

Tomcat学习四步走：内核、集群、参数及性能



DBAplus社群

2018-01-15 · 互联网领域创作者

订阅

本文根据DBAplus社群第135期线上分享整理而成

讲师介绍



汪建

《Tomcat内核设计剖析》作者

- 笔名seaboat，中国电信资深架构师，从事航空系统、电信系统、中间件、基础架构、智能客服等研发工作，目前主要关注分布式、高并发、大数据、搜索引擎、机器学习、深度学习等方面的技术。
- 崇尚开源，崇尚技术自由，更崇尚思想自由。个人公众号：远洋号（seaboat-top）

内核实现原理

分布式集群

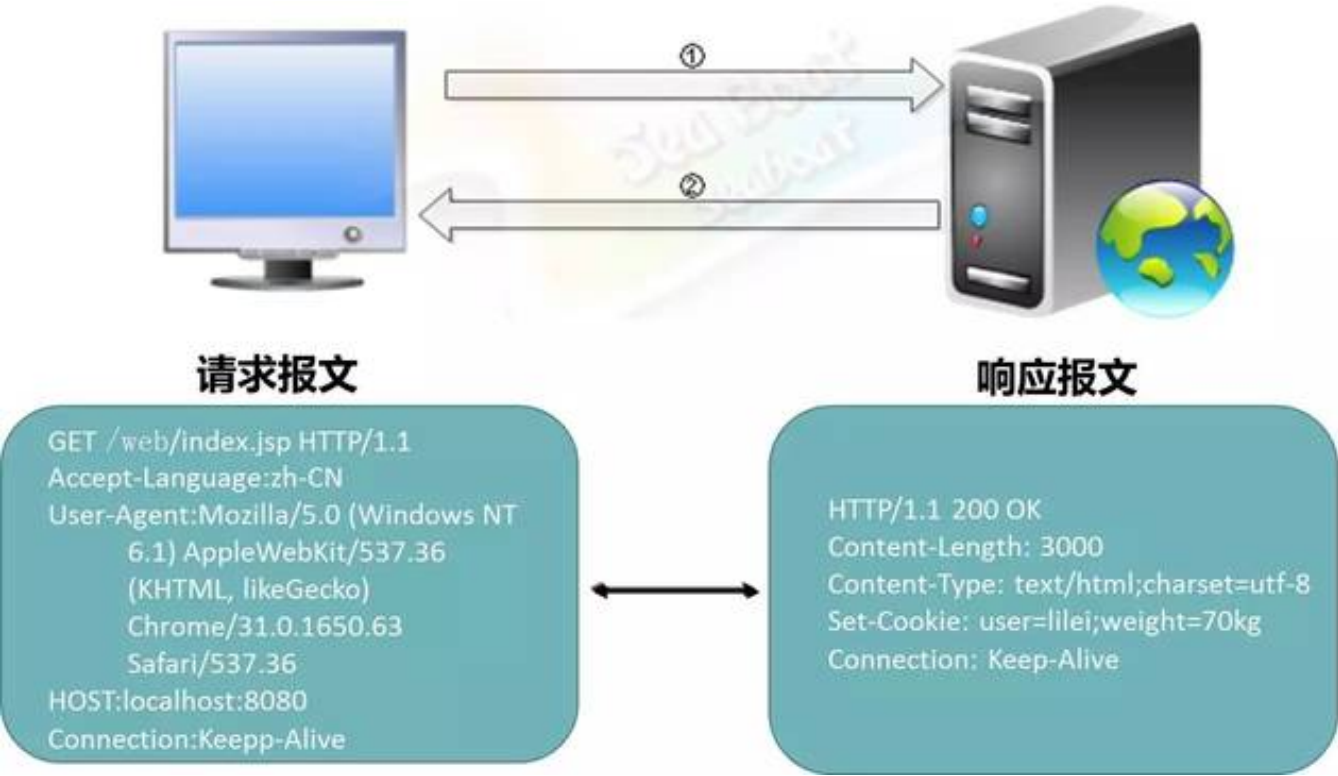
生产部署关键参数

性能监控和分析

一、内核实现原理HTTP



Web服务器与浏览器之间以HTTP协议通信，浏览器要访问服务器即向服务器发送HTTP请求报文。



如图，此处用get方法访问了localhost的8080端口的Web、Index、JSP，服务器返回200状态码并将一些HTTP报文返回到客户端。

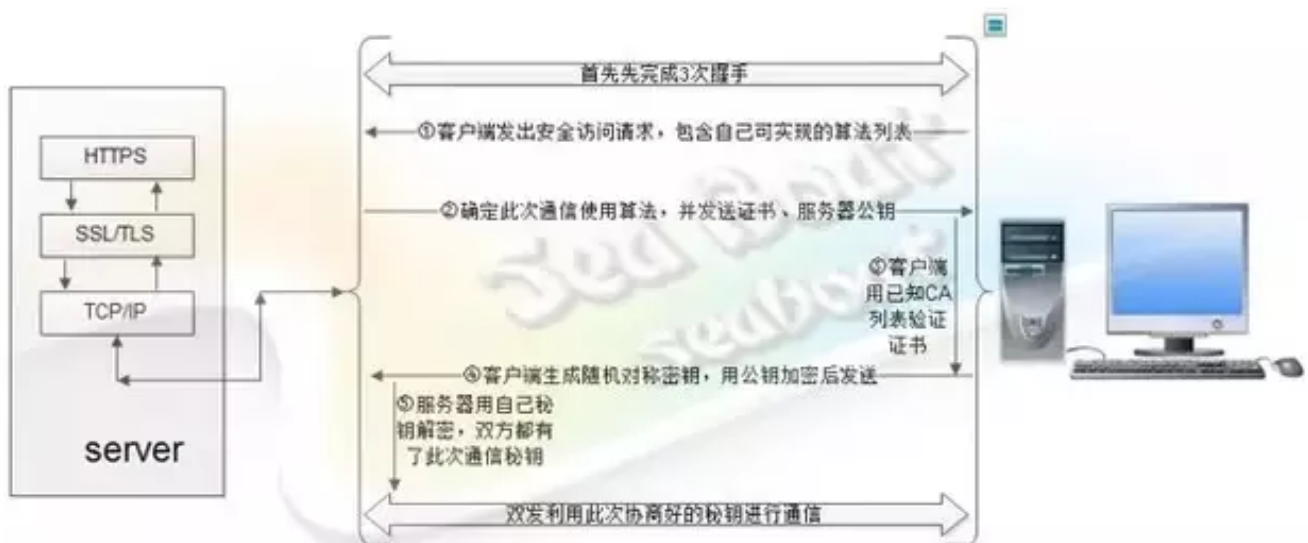
HTTP报文



从图中可以看到，**HTTP**报文中的请求报文和响应报文都由三部分组成。请求报文由请求行、请求头和请求体三部分组成，其中请求行主要包括**method**、**uri**和协议版本；请求头主要包含**kv**对；请求体一般以**post**方法来存放参数；而响应报文则由响应行、响应头和响应体组成，其中响应行主要包括协议版本和状态码；响应头包含**kv**对；响应体则包含真正的报文。

HTTPS协议

我们也可以把**HTTPS**看成是**HTTP**的安全版本，此时它不再是明文通信，而是双方协商出密钥后对报文进行加密后再通信。在这过程中，加密后需要对其进行解密，然后才能进行下一步处理。

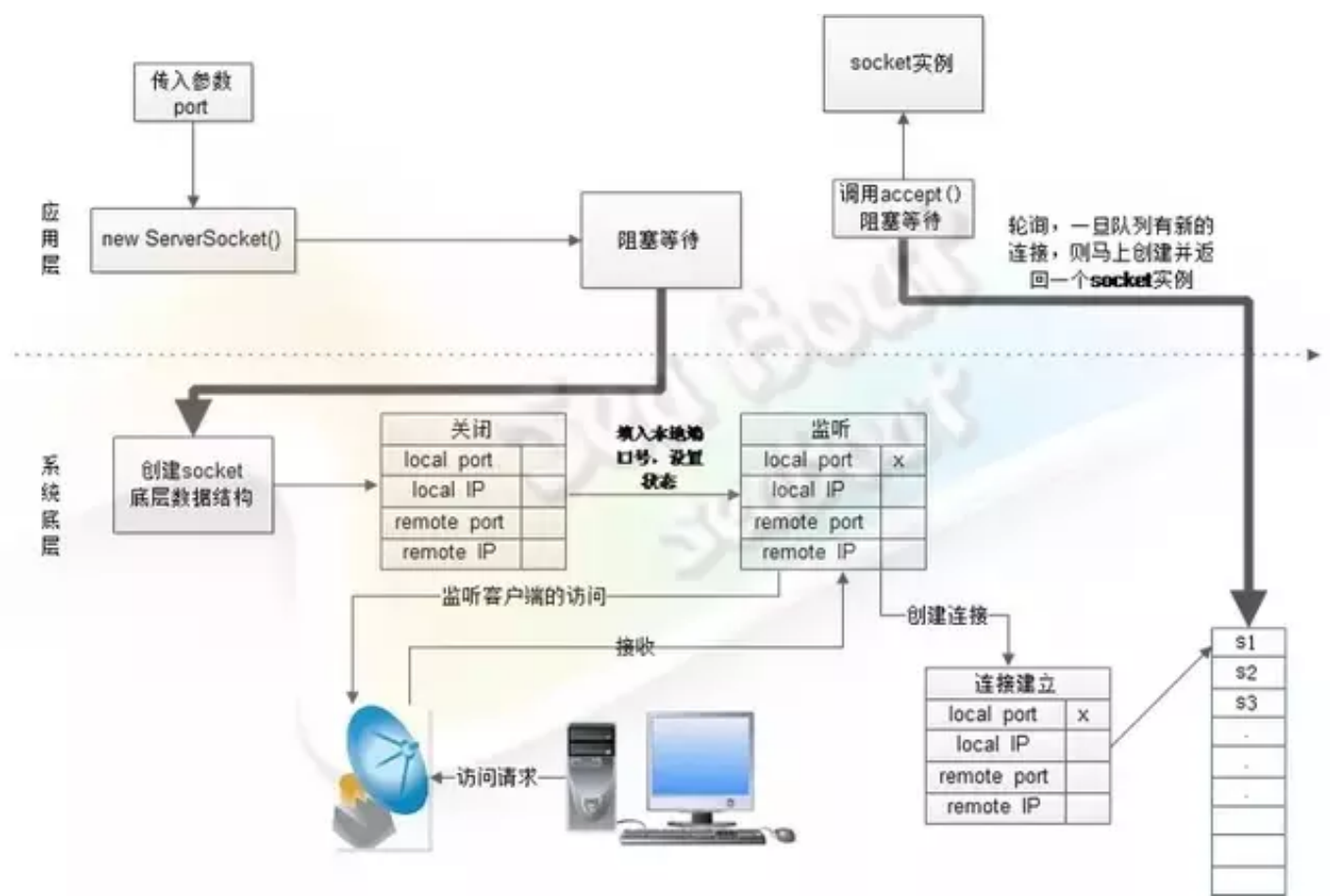


HTTPS在**TCP/TP**协议上层多加了一层**SSL/TLS**层，所以它能做到对**Web**应用的透明化。我们可以看到，客户端连服务端后通过一定的步骤来协商确定密钥，而**Java**也已经有了**SSL/TLS**协议过程的包，就无需自己再做了。

套接字通信

大家应该都很熟悉套接字了，那我们再深入地探讨下服务端套接字的过程：

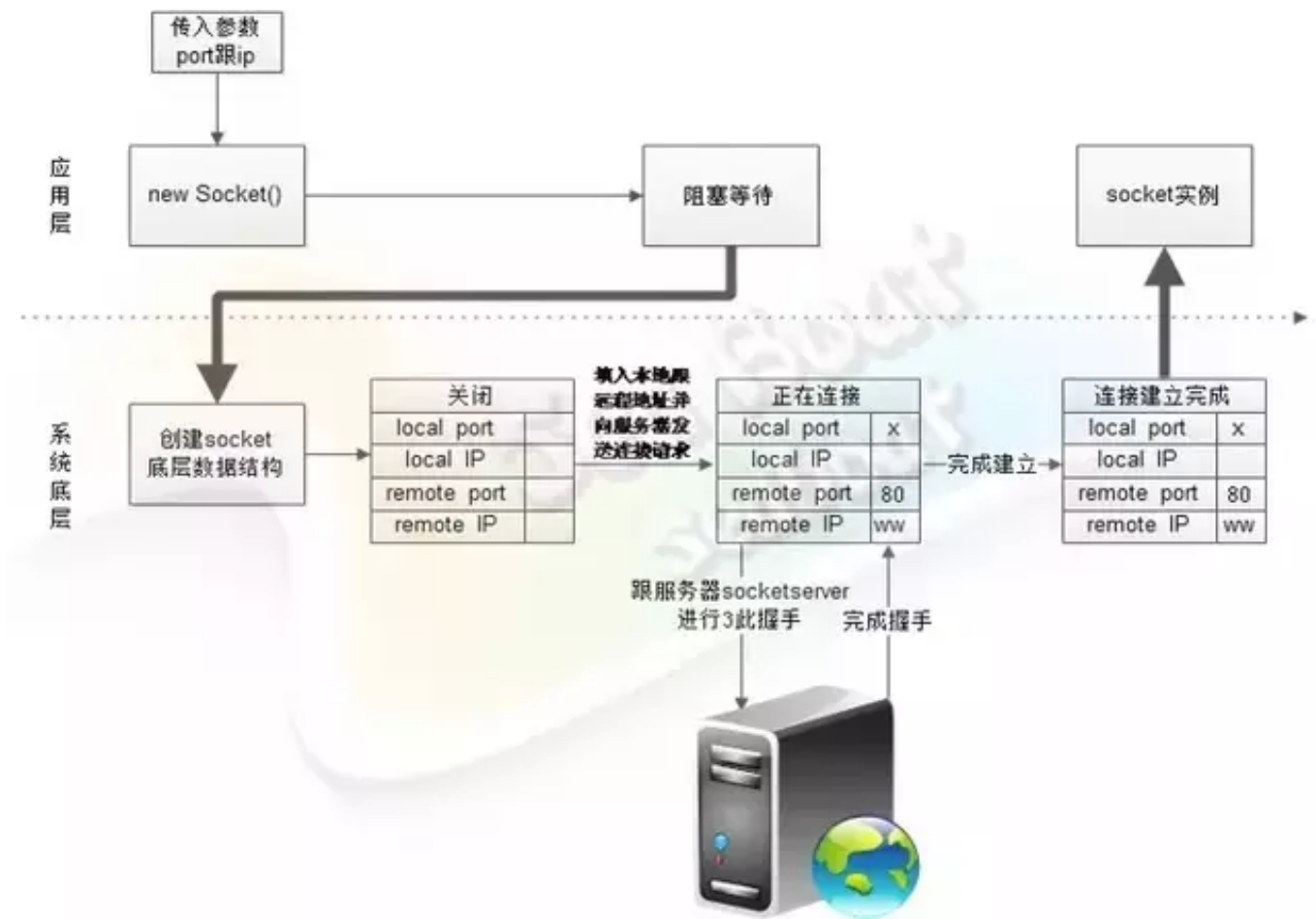
当应用层new ServerSocket 阻塞等待，操作系统会做一系列操作并监听客户端的访问。而当服务端接收到客户端连接时，就会创建一个socket数据结构并放到队列中，随后应用层的accept就会轮询获取客户端socket。



套接字通信

当客户端Socket在new Socket后阻塞等待，操作系统会负责发起对服务端的连接请求，直到完成三次握手，应用层才会解除等待。





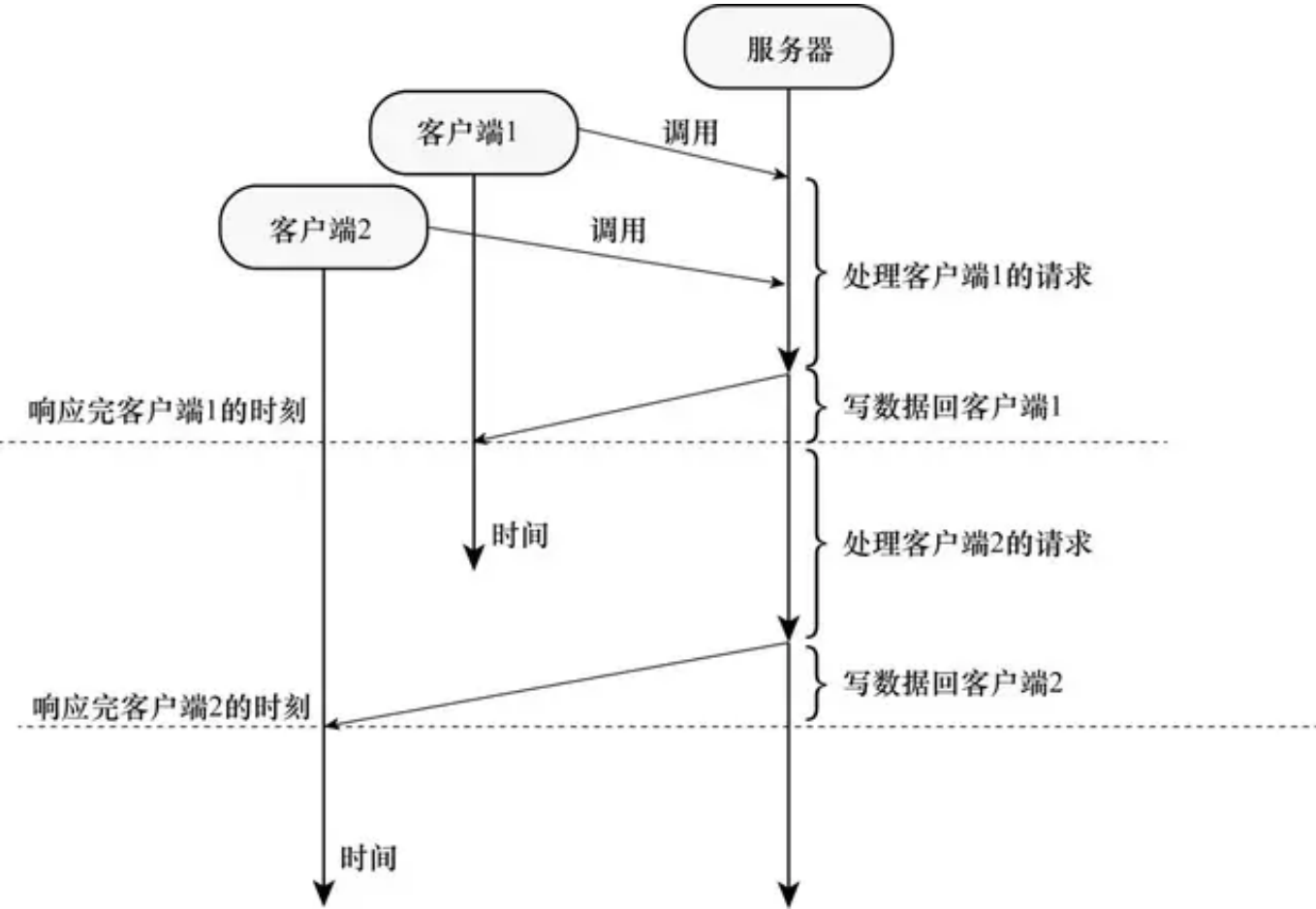
服务器模型

(1) 线程阻塞模式

单线程阻塞模式

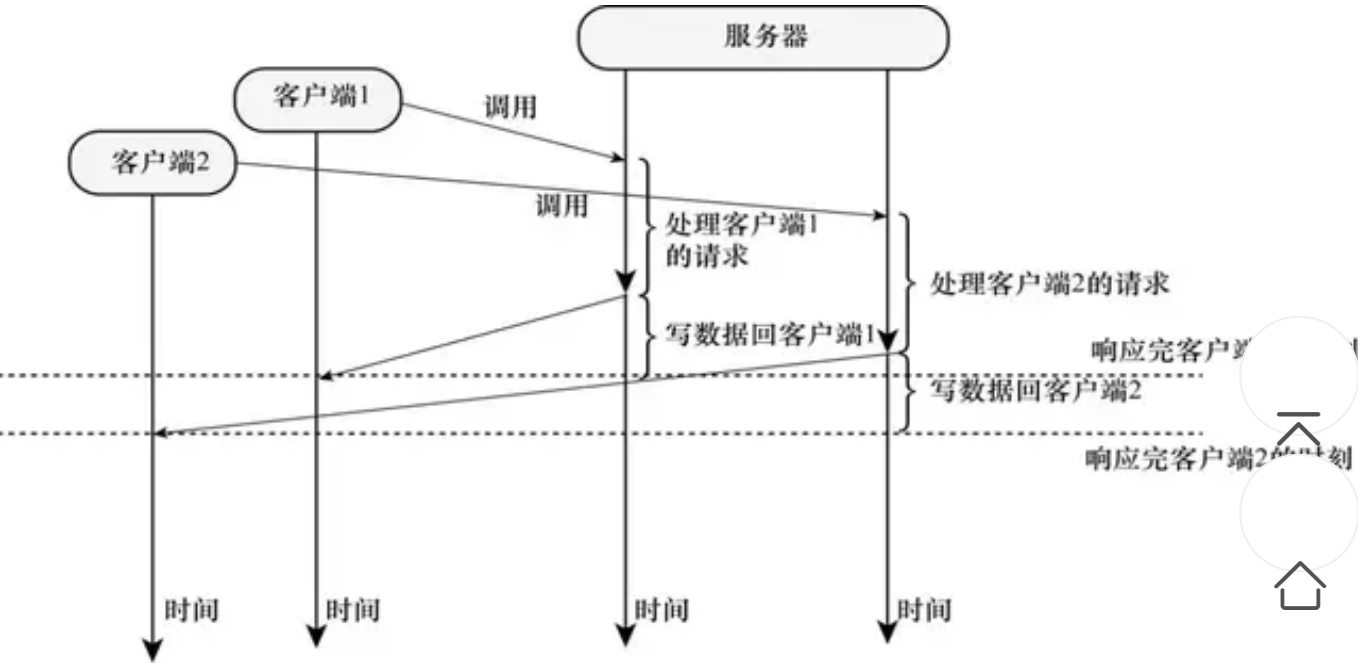
从单线程阻塞模式来看，共有两个客户端请求服务器，其中第二个客户端必须等到第一个客户端处理完成后才能开始处理。





多线程阻塞模式

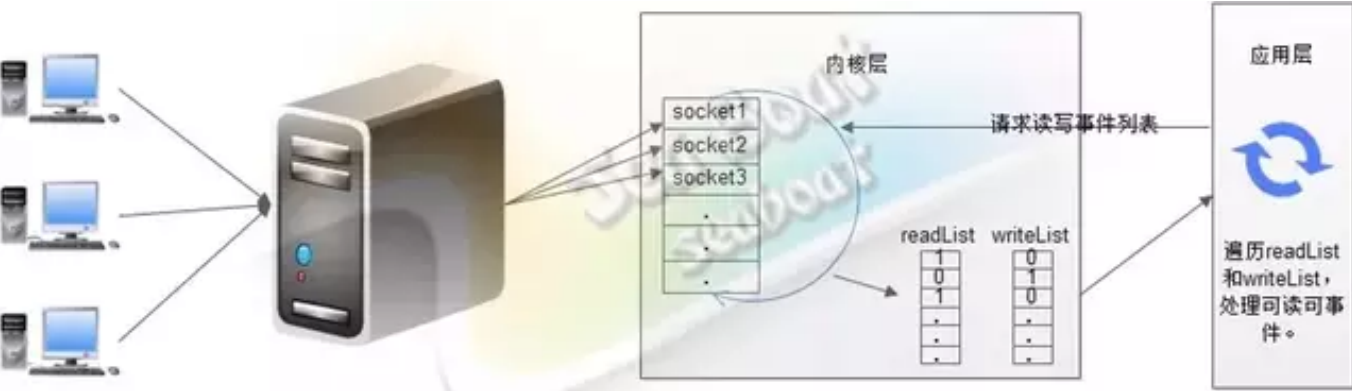
多线程阻塞模式同样也有两个客户端请求服务器，但此模式的第二个客户端不需要等到第一个客户端处理完，而是两个客户端并发的被处理。



单线程非阻塞模式



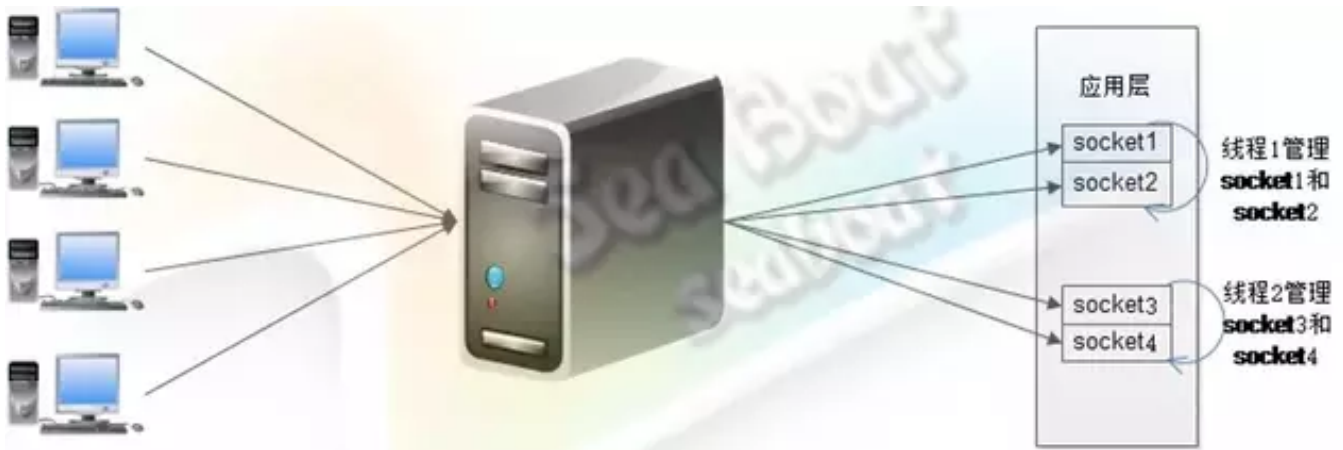
在单线程非阻塞模式中，服务器一个线程维护着多个客户端的请求，该线程不断遍历处理所有socket，尝试读写。基于事件监测模式，服务器会告诉操作系统需要关注的事件，接着操作系统负责检测所有客户端的连接并将检测到的事件放进两个列表中，最后，应用层只需要遍历这两个列表即可开始处理。



多线程非阻塞模式

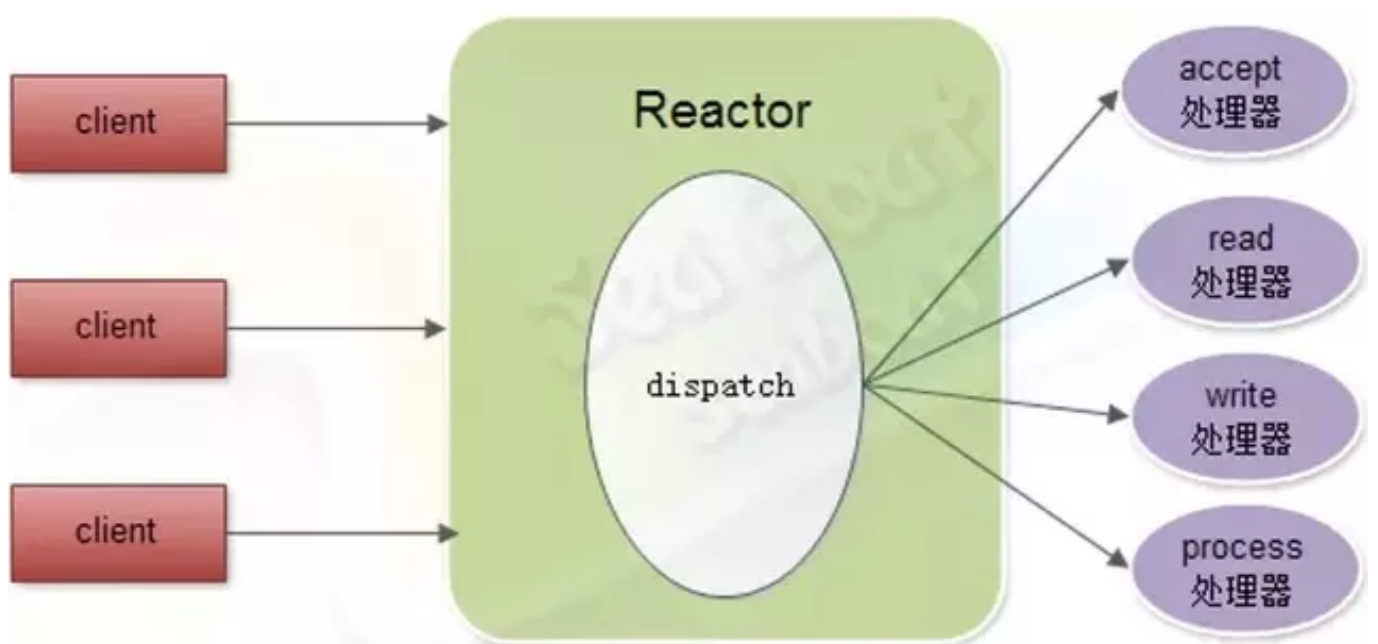
在多线程非阻塞模式中，服务器有多个线程共同负责多个客户端，客户端的连接会均匀分配给每个线程管理。





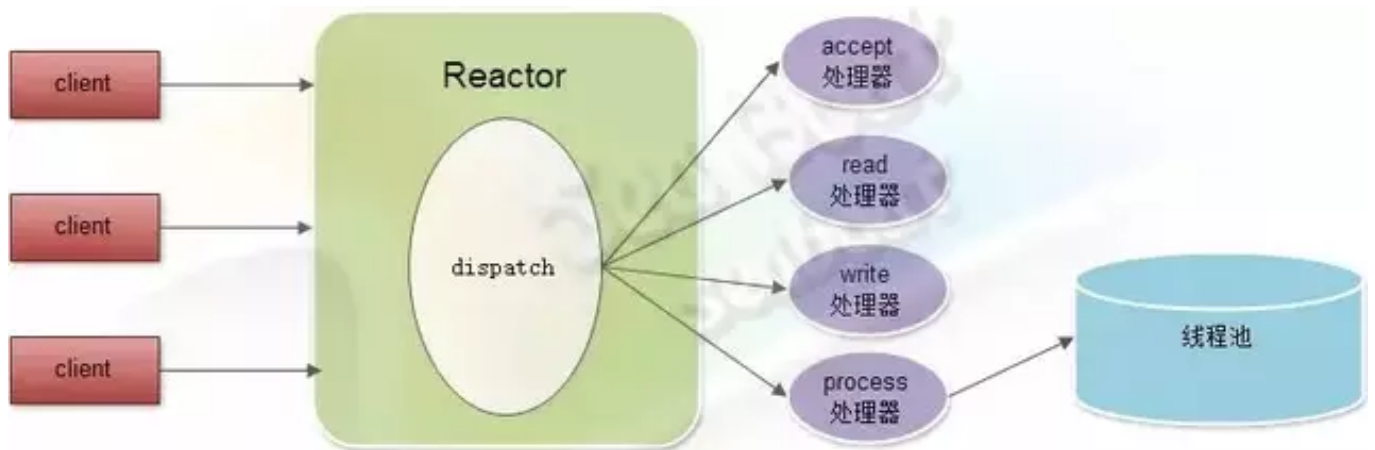
(2) Reactor模式

实际项目中，我们较常用的模式是Reactor模式。Reactor线程负责将客户端连接的不同事件分配到不同的处理器中进行处理，如 **accept** 处理器、**read** 处理器、**write** 处理器和 **process** 处理器。

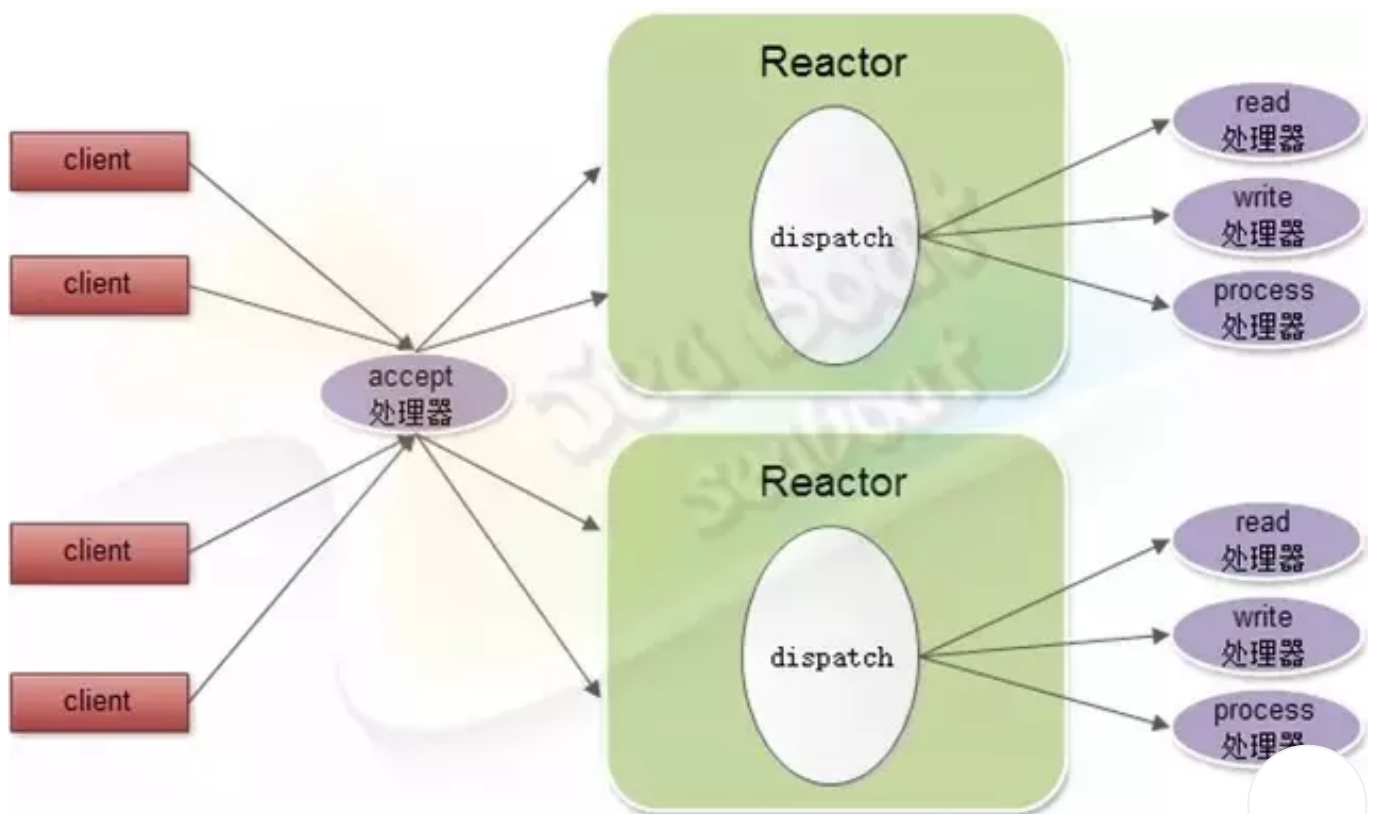


但事实上，Reactor模式有个不容忽视的缺点，比如处理耗时较长操作的处理器有可能会影响到整体的处理能力，需要在 **process** 处理器中引入一个线程池，并将比较耗时的操作放到线程池中处理，从而使得Reactor的整体运转正常状态。





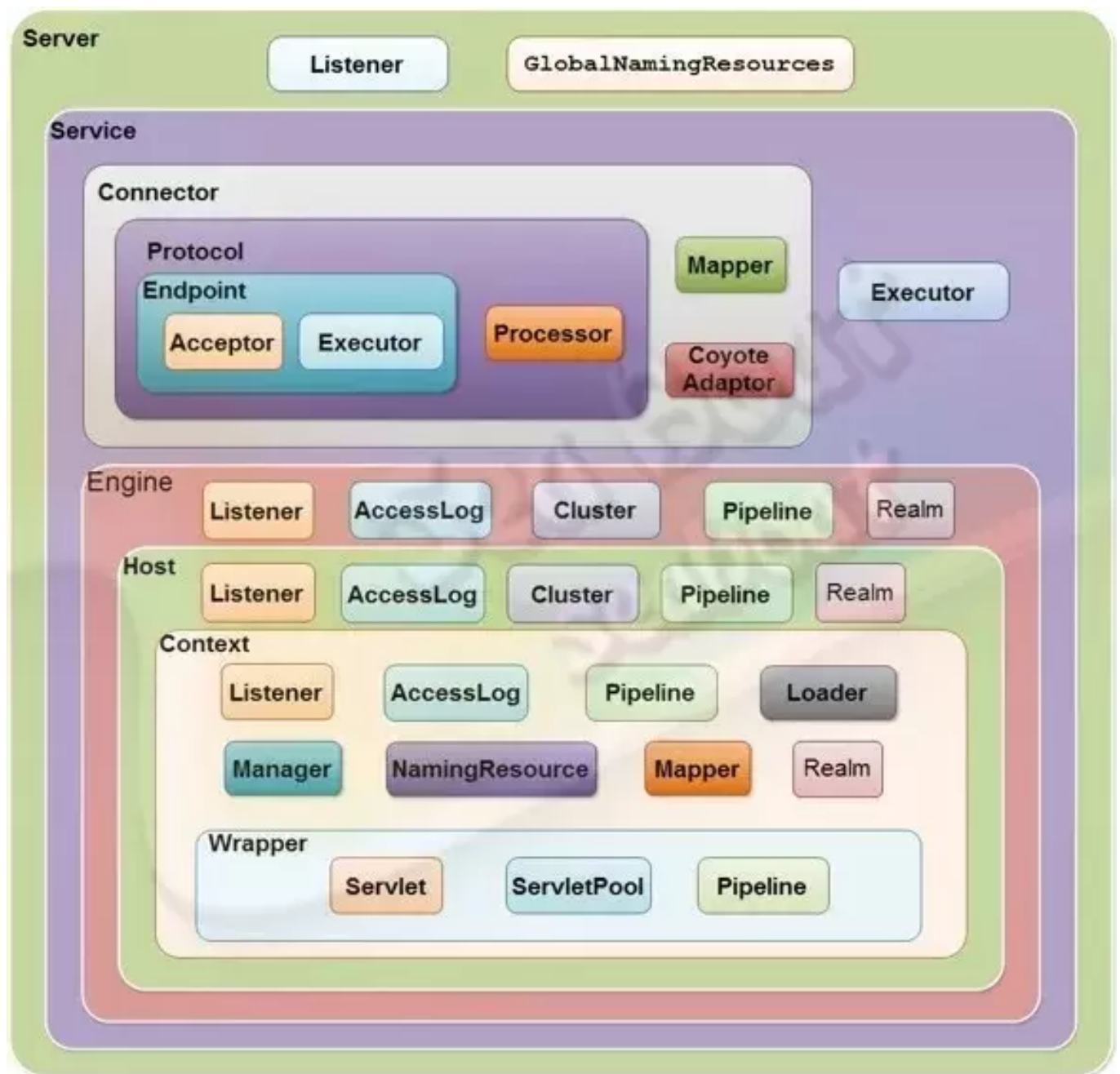
除此之外，还有一种改良的Reactor模式，即如果一个Reactor不够用，那么就创建多几个Reactor来同时处理。如下图，这里有两个Reactor对象，每个对象都有read处理器、write处理器和process处理器。而客户端连接的分发工作则共同由一个accept处理器完成，再均匀分配到不同的Reactor对象中。



整体结构

我们先来认识Tomcat的整体结构。它的顶层容器为Server，下面包括service、监听器和全局资源。Tomcat的主要对象

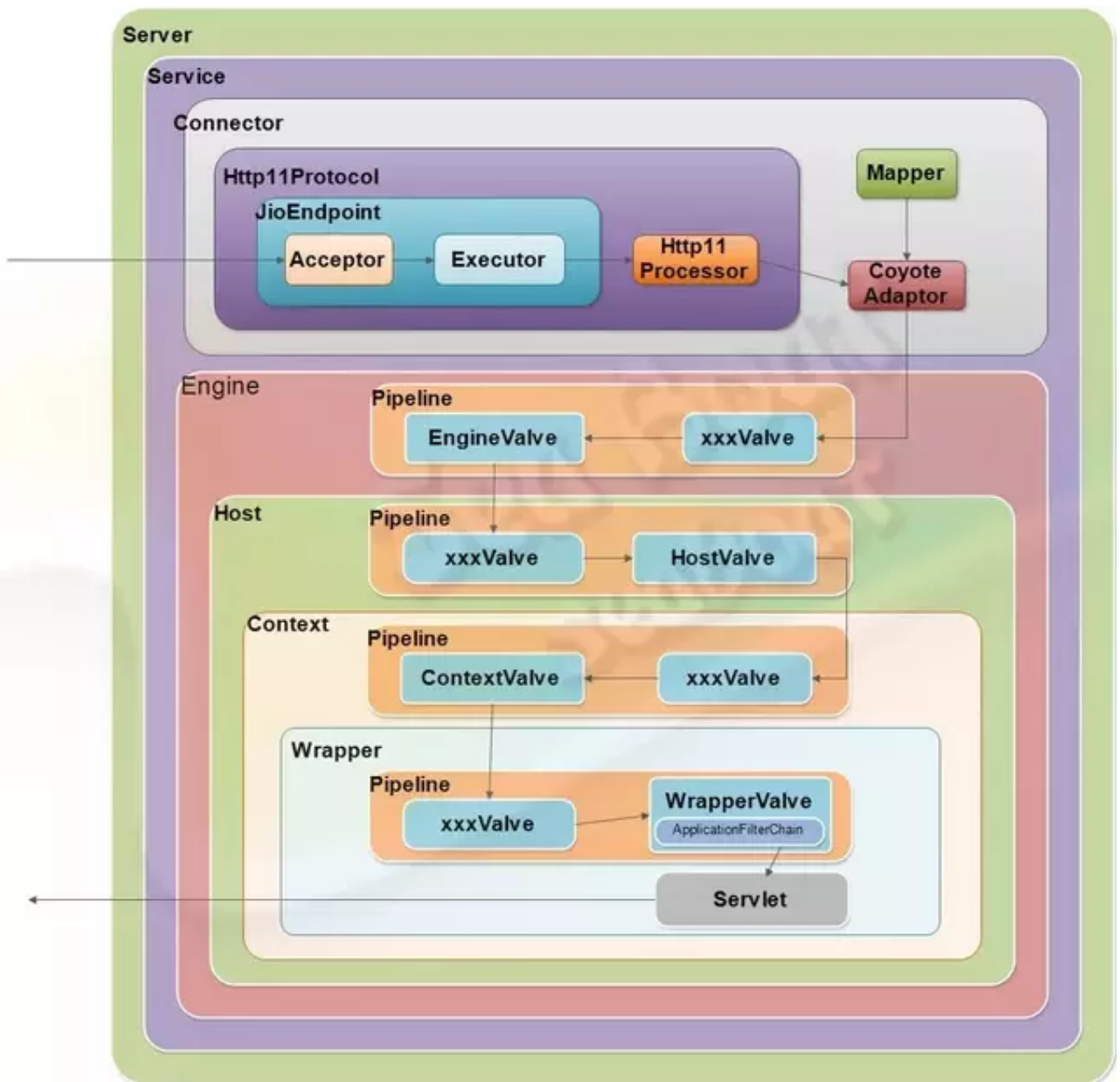
为 Connector（可有多 个）和 Container，其中每个 Connector 对应一个端口，用于处理不同的协议。



Container 包含了四个级别，分别为 Engine、Host、Context 和 Wrapper，其中 Engine 是全局的 Servlet 引擎，Host 是虚拟主机，Context 对应 web 应用，Wrapper 则对应 web 应用 servlet 对象。

请求处理过程

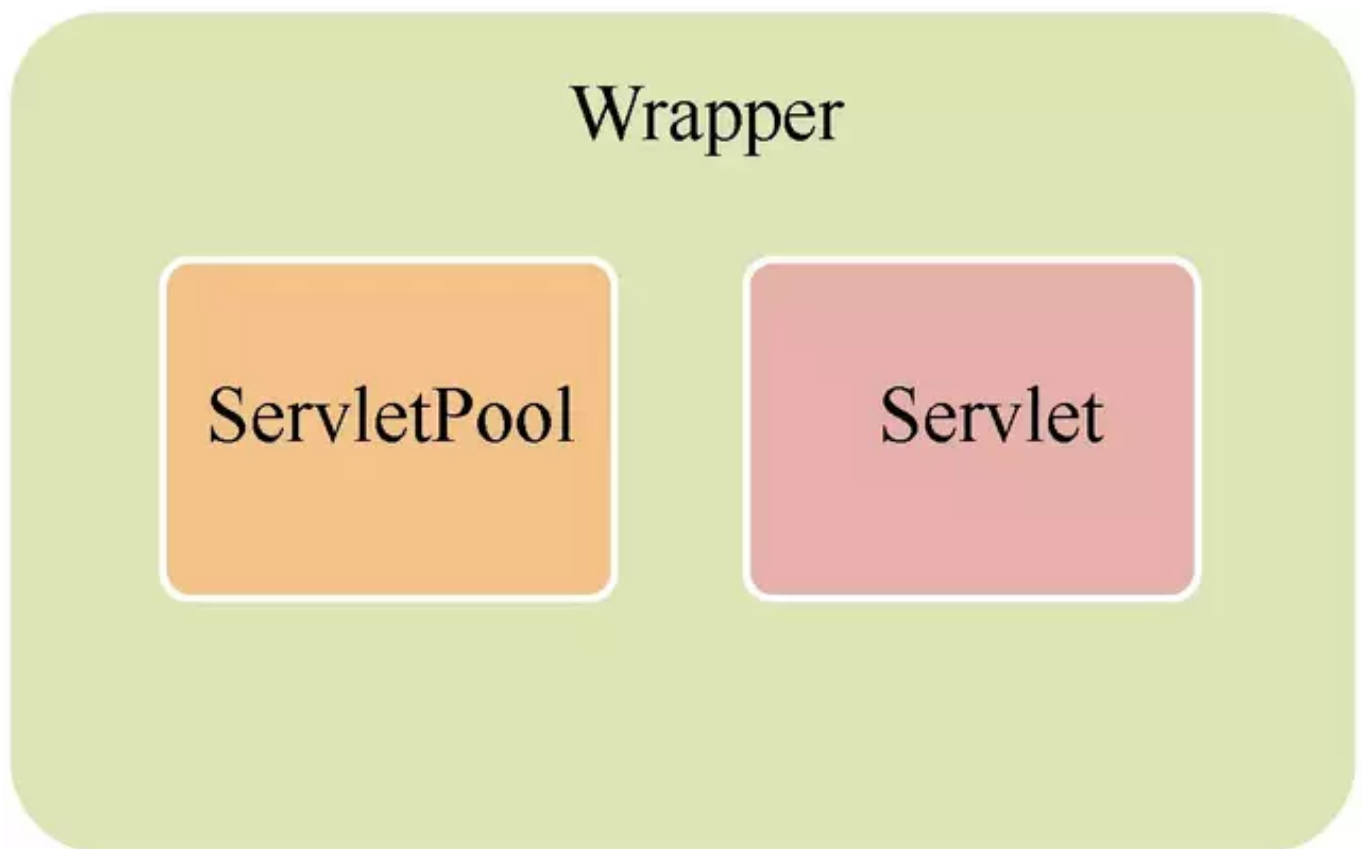




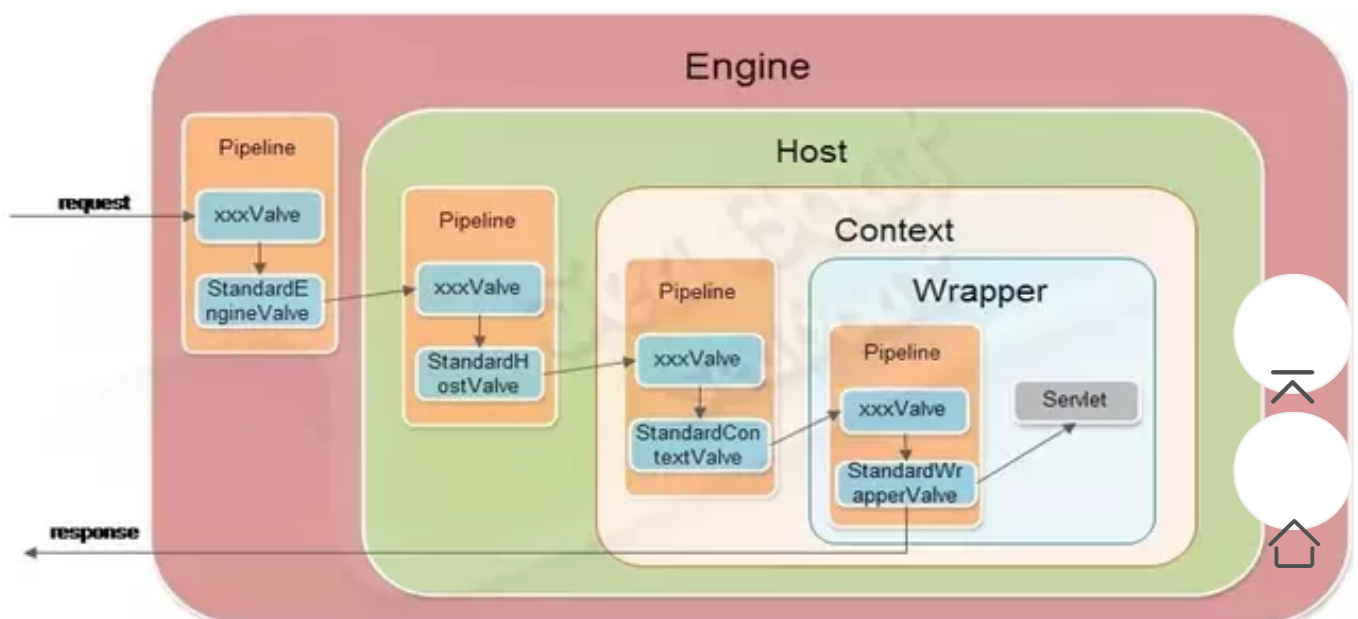
一个完整的请求处理过程是怎样的呢？如图，**Connector**启动后会由**JioEngdpoint**来负责接受客户端的请求连接，并在接收后交由任务池进行处理。该任务池会根据**Http11Processor**的逻辑（按照HTTP1.1协议）来对请求报文进行解析处理。接着，**CoyoteAdapter**适配器会适配到对应的**servlet**来进行业务逻辑处理。这一过程会经历四道管道，每个管道可能有若干个阀门，处理后最后将请求交给**Wrapper**容器的**servlet**来处理，并将响应报文返回到客户端，完成整个请求过程。

Servlet工作机制

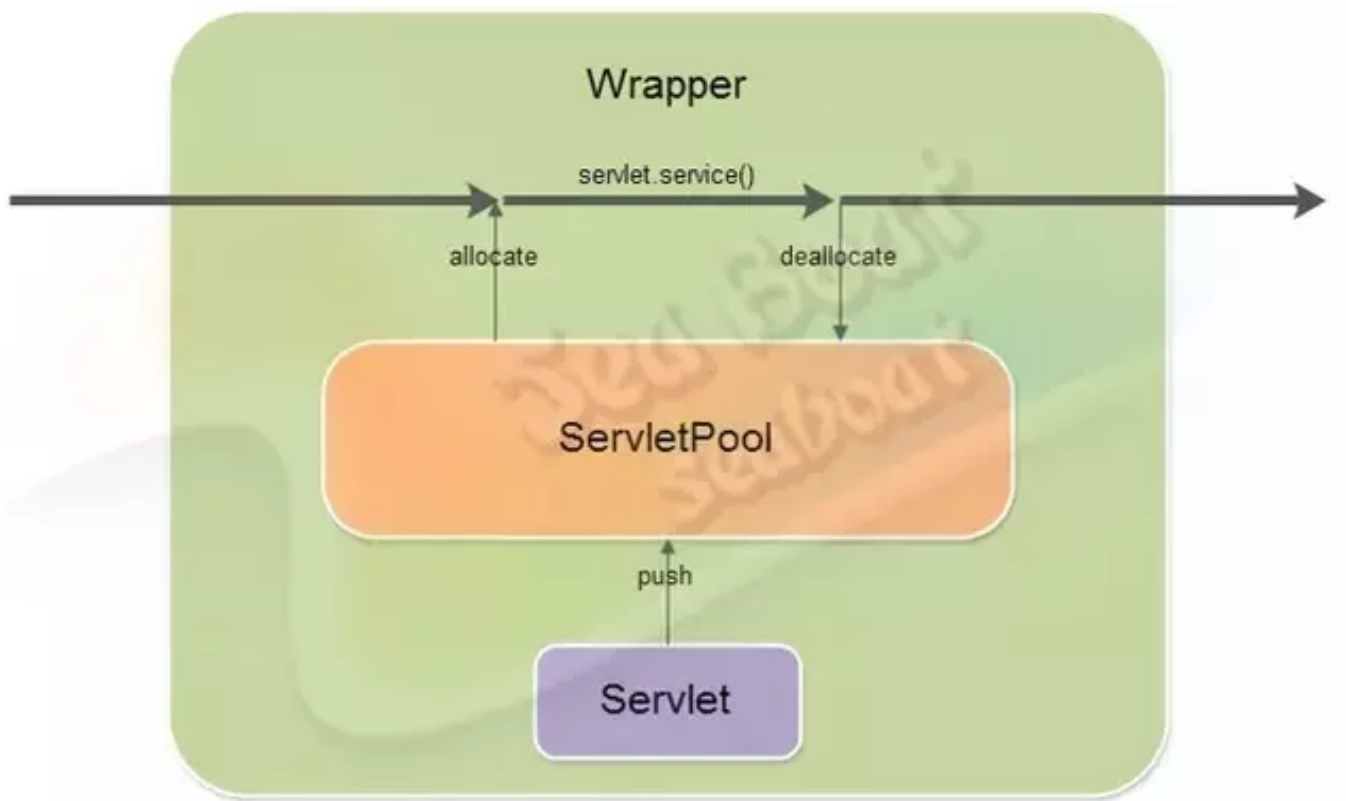
这里主要想说的是**Servlet**的非线程安全。正常的**Servlet**只有一个对象，而实现了指定接口的**Servlet**则会有一个**Servlet**对象池，该池默认的对象数是20。



前文已经简单提及**Servlet**的工作机制，就是通过四个级别的容器，通过管道一层层往下找到请求对应的**servlet**，执行完逻辑处理后将响应报文返回到客户端。

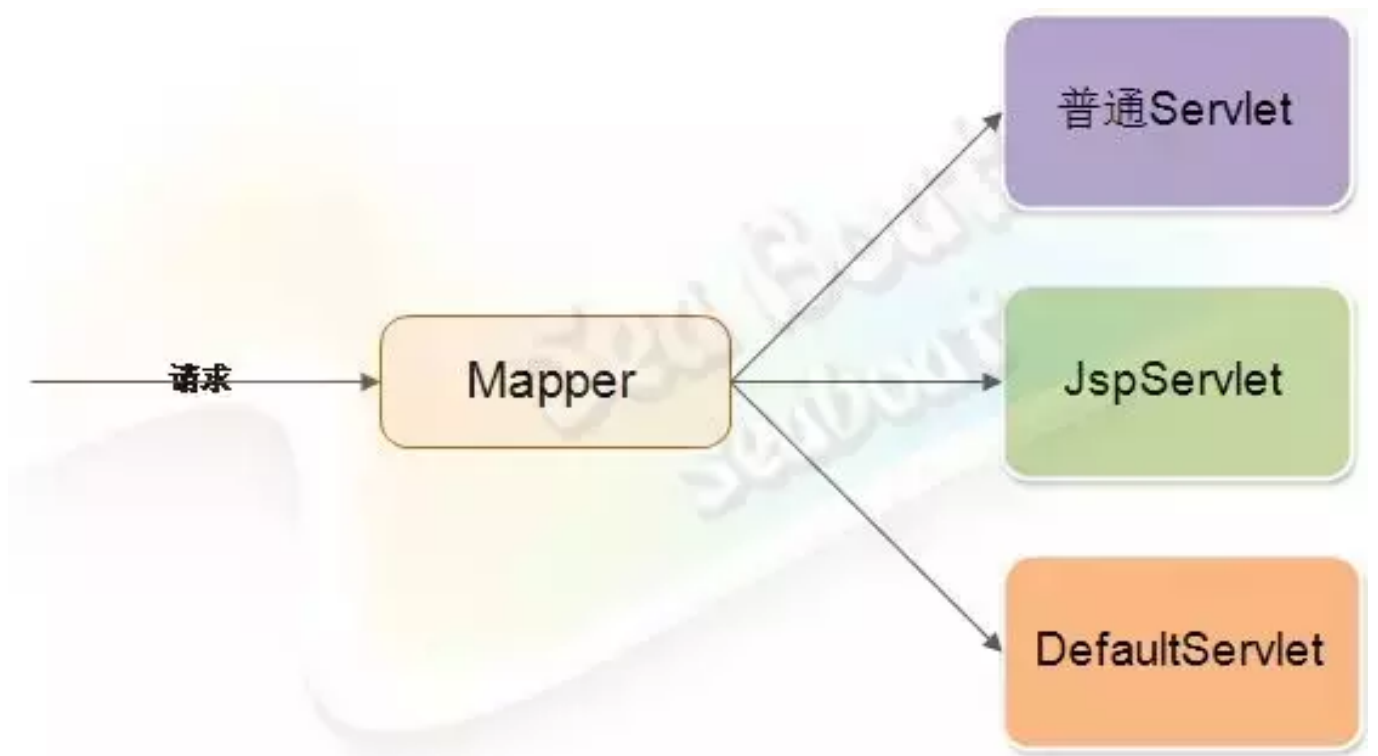


实现了SingleThreadModel接口的servlet则会在请求过程中先从Servlet池中allocate一个对象，使用完后再deallocate回池里，给其它线程使用。



根据请求资源的不同种类，可以把Servlet分成三种类别，比如普通Servlet、JspServlet和DefaultServlet。其中不同类别的请求资源会通过Mapper映射到对应类型的Servlet上。

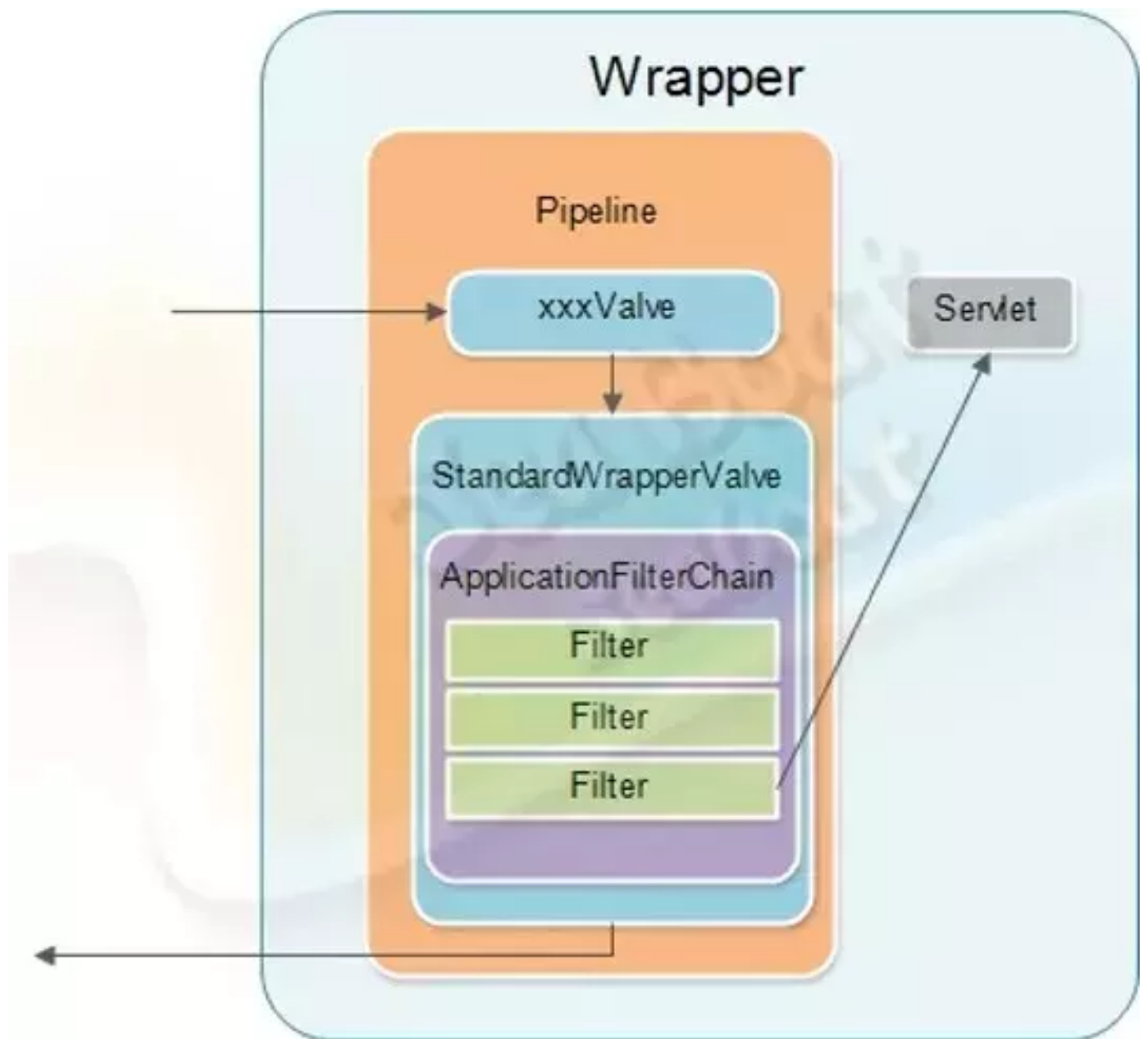




过滤连机制

这一处理过程中还有过滤连机制，即先通过不同的**filter**，最后才到**servlet**中。

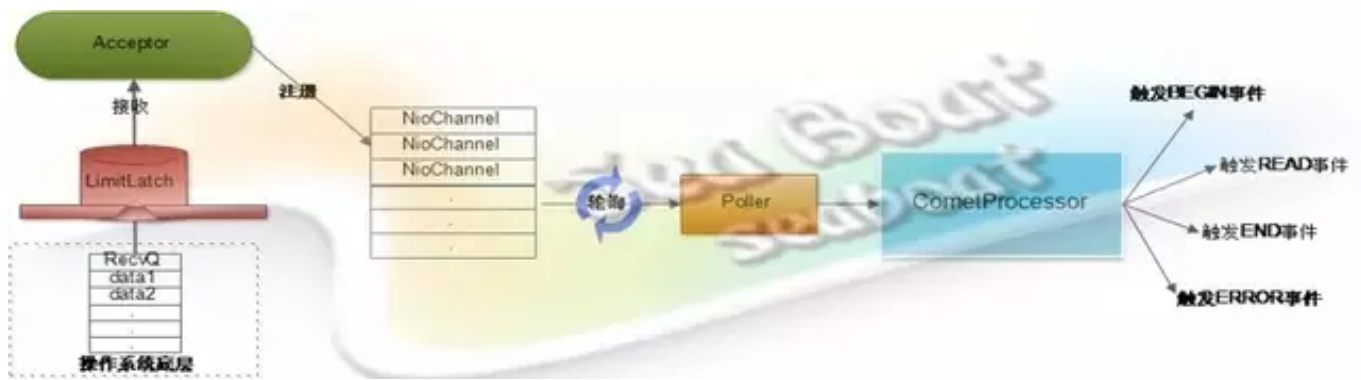




Comet模式

客户端发送一个请求到服务器，服务器接收后就将其注册到 **NioChannel** 队列中，随后 **Poller** 组件不断轮询是否有需要处理的 **NioChannel**。如果有需要处理的 **NioChannel**，那就调用前面实例化的 **Comet模式Servlet**。





这里主要用到CometProcessor借口的event方法，Poller会将对应的请求对象、响应对象和事件封装成CometEvent对象并传入event方法，随后执行event方法的逻辑，完成对不同事件的处理，从而实现Comet模式。

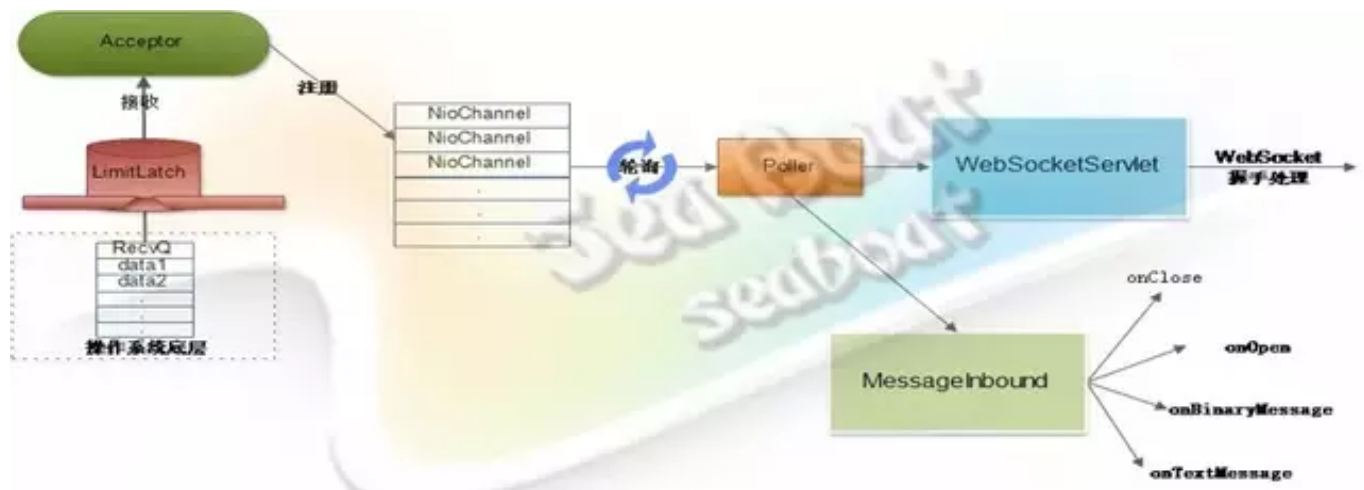
WebSocket模式

首先，客户端先发送一个“WebSocket协议升级”的握手包到服务器端；如果服务器端支持WebSocket协议，则会返回一个“升级确认”的握手包。这时就成功建立起了一条可双向通信的WebSocket连接，可以使用WebSocket协议的数据帧格式来发送消息。



当WebSocket的客户端连接被接收器接收并注册到NioChannel队列后，Poller组件不断轮询是否有需要处理的

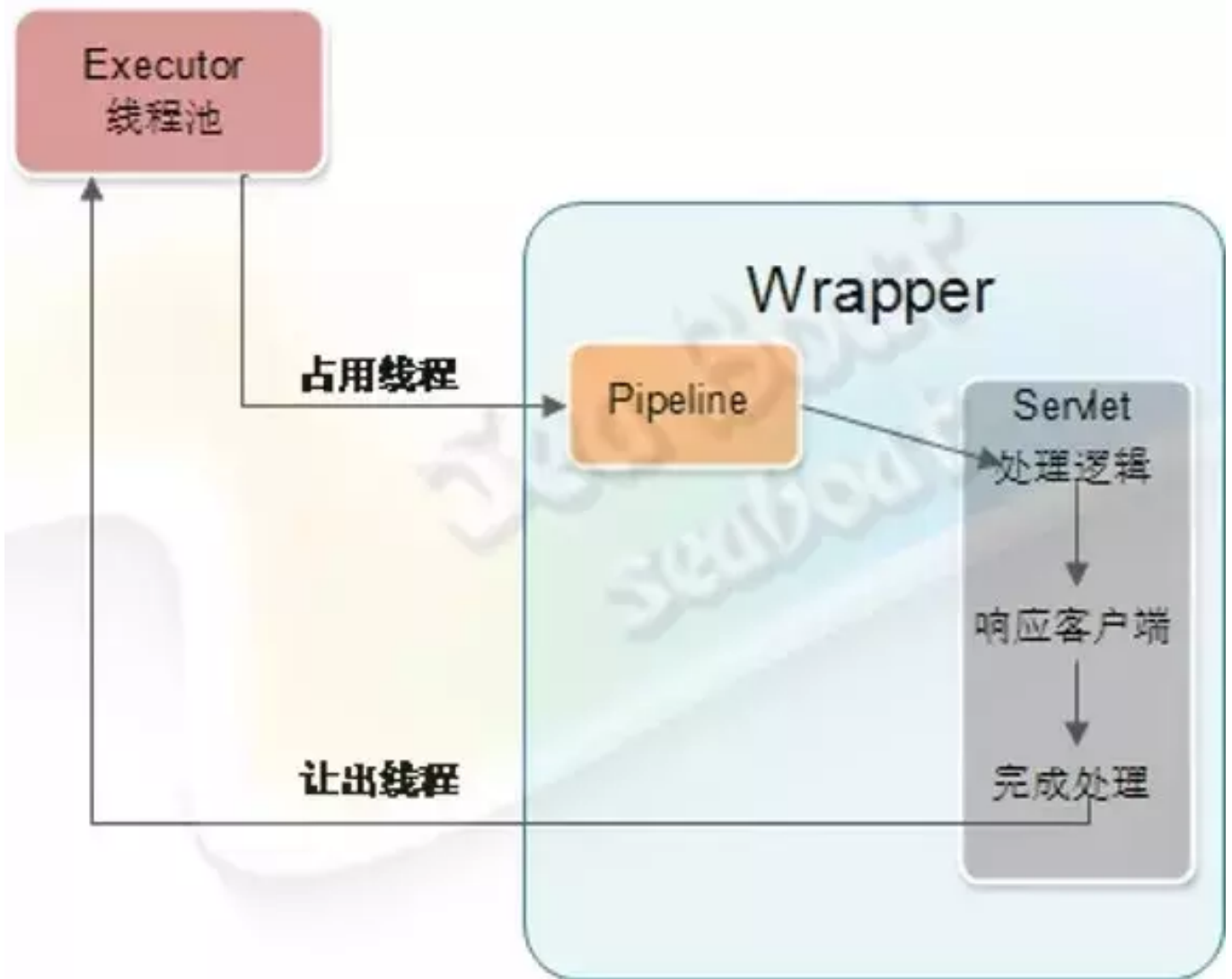
NioChannel。如果有，则经过处理管道后进入到继承了WebSocketServlet的Servlet上。WebSocketServlet的doGet方法会处理WebSocket握手，告知客户端同意升级协议。随后Poller继续轮询相关NioChannel，一旦发现使用WebSocket协议的管道，则会调用MessageInbound的相关方法，完成不同事件的处理，从而实现对WebSocket协议的支持。



同步Servlet

Servlet在同步情况下的处理过程，如图所示。



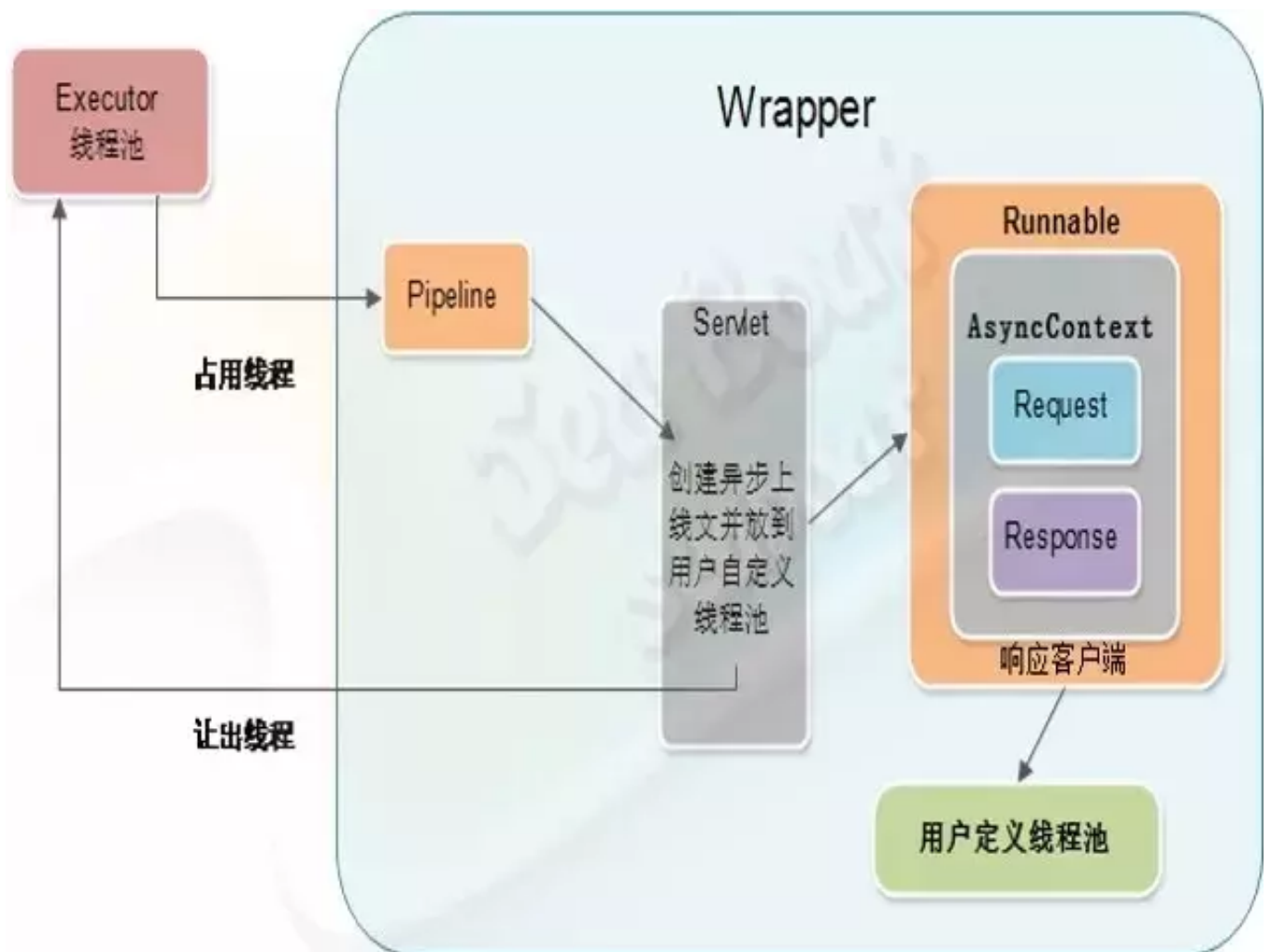


Tomcat的客户端请求由管道处理，最后会通过Wrapper容器的管道，这时它会调用Servlet实例的service方法进行逻辑处理，处理完后响应客户端。整个处理由Tomcat的Executor线程池的线程处理，而线程池的最大线程数是有限制的，所以这个处理过程越短，就能更快地将线程释放回线程池。但如果Servlet中的处理逻辑耗时越长，就会导致长期地占用Tomcat的处理线程池，最终影响Tomcat的整体处理能力。

异步Servlet

为了解决上面的问题，我们可以引入支持异步的Servlet，如下图所示。





同样，当客户端请求到来时，首先通过管道，然后进入到Wrapper容器的管道，再调用Servlet实例的service后，创建一个异步Servlet将耗时的逻辑操作封装起来，交给用户自己定义的线程池。这样就可以避免因Servlet中的处理逻辑耗时长而影响Tomcat的整体处理能力。

二、分布式集群为什么要使用集群？

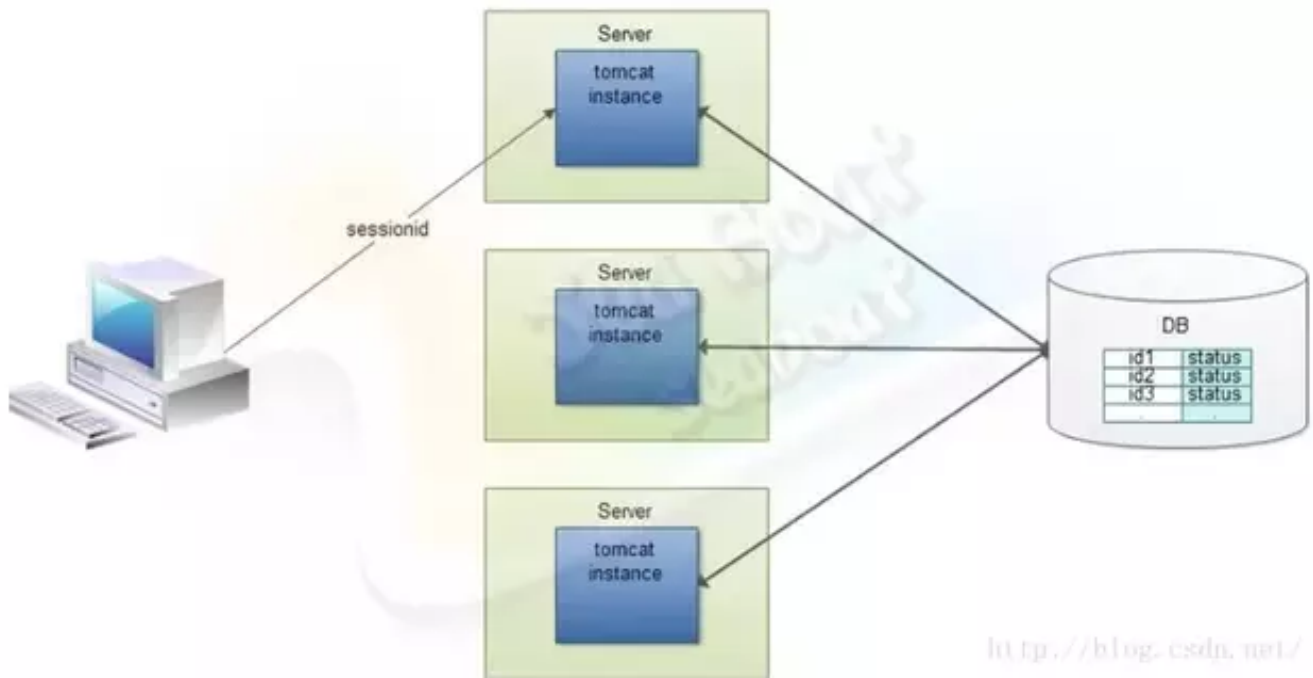
这主要有两个原因：

一是对于一些核心系统要求长期不能中断服务，为了提高可用性我们需要由多台机器组成的集群；

二是随着访问量越来越大且业务逻辑越来越复杂，单台的处理能力已经不足以处理如此多且复杂的逻辑，于是需要增加若干台机器使整个服务处理能力得到提升。

集群难在哪？

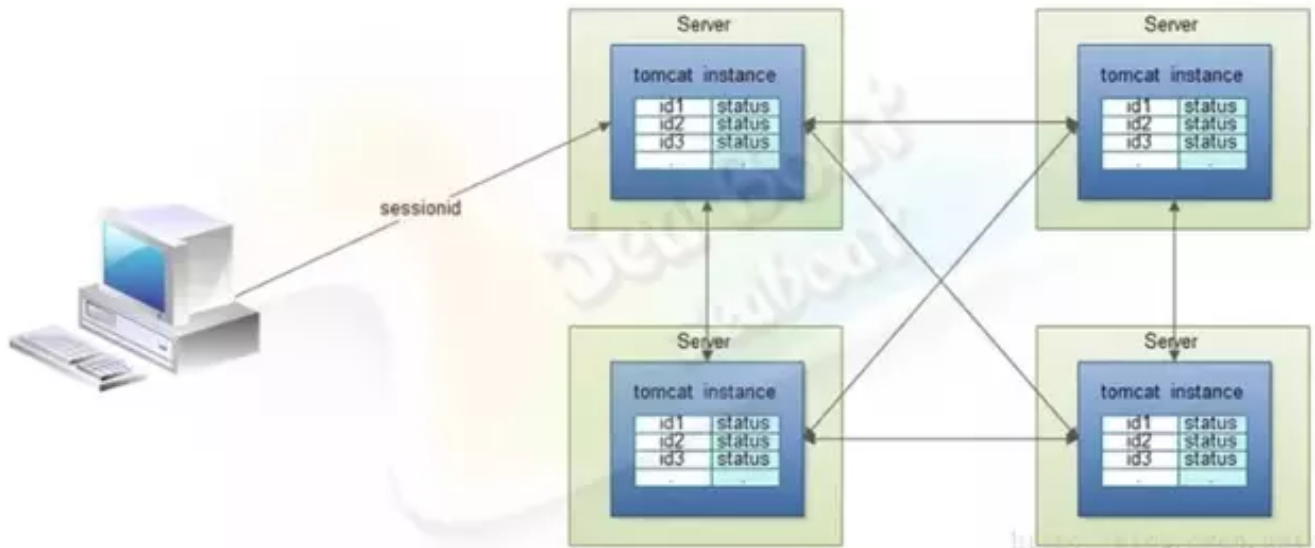
如果没有状态，那么做集群很简单，直接堆机器即可，请求不管到哪个节点上都能正确处理。但在有状态的情况下，则需在对应节点能获取该客户端对应的会话信息后才能正确处理，最简单的处理方法就是将会话信息放到**DB**，所有节点都从**DB**去拿客户端会话信息。



全节点会话同步模型

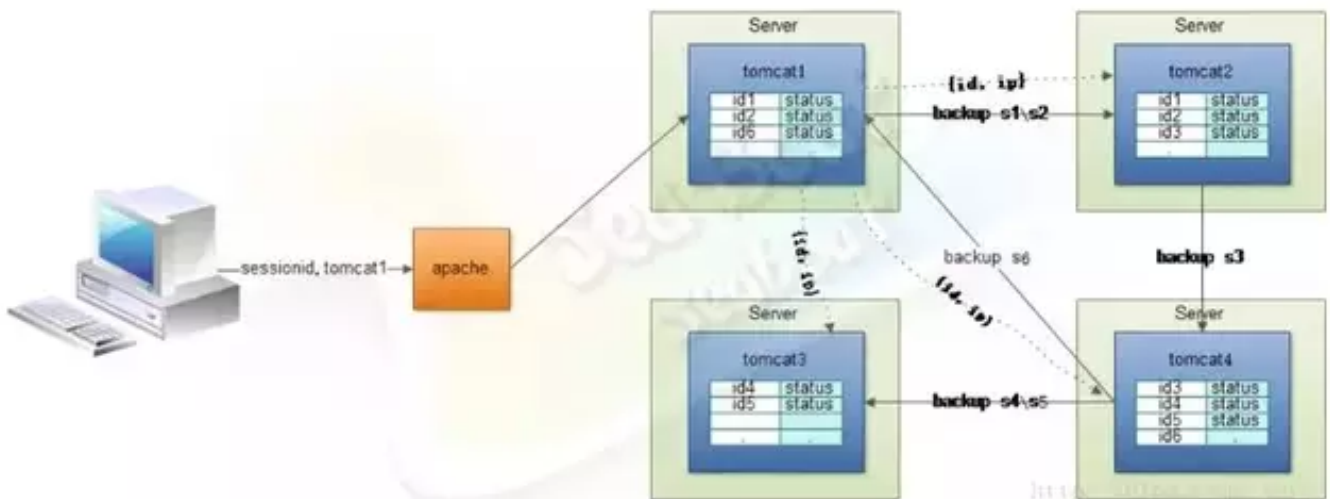
全节点会话同步模型可在服务端所有的节点之间分享所有会话信息，而每个节点都包含了所有客户端的会话信息，可以保障服务端能准确获取到客户端的会话信息并正确处理。但全节点会话同步模型也可能会引入网络堵塞的风险。





会话备份单节点模型

一个请求经由**Apache**分发到**Tomcat**集群中某个节点，再生成会话信息。这些会话信息可以通过一定的备份机制，只将信息都同步在某一个节点上，而不是同步到所有节点，这样大大减少了网络开销，能有效避免网络阻塞。



生产部署选型

1、较小的应用可直接用**Tomcat**内置的会话共享方案

对于全节点会话同步模型

此种方案在实际生产上推荐的集群节点个数为**3-6**个，它无法组建更大的集群，而且冗余了大量数据，利用率低。

对于会话备份模型

此种方案在实际生产上推荐的集群节点个数可达到**10**个以上。

2、较大的应用一般会把会话剥离出来放到缓存集群中

Redis

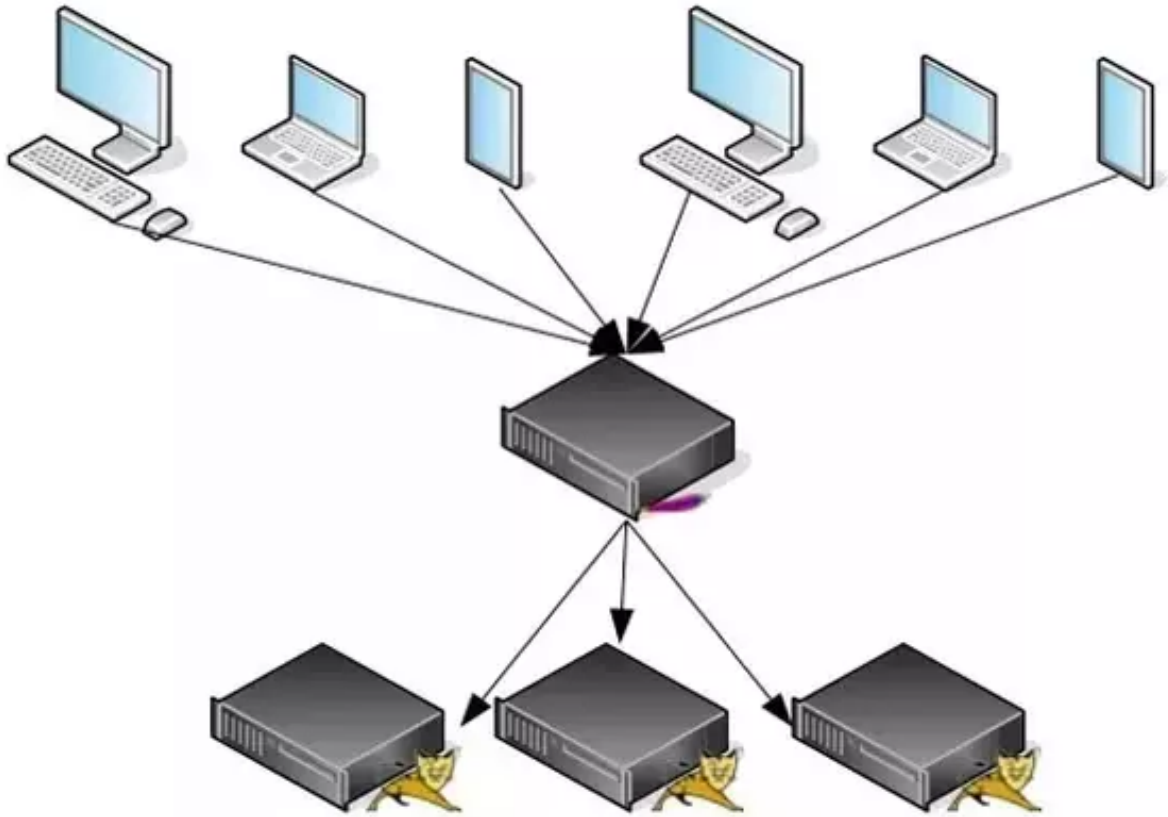
memcached

这两者都有相关的**jar**包，便于集成。

部署

常见的部署方式如下图，通过一个负载均衡器拖若干个**Tomcat**节点，前端不同的客户端通过访问负载均衡来访问**Tomcat**。





反向代理

常见的负载均衡器可分为软件和硬件。硬件包括F5、A10、Cisco等，软件包括Nginx、Apache httpd、Lighttpd、Squid等。

三、生产部署关键参数JVM设置

由于Tomcat也是运行在JVM上，所以JVM也有一些参数需要设置，加上-server参数，java堆初始化和最大值，默认是1/64物理内存和1/4物理内存，一般不超过物理内存的80%，且这两个最好设置成一样，够用就好，太高会导致浪费内存和GC回收周期长。其它参数如下所示。

一般使用HotSpot JVM

加上-server



-Xms/-Xmx:设置java堆初始化和最大值，默认为1/64物理内存和1/4物理内存一般不超过物理内存的80%，且这两个最好设置成一样，够用就好，太高会导致浪费内存和GC回收周期长。

-XX:NewSize/-XX:NewRatio：设置成25%-33%java堆总量，太高太低都会导致无效GC。

-XX:PermSize/-XX:MaxPermSize：非堆内内存初始值最大值分别设为128M，256M。

-XX:+AggressiveOpts：使用最新优化技术。


参考oracle官网


<http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html>，还有其它参数可根据实际配置。

集群设置

负载均衡用四层还是七层，根据实际情况选择。其中：

四层七层：四层不认识http协议，只按照客户端ip和port分配流量，但性能好；七层认识http协议，可用http某些头部分配流量，由于需要计算，性能相对差点。

连接池：负载均衡器到tomcat的连接数，一般小于等于tomcat集群节点处理连接能力之和。例如集群有4个节点每个tomcat预计处理500个链接，那么连接池的长连接，大设为2000。

全节点复制（DeltaManager）模式集群节点数3-6为宜。

主备复制（**BackupMnagager**）模式集群节点可到10台以上。

设置

一共有三种模式：

JAVA BIO，最原始最稳定的堵塞模式，也是tomcat7之前的默认模式。它支持较小的并发处理，高并发且短连接的处理也可以为首选。**BIO**模式下有一个非常重要的参数：**maxThreads**，它表示最大同时处理请求书，一般范围为200-800，看从400开始根据实际调节。如果是**CPU**密集型的应用可以减少，而非**CPU**密集型的应用可以增加。

JAVA NIO，是tomcat8的后默认模式，能支持发并发多连接处理，属于非堵塞模式。

Native APR，为提高性能而使用本地代码的一种非堵塞模式，由**C++**编写，支持更大并发处理。

四、性能监控和分析步骤

性能调优是不断找瓶颈的动态过程，包括：

确定应用的性能指标

搞清楚应用的系统架构

测试目前应用的性能参数

分析性能问题找到瓶颈

解决优化瓶颈



不断重复上述几步直到满足性能指标

分析Connector

Tomcat性能相关因素有很多，一般包括网络网卡、TCP连接参数、HTTP长短连接、SSL、BIO&NIO、Connector自身参数、负载均衡的选择和负载均衡参数等。分析性能瓶颈应该考虑如上多个相关因素。

JVM分析

在JVM分析上，我们要关注Java堆内存、直接内存、永久代、GC、线程栈、本地代码和TCP缓存等。

常用分析工具

Jmeter 压测：得到并发数、TPS、响应时间等

Druid 自带：SQL耗时、池使用率

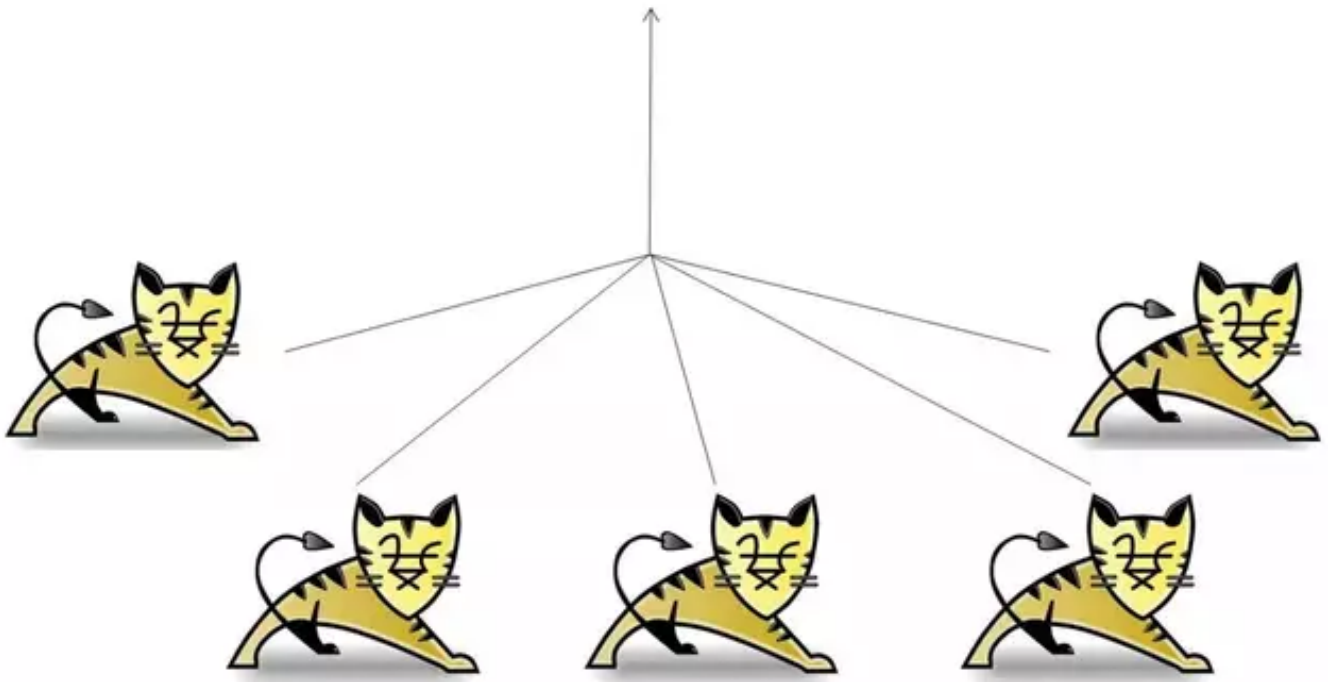
JVM自带：JPS、jinfo、jstat、jmap、jstack等

Linux监控：CPU、内存、磁盘io、网卡、swap等

常用工具：top、tail、grep、iotop、iftop等

整体压测





单个**Tomcat**压测并调优后就对整个集群进行集体压测，关键是看性能能否基本满足线性增长。

问答环节

【问题1】如果使用**Tomcat**部署了一个**Webserver**接口，将一个应用通过接口反复调用同步数据后发现：数据量达到1万多条数据后，数据的同步时间变得越来越长，甚至达到1个小时。请问一般是什么达到瓶颈才导致这样的结构，又该如何调优？

答：如果想调优，建议你先通过工具找出问题在哪，比如数据同步时间长时，电脑的内存、**CPU**、**JVM**分别是什么情况（是否正常），再把堆栈打出来。调优没有固定的公式，只能用工具来不断找瓶颈，进而完成调优。

【追问】如果内存，**CPU**方面都很正常呢？

答：那就检查其它一些参数，比如网卡、磁盘**IO**等，建议你使用一套全面的监控系统。



【问题2】 如果有一个HTTP请求在自己的应用过滤器出错了，请问这个请求到应用的servlontroller了吗？

答：异常catch住了是能到的。

【问题3】 请问如何分析非堆内存占用过多的情况？

答：直接内存不归JVM管理，建议用火焰图来分析。

【问题4】 Java进程异常终止但没有任何日志，请问如何分析这种情况？

答：这种情况一般是因操作系统资源吃紧被kill掉而产生的。你可以检查操作系统级别的日志。

【问题5】 能否给个生产的参数配置？

答：我的博客里面有生产的参数配置，可以仔细看看，地址：
<http://blog.csdn.net/wangyangzhizhou/article/details/50359012>

【问题6】 操作中发现异步日志，Tomcat 脚本语法停止，只能杀进程，这个问题该如何处理？

答：这应该是Web应用new了非daemon线程而该线程一直在运行不结束，所以导致shutdown脚本关不了。建议你打印出执行栈，找出出问题的线程，再将其设置为deamo

直播链接

<https://m.qlchat.com/topic/details?topicId=2000000607962364>



