架构师面试题 - Tomcat 面试专题

目录

1、Tomcat 的缺省端口是多少，怎么修改？

1）找到 Tomcat 目录下的 conf 文件夹
2）进入 conf 文件夹里面找到 server.xml 文件
3）打开 server.xml 文件
4）在 server.xml 文件里面找到下列信息

```
<!--
<Connector connectionTimeout="20000" port="8000"
```

```
protocol="HTTP/1.1" redirectPort="8443" uriEncoding="utf-8"/>
 -->
```

2、tomcat 有哪几种 Connector 运行模式(优化)？

bio：传统的 Java I/O 操作，同步且阻塞 IO。maxThreads=”150”//Tomcat 使用线程来处理接收的每个请求。这个值表示 Tomcat 可创建的最大的线程数。默认值 200。可以根据机器的时期性能和内存大小调整，一般可以在 400-500。最大可以在 800 左右。minSpareThreads=”25”—Tomcat 初始化时创建的线程数。默认值 4。如果当前没有空闲线程，且没有超过 maxThreads，一次性创建的空闲线程数量。Tomcat 初始化时创建的线程数量也由此值设置。maxSpareThreads=”75” - 一旦创建的线程超过这个值，Tomcat 就会关闭不再需要的 socket 线程。默认值 50。一旦创建的线程超过此数值，Tomcat 会关闭不再需要的线程。线程数可以大致上用 “同时在线人数每秒用户操作次数系统平均操作时间” 来计算。acceptCount=”100”——指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理。默认值 10。如果当前可用线程数为 0，则将请求放入处理队列中。这个值限定了请求队列的大小，超过这个数值的请求将不予处理。connectionTimeout=”20000” - 网络连接超时，默认值 20000，单位：毫秒。设置为 0 表示永不超时，这样设置有隐患的。通常可设置为 30000 毫秒。nio：JDK1.4 开始支持，同步阻塞或同步非阻塞 IO。指定使用 NIO 模型来接受 HTTP 请求 protocol=”org.apache.coyote.http11.Http11NioProtocol” 指定使用 NIO 模型来接受 HTTP 请求。默认是 BlockingIO，配置为 protocol=”HTTP/1.1” acceptorThreadCount=”2” 使用 NIO 模型时接收线程的数目 aio(nio.2)：JDK7 开始支持，异步非阻塞 IO。apr：Tomcat 将以 JNI 的形式调用 Apache HTTP 服务器的核心动态链接库来处理文件读取或网络传输操作，从而大大地提高 Tomcat 对静态文件的处理性能。

```
 <!-- protocol 启用 nio 模式，(tomcat8 默认使用的是 nio)(apr 模式利用系统级异步 io) -->
 <!-- minProcessors 最小空闲连接线程数-->
 <!-- maxProcessors 最大连接线程数-->
 <!-- acceptCount 允许的最大连接数，应大于等于 maxProcessors-->
 <!-- enableLookups 如果为 true,requst.getRemoteHost 会执行 DNS 查找，反向解析 ip 对应域名或主机名-->
 <Connector port="8080"
protocol="org.apache.coyote.http11.Http11NioProtocol"
connectionTimeout="20000" redirectPort="8443 maxThreads=“500”
minSpareThreads=“100” maxSpareThreads=“200” acceptCount="200"
enableLookups="false"/>
```

其他配置 maxHttpHeaderSize="8192" http 请求头信息的最大程度，超过此长度的部分不予处理。一般 8K。URIEncoding="UTF-8" 指定 Tomcat 容器的 URL 编码格式。disableUploadTimeout="true" 上传时是否使用超时机制 enableLookups="false"--是否反查域名，默认值为 true。为了提高处理能力，应设置为 false compression="on" 打开压缩功能 compressionMinSize="10240" 启用压缩的输出内容大小，默认为 2KB noCompressionUserAgents="gozilla, traviata" 对于以下的浏览器，不启用压缩 compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain"

3、Tomcat 有几种部署方式？

1）直接把 Web 项目放在 webapps 下，Tomcat 会自动将其部署
2）在 server.xml 文件上配置 节点，设置相关的属性即可
3）通过 Catalina 来进行配置:进入到 conf\Catalina\localhost 文件下，创建一个 xml 文件，该文件的名字就是站点的名字。编写 XML 的方式来进行设置。

4、tomcat 容器是如何创建 servlet 类实例？用到了什么原理？

当容器启动时，会读取在 webapps 目录下所有的 web 应用中的 web.xml 文件，然后对 xml 文件进行解析，并读取 servlet 注册信息。然后，将每个应用中注册的 servlet 类都进行加载，并通过反射的方式实例化。（有时候也是在第一次请求时实例化）在 servlet 注册时加上如果为正数，则在一开始就实例化，如果不写或为负数，则第一次请求实例化。

5.tomcat 如何优化？

1、优化连接配置.这里以 tomcat7 的参数配置为例，需要修改 conf/server.xml 文件，修改连接数，关闭客户端 dns 查询。参数解释：URIEncoding="UTF-8" :使得 tomcat 可以解析含有中文名的文件的 url，真方便，不像 apache 里还有搞个 mod_encoding，还要手工编译 maxSpareThreads : 如果空闲状态的线程数多于设置的数目，则将这些线程中止，减少这个池中的线程总数。minSpareThreads : 最小备用线程数，tomcat 启动时的初始化的线程数。enableLookups : 这个功效和 Apache 中的 HostnameLookups 一样，设为关闭。connectionTimeout : connectionTimeout 为网络连接超时时间毫秒数。maxThreads : maxThreads Tomcat 使用线程来处理接收的每个请求。这个值表示 Tomcat 可创建的最大的线程数，即最大并发数。acceptCount : acceptCount 是当线程数达到 maxThreads 后，后续请求会被放入一个等待队列，这个 acceptCount 是这个队列的大小，如果这个队列也满了，就直接 refuse connection maxProcessors 与 minProcessors : 在 Java 中线程是程序运行时的路径，是在一个程序中与其它控制线程无关的、能够独立运行的代码段。它们共享相同的地址空间。多线程帮助程序员写出 CPU 最 大利用率的高效程序，使空闲时间保持

最低，从而接受更多的请求。通常 Windows 是 1000 个左右，Linux 是 2000 个左右。
useURIValidationHack:我们来看一下 tomcat 中的一段源码：

```
    <!-enable tomcat ssl->
<Connector port=”8443” protocol=”HTTP/1.1”
URIEncoding=”UTF-8” minSpareThreads=”25” maxSpareThreads=”
75”
enableLookups=”false” disableUploadTimeout=”true”
connectionTimeout=”20000”
acceptCount=”300” maxThreads=”300” maxProcessors=”1000”
minProcessors=”5”
useURIValidationHack=”false”
compression=”on” compressionMinSize=”2048”
compressableMimeType=”text/html,text/xml,text/javascript,text/css,text/
plain”
SSLEnabled=”true”
scheme=”https” secure=”true”
clientAuth=”false” sslProtocol=”TLS”
keystoreFile=”d:/tomcat2/conf/shnlap93.jks” keystorePass=”aaaaaa”
/>
```

好了，所有的 Tomcat 优化的地方都加上了。

6.内存调优

内存方式的设置是在 catalina.sh 中，调整一下 JAVA_OPTS 变量即可，因为后面的启动参数会把 JAVA_OPTS 作为 JVM 的启动参数来处理。具体设置如下：

```
XX:NewRatio=4 -XX:SurvivorRatio=4"
```

其各项参数如下：-Xmx3550m：设置 JVM 最大可用内存为 3550M。-Xms3550m：设置 JVM 促使内存为 3550m。此值可以设置与-Xmx 相同，以避免每次垃圾回收完成后 JVM 重新分配内存。-Xmn2g：设置年轻代大小为 2G。整个堆大小=年轻代大小 + 年老代大小 +持久代大小。持久代一般固定大小为 64m，所以增大年轻代后，将会减小年老 代大小。此值对系统性能影响较大，Sun 官方推荐配置为整个堆的 3/8。-Xss128k：设置每个线程的堆栈大小。JDK5.0 以后每个线程堆栈大小为 1M，以前每个线程堆栈大小为 256K。更具应用的线程所需内存大小进行调整。在相同物理内存下，减小这个值能生成更多的线程。但是操作系统对一个进程内的线程数还是有限制的，不能无限生成，经验值在 3000~5000 左右。-XX:NewRatio=4:设置年轻代（包括 Eden 和两个 Survivor 区）与年老代的比值（除去持久代）。设置为 4，则年轻代与年老代所占比值为 1：4，年轻代占整个堆栈的 1/5-XX:SurvivorRatio=4：设置年轻代中 Eden 区与 Survivor 区的大小比值。设置为 4，则两个

Survivor 区与一个 Eden 区的比值为 2:4，一个 Survivor 区占整个年轻代的 1/6 -XX:MaxPermSize=16m:设置持久代大小为 16m。-XX:MaxTenuringThreshold=0：设置垃圾最大年龄。如果设置为 0 的话，则年轻代对象不经过 Survivor 区，直接进入年老代。对于年老代比较多的应用，可以提高效率。如果将此值设置为一个较大值，则年轻代对象会在 Survivor 区进行多次复制，这样可以增加对象再年轻代的存活时间，增加在年轻代即被回收的概论。

7.垃圾回收策略调优

垃圾回收的设置也是在 catalina.sh 中，调整 JAVA_OPTS 变量。具体设置如下：

```
XX:+UseParallelGC -XX:MaxGCPauseMillis=100"
```

具体的垃圾回收策略及相应策略的各项参数如下：串行收集器（JDK1.5 以前主要的回收方式） -XX:+UseSerialGC:设置串行收集器 并行收集器（吞吐量优先） 示例：java -Xmx3550m -Xms3550m -Xmn2g -Xss128k -XX:+UseParallelGC - XX:MaxGCPauseMillis=100 -XX:+UseParallelGC：选择垃圾收集器为并行收集器。此配置仅对年轻代有效。即上述配置下，年轻代使用并发收集，而年老代仍旧使用串行收集。-XX:ParallelGCThreads=20：配置并行收集器的线程数，即：同时多少个线程一起进行垃圾回收。此值最好配置与处理器数目相等。-XX:+UseParallelOldGC：配置年老代垃圾收集方式为并行收集。JDK6.0 支持对年老代并行收集 -XX:MaxGCPauseMillis=100:设置每次年轻代垃圾回收的最长时间，如果无法满足此时间，JVM 会自动调整年轻代大小，以满足此值。-XX:+UseAdaptiveSizePolicy：设置此选项后，并行收集器会自动选择年轻代区大小和相应的 Survivor 区比例，以达到目标系统规定的最低相应时间或者收集频率等，此值建议使用并行收集器时，一直打开。并发收集器（响应时间优先） 示例：java -Xmx3550m -Xms3550m -Xmn2g -Xss128k -XX:+UseConcMarkSweepGC-XX:+UseConcMarkSweepGC：设置年老代为并发收集。测试中配置这个以后，-XX:NewRatio=4 的配置失效了，原因不明。所以，此时年轻代大小最好用-Xmn 设置。-XX:+UseParNewGC: 设置年轻代为并行收集。可与 CMS 收集同时使用。JDK5.0 以上，JVM 会根据系统配置自行设置，所以无需再设置此值。-XX:CMSFullGCsBeforeCompaction：由于并发收集器不对内存空间进行压缩、整理，所以运行一段时间以后会产生“碎片”，使得运行效率降低。此值设置运行多少次 GC 以后对内存空间进行压缩、整理。-XX:+UseCMSCompactAtFullCollection：打开对年老代的压缩。可能会影响性能，但是可以消除碎片 **8.共享 session 处理**目前的处理方式有如下几种：1).使用 Tomcat 本身的 Session 复制功能 参考 http://ajita.iteye.com/blog/1715312（Session 复制的配置）方案的有点是配置简单，缺点是当集群数量较多时，Session 复制的时间会比 较长，影响响应的效率 2).使用第三方来存放共享 Session 目前用的较多的是使用 memcached 来管理共享 Session，借助于 memcached-sesson-manager 来进行 Tomcat 的 Session 管理参考 http://ajita.iteye.com/blog/1716320（使用 MSM 管理 Tomcat 集群 session） 3).使用黏性 session 的策略 对于会话要求不太强（不涉及到计费，失败了允许重新请求下等）的场合，同一个用户的 session 可以由 nginx 或者 apache 交给同一个 Tomcat 来处理，这就是所谓的 session sticky 策略，目前应用也比较多 参考： http://ajita.iteye.com/blog/1848665（tomcat session sticky）nginx 默认不包含 session sticky 模块，需要重新编译才行 优点是处理效率高多了，缺点是强会话要求的场合不合适

8.添加 JMS 远程监控

对于部署在局域网内其它机器上的 Tomcat，可以打开 JMX 监控端口，局域网其它机器就可以通过这个端口查看一些常用的参数（但一些比较复杂的功能不支持），同样是在 JVM 启动参数中配置即可，配置如下：-Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false -Djava.rmi.server.hostname=192.168.71.38 设置 JVM 的 JMS 监控监听的 IP 地址，主要是为了防止错误的监听成 127.0.0.1 这个内网地址 -Dcom.sun.management.jmxremote.port=1090 设置 JVM 的 JMS 监控的端口 -Dcom.sun.management.jmxremote.ssl=false 设置 JVM 的 JMS 监控不实用 SSL -Dcom.sun.management.jmxremote.authenticate=false 设置 JVM 的 JMS 监控不需要认证

9.专业点的分析工具有哪些？

IBM ISA，JProfiler、probe 等，具体监控及分析方式去网上搜索即可

10.关于 Tomcat 的 session 数目

这个可以直接从 Tomcat 的 web 管理界面去查看即可 ；或者借助于第三方工具 Lambda Probe 来查看，它相对于 Tomcat 自带的管理稍微多了点功能，但也不多 ；

11.监视 Tomcat 的内存使用情况

使用 JDK 自带的 jconsole 可以比较明了的看到内存的使用情况，线程的状态，当前加载的类的总量等；JDK 自带的 jvisualvm 可以下载插件（如 GC 等），可以查看更丰富的信息。如果是分析本地的 Tomcat 的话，还可以进行内存抽样等，检查每个类的使用情况

12.打印类的加载情况及对象的回收情况

这个可以通过配置 JVM 的启动参数，打印这些信息（到屏幕（默认也会到 catalina.log 中）或者文件），具体参数如下：-XX:+PrintGC：输出形式：[GC 118250K->113543K(130112K), 0.0094143secs] [Full GC 121376K->10414K(130112K), 0.0650971 secs] -XX:+PrintGCDetails：输出形式：[GC [DefNew: 8614K->781K(9088K),0.0123035 secs] 118250K->113543K(130112K), 0.0124633 secs] [GC [DefNew: 8614K->8614K(9088K), 0.0000665 secs][Tenured: 112761K->10414K(121024K), 0.0433488 secs]

121376K->10414K(130112K),0.0436268 secs] -XX:+PrintGCTimeStamps -XX:+PrintGC：PrintGCTimeStamps 可与上面两个混合使用，输出形式：11.851: [GC 98328K->93620K(130112K), 0.0082960secs] -XX:+PrintGCApplicationConcurrentTime：打印每次垃圾回收前，程序未中断的执行时间。可与上面混合使用。输出形式：Application time: 0.5291524seconds -XX:+PrintGCApplicationStoppedTime：打印垃圾回收期间程序暂停的时间。可与上面混合使用。输出形式：Total time for which application threads were stopped: 0.0468229 seconds -XX:PrintHeapAtGC: 打印 GC 前后的详细堆栈信息 -Xloggc:filename:与上面几个配合使用，把相关日志信息记录到文件以便分析 -verbose:class 监视加载的类的情况 -verbose:gc 在虚拟机发生内存回收时在输出设备显示信息 -verbose:jni 输出 native 方法调用的相关情况，一般用于诊断 jni 调用错误信息

13.Tomcat 一个请求的完整过程

```
    <Connector port="8080"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" redirectPort="8443" acceptCount="100"
debug="0" connectionTimeout="20000"
disableUploadTimeout="true" />
```

对于其他端口的侦听配置，以此类推。

3. tomcat 中如何禁止列目录下的文件

在{tomcat_home}/conf/web.xml 中，把 listings 参数设置成 false 即可，如下：

```
<init-param>
<param-name>listings</param-name>
<param-value>false</param-value>
</init-param>
<init-param>
<param-name>listings</param-name>
<param-value>false</param-value>
</init-param>
```

首先 dns 解析 wo.de.tian 机器，一般是 ng 服务器 ip 地址 然后 ng 根据 server 的配置，寻找路径为 yy/的机器列表，ip 和端口 最后 选择其中一台机器进行访问—->下面为详细过程 1) 请求被发送到本机端口 8080，被在那里侦听的 Coyote HTTP/1.1Connector 获得 2) Connector 把该请求交给它所在的 Service 的 Engine 来处理，并等待来自 Engine 的回应 3) Engine 获得请求 localhost/yy/index.jsp，匹配它所拥有的所有虚拟主机 Host 4) Engine 匹配到名为 localhost 的 Host（即使匹配不到也把请求交给该 Host 处理，因为该 Host 被定义为该 Engine 的默认主机） 5) localhost Host 获得请求/yy/index.jsp，匹配它所拥有的所有 Context 6) Host 匹配到路径为/yy 的 Context（如果匹配不到就把该请求交给路径名为”“的 Context 去处

理） 7) path=”/yy”的 Context 获得请求/index.jsp，在它的 mapping table 中寻找对应的 servlet 8) Context 匹配到 URL PATTERN 为*.jsp 的 servlet，对应于 JspServlet 类 9) 构造 HttpServletRequest 对象和 HttpServletResponse 对象，作为参数调用 JspServlet 的 doGet 或 doPost 方法 10)Context 把执行完了之后的 HttpServletResponse 对象返回给 Host 11)Host 把 HttpServletResponse 对象返回给 Engine 12)Engine 把 HttpServletResponse 对象返回给 Connector 13)Connector 把 HttpServletResponse 对象返回给客户 browser

14.Tomcat 工作模式？

Tomcat 是一个 JSP/Servlet 容器。其作为 Servlet 容器，有三种工作模式：独立的 Servlet 容器、进程内的 Servlet 容器和进程外的 Servlet 容器。进入 Tomcat 的请求可以根据 Tomcat 的工作模式分为如下两类：Tomcat 作为应用程序服务器：请求来自于前端的 web 服务器，这可能是 Apache, IIS, Nginx 等；Tomcat 作为独立服务器：请求来自于 web 浏览器；

15.你怎样给 tomcat 去调优?

JVM 参数调优：-Xms 表示 JVM 初始化堆的大小，-Xmx 表示 JVM 堆的最大值。这两个值的大小一般根据需要进行设置。当应用程序需要的内存超出堆的最大值时虚拟机就会提示内存溢出，并且导致应用服务崩溃。因此一般建议堆的最大值设置为可用内存的最大值的 80%。在 catalina.bat 中，设置 JAVA_OPTS=’-Xms256m -Xmx512m’，表示初始化内存为 256MB，可以使用的最大内存为 512MB。
禁用 DNS 查询 当 web 应用程序向要记录客户端的信息时，它也会记录客户端的 IP 地址或者通过域名服务器查找机器名转换为 IP 地址。DNS 查询需要占用网络，并且包括可能从很多很远的服务器或者不起作用的服务器上去获取对应的 IP 的过程，这样会消耗一定的时间。为了消除 DNS 查询对性能的影响我们可以关闭 DNS 查询，方式是修改 server.xml 文件中的 enableLookups 参数值：Tomcat4 Tomcat5
调整线程数 通过应用程序的连接器（Connector）进行性能控制的的参数是创建的处理请求的线程数。Tomcat 使用线程池加速响应速度来处理请求。在 Java 中线程是程序运行时的路径，是在一个程序中与其它控制线程无关的、能够独立运行的代码段。它们共享相同的地址空间。多线程帮助程序员写出 CPU 最大利用率的高效程序，使空闲时间保持最低，从而接受更多的请求。Tomcat4 中可以通过修改 minProcessors 和 maxProcessors 的值来控制线程数。这些值在安装后就已经设定为默认值并且是足够使用的，但是随着站点的扩容而改大这些值。minProcessors 服务器启动时创建的处理请求的线程数应该足够处理一个小量的负载。也就是说，如果一天内每秒仅发生 5 次单击事件，并且每个请求任务处理需要 1 秒钟，那么预先设置线程数为 5 就足够了。但在你的站点访问量较大时就需要设置更大的线程数，指定为参数 maxProcessors 的值。maxProcessors 的值也是有上限的，应防止流量不可控制（或者恶意的服务攻击），从而导致超出了虚拟机使用内存的大小。如果要加大并发连接数，应同时加大这两个参数。web server 允许的最大连接数还受制于操作系统的内核参数设置，通常 Windows 是 2000 个左右，Linux 是 1000 个左右。

在 Tomcat5 对这些参数进行了调整，请看下面属性：maxThreads Tomcat 使用线程来处理接收的每个请求。这个值表示 Tomcat 可创建的最大的线程数。acceptCount 指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理。connnectionTimeout 网络连接超时，单位：毫秒。设置为 0 表示永不超时，这样设置有隐患的。通常可设置为 30000 毫秒。minSpareThreads Tomcat 初始化时创建的线程数。maxSpareThreads 一旦创建的线程超过这个值，Tomcat 就会关闭不再需要的 socket 线程。最好的方式是多设置几次并且进行测试，观察响应时间和内存使用情况。在不同的机器、操作系统或虚拟机组合的情况下可能会不同，而且并不是所有人的 web 站点的流量都是一样的，因此没有一刀切的方案来确定线程数的值。

16.如何加大 tomcat 连接数

在 tomcat 配置文件 server.xml 中的 配置中，和连接数相关的参数有：
minProcessors：最小空闲连接线程数，用于提高系统处理性能，默认值为 10
maxProcessors：最大连接线程数，即：并发处理的最大请求数，默认值为 75
acceptCount：允许的最大连接数，应大于等于 maxProcessors，默认值为 100
enableLookups：是否反查域名，取值为：true 或 false。为了提高处理能力，应设置为 false connectionTimeout：网络连接超时，单位：毫秒。设置为 0 表示永不超时，这样设置有隐患的。通常可设置为 30000 毫秒。其中和最大连接数相关的参数为 maxProcessors 和 acceptCount。如果要加大并发连接数，应同时加大这两个参数。web server 允许的最大连接数还受制于操作系统的内核参数设置，通常 Windows 是 2000 个左右，Linux 是 1000 个左右。tomcat5 中的配置示例：

17.怎样加大 tomcat 的内存。

首先检查程序有没有限入死循环 这个问题主要还是由这个问题
java.lang.OutOfMemoryError: Java heap space 引起的。第一次出现这样的的问题以后，引发了其他的问题。在网上一查可能是 JAVA 的堆栈设置太小的原因。跟据网上的答案大致有这两种解决方法：
1、设置环境变量 解决方法：手动设置 Heap size 修改 TOMCAT_HOME/bin/catalina.sh set JAVA_OPTS= -Xms32m -Xmx512m 可以根据自己机器的内存进行更改。
2、java -Xms32m -Xmx800m className 就是在执行 JAVA 类文件时加上这个参数，其中 className 是需要执行的确类名。（包括包名） 这个解决问题了。而且执行的速度比没有设置的时候快很多。如果在测试的时候可能会用 Eclispe 这时候就需要在 Eclipse ->run -arguments 中的 VM arguments 中输入-Xms32m -Xmx800m 这个参数就可以了。后来在 Eclilpse 中修改了启动参数，在 VM arguments 加入了-Xms32m -Xmx800m，问题解决。
一、java.lang.OutOfMemoryError: PermGen space PermGen space 的全称是 Permanent Generation space,是指内存的永久保存区域,这块内存主要是被 JVM 存放 Class 和 Meta 信息的,Class 在被 Loader 时就会被放到 PermGen space 中,它和存放类实例(Instance)的 Heap 区域不同,GC(Garbage Collection)不会在主程序运行期对 PermGen space 进行清理，所

以如果你的应用中有很多 CLASS 的话,就很可能出现 PermGen space 错误,这种错误常见在 web 服务器对 JSP 进行 pre compile 的时候。如果你的 WEB APP 下都用了大量的第三方 jar, 其大小超过了 jvm 默认的大小(4M)那么就会产生此错误信息了。解决方法：手动设置 MaxPermSize 大小 修改 TOMCAT_HOME/bin/catalina.sh 在“echo "Using CATALINA_BASE: $CATALINA_BASE"”上面加入以下行：

```
JAVA_OPTS="-server -XX:PermSize=64M -XX:MaxPermSize=128m
```

建议：将相同的第三方 jar 文件移置到 tomcat/shared/lib 目录下，这样可以达到减少 jar 文档重复占用内存的目的。二、java.lang.OutOfMemoryError: Java heap spaceHeap size 设置 JVM 堆的设置是指 java 程序运行过程中 JVM 可以调配使用的内存空间的设置.JVM 在启动的时候会自动设置 Heap size 的值， 其初始空间(即-Xms)是物理内存的 1/64，最大空间(-Xmx)是物理内存的 1/4。可以利用 JVM 提供的-Xmn -Xms -Xmx 等选项可进行设置。Heap size 的大小是 Young Generation 和 Tenured Generaion 之和。提示：在 JVM 中如果 98％的时间是用于 GC 且可用的 Heap size 不足 2％的时候将抛出此异常信息。提示：Heap Size 最大不要超过可用物理内存的 80％，一般的要将-Xms 和-Xmx 选项设置为相同，而 Xmn 为 1/4 的-Xmx 值。解决方法：手动设置 Heap size 修改 TOMCAT_HOME/bin/catalina.sh 在“echo "Using CATALINA_BASE: $CATALINA_BASE"”上面加入以下行：

```
JAVA_OPTS="-server -Xms800m -Xmx800m -XX:MaxNewSize=256m"
```

3、实例，以下给出 1G 内存环境下 java jvm 的参数设置参考：JAVA_OPTS=″-server -Xms800m -Xmx800m -XX:PermSize=64M -XX:MaxNewSize=256m - XX:MaxPermSize=128m -Djava.awt.headless=true ″ 很大的 web 工程，用 tomcat 默认分配的内存空间无法启动，如果不是在 myeclipse 中启动 tomcat 可以对 tomcat 这样设置：TOMCAT_HOME/bin/catalina.bat 中添加这样一句话：set JAVA_OPTS=-server -Xms2048m -Xmx4096m -XX:PermSize=512M - XX:MaxPermSize=1024M -Duser.timezone=GMT+08 或者 set JAVA_OPTS= -Xmx1024M -Xms512M -XX:MaxPermSize=256m 如果要在 myeclipse 中启动，上述的修改就不起作用了，可如下设置：Myeclipse->preferences->myeclipse->servers->tomcat->tomcat×.×->JDK 面板中的 Optional Java VM arguments 中添加：-Xmx1024M -Xms512M -XX:MaxPermSize=256m 以上是转贴，但本人遇见的问题是：在 myeclipse 中启动 Tomcat 时，提示 ″ava.lang.OutOfMemoryError: Java heap space″，解决办法就是：Myeclipse->preferences->myeclipse->servers->tomcat->tomcat×.×->JDK 面板中的 Optional Java VM arguments 中添加：-Xmx1024M -Xms512M -XX:MaxPermSize=256m

18.Tomcat 有几种部署方式

tomcat 中四种部署项目的方法 第一种方法：在 tomcat 中的 conf 目录中，在 server.xml 中的， 节点中添加：

```
<Context path="/hello"
docBase="D:/eclipse3.2.2/forwebtoolsworkspacehello/WebRoot"
debug="0" privileged="true">
</Context>
```

至于 Context 节点属性，可详细见相关文档。第二种方法：将 web 项目文件件拷贝到 webapps 目录中。第三种方法：很灵活，在 conf 目录中，新建 Catalina（注意大小写）\localhost 目录，在该目录中新建一个 xml 文件，名字可以随意取，只要和当前文件中的文件名不重复就行了，该 xml 文件的内容为：

```
<Context path="/hello"
docBase="D:eclipse3.2.2forwebtoolsworkspacehelloWebRoot"
debug="0" privileged="true">
</Context>
```

第 3 个方法有个优点，可以定义别名。服务器端运行的项目名称为 path，外部访问的 URL 则使用 XML 的文件名。这个方法很方便的隐藏了项目的名称，对一些项目名称被固定不能更换，但外部访问时又想换个路径，非常有效。第 2、3 还有优点，可以定义一些个性配置，如数据源的配置等。第四种办法,：可以用 tomcat 在线后台管理器,一般 tomcat 都打开了,直接上传 war 就可以

19.Tomcat 的优化经验。

Tomcat 作为 Web 服务器，它的处理性能直接关系到用户体验，下面是几种常见的优化措施：• 去掉对 web.xml 的监视，把 jsp 提前编辑成 Servlet。有富余物理内存的情况，加大 tomcat 使用的 jvm 的内存。• 服务器资源 服务器所能提供 CPU、内存、硬盘的性能对处理能力有决定性影响。o 对于高并发情况下会有大量的运算，那么 CPU 的速度会直接影响到处理速度。 o 内存在大量数据处理的情况下，将会有较大的内存容量需求，可以用 -Xmx -Xms -XX:MaxPermSize 等参数对内存不同功能块进行划分。我们之前就遇到过内存分配不足，导致虚拟机一直处于 full GC，从而导致处理能力严重下降。 o 硬盘主要问题就是读写性能，当大量文件进行读写时，磁盘极容易成为性能瓶颈。最好的办法还是利用下面提到的缓存。• 利用缓存和压缩对于静态页面最好是能够缓存起来，这样就不必每次从磁盘上读。这里我们采用了 Nginx 作为缓存服务器，将图片、css、js 文件都进行了缓存，有效的减少了后端 tomcat 的访问。另外，为了能加快网络传输速度，开启 gzip 压缩也是必不可少的。但考虑到 tomcat 已经需要处理很多东西了，所以把这个压缩的工作就交给前端的 Nginx 来完成。除了文本可以用 gzip 压缩，其实很多图片也可以用图像处理工具预先进行压缩，找到一个平衡点可以让画质损失很小而文件可以减小很多。曾经我就见过一个图片从 300 多 kb 压缩到几十 kb，自己几乎看不出来区别。• 采用集群 单个服务器性能总是有限的，最好的办法自然是实现横向扩展，那么组建 tomcat 集群是有效提升性能的手段。我们还是采用了 Nginx 来作为请求分 流的服务器，后端多个 tomcat 共享 session 来协同工作。可以参考之前写的《利用 nginx+tomcat+memcached 组建 web 服务器负载均衡》。• 优化 tomcat 参数 这里以 tomcat7 的参数配置为例，需要修改 conf/server.xml 文件，主要是优化连接配置，关闭客户端 dns 查询。

```
<Connector port="8080"
 protocol="org.apache.coyote.http11.Http11NioProtocol"
 connectionTimeout="20000"
 redirectPort="8443"
 maxThreads="500"
 minSpareThreads="20"
 acceptCount="100"
 disableUploadTimeout="true"
 enableLookups="false"
 URIEncoding="UTF-8" />
```