会用私钥对其解密得到真正的随机数,随后用这个随机数当做私钥对需要发送的数据进行对称加密

客户端在接收到加密后的数据使用私钥(即生成的随机值)对数据进行解密并且解析数据呈现结果给客户

SSL加密建立

# 多线程

线程状态



Jmm

