Tomcat 应用服务

目录

文档信息

文档作者	房佳亮
文档版本	Version1.0
文档版权	内部资料禁止传播
文档归类	Linux 运维架构师系列
系统环境	CentOS-7.X-x86_64
作者邮箱	crushlinux@163.com
修订信息	2021-02-04
技术交流	

文档约定

[绿色背景] 知识重点
[红色背景] 错误警告
[黄色背景] 注意事项

执行命令

中间件产品介绍

目前来说 IBM 的 WebSphere，Oracle 的 Weblogic 占据了市场上 Java 语言 Web 站点的部分份额，该两种软件由于无与伦比的性能及可靠性等优势被广泛应用于大型互联网公司的 Web 场景中，但是其高昂的价格也使得中小型互联网公司对此望而却步。

Tomcat 自 5.x 版本以来，其性能上已经得到很大幅度的提升，加上其开放性的框架和二次开发等特性，已经完全可以用在访问量不是很大的生产环境下，目前大多数用于 JSP 技术开发的电子商务网站基本上都应用了 Tomcat。

中间件产品：RedHat JBoss、Oracle Tuxedo、caucho Resin。

Tomcat 软件简介

Tomcat 是 Apache 软件基金会（Apache Software Foundation）的 Jakarta 项目中的一个核心项目，由 Apache、Sun 和其他一些公司及个人共同开发而成。Tomcat 最初是由 Sun 的软件构架师詹姆斯 •邓肯 •戴维森开发的。后来他将其变为开源项目，并由 Sun 贡献给 Apache

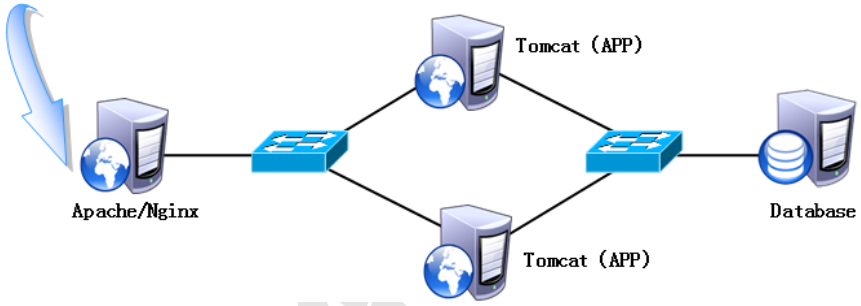
软件基金会。由于大部分开源项目 O'Reilly 都会出一本相关的书，并且将其封面设计成某个动物的素描，因此他希望将此项目以一个动物的名字命名。因为他希望这种动物能够自己照顾自己，最终，他将其命名为 Tomcat（公猫）。而 O'Reilly 出版的介绍 Tomcat 的书籍的封面也被设计成了一个公猫的形象。而 Tomcat 的 Logo 兼吉祥物也被设计成了一只公猫。

其实 Tomcat 最早在开始研发的时候并不叫这个名字，早期的 Tomcat 的 Logo 项目的名字叫 Catalina，所以当我们安装完 Tomcat 后会发现安装路径下有很多 Catalina 有关的目录和文件。这些文件是我们配置或使用 Tomcat 的重要文件所在。

Tomcat 应用场景

Tomcat 服务器是一个免费的开放源代码的 web 应用服务器，属于轻量级应用服务器，应用在中小型系统和并发访问用户不是很多的场合，是开发和调试 JSP 页面的首选，Tomcat 也可处理静态的 HTML 页面但是能力不及 Apache 或 Nginx，所以 Tomcat 通常作为一个 servlet 和 JSP 容器，单独运行在后端。

Tomcat 官网： http://tomcat.apache.org



JDK 软件简介

在安装 Tomcat 之前必须先安装 JDK，JDK 全称是 Java Development Kit，是 SUN 公司免费提供的 java 语言的软件开发工具包，其中包含 Java 虚拟机（JVM），编写好的 java 源程序经过编译可生产 java 字节码，只要安装了 JDK，就可以利用 JVM 解释这些字节码文件，从而保证了 Java 的跨平台性。

在平台兼容性方面，JDK 作为解释字节码文件并根据此调用操作系统的 API 实现对应功能的 java 虚拟机，与操作系统位数密切相关，因此存在不同类型的版本，Tomcat 也具有上述特征，CentOS7 系统中默认已经安装了 JDK。

JDK 包含了一批用于 Java 开发的组件，其中包括：

javac：编译器，将后缀名为.java 的源代码编译成后缀名为“.class”的字节码
java：运行工具，运行.class 的字节码
jar：打包工具，将相关的类文件打包成一个文件
javadoc：文档生成器，从源码注释中提取文档，注释需匹配规范
jdb debugger：调试工具
jps：显示当前 java 程序运行的进程状态
javap：反编译程序
appletviewer：运行和调试 applet 程序的工具，不需要使用浏览器
javah：从 Java 类生成 C 头文件和 C 源文件。这些文件提供了连接胶合，使 Java 和 C 代码

```
可进行交互。
javaws：运行 JNLP 程序
extcheck：一个检测 jar 包冲突的工具
apt：注释处理工具
jhat：java 堆分析工具
jstack：栈跟踪程序
jstat：JVM 检测统计工具
jstatd：jstat 守护进程
jinfo：获取正在运行或崩溃的 java 程序配置信息
jmap：获取 java 进程内存映射信息
idlj：IDL-to-Java 编译器。将 IDL 语言转化为 java 文件
policytool：一个 GUI 的策略文件创建和管理工具
jrunscript：命令行脚本运行
```

JDK 中还包括完整的 JRE（Java Runtime Environment），Java 运行环境，也被称为 private runtime。包括了用于产品环境的各种库类，如基础类库 rt.jar，以及给开发人员使用的补充库，如国际化与本地化的类库、IDL 库等等。JDK 中还包括各种样例程序，用以展示 Java API 中的各部分。

安装配置 Tomcat

安装时候选择 tomcat 软件版本要与程序开发使用的版本一致。jdk 版本要进行与 tomcat 保持一致。

1、所有主机关闭防火墙和 selinux：

```
[root@localhost ~]# iptables -F
[root@localhost ~]# setenforce 0
setenforce: SELinux is disabled
[root@localhost ~]# systemctl stop firewalld
```

2、查看 JDK 是否安装

```
[root@localhost ~]# java -version
openjdk version "1.8.0_161"       //这是系统自带的 rpm 方式安装
OpenJDK Runtime Environment (build 1.8.0_161-b14)
OpenJDK 64-Bit Server VM (build 25.161-b14, mixed mode)
```

3、卸载 rpm 方式安装的 jdk

方法一：

```
[root@localhost ~]# which java
/usr/bin/java
[root@localhost ~]# rm -rf /usr/bin/java
```

方法二：

```
[root@localhost ~]# rpm -qa | grep -i openjdk
```

```
java-1.7.0-openjdk-1.7.0.171-2.6.13.2.el7.x86_64
java-1.7.0-openjdk-headless-1.7.0.171-2.6.13.2.el7.x86_64
java-1.8.0-openjdk-1.8.0.161-2.b14.el7.x86_64
java-1.8.0-openjdk-headless-1.8.0.161-2.b14.el7.x86_64

[root@localhost ~]# rpm -e java-1.7.0-openjdk
[root@localhost ~]# rpm -e java-1.7.0-openjdk-headless
[root@localhost ~]# rpm -e java-1.8.0-openjdk --nodeps
[root@localhost ~]# rpm -e java-1.8.0-openjdk-headless
[root@localhost ~]# rpm -qa | grep -i openjdk
```

4、JDK 安装

```
[root@localhost ~]# tar xf jdk-8u191-linux-x64.tar.gz
[root@localhost ~]# mv jdk1.8.0_191/ /usr/local/java
[root@localhost ~]# vim /etc/profile
export JAVA_HOME=/usr/local/java    #设置 java 跟目录
export PATH=$PATH:$JAVA_HOME/bin#在 PATH 环境变量中添加 java 跟目录的 bin 子目录
[root@localhost ~]# source /etc/profile
[root@localhost ~]# java -version
java version "1.8.0_191"
Java(TM) SE Runtime Environment (build 1.8.0_191-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)
```

5、解压 apache-tomcat-8.5.16.tar.gz 包

```
[root@localhost ~]# tar xf apache-tomcat-8.5.16.tar.gz
```

6、解压后生成 apache-tomcat-8.5.16 文件夹，将该文件夹移动到/usr/local 下，并改名为 tomcat8

```
[root@localhost ~]# mv apache-tomcat-8.5.16 /usr/local/tomcat8
```

7、启动 Tomcat

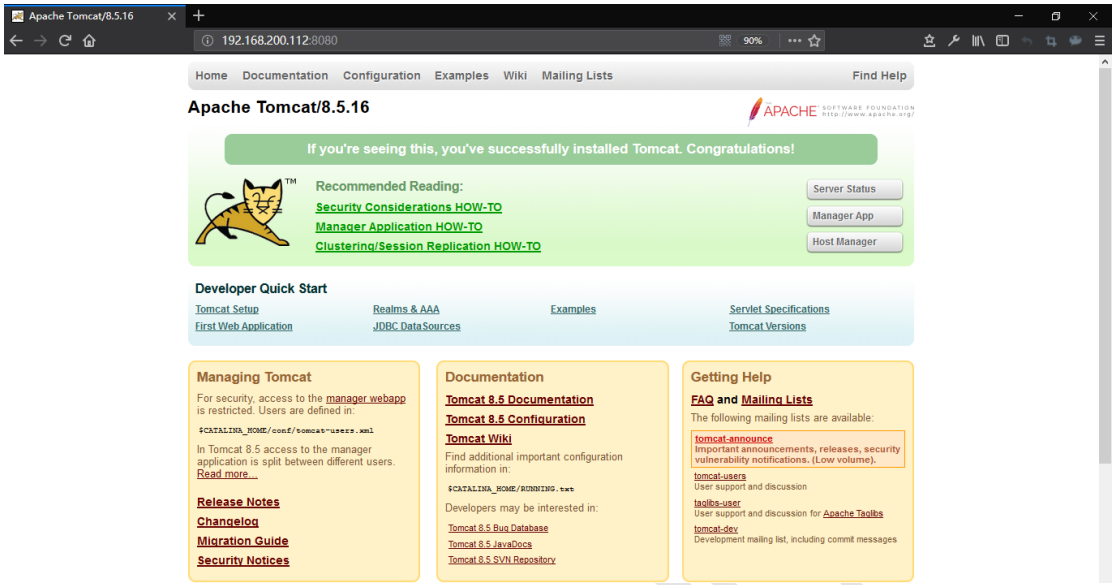
```
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
Using CATALINA_BASE:   /usr/local/tomcat8
Using CATALINA_HOME:   /usr/local/tomcat8
Using CATALINA_TMPDIR: /usr/local/tomcat8/temp
Using JRE_HOME:        /usr
Using CLASSPATH:
/usr/local/tomcat8/bin/bootstrap.jar:/usr/local/tomcat8/bin/tomcat-juli.jar
Tomcat started.
```

8、Tomcat 默认运行在 8080 端口

```
[root@localhost ~]# netstat -lnpt | grep :8080
tcp6       0      0 :::8080                 :::*                    LISTEN
```

```
1533/java
```

9、浏览器访问测试 http://192.168.200.112:8080



10、关闭 Tomcat

```
[root@localhost ~]# /usr/local/tomcat8/bin/shutdown.sh
```

Tomcat 配置目录及文件说明

```
[root@localhost ~]# ll /usr/local/tomcat8/
总用量 92
drwxr-x--- 2 root root  4096 1 月  21 22:53 bin
drwx------ 3 root root   254 1 月  21 22:54 conf
drwxr-x--- 2 root root  4096 1 月  21 22:53 lib
-rw-r----- 1 root root 57092 6 月  22 2017 LICENSE
drwxr-x--- 2 root root   197 1 月  21 22:54 logs
-rw-r----- 1 root root  1723 6 月  22 2017 NOTICE
-rw-r----- 1 root root  7064 6 月  22 2017 RELEASE-NOTES
-rw-r----- 1 root root 15946 6 月  22 2017 RUNNING.txt
drwxr-x--- 2 root root    30 1 月  21 22:53 temp
drwxr-x--- 7 root root    81 6 月  22 2017 webapps
drwxr-x--- 3 root root    22 1 月  21 22:54 work
```

tomcat 主目录介绍

bin //存放 windows 或 linux 平台上启动或关闭的 Tomcat 的脚本文件
conf //存放 Tomcat 的各种全局配置文件，其中最主要的是 server.xml 和 web.xml
lib //存放 Tomcat 运行需要的库文件(JARS)
logs //存放 Tomcat 执行时的 LOG 文件

webapps //Tomcat 的主要 Web 发布目录、类似于 nginx 的 html（包括应用程实例）
work //存放 jsp 编译后产生的.class 文件
temp //存放临时文件

webapps 目录介绍

docs //tomcat 帮助文档
examples //web 应用实例
host-manager //主机管理
manager //管理
ROOT //默认站点根目录

```
[root@localhost ~]# ll /usr/local/tomcat8/conf/
总用量 224
drwxr-x--- 3 root root     23 1 月  21 22:54 Catalina
-rw------- 1 root root  13816 6 月  22 2017 catalina.policy
-rw------- 1 root root   7376 6 月  22 2017 catalina.properties
-rw------- 1 root root   1338 6 月  22 2017 context.xml
-rw------- 1 root root   1149 6 月  22 2017 jaspic-providers.xml
-rw------- 1 root root   2358 6 月  22 2017 jaspic-providers.xsd
-rw------- 1 root root   3622 6 月  22 2017 logging.properties
-rw------- 1 root root   7511 6 月  22 2017 server.xml
-rw------- 1 root root   2164 6 月  22 2017 tomcat-users.xml
-rw------- 1 root root   2633 6 月  22 2017 tomcat-users.xsd
-rw------- 1 root root 168251 6 月  22 2017 web.xml
```

catalina.policy //权限控制配置文件
catalina.properties //Tomcat 属性配置文件
context.xml //context 用于指定额外的 web 目录
logging.properties //日志 log 相关配置文件
server.xml //主配置文件
tomcat-users.xml //manager-gui 管理用户配置文件（Tomcat 安装后生成的管理界面，该文件可开启访问）
web.xml //Tomcat 的 servlet，servlet-mapping，filter，MIME 等相关配置

Tomcat 主配置文件说明

server.xml 主要配置文件，可修改启动端口，设置网站根目录，虚拟主机，多实例、开启 https 加密等功能。

server.xml 的结构构成：

```
<Server>
    <Service>
        <Connector />
```

```
        <Engine>
            <Host>
                <Context> </Context>
            </Host>
        </Engine>
    </Service>
</Server>
```

<!-- --> 内的内容是注视信息

//tomcat 关闭端口，默认只对本机地址开放，可以在本机通过 telnet 127.0.0.1 8005 访问，对 Tomcat 进行关闭从操作。

```
<Server port="8005" shutdown="SHUTDOWN">
```

//tomcat 启动的默认端口号 8080，可以根据需要进行修改

```
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

国家信息安全漏洞共享平台

https://www.cnvd.org.cn/webinfo/show/5415

//tomcat 启动 AJP1.3 连接器时默认的端口号，可以根据需要进行修改

```
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

//以下是 tomcat 定义虚拟主机时的配置及日志配置

```
      <Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true">
            <Context docBase="/web/webapp" path="" reloadable="flase" >
            </Context>
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log" suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
      </Host>
```

Server.xml 主要参数说明

组件名称	功能介绍
server	表示一个运行于 JVM 中的 tomcat 实例。
service（服务）	将 connector 关联至 engine，因此一个 service 内部可以有多个 connector，但只能有一个引擎 engine，负责处理所有 Connector 所获得的客户请求。service 内部有两个 connector，一个 engine。因此，一般情况下一个 server 内部只有一个 service，一个 service 内部只有一个 engine，但一个 service 内部可以有多个 connector。

connector	接收用户请求，类似于 httpd 的 listen 配置监听端口的。 一个 Connector 在某个指定端口上侦听客户请求，并将获得的请求交给 Engine 来处理，从 Engine 处获得回应并返回客户。 Tomcat Engine 有两个典型的 Connector，一个直接侦听来自 browser 的 http 请求，一个侦听来自其他 webserver 的请 ote Http/1.1 Connector 在端口 8080 处侦听来自客户 Browser 的 http 请求。 Coyote JK2 Connector 在端口 8009 处侦听来自其他 webserver（Apache）的 servlet/jsp 代理请求。
engine	核心容器组件，catalina 引擎，负责通过 connector 接收用户请求，并处理请求，将请求转至对应的虚拟主机 host。 Engine 有一个默认的虚拟主机，当请求无法匹配到任何一个 Host 上的时候，将交给该默认 Host 来处理。
host	代表一个 Virtual Host，虚拟主机，每个虚拟主机和某个网络域名 Domain Name 相匹配 每个虚拟主机下都可以部署（deploy）一个或者多个 Web app，每个 web app 对应一个 Context，有一个 Context path。 当 Host 获得一个请求时，将把该请求匹配到某个 Context 上，然后把该请求交给该 Context 来处理，匹配的方法是最长匹配，所以一个 path==“”的 Context 将成为该 Host 的默认 Context 匹配。 所有无法和凌然 Context 的路径名匹配的请求都将最终和该默认 Context 匹配。
context	定义一个应用程序，是一个最内层的容器类组件（不能再嵌套）。配置 context 的主要目的指定对应对的 webapp 的根目录，类似于 httpd 的 alias，其还能为 webapp 指定额外的属性，如部署方式等。
realm	可以用于任意容器类的组件中，关联一个用户认证库，实现认证和授权。可以关联的认证库有两种：UserDatabaseRealm、MemoryRealm 和 JDBCRealm。
UserDatabaseRealm	使用 JNDI 自定义的用户认证库。
MemoryRealm	认证信息定义在 tomcat-users.xml 中。
JDBCRealm	认证信息定义在数据库中，并通过 JDBC 连接至数据库中查找认证用户。

host 参数详解

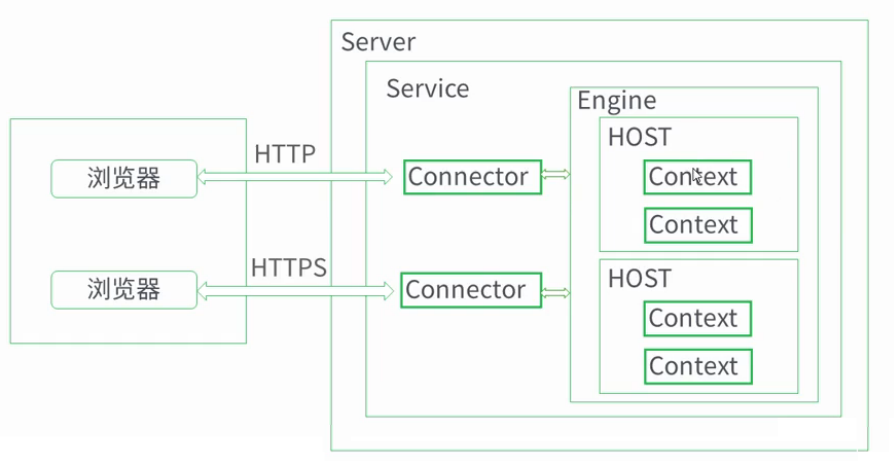
参数	参数说明
host	表示一个虚拟主机
name	指定主机名（域名）
appBase	应用程序基本目录，即存放应用程序的目录.一般为 appBase="webapps" ，相对于 CATALINA_HOME 而言的，也可以写绝对路径。
unpackWARs	如果为 true，则 tomcat 会自动将 WAR 文件解压，否则不解压，直接从 WAR 文件中运行应用程序
autoDeploy	在 tomcat 启动时，是否自动部署。

xmlValidation	是否启动 xml 的校验功能，一般 xmlValidation="false"。
xmlNamespaceAware	检测名称空间，一般 xmlNamespaceAware="false"。

Context 参数说明

参数	参数说明
Context	表示一个 web 应用程序加载位置，通常为 WAR 文件位置
docBase	应用程序的路径或者是 WAR 文件存放的路径,也可以使用相对路径，起始路径为此 Context 所属 Host 中 appBase 定义的路径。
path	表示此 web 应用程序的 url 的前缀，这样请求的 url 为 http://localhost:8080/path/****
reloadable	这个属性非常重要，如果为 true，则 tomcat 会自动检测应用程序的/WEB-INF/lib 和/WEB-INF/classes 目录的变化，自动装载新的应用程序，可以在不重启 tomcat 的情况下改变应用程序

请求处理流程



处理流程：用户发送请求到 WEB 服务器，该请求会被正在监听的 Connector 连接器接收，并把该请求交给 Service 下的 Engine 来处理，并等待 Engine 处理的结果。Engine 获得请求后会根据请求的主机信息来匹配相应的 Host 主机，Host 主机会根据请求的路径匹配对应的 Context，Context web 应用匹配上之后就构建 request、response 请求对象，调用指定的 Servlet 来处理请求。请求处理完成后会将 response 对象返回给 Host 主机，Host 主机将 response 对象返回给 Engine 引擎，Engine 再将 response 对象返回给 Connector 链接器，最后 Connector 连接器将 response 返回给浏览器。

自定义默认网站目录

首先在跟目录下建立一个 web 目录，并在里面建立一个 webapp 目录，用于存放网站文件

```
[root@localhost ~]# mkdir -pv /web/webapp
```

```
mkdir: 已创建目录 "/web"
mkdir: 已创建目录 "/web/webapp"
```

在 webapp 目录下建立一个 index.jsp 的测试页面

```
[root@localhost ~]# vim /web/webapp/index.jsp
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>JSP test page</title>
    </head>
    <body>
        <% out.println("Welcome to test site,http://www.test1.com");%>
    </body>
</html>
```

修改 Tomcat 的 server.xml 文件

定义一个虚拟主机，并将网站文件路径指向已经建立的/web/webapp，在 host 段增加 context 段

```
[root@localhost ~]# cp /usr/local/tomcat8/conf/server.xml{,.-$(date +%F)}
[root@localhost ~]# vim /usr/local/tomcat8/conf/server.xml
148         <Host name="localhost"    appBase="webapps"
149                 unpackWARs="true" autoDeploy="true">
150                 <Context docBase="/web/webapp" path="" reloadable="flase" >
151                 </Context>
```

重新启动 tomcat 服务

```
[root@localhost ~]# /usr/local/tomcat8/bin/shutdown.sh
Using CATALINA_BASE:    /usr/local/tomcat8
Using CATALINA_HOME:    /usr/local/tomcat8
Using CATALINA_TMPDIR: /usr/local/tomcat8/temp
Using JRE_HOME:          /usr
Using CLASSPATH:
/usr/local/tomcat8/bin/bootstrap.jar:/usr/local/tomcat8/bin/tomcat-juli.jar
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
Using CATALINA_BASE:    /usr/local/tomcat8
Using CATALINA_HOME:    /usr/local/tomcat8
Using CATALINA_TMPDIR: /usr/local/tomcat8/temp
Using JRE_HOME:          /usr
Using CLASSPATH:
/usr/local/tomcat8/bin/bootstrap.jar:/usr/local/tomcat8/bin/tomcat-juli.jar
Tomcat started.
```

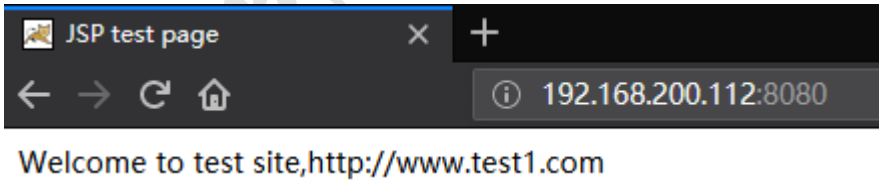
说明：所有与 java 相关的，服务启动都是 java 命名的进程

```
[root@localhost ~]# tail -f /usr/local/tomcat8/logs/catalina.out
```

```
21-Jan-2019 23:24:01.391 信息 [localhost-startStop-1]
org.apache.catalina.startup.HostConfig.deployDirectory Deployment o
f web application directory [/usr/local/tomcat8/webapps/docs] has finished in [134] ms21-Jan-
2019 23:24:01.391 信息 [localhost-startStop-1]
org.apache.catalina.startup.HostConfig.deployDirectory Deploying we
b application directory [/usr/local/tomcat8/webapps/examples]21-Jan-2019 23:24:04.183 信
息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deployment
o
f web application directory [/usr/local/tomcat8/webapps/examples] has finished in [2,792]
ms21-Jan-2019 23:24:04.184 信息 [localhost-startStop-1]
org.apache.catalina.startup.HostConfig.deployDirectory Deploying we
b application directory [/usr/local/tomcat8/webapps/host-manager]21-Jan-2019 23:24:04.433
信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory
Deployment o
f web application directory [/usr/local/tomcat8/webapps/host-manager] has finished in [248]
ms21-Jan-2019 23:24:04.434 信息 [localhost-startStop-1]
org.apache.catalina.startup.HostConfig.deployDirectory Deploying we
b application directory [/usr/local/tomcat8/webapps/manager]21-Jan-2019 23:24:04.859 信
息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deployment
o
f web application directory [/usr/local/tomcat8/webapps/manager] has finished in [415]
ms21-Jan-2019 23:24:04.941 信息 [main] org.apache.coyote.AbstractProtocol.start Starting
ProtocolHandler ["http-nio-8080"]
21-Jan-2019 23:24:05.167 信息 [main] org.apache.coyote.AbstractProtocol.start Starting
ProtocolHandler ["ajp-nio-8009"]
21-Jan-2019 23:24:05.190 信息 [main] org.apache.catalina.startup.Catalina.start Server
startup in 6892 ms
```

注意：tomcat 未启动的情况下使用 shutdown 脚本，会有大量的输出信息。

浏览器访问测试 http://192.168.200.200:8080



Nginx+Tomcat 负载均衡集群（群集 Cluster）

Nginx 代理介绍

Nginx 服务器的反向代理服务是其最常用的重要功能，由反向代理服务也可以衍生出很多与此相关的 Nginx 服务器重要功能，比如后面会介绍的负载均衡。当然在了解反向代理

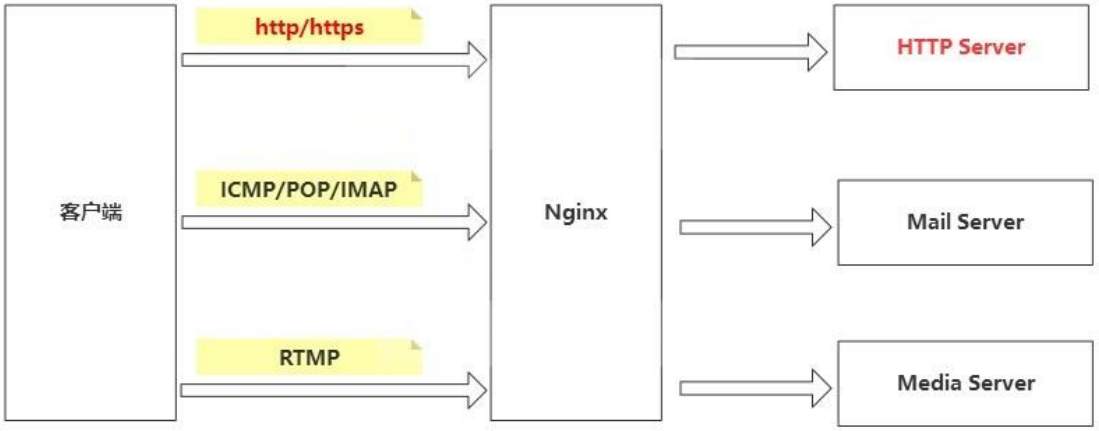
之前，我们需要先知道什么是代理以及什么是正反向代理。

1、代理概述

举一个现实生活中的例子：比如我们要买一间二手房，虽然我们可以自己去找房源，但是这太花费时间精力了，而且房屋质量检测以及房屋过户等一系列手续也都得我们去办，再说现在这个社会，等我们找到房源，说不定房子都已经涨价了，那么怎么办呢？最简单快捷的方法就是找二手房中介公司（为什么？别人那里房源多啊），于是我们就委托中介公司来给我找合适的房子，以及后续的质量检测过户等操作，我们只需要选好自己想要的房子，然后交钱就行了。

代理简单来说，就是如果我们想做什么，但又不想直接去做，那么这时候就找另外一个人帮我们去做。那么这个例子里面的中介公司就是给我们做代理服务的，我们委托中介公司帮我们找房子。

Nginx 主要能够代理如下几种协议，其中用到的最多的就是做 Http 代理服务器。



2、正向代理

弄清楚什么是代理了，那么什么又是正向代理呢？

这里我再举一个例子：大家都知道，现在国内是访问不了 Google 的，那么怎么才能访问 Google 呢？我们又想，美国人不是能访问 Google 吗（这不废话，Google 就是美国的），如果我们电脑的对外公网 IP 地址能变成美国的 IP 地址，那不就可以访问 Google 了。你很聪明，VPN 就是这样产生的。我们在访问 Google 时，先连上 VPN 服务器将我们的 IP 地址变成美国的 IP 地址，然后就可以顺利的访问了。

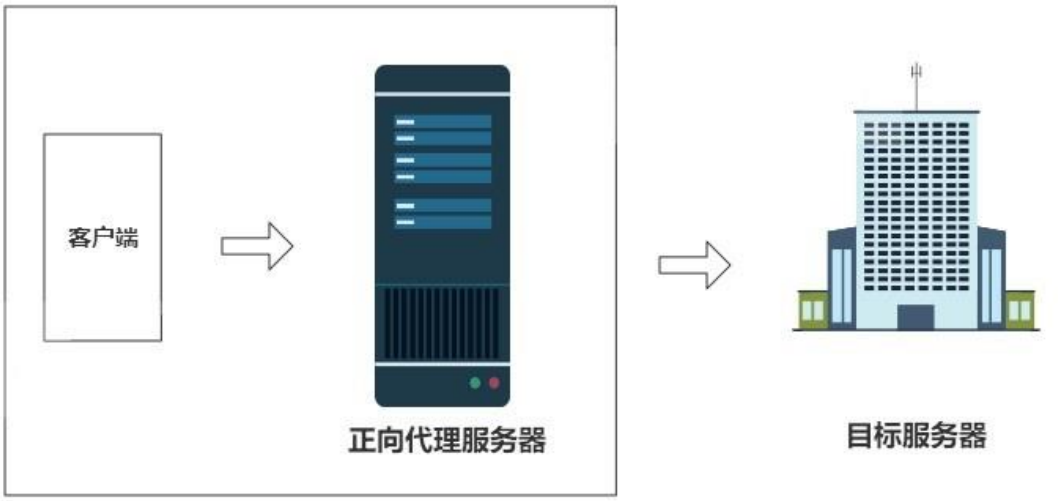
这里的 VPN 就是做正向代理的。正向代理服务器位于客户端和服务器之间，为了向服务器获取数据，客户端要向代理服务器发送一个请求，并指定目标服务器，代理服务器将目标服务器返回的数据转交给客户端。这里客户端是要进行一些正向代理的设置的。

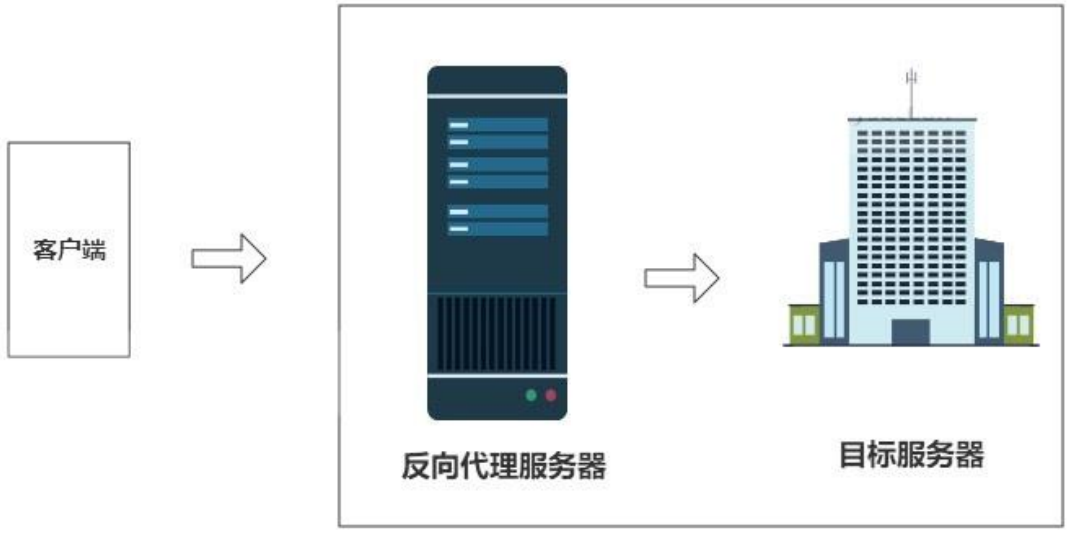
3、反向代理

反向代理，其实客户端对代理是无感知的，因为客户端不需要任何配置就可以访问，我们只需要将请求发送到反向代理服务器，由反向代理服务器去选择目标服务器获取数据后，

在返回给客户端，此时反向代理服务器和目标服务器对外就是一个服务器，暴露的是代理服务器地址，隐藏了真实服务器 IP 地址。

下面我们通过两张图来对比正向代理和方向代理：





理解这两种代理的关键在于代理服务器所代理的对象是什么，正向代理代理的是客户端，我们需要在客户端进行一些代理的设置。而反向代理代理的是服务器，作为客户端的我们是无法感知到服务器的真实存在的。

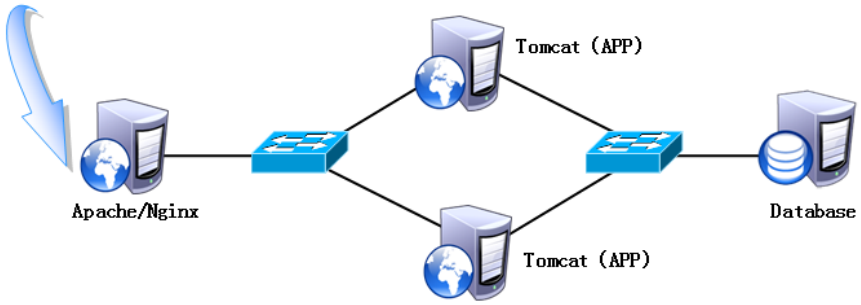
反向代理和正向代理的区别就是：正向代理代理客户端，反向代理代理服务器。

一台 Tomcat 站点由于可能出现单点故障及无法应付过多客户复杂多样的请求等问题，不能单独应用于生产环境下，所以需要一套可靠的解决方案来完善 web 站点架构。

Nginx 是一款非常优秀的 http 服务器软件，它能够支持高达 50000 个并发连接数的响应，拥有强大的静态资源处理能力，运行稳定，并且内存，CPU 等系统资源消耗非常低，目前很多大型网站都用 Nginx 服务器做后端网站程序的反向代理及负载均衡器，来提升整个站点的负载并发能力。

Nginx 服务器 CentOS 7.5 x86_64 192.168.200.111 nginx

Tomcat 服务器 1 CentOS 7.5 x86_64 192.168.200.112 jdk tomcat
Tomcat 服务器 2 CentOS 7.5 x86_64 192.168.200.113 jdk tomcat

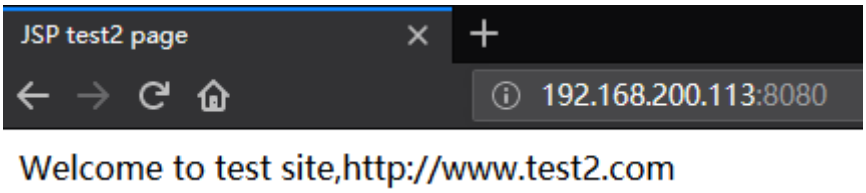


Tomcat 2 配置方法基本与 Tomcat 1 相同
1）关闭防火墙，selinux
2）安装 JDK，配置 Java 环境，版本保持一致
3）安装 Tomcat，版本与 Tomcat 1 保持一致
4）建立一个 web 目录，并在里面建立一个 webapp 目录，用于存放网站文件
5）在 webapp 目录下建立一个 index.jsp 的测试页面

```
[root@localhost ~]# tar xf apache-tomcat-8.5.16.tar.gz
[root@localhost ~]# mv apache-tomcat-8.5.16 /usr/local/tomcat8
[root@localhost ~]# cp /usr/local/tomcat8/conf/server.xml{,.bak}
[root@localhost ~]# vim /usr/local/tomcat8/conf/server.xml
148          <Host name="localhost"    appBase="webapps"
149                    unpackWARs="true" autoDeploy="true">
150                    <Context docBase="/web/webapp" path="" reloadable="flase" >
151                    </Context>
```

```
[root@localhost ~]# vim /web/webapp/index.jsp
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>JSP test2 page</title>
    </head>
    <body>
        <% out.println("Welcome to test site,http://www.test2.com");%>
    </body>
</html>
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
Using CATALINA_BASE:    /usr/local/tomcat8
Using CATALINA_HOME:    /usr/local/tomcat8
Using CATALINA_TMPDIR: /usr/local/tomcat8/temp
Using JRE_HOME:          /usr
Using CLASSPATH:
/usr/local/tomcat8/bin/bootstrap.jar:/usr/local/tomcat8/bin/tomcat-juli.jar
Tomcat started.
```

浏览器访问测试 http://192.168.200.113:8080



Nginx 服务器配置

```
[root@nginx ~]# yum -y install pcre-devel zlib-devel openssl-devel lrzsz
[root@nginx ~]# useradd -M -s /sbin/nologin nginx
[root@nginx ~]# tar xf nginx-1.14.2.tar.gz -C /usr/src/
[root@nginx ~]# cd /usr/src/nginx-1.14.2/
[root@nginx nginx-1.14.2]# ./configure --prefix=/usr/local/nginx    --user=nginx --group=nginx --with-file-aio --with-http_stub_status_module --with-http_ssl_module --with-http_flv_module --with-http_gzip_static_module && make && make install
--prefix=/usr/local/nginx            //指定安装目录
--user=nginx --group=nginx           //指定运行的用户和组
--with-file-aio                      //启用文件修改支持
--with-http_stub_status_module       //启用状态统计
--with-http_ssl_module               //启用 ssl 模块
--with-http_flv_module               //启用 flv 模块，提供寻求内存使用基于时间的偏移量文件
--with-http_gzip_static_module       //启用 gzip 静态压缩
```

配置 nginx.conf

```
[root@nginx nginx-1.14.2]# cp /usr/local/nginx/conf/nginx.conf{,.bak}
[root@nginx nginx-1.14.2]# vim /usr/local/nginx/conf/nginx.conf
user    nginx nginx;
worker_processes    2;
error_log    logs/error.log;
pid              logs/nginx.pid;

events {
      use epoll;
      worker_connections    10240;
}

http {
      include            mime.types;
      default_type    application/octet-stream;

      log_format    main    '$remote_addr - $remote_user [$time_local] "$request" '
                            '$status $body_bytes_sent "$http_referer" '
                            '"$http_user_agent" "$http_x_forwarded_for"';
```

```
    access_log  logs/access.log  main;
    sendfile        on;
    keepalive_timeout  65;
    upstream tomcat {
    server 192.168.200.112:8080 weight=1;
    server 192.168.200.113:8080 weight=1;
    }

    server {
        listen       80;
        server_name  localhost;

        location / {
            root   html;
            index  index.html index.htm;
        }
        location ~ \.jsp$ {
            proxy_pass http://tomcat;
        }
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
    }
}
[root@nginx nginx-1.14.2]# /usr/local/nginx/sbin/nginx -t
nginx: the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok
nginx: configuration file /usr/local/nginx/conf/nginx.conf test is successful

[root@nginx nginx-1.14.2]# /usr/local/nginx/sbin/nginx
[root@nginx nginx-1.14.2]# netstat -anpt |grep :80
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
4632/nginx: master

[root@nginx nginx-1.14.2]# ps aux | grep nginx | grep -v grep
root       4632  0.0  0.1  45936  1148 ?        Ss   23:41   0:00 nginx: master
process /usr/local/nginx/sbin/nginx -c /u
sr/local/nginx/conf/nginx.confnginx      4633  0.0  0.5  52276  5708 ?        S
23:41   0:00 nginx: worker process
nginx      4634  0.0  0.5  52276  5708 ?        S    23:41   0:00 nginx: worker
process
```

客户端测试：

打开浏览器访问: http://192.168.200.111/index.jsp、不断刷新可看到由于权重相同，页面会反复切换

Nginx 负载均衡算法

1、轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务，如果后端某台服务器死机，自动剔除故障系统，使用户访问不受影响。

```
upstream tomcat_server {
    server 192.168.200.112:8080 weight=1;
    server 192.168.200.113:8080 weight=1;
```

weight（轮询权值）

weight 的值越大分配到的访问概率越高，主要用于后端每台服务器性能不均衡的情况下。或者仅仅为在主从的情况下设置不同的权值，达到合理有效的地利用主机资源。

```
upstream tomcat_server {
    server 192.168.200.112:8080 weight=1;
    server 192.168.200.113:8080 weight=2;
```

2、least_conn

least-connected 方式可以更公平的将负载分配到多个机器上面。使用 least-connected，nginx 不会将请求分发到繁忙的机器上面，而且将新的请求分发的较清闲的机器上面。

```
upstream tomcat_server {
    least_conn;
    server 192.168.200.112:8080 weight=1;
    server 192.168.200.113:8080 weight=1;
```

3、ip_hash

每个请求按访问 IP 的哈希结果分配，使来自同一个 IP 的访客固定访问一台后端服务器，并且可以有效解决动态网页存在的 session 共享问题。

```
upstream tomcat_server {
    ip_hash;
    server 192.168.200.112:8080 weight=1;
```

```
    server 192.168.200.113:8080 weight=1;
```

4、fair

比 weight、ip_hash 更加智能的负载均衡算法，fair 算法可以根据页面大小和加载时间长短智能地进行负载均衡，也就是根据后端服务器的响应时间来分配请求，响应时间短的优先分配。Nginx 本身不支持 fair，如果需要这种调度算法，则必须安装 upstream_fair 模块。

```
upstream tomcat_server {
    fair;
    server 192.168.200.112:8080 weight=1;
    server 192.168.200.113:8080 weight=1;
```

5、url_hash

按访问的 URL 的哈希结果来分配请求，使每个 URL 定向到一台后端服务器，可以进一步提高后端缓存服务器的效率。Nginx 本身不支持 url_hash，如果需要这种调度算法，则必须安装 Nginx 的 hash 软件包。

```
upstream tomcat_server {
    hash $request_uri;
    hash_method crc32;
    server 192.168.200.112:8080;
    server 192.168.200.113:8080;
```

Nginx 负载均衡调度状态

在 Nginx upstream 模块中，可以设定每台后端服务器在负载均衡调度中的状态，常用的状态有：

down：表示当前的 server 暂时不参与负载均衡。

backup：预留的备份机器。当其他所有的非 backup 机器出现故障的时候，才会请求 backup 机器，因此这台机器的访问压力最低。

max_fails：允许请求失败的次数，默认为 1，当超过最大次数时，返回 proxy_next_upstream 模块定义的错误。

fail_timeout：请求失败超时时间，在经历了 max_fails 次失败后，暂停服务的时间。max_fails 和 fail_timeout 可以一起使用。

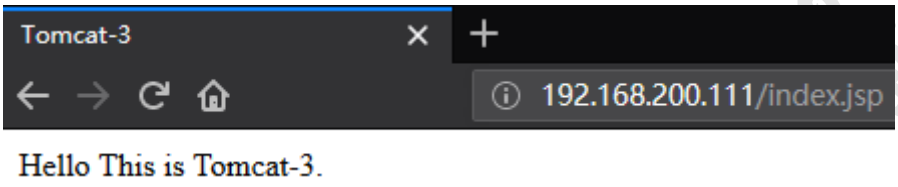
```
[root@localhost nginx-1.14.2]# vim /usr/local/nginx/conf/nginx.conf  修改为以下配置
    upstream tomcat_server {
        server 192.168.200.112:8080 weight=1;
        server 192.168.200.112:8090 weight=1;
        server 192.168.200.112:8070 weight=1 backup;
    }
[root@localhost nginx-1.14.2]# killall -HUP nginx
```

```
[root@localhost tomcat3]# /usr/local/tomcat_instance/tomcat1/bin/shutdown.sh
[root@localhost tomcat3]# /usr/local/tomcat_instance/tomcat2/bin/shutdown.sh
```

```
[root@localhost tomcat3]# netstat -lnpt  | grep java
tcp6       0      0 127.0.0.1:8004          :::*                    LISTEN
66235/java
tcp6       0      0 :::8070                 :::*                    LISTEN
66235/java
tcp6       0      0 :::8008                 :::*                    LISTEN
66235/java
```

客户端测试：

打开浏览器访问: http://192.168.200.111/index.jsp、不断刷新可看到由于 tomcat1 和 tomcat2，页面会切换到 tomcat3 实例页面。



在 nginx 中 HTTP, HTTPS, FastCGI, uwsgi, SCGI, and memcached 的负载均衡都是通过反向代理实现的。

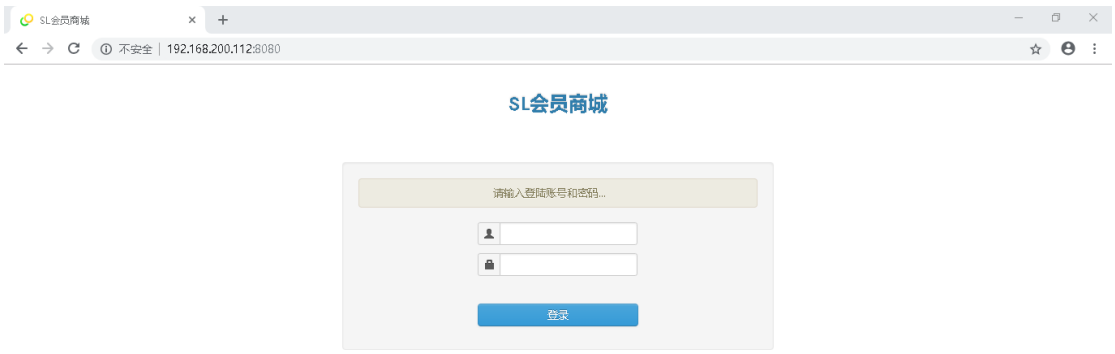
要配置 https 的负载均衡只需要将 http 协议改 https 就可以了，其他配置不变。

要实现 FastCGI, uwsgi, SCGI, or memcached 的负载均衡可以分别使用 fastcgi_pass, uwsgi_pass, scgi_pass, and memcached_pass 指令。

Tomcat 连接数据库

```
[root@localhost ~]# tar xf SLSaleSystem.tar.gz -C /web/webapp/
[root@localhost ~]# ls /web/webapp/SLSaleSystem/
logs  META-INF  statics  WEB-INF
[root@localhost ~]# vim /usr/local/tomcat8/conf/server.xml
<Context docBase="/web/webapp/SLSaleSystem" path="" reloadable="flase" >
</Context>
[root@localhost ~]# /usr/local/tomcat8/bin/shutdown.sh
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
```

浏览器访问：http://192.168.200.112:8080



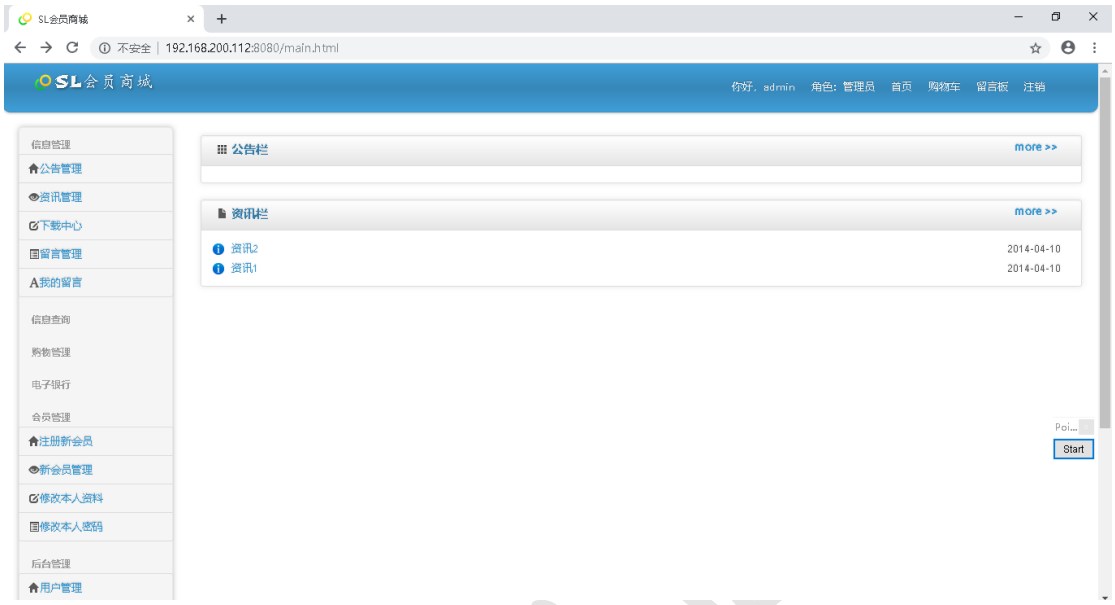
192.168.200.117

```
[root@localhost ~]# yum -y install mariadb-server mariadb
[root@localhost ~]# systemctl start mariadb
[root@localhost ~]# mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 5.5.56-MariaDB MariaDB Server

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database slsaledb;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> grant all on slsaledb.* to admin@'%' identified by '123456';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> flush privileges;
Query OK, 0 rows affected (0.00 sec)

[root@localhost ~]# mysql -uroot < slsaledb-2014-4-10.sql
```

```
[root@localhost ~]# vim /web/webapp/SLSaleSystem/WEB-INF/classes/jdbc.properties
driverClassName=com.mysql.jdbc.Driver
url=jdbc\:mysql\://192.168.200.117\:3306/slsaledb?useUnicode\=true&characterEncoding\=UTF-8
uname=admin
password=123456
minIdle=10
maxIdle=50
initialSize=5
maxActive=100
maxWait=100
removeAbandonedTimeout=180
```

```
removeAbandoned=true
[root@localhost ~]# /usr/local/tomcat8/bin/shutdown.sh
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
```

用户密码 admin、123456



Tomcat 配置 SSL 加密

说明：当通过 Nginx 调度 tomcat 进行工作时，如果 Nginx 配置了 ssl，Tomcat 是无需配置的。
扩展实验：Nginx+OpenSSL 实现 https 协议
创建加密用的私钥和证书文件

```
[root@localhost ~]# keytool -genkeypair -alias tomcat -keyalg RSA -keystore
/usr/local/tomcat8/keystore
输入密钥库口令:
再次输入新口令:
您的名字与姓氏是什么?
  [Unknown]:  crushlinux
您的组织单位名称是什么?
  [Unknown]:  cloud
您的组织名称是什么?
  [Unknown]:  cloud
您所在的城市或区域名称是什么?
  [Unknown]:  beijing
您所在的省/市/自治区名称是什么?
  [Unknown]:  haidian
该单位的双字母国家/地区代码是什么?
  [Unknown]:  CN
CN=crushlinux, OU=cloud, O=cloud, L=beijing, ST=haidian, C=CN 是否正确?
  [否]:  y
```

```
输入 <tomcat> 的密钥口令
    (如果和密钥库口令相同，按回车):
再次输入新口令:

Warning:
JKS 密钥库使用专用格式。建议使用 "keytool -importkeystore -srckeystore
/usr/local/tomcat8/keystore -destkeystore /usr/local/tomcat8/keystore -deststoretype
pkcs12" 迁移到行业标准格式 PKCS12。
```

修改 server.xml 配置文件，创建支持加密连接的 Connector

```
[root@localhost ~]# vim /usr/local/tomcat8/conf/server.xml
```

```
<!--
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
           maxThreads="150" SSLEnabled="true">
    <SSLHostConfig>
        <Certificate certificateKeystoreFile="conf/localhost-rsa.jks"
                     type="RSA" />
    </SSLHostConfig>
</Connector>
-->
```

将上述配置注释去掉并修改为如下配置

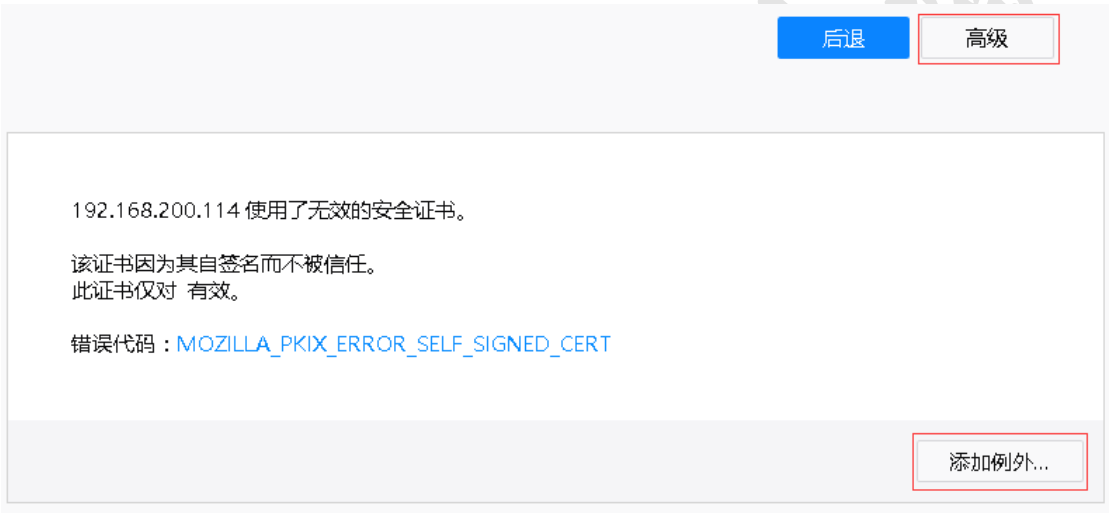
```
    <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
                maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS" keystoreFile="/usr/local/tomcat8/keystore"
keystorePass="123456">
</Connector>
[root@localhost conf]# /usr/local/tomcat8/bin/shutdown.sh
[root@localhost conf]# /usr/local/tomcat8/bin/startup.sh
```

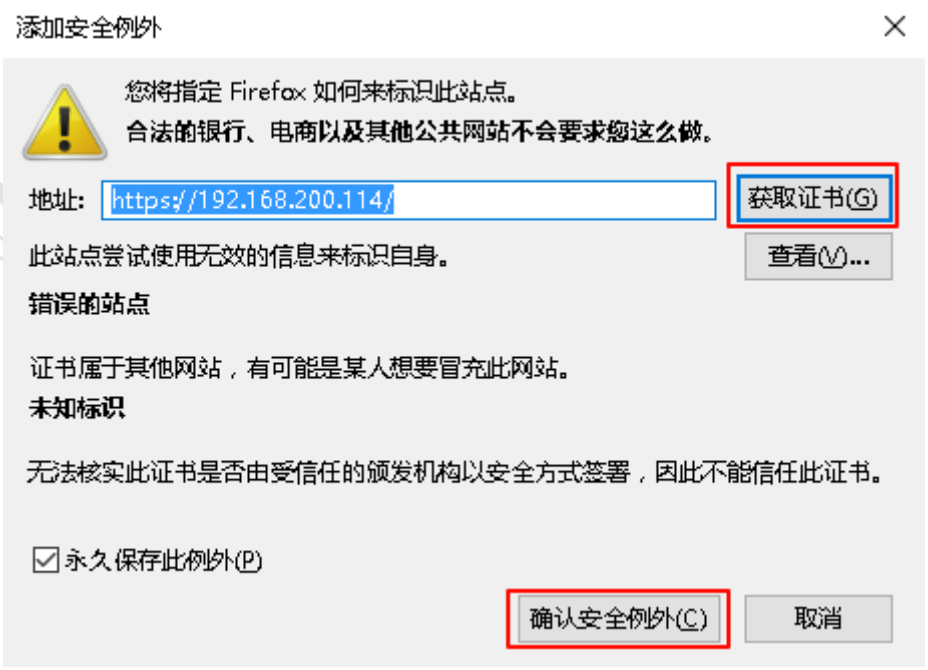
密码和证书的位置根据个人的具体环境而设置，属性参数如下所述：

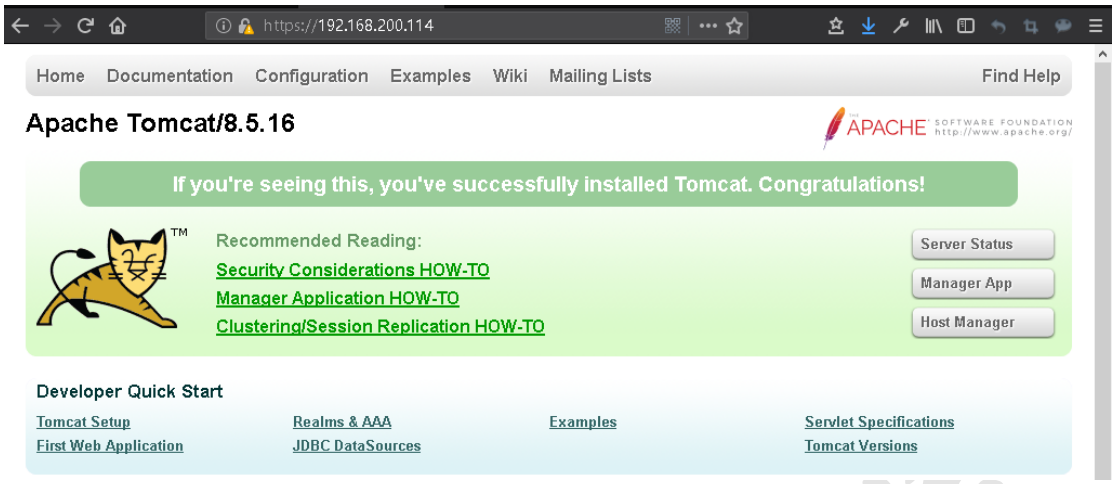
属性	描述
clientAuth	如果设为 true，表示 Tomcat 要求所有的 SSL 客户出示安全证书，对 SSL 客户进行身份验证
keystoreFile	指定 keystore 文件的存放位置，可以指定绝对路径，也可以指定相对于<CATALINA_HOME>（Tomcat 安装目录）环境变量 的相对路径。如果此项没有设定，默认情况下，Tomcat 将从当前操作系统用户的用户目录下读取名为“.keystore”的文件。
keystorePass	指定 keystore 的密码，如果此项没有设定，在默认情况下，Tomcat 将使用“changeit”作为默认密码。
sslProtocol	指定套接字（Socket）使用的加密/解密协议，默认值为 TLS，用户不应该修改这个默认值。
ciphers	指定套接字可用的用于加密的密码清单，多个密码间以逗号（,）分隔。如果此项没有设定，在默认情况下，套接字可以使用任意一个可用的密码。

建议用 google 浏览器访问：https://192.168.200.114:8443


https://192.168.200.114
您的连接并不安全
192.168.200.114 的管理员没有正确配置好此网站。为避免您的信息失窃，Firefox 并未与此网站建立连接。
详细了解...
报告此类错误，帮助 Mozilla 识别与拦截恶意网站
后退
高级


后退
高级
192.168.200.114 使用了无效的安全证书。
该证书因为其自签名而不被信任。
此证书仅对 有效。
错误代码：MOZILLA_PKIX_ERROR_SELF_SIGNED_CERT
添加例外...


添加安全例外
您将指定 Firefox 如何来标识此站点。
合法的银行、电商以及其他公共网站不会要求您这么做。
地址: https://192.168.200.114/
获取证书(G)
此站点尝试使用无效的信息来标识自身。
查看(V)...
错误的站点
证书属于其他网站，有可能是某人想要冒充此网站。
未知标识
无法核实此证书是否由受信任的颁发机构以安全方式签署，因此不能信任此证书。
永久保存此例外(P)
确认安全例外(C)
取消



Tomcat 虚拟主机

虚拟主机用于在一台物理机上搭建多个 web 站点，每个 web 站点独立运行，互不干扰，这些站点就是"虚拟主机"。

基于域名的虚拟主机

多个域名解析到同一个 IP 地址，在 WEB 服务器里添加多个站点，每个站点绑定一个域名。HTTP 协议请求里包含了域名信息，当 WEB 服务器收到访问请求时，就可以根据不同的域名来访问不同的网站。

```
[root@localhost ~]# tar xf apache-tomcat-8.5.16.tar.gz
[root@localhost ~]# mv apache-tomcat-8.5.16 /usr/local/tomcat8
```

配置域名与 Ip 的映射管理

对于本地局域网我们使用在 host 文件中添加。对于大型网络或者外网网络则需要配置 DNS 服务器中 Ip 地址与域名的映射关系。

```
[root@localhost ~]# tail -2 /etc/hosts
192.168.200.114    www.crushlinux.com
192.168.200.114    www.cloud.com
```

修改 server.xml

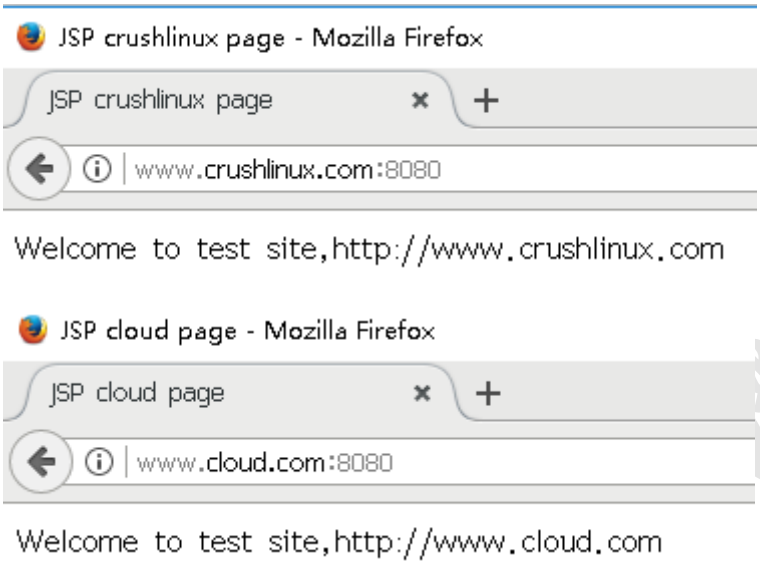
```
[root@localhost conf]# cp server.xml server.xml_$(date +%F)
[root@localhost conf]# vim server.xml
准备 jsp 网页文件
        <Host name="www.crushlinux.com"    appBase="webapps"
                unpackWARs="true" autoDeploy="true">
                <Context docBase="/web/crushlinux" path="" reloadable="flase" />
          <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
                    prefix="localhost_access_log" suffix=".txt"
```

```
                pattern="%h %l %u %t &quot;%r&quot; %s %b" />
        </Host>

        <Host name="www.cloud.com"    appBase="webapps"
                unpackWARs="true" autoDeploy="true">
                <Context docBase="/web/cloud" path="" reloadable="flase" />
          <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
                prefix="localhost_access_log" suffix=".txt"
                pattern="%h %l %u %t &quot;%r&quot; %s %b" />
        </Host>
[root@localhost conf]# mkdir -pv /web/{crushlinux,cloud}
```

```
[root@localhost conf]# vim /web/crushlinux/index.jsp
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>JSP crushlinux page</title>
    </head>
    <body>
        <% out.println("Welcome to test site,http://www.crushlinux.com");%>
    </body>
</html>

[root@localhost conf]# vim /web/cloud/index.jsp
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>JSP cloud page</title>
    </head>
    <body>
        <% out.println("Welcome to test site,http://www.cloud.com");%>
    </body>
</html>

[root@localhost conf]# ../bin/startup.sh
Using CATALINA_BASE:    /usr/local/tomcat8
Using CATALINA_HOME:    /usr/local/tomcat8
Using CATALINA_TMPDIR: /usr/local/tomcat8/temp
Using JRE_HOME:          /usr
Using CLASSPATH:
/usr/local/tomcat8/bin/bootstrap.jar:/usr/local/tomcat8/bin/tomcat-
juli.jarTomcat started.
```

访问测试

http://www.crushlinux.com:8080
http://www.cloud.com:8080



基于端口的虚拟主机

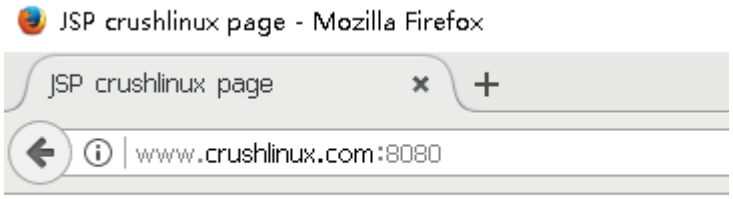
主机只拥有一个 IP 地址，通过不同的端口实现不同 WEB 站点的访问。
在 server.xml 中设置两个 service 组件

```
[root@localhost conf]# vim server.xml
  <Service name="Catalina1">
    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
    <Engine name="Catalina" defaultHost="localhost">
      <Realm className="org.apache.catalina.realm.LockOutRealm">
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
               resourceName="UserDatabase"/>
      </Realm>

      <Host name="www.crushlinux.com"    appBase="webapps"
            unpackWARs="true" autoDeploy="true">
            <Context docBase="/web/crushlinux" path="" reloadable="flase" />
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log" suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
      </Host>
    </Engine>
  </Service>
```

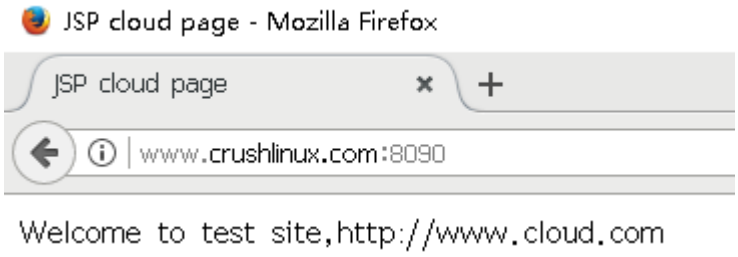
```
<Service name="Catalina2">
    <Connector port="8090" protocol="HTTP/1.1"
                connectionTimeout="20000"
                redirectPort="8443" />
    <Connector port="8010" protocol="AJP/1.3" redirectPort="8443" />
    <Engine name="Catalina" defaultHost="localhost">
      <Realm className="org.apache.catalina.realm.LockOutRealm">
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                resourceName="UserDatabase"/>
      </Realm>

      <Host name="www.crushlinux.com"    appBase="webapps"
            unpackWARs="true" autoDeploy="true">
            <Context docBase="/web/cloud" path="" reloadable="flase" />
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
                prefix="localhost_access_log" suffix=".txt"
                pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
      </Host>
    </Engine>
  </Service>
```

访问测试

http://www.crushlinux.com:8080



http://www.crushlinux.com:8090



基于 IP 地址的虚拟主机

服务器使用多网卡配置多个 IP 地址，然后配置 WEB 服务器，把多个网站绑定在不同的 IP 上。（该方式浪费 Ip 资源，Tomcat 不支持该方式）。

Tomcat 多实例配置

WEB 应用程序需要部署在服务器上时 Tomcat 软件的部署方式可以分为以下几种：

- 单实例单应用（webapps/a）
- 单实例多应用（webapps/{a,b}）
- 多实例单应用
- 多实例多应用

实例的概念可以先理解为一个 Tomcat 目录中的一个工作子目录

- 单实例单应用：比较常用的一种方式，只需要把做好的 war 包丢在 webapps 目录下，执行启动 Tomcat 的脚本就行了。
- 单实例多应用:有两个不同的 Web 项目的 war 包,还是只需要丢在 webapps 目录下，执行启动 Tomcat 的脚本，访问不同项目会加载不同的 WEB 虚拟目录。这种方式在生产环境中要慎用，因为重启或挂掉 Tomcat 程序后会影响另外一个应用的访问。
- 多实例单应用：多个 Tomcat 实例部署同一个项目，端口号不同，可以利用 Nginx 做负载均衡。
- 多实例多应用：多个 Tomcat 实例部署多个不同的项目。这种模式在服务器资源有限，或者对服务器资源要求并不是很高的情况下,可以实现多个不同项目部署在同一台服务器上的需求，来实现资源使用的最大化。

a、解压并部署 tomcat 程序

```
[root@localhost ~]# tar xf apache-tomcat-8.5.16.tar.gz
[root@localhost ~]# mv apache-tomcat-8.5.16 /usr/local/
```

b、创建 2 个实例的工作目录

```
[root@localhost ~]# mkdir -p /usr/local/tomcat_instance/tomcat1
[root@localhost ~]# mkdir -p /usr/local/tomcat_instance/tomcat2
```

c、拷贝 tomcat 程序目录下的 conf 目录分别放入 2 个实例目录中

```
[root@localhost ~]# cp -R /usr/local/apache-tomcat-8.5.16/conf/
/usr/local/tomcat_instance/tomcat1
[root@localhost ~]# cp -R /usr/local/apache-tomcat-8.5.16/conf/
/usr/local/tomcat_instance/tomcat2
```

d、在 tomcat 实例目录下的 bin 目录中创建实例启动和停止脚本

启动脚本

```
[root@localhost ~]# mkdir /usr/local/tomcat_instance/{tomcat1,tomcat2}/bin
[root@localhost ~]# touch /usr/local/tomcat_instance/{tomcat1,tomcat2}/bin/startup.sh
[root@localhost ~]# chmod +x /usr/local/tomcat_instance/{tomcat1,tomcat2}/bin/startup.sh
```

注意 CATALINA_BASE 所指向的 tomcat 实例目录路径：

```
[root@localhost ~]# vim /usr/local/tomcat_instance/tomcat1/bin/startup.sh
```

```
#!/bin/bash
export CATALINA_HOME="/usr/local/apache-tomcat-8.5.16"
export CATALINA_BASE="/usr/local/tomcat_instance/tomcat1"
export CATALINA_TMPDIR="$CATALINA_BASE/temp"
export CATALINA_PID="$CATALINA_BASE/bin/tomcat.pid"
export JAVA_OPTS="-server -Xms1024m -Xmx1024m -Djava.awt.headless=true -
Dtomcat.name=tomcat1"

#创建 logs 目录
if [ ! -d "$CATALINA_BASE/logs" ]; then
   mkdir $CATALINA_BASE/logs
fi

#创建 temp 目录
if [ ! -d "$CATALINA_BASE/temp" ]; then
   mkdir $CATALINA_BASE/temp
fi

#  调用 tomcat 启动脚本
bash $CATALINA_HOME/bin/startup.sh "$@"

[root@localhost ~]# vim /usr/local/tomcat_instance/tomcat2/bin/startup.sh
#!/bin/bash
export CATALINA_HOME="/usr/local/apache-tomcat-8.5.16"
export CATALINA_BASE="/usr/local/tomcat_instance/tomcat2"
export CATALINA_TMPDIR="$CATALINA_BASE/temp"
export CATALINA_PID="$CATALINA_BASE/bin/tomcat.pid"
export JAVA_OPTS="-server -Xms1024m -Xmx1024m -Djava.awt.headless=true -
Dtomcat.name=tomcat2"

#创建 logs 目录
if [ ! -d "$CATALINA_BASE/logs" ]; then
   mkdir $CATALINA_BASE/logs
fi

#创建 temp 目录
if [ ! -d "$CATALINA_BASE/temp" ]; then
   mkdir $CATALINA_BASE/temp
fi

#  调用 tomcat 启动脚本
bash $CATALINA_HOME/bin/startup.sh "$@"
```

停止脚本

```
[root@localhost ~]# touch /usr/local/tomcat_instance/{tomcat1,tomcat2}/bin/shutdown.sh
[root@localhost ~]# chmod +x
/usr/local/tomcat_instance/{tomcat1,tomcat2}/bin/shutdown.sh
[root@localhost ~]# vim /usr/local/tomcat_instance/tomcat1/bin/shutdown.sh
#!/bin/bash
export CATALINA_HOME="/usr/local/apache-tomcat-8.5.16"
export CATALINA_BASE="/usr/local/tomcat_instance/tomcat1"
export CATALINA_TMPDIR="$CATALINA_BASE/temp"
export CATALINA_PID="$CATALINA_BASE/bin/tomcat.pid"

bash $CATALINA_HOME/bin/shutdown.sh "$@"

[root@localhost ~]# vim /usr/local/tomcat_instance/tomcat2/bin/shutdown.sh
#!/bin/bash
export CATALINA_HOME="/usr/local/apache-tomcat-8.5.16"
export CATALINA_BASE="/usr/local/tomcat_instance/tomcat2"
export CATALINA_TMPDIR="$CATALINA_BASE/temp"
export CATALINA_PID="$CATALINA_BASE/bin/tomcat.pid"

bash $CATALINA_HOME/bin/shutdown.sh "$@"
```

e、修改每个 tomcat 实例中 server.xml 中的端口（分别修改以上三个端口（Server port、Connector port、AJP），不要和其它实例的端口或系统已经占用的端口发生冲突）。

```
[root@localhost ~]# cat /usr/local/tomcat_instance/tomcat2/conf/server.xml
<?xml version="1.0" encoding="UTF-8"?>
<Server port="8006" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
  <GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>
  <Service name="Catalina">
    <Connector port="8090" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    <Connector port="8010" protocol="AJP/1.3" redirectPort="8443" />
    <Engine name="Catalina" defaultHost="localhost">
```

```
        <Realm className="org.apache.catalina.realm.LockOutRealm">
          <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                 resourceName="UserDatabase"/>
        </Realm>
        <Host name="localhost"  appBase="webapps"
              unpackWARs="true" autoDeploy="true">
          <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
                 prefix="localhost_access_log" suffix=".txt"
                 pattern="%h %l %u %t &quot;%r&quot; %s %b" />

        </Host>
      </Engine>
    </Service>
</Server>
```

注意 server.xml 中 Host 节点的 appBase 属性的值指向。

f、创建测试页

```
[root@localhost ~]# mkdir -p /usr/local/tomcat_instance/{tomcat1,tomcat2}/webapps/ROOT
[root@localhost ~]# vim /usr/local/tomcat_instance/tomcat1/webapps/ROOT/index.jsp
<html>
<title>Tomcat-1</title>
<body>
      Hello This is Tomcat-1.
</body>
</html>

[root@localhost ~]# vim /usr/local/tomcat_instance/tomcat2/webapps/ROOT/index.jsp
<html>
<title>Tomcat-2</title>
<body>
      Hello This is Tomcat-2.
</body>
</html>
```

g、启动 tomcat 实例

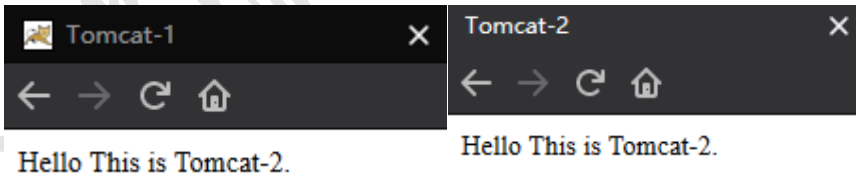
```
[root@localhost ~]# /usr/local/tomcat_instance/tomcat1/bin/startup.sh
Using CATALINA_BASE:     /usr/local/tomcat_instance/tomcat1
Using CATALINA_HOME:     /usr/local/apache-tomcat-8.5.16
Using CATALINA_TMPDIR: /usr/local/tomcat_instance/tomcat1/temp
Using JRE_HOME:            /usr
Using CLASSPATH:          /usr/local/apache-tomcat-
8.5.16/bin/bootstrap.jar:/usr/local/apache-tomcat-8.5.16/bin/tomcat-juli.jar
Using CATALINA_PID:      /usr/local/tomcat_instance/tomcat1/bin/tomcat.pid
Tomcat started.
```

```
[root@localhost ~]# /usr/local/tomcat_instance/tomcat2/bin/startup.sh
Using CATALINA_BASE:   /usr/local/tomcat_instance/tomcat2
Using CATALINA_HOME:   /usr/local/apache-tomcat-8.5.16
Using CATALINA_TMPDIR: /usr/local/tomcat_instance/tomcat2/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /usr/local/apache-tomcat-
8.5.16/bin/bootstrap.jar:/usr/local/apache-tomcat-8.5.16/bin/tomcat-juli.jar
Using CATALINA_PID:    /usr/local/tomcat_instance/tomcat2/bin/tomcat.pid
Tomcat started.
```

查看监听端口

```
[root@localhost ~]# netstat -lnpt  | grep java
tcp6       0      0 127.0.0.1:8005          :::*                    LISTEN
65208/java
tcp6       0      0 127.0.0.1:8006          :::*                    LISTEN
65246/java
tcp6       0      0 :::8009                 :::*                    LISTEN
65208/java
tcp6       0      0 :::8010                 :::*                    LISTEN
65246/java
tcp6       0      0 :::8080                 :::*                    LISTEN
65208/java
tcp6       0      0 :::8090                 :::*                    LISTEN
65246/java
```

h、访问测试实例内容



服务器启动之后，会分别在相应的实例目录下生成 logs、temp、work 目录。另外也可以在实例目录下创建 lib 目录，用于存放 app 的 jar。现在来看实例的安装目录，就和 tomcat 的安装包解压后的目录结构一样了，但所有实例共享同一套 tomcat 安装程序的 bin 和 lib。后面如果需要升级 tomcat 或修改 tomcat 脚本的相关配置，只需要更新这一套程序就行，也方便了日后的维护。多实例部署最大作用就是最大化利用服务器资源。

Nginx 基于 Tomcat 多实例的负载均衡

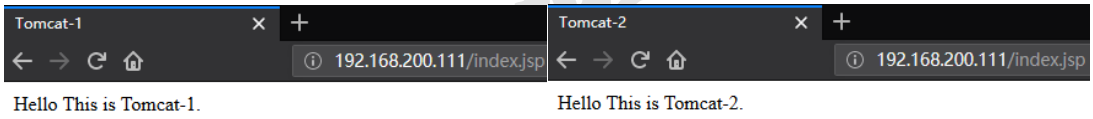
配置 nginx.conf

```
[root@nginx nginx-1.14.2]# cp /usr/local/nginx/conf/nginx.conf{,.bak}
[root@nginx nginx-1.14.2]# vim /usr/local/nginx/conf/nginx.conf
user    nginx nginx;
worker_processes    2;
error_log    logs/error.log;
pid             logs/nginx.pid;

events {
      use epoll;
      worker_connections    10240;
}

http {
      include            mime.types;
      default_type    application/octet-stream;

      log_format    main    '$remote_addr - $remote_user [$time_local] "$request" '
                            '$status $body_bytes_sent "$http_referer" '
                            '"$http_user_agent" "$http_x_forwarded_for"';

      access_log    logs/access.log    main;
      sendfile             on;
      keepalive_timeout    65;
      upstream tomcat_server {
      server 192.168.200.112:8080 weight=1;
      server 192.168.200.112:8090 weight=1;
      }

      server {
            listen           80;
            server_name    localhost;

            location / {
                  root     html;
                  index    index.html index.htm;
            }
            location ~ \.jsp$ {
                  proxy_pass http://tomcat_server;
            }
            error_page     500 502 503 504    /50x.html;
            location = /50x.html {
                  root     html;
            }
      }
```

```
}
[root@nginx nginx-1.14.2]# /usr/local/nginx/sbin/nginx -t
nginx: the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok
nginx: configuration file /usr/local/nginx/conf/nginx.conf test is successful

[root@nginx nginx-1.14.2]# /usr/local/nginx/sbin/nginx -c /usr/local/nginx/conf/nginx.conf
[root@nginx nginx-1.14.2]# netstat -anpt |grep :80
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
4632/nginx: master

[root@nginx nginx-1.14.2]# ps aux | grep nginx | grep -v grep
root       4632  0.0  0.1  45936  1148 ?        Ss   23:41   0:00 nginx: master
process /usr/local/nginx/sbin/nginx -c /u
sr/local/nginx/conf/nginx.confnginx        4633  0.0  0.5  52276  5708 ?        S
23:41   0:00 nginx: worker process
nginx      4634  0.0  0.5  52276  5708 ?        S    23:41   0:00 nginx: worker
process
```

客户端测试：

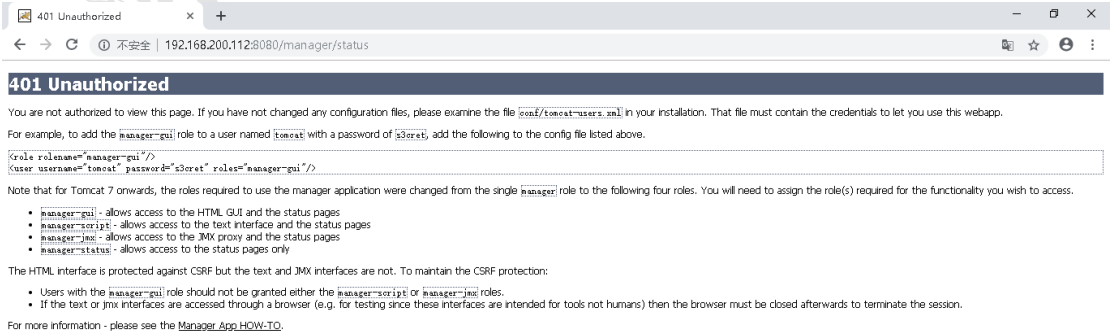
打开浏览器访问: http://192.168.200.111/index.jsp、不断刷新可看到由于权重相同，页面会反复切换



Tomcat 管理功能使用

注意：测试功能，生产环境不要用

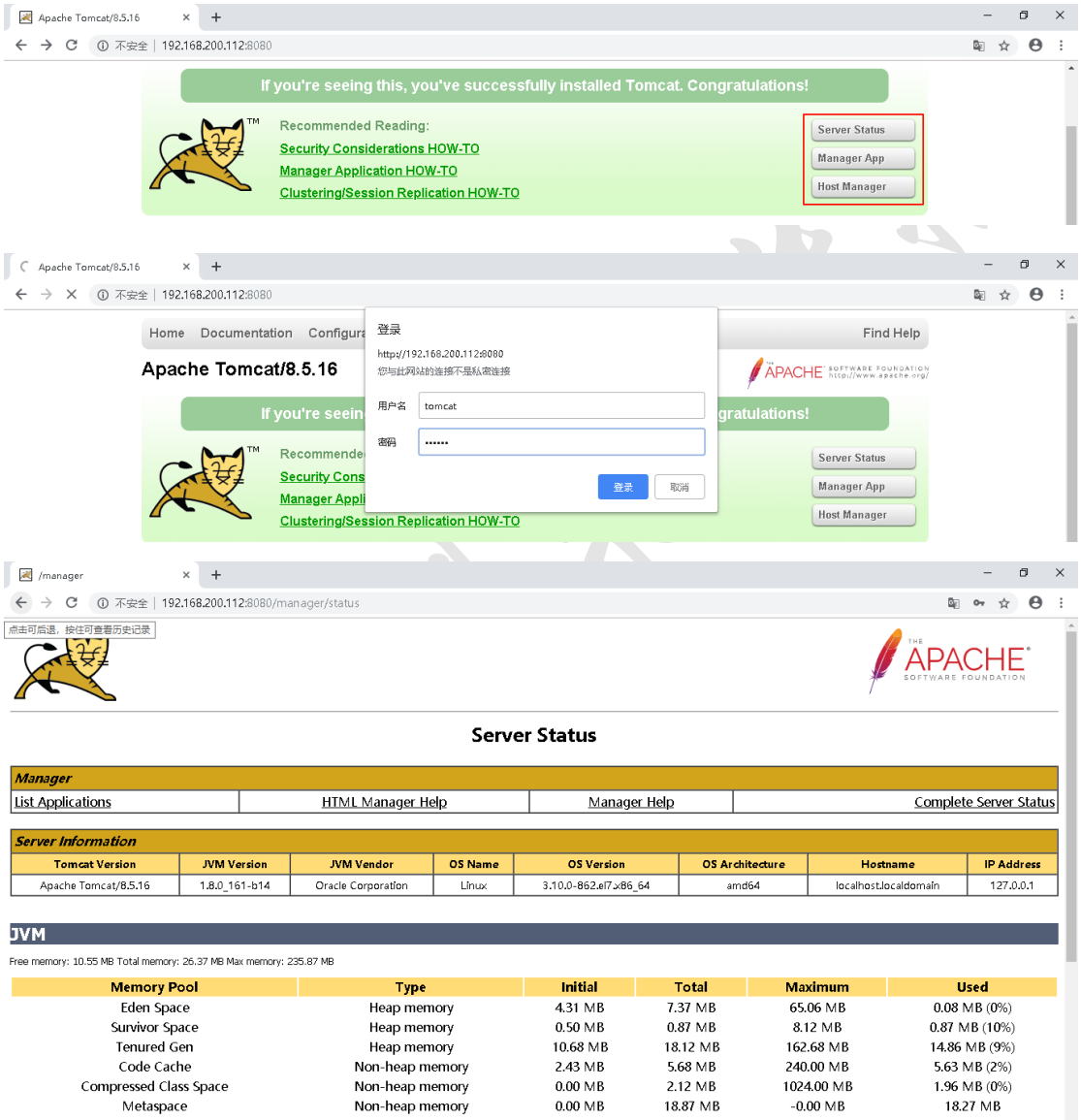
Tomcat 管理功能用于对 Tomcat 自身以及部署在 Tomcat 上的应用进行管理的 web 应用。在默认情况下是处于禁用状态的。如果需要开启这个功能，就需要配置管理用户，即配置 tomcat-users.xml 文件。




```
[root@localhost ~]# vim /usr/local/tomcat8/conf/tomcat-users.xml
  <role rolename="manager-gui"/>
  <role rolename="admin-gui"/>
  <user username="tomcat" password="tomcat" roles="manager-gui,admin-gui"/>
```

```
</tomcat-users>    # 在此行前加入上面三行
```

```
[root@localhost ~]# vim /usr/local/tomcat8/webapps/manager/META-INF/context.xml
<!--    <Valve className="org.apache.catalina.valves.RemoteAddrValve"
              allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->

[root@localhost ~]# /usr/local/tomcat8/bin/shutdown.sh
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
```



WEB 站点部署

上线的代码有两种方式：
第一种方式是直接将程序目录放在 webapps 目录下面，这种方式大家已经明白了，就不多说了。

第二种方式是使用开发工具将程序打包成 war 包，然后上传到 webapps 目录下面。

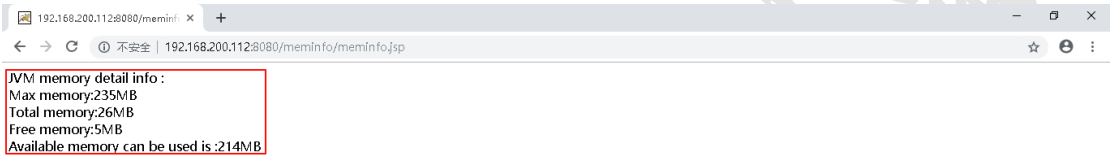
使用 war 包部署 web 站点
部署 tomcat 内存检测包
上传 meminfo.war 包到/usr/local/tomcat8/webapps 目录中

```
[root@localhost webapps]# ls
docs  examples  host-manager  manager  meminfo.war  ROOT
```

站点主动解压部署

```
[root@localhost webapps]# ls
docs  examples  host-manager  manager  meminfo  meminfo.war  ROOT
```

浏览器访问：
http://192.168.200.112:8080/meminfo/meminfo.jsp



部署开源站点（jpress）
jpress 官网：http://jpress.io
下载地址：https://github.com/JpressProjects/jpress

安装配置数据库

```
[root@localhost ~]# yum -y install mariadb-server mariadb
[root@localhost ~]# systemctl start mariadb
```

配置数据库

```
[root@localhost ~]# mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 5.5.56-MariaDB MariaDB Server

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database jpress DEFAULT CHARACTER SET utf8;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> grant all on jpress.* to jpress@'localhost' identified by '123456';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

```
MariaDB [(none)]> exit
Bye
```

jpress 站点上线

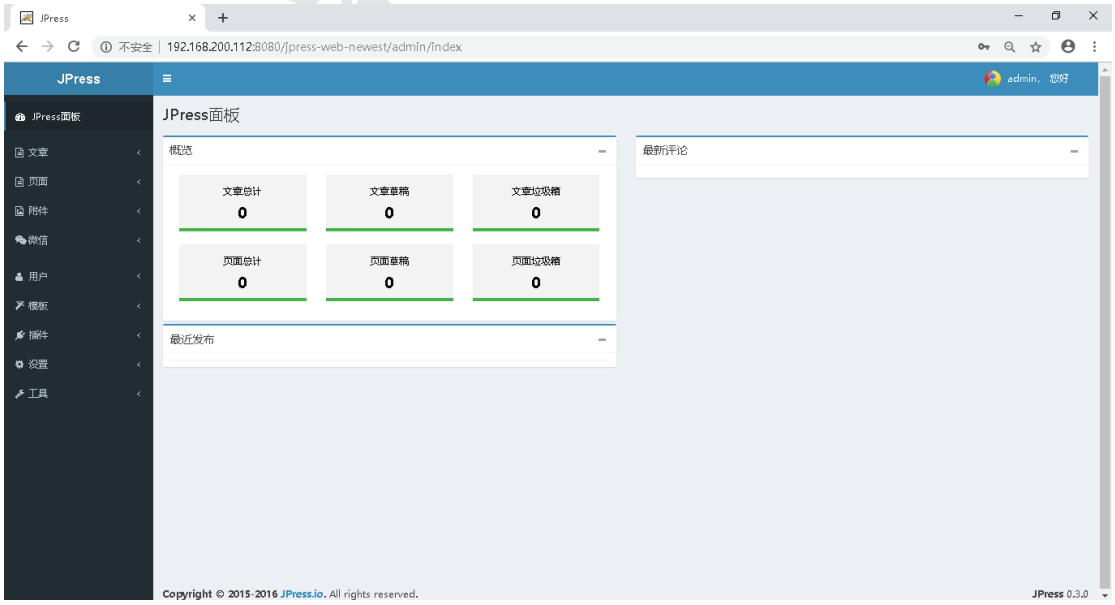
```
[root@localhost ~]# wget  https://github.com/JpressProjects/jpress/archive/0.4.0.tar.gz
[root@localhost ~]# tar xf 0.4.0.tar.gz
[root@localhost ~]# cd jpress-0.4.0/wars/
[root@localhost wars]# mv jpress-web-newest.war /usr/local/tomcat8/webapps/
[root@localhost wars]# ls /usr/local/tomcat8/webapps/
docs examples host-manager jpress-web-newest jpress-web-newest.war manager
ROOT
```

浏览器访问：http://192.168.200.112:8080/jpress-web-newest/install


JPress安装向导
安装即将完成。
恭喜，JPress安装即将完成...
网站名称
Crushlinux工作室
给网站起个漂亮名字吧
管理员
admin
极为重要，它就是您的网站的超级管理员的账号。
登录密码
admin
...及其密码
很重要：请务必牢记管理员账号及登录密码！！！
完成


JPress安装向导
安装完成。
安装完成！稍等几秒后自动刷新。若一直停留在此页面，请重启你的web容器。

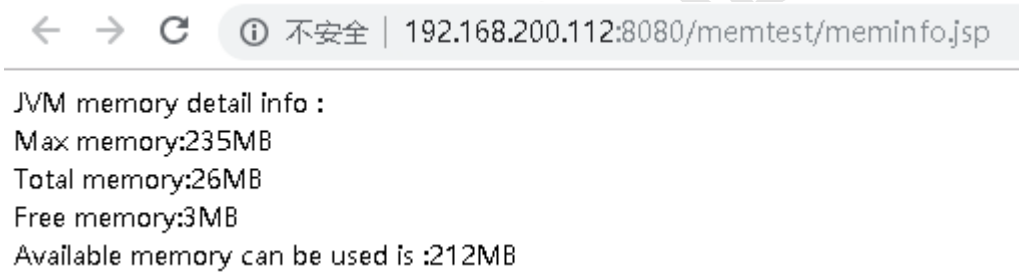

JPress
管理员登陆
用户名:
admin
密码:
记住我的登陆信息
登陆


JPress
admin，您好
JPress面板
概览
文章总计
0
文章草稿
0
文章垃圾箱
0
页面总计
0
页面草稿
0
页面垃圾箱
0
最新评论
最近发布
文章
页面
附件
微信
用户
模板
插件
设置
工具
Copyright © 2015-2016 JPress.io. All rights reserved.
JPress 0.3.0

监控 Tomcat 状态

方法一：开发 java 监控页面

```
[root@localhost ~]# mkdir /usr/local/tomcat8/webapps/memtest/
[root@localhost ~]# vim /usr/local/tomcat8/webapps/memtest/meminfo.jsp
<%
Runtime rtm = Runtime.getRuntime();
long mm = rtm.maxMemory()/1024/1024;
long tm = rtm.totalMemory()/1024/1024;
long fm = rtm.freeMemory()/1024/1024;

out.println("JVM memory detail info :<br>");
out.println("Max memory:"+mm+"MB"+"<br>");
out.println("Total memory:"+tm+"MB"+"<br>");
out.println("Free memory:"+fm+"MB"+"<br>");
out.println("Available memory can be used is :"+(mm+fm-tm)+"MB"+"<br>");
%>
```



方法二：使用 jps 命令进行监控

```
[root@localhost ~]# jps -lvm
80106 sun.tools.jps.Jps -lvm -Dapplication.home=/usr/local/java -Xms8m
79788 org.apache.catalina.startup.Bootstrap start -
Djava.util.logging.config.file=/usr/local/tomcat8/conf/logging.propertie
s -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -
Djdk.tls.ephemeralDHKeySize=2048 -
Djava.protocol.handler.pkgs=org.apache.catalina.webresources -
Dcatalina.base=/usr/local/tomcat8 -Dcatalina.home=/usr/local/tomcat8 -
Djava.io.tmpdir=/usr/local/tomcat8/temp
```

方法三：Tomcat 远程监控功能

修改配置文件，开启远程监控

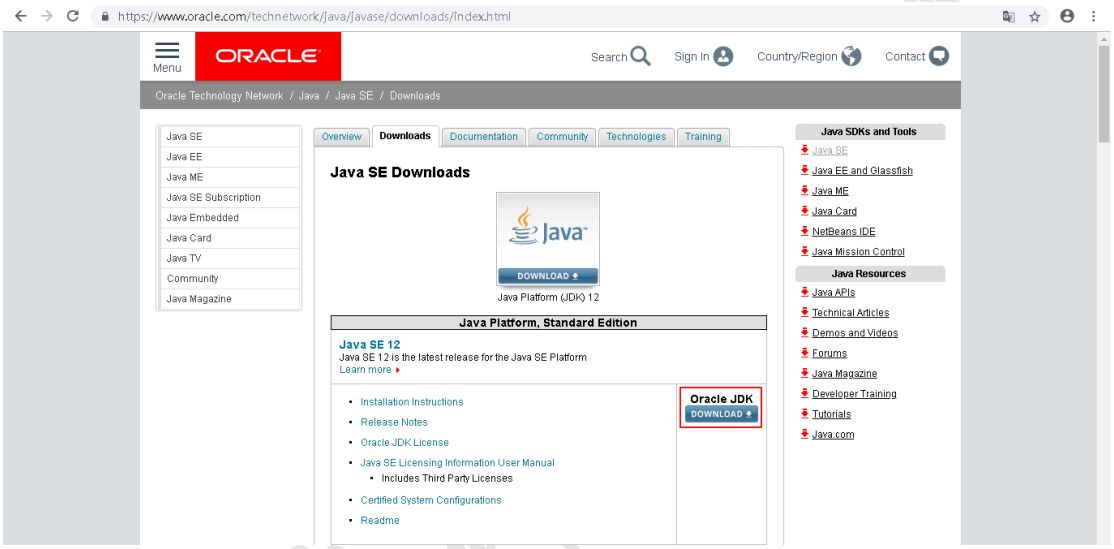
```
[root@localhost ~]# vim /usr/local/tomcat8/bin/catalina.sh
#!/bin/sh
CATALINA_OPTS="$CATALINA_OPTS
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=12345
-Dcom.sun.management.jmxremote.authenticate=false
```

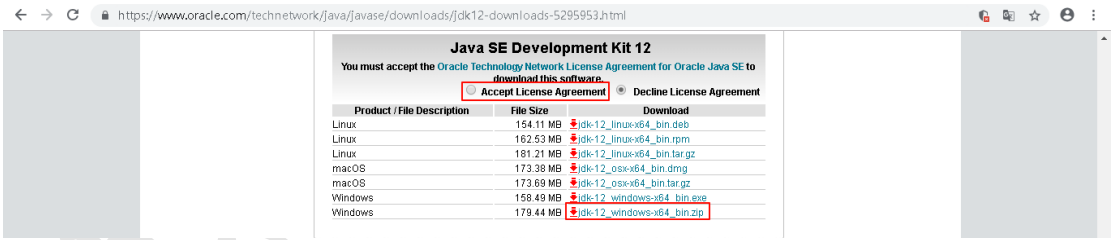
```
-Dcom.sun.management.jmxremote.ssl=false
-Djava.rmi.server.hostname=192.168.200.112"
[root@localhost ~]# /usr/local/tomcat8/bin/shutdown.sh
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
[root@localhost ~]# netstat -lnpt | grep 12345
tcp6       0      0 :::12345                :::*                    LISTEN
80262/java
```

在 windows 上监控 tomcat

注意：windwos 需要安装 jdk 环境！

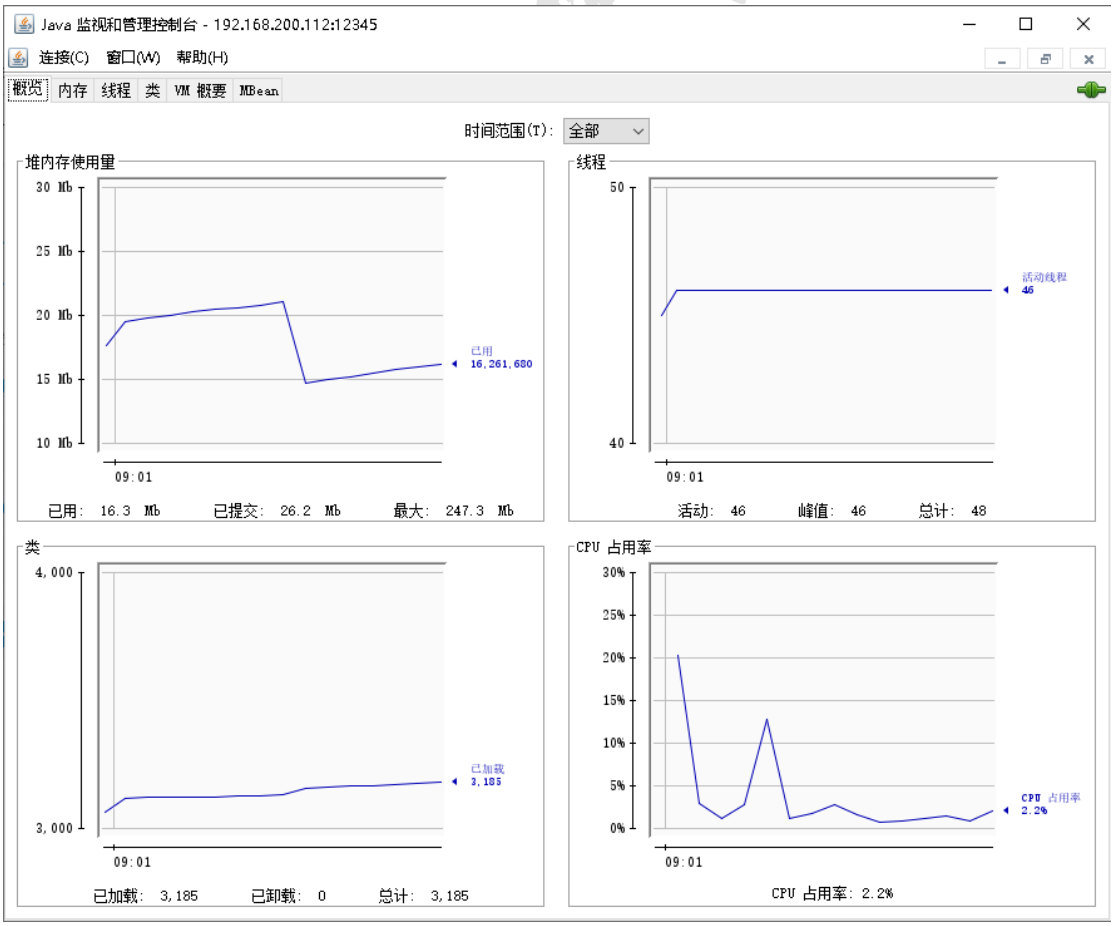
下载地址：http://www.oracle.com/technetwork/java/javase/downloads/index.html





将 jdk 直接解压到 C:\java 中，双击运行 C:\java\jdk-12\bin\jconsole.exe

连接成功即可进行监控，连接的时候注意端口信息。



Tomcat 故障排查步骤

a. 查看 catalina.out
b. 使用 sh show-busy-java-threads.sh 脚本进行检测
脚本下载地址
https://files.cnblogs.com/files/clsn/show-busy-java-threads.sh

Tomcat 安全优化

1.telnet 管理端口保护（强制）

类别	配置内容及说明	标准配置	备注
telnet 管理端口保护	1.修改默认的 8005 管理端口为不易猜测的端口（大于 1024）； 2.修改 SHUTDOWN 指令为其他字符串；	<Server port="8527" shutdown="crushlinux">	1.以上配置项的配置内容只是建议配置，可以按照服务实际情况进行合理配置，但要求端口配置在 8000~8999 之间；

2. ajp 连接端口保护（强制）

类别	配置内容及说明	标准配置	备注
Ajp 连接端口保护	1.修改默认的 ajp 8009 端口为不易冲突的大于 1024 端口； 2.通过 iptables 规则限制 ajp 端口访问的权限仅为线上机器；	<Connector port="8528" protocol="AJP/1.3" />	以上配置项的配置内容仅为建议配置，请按照服务实际情况进行合理配置，但要求端口配置在 8000~8999 之间； 保护此端口的目的在于防止线下的测试流量被 mod_jk 转发至线上 tomcat 服务器；

3. 禁用管理端（强制）

类别	配置内容及说明	标准配置	备注
禁用管理端	1. 删除默认的{Tomcat 安装目录}/conf/tomcat-users.xml 文件，重启 tomcat 后将会自动生成新的文件； 2. 删除{Tomcat 安装目录}/webapps 下默认的所有目录和文件； 3.将 tomcat 应用根目录配置为 tomcat 安装目录以外的目录；	<Context path="" docBase="/web/webapps" debug="0" reloadable="false" crossContext="true"/>	对于前端 web 模块，Tomcat 管理端属于 tomcat 的高危安全隐患，一旦被攻破，黑客通过上传 web shell 的方式将会直接取得服务器的控制权，后果极其严重；

4. 降权启动（强制）

类别	配置内容及说明	标准配置	备注
降权启动	1.tomcat 启动用户权限必须为非 root 权限，尽量降低 tomcat 启动用户的目录访问权限； 2.如需直接对外使用 80 端口，可通过普通账号启动后，配置 iptables 规则进行转发；		避免一旦 tomcat 服务被入侵，黑客直接获取高级用户权限危害整个 server 的安全；

```
useradd tomcat
cp -a /usr/local/tomcat8    /home/tomcat/tomcat8_1/
chown -R tomcat.tomcat    /home/tomcat/tomcat8_1/
su -c '/home/tomcat/tomcat8_1/bin/startup.sh' tomcat
ps -ef|grep tomcat
```

5.文件列表访问控制（强制）

类别	配置内容及说明	标准配置	备注
文件列表访问控制	1.conf/web.xml 文件中 default 部分 listings 的配置必须为 false；	<init-param> <param-name>listings</param-name> <param-value>false</param-value> </init-param>	false 为不列出目录文件，true 为允许列出，默认为 false；

```
[root@localhost ~]# vim /usr/local/tomcat8/conf/web.xml
110             <init-param>
111                 <param-name>listings</param-name>
112                 <param-value>false</param-value>
113             </init-param>
```

6. 版本信息隐藏（强制）

类别	配置内容及说明	标准配置	备注
版本信息隐藏	1. 修改 conf/web.xml，重定向 403、404 以及 500 等错误到指定的错误页面；2.也可以通过修改应用程序目录下的 WEB-INF/web.xml 下的配置进行错误页面的重定向；	<error-page> <error-code>403</error-code> <location>/forbidden.jsp</location> </error-page> <error-page> <error-code>404</error-code> <location>/notfound.jsp</location> </error-page> <error-page> <error-code>500</error-code> <location>/systembusy.jsp</location> </error-page>	在配置中对一些常见错误进行重定向，避免当出现错误时 tomcat 默认显示的错误页面暴露服务器和版本信息； 必须确保程序根目录下的错误页面已经存在；

7. Server header 重写（推荐）

类别	配置内容及说明	标准配置	备注
Server header 重写	在 HTTP Connector 配置中加入 server 的配置；	server="webserver"	当 tomcat HTTP 端口直接提供 web 服务时此配置生效，加入此配置，将会替换 http 响应 Server header 部分的默认配置，默认是 Apache-Coyote/1.1

8. 访问限制（可选）

类别	配置内容及说明	标准配置或操作	备注
访问限制	通过配置，限定访问的 ip 来源	<Context path="" docBase="/web/webapps" debug="0" reloadable="false" crossContext="true"> <Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="61.148.18.138,61.135.165.*" deny="*.*.*.*"/> </Context>	通过配置信任 ip 的白名单，拒绝非白名单 ip 的访问，此配置主要是针对高保密级别的系统，一般产品线不需要；

9. 起停脚本权限回收（推荐）

类别	配置内容及说明	标准配置或操作	备注
起停脚本权限回收	去除其他用户对 Tomcat 的 bin 目录下 shutdown.sh、startup.sh、catalina.sh 的可执行权限；	chmod -R 744 tomcat/bin/*	防止其他用户有起停线上 Tomcat 的权限；

10. 访问日志格式规范（推荐）

类别	配置内容及说明	标准配置或操作	备注
访问日志格式规范	开启 Tomcat 默认访问日志中的 Referer 和 User-Agent 记录	<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs" prefix="localhost_access_log." suffix=".txt" pattern="%h %l %u %t %r %s %b %{Referer}i %{UserAgent}i %D" resolveHosts="false"/>	开启 Referer 和 User-Agent 是为了一旦出现安全问题能够更好的根据日志进行问题排查；

11、页面超时

```
581     <session-config>
582         <session-timeout>30</session-timeout>
583     </session-config>
```

12、默认页面

```
[root@localhost ~]# vim /usr/local/tomcat8/conf/web.xml
4679     <welcome-file-list>
4680         <welcome-file>index.html</welcome-file>
4681         <welcome-file>index.htm</welcome-file>
4682         <welcome-file>index.jsp</welcome-file>
4683     </welcome-file-list>
```

13、配置网页传输压缩

注：如若有 apache、nginx 等做代理，tomcat 则不必配置传输压缩；

```
[root@localhost ~]# vim /usr/local/tomcat8/conf/server.xml
 69     <Connector port="8080" protocol="HTTP/1.1"
 70                connectionTimeout="20000"
 71                redirectPort="8443"
 72                compression="on"  #开启压缩
 73                compressionMinSize="50" #指定最小的压缩文件，单位是字节
 74                noCompressionUserAgents="gozilla, Traviata"    #此浏览器类型不进行压缩
 75                compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain" />  #文件的格式
```

```
[root@localhost ~]# /usr/local/tomcat8/bin/shutdown.sh
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
```

Tomcat 运行模式

1、同步概念：

- 同步 ： 自己亲自出马持银行卡到银行取钱（使用同步 IO 时，Java 自己处理 IO 读写）。
- 异步 ： 委托一小弟拿银行卡到银行取钱，然后给你（使用异步 IO 时，Java 将 IO 读写委托给 OS 处理，需要将数据缓冲区地址和大小传给 OS(银行卡和密码)，OS 需要支持异步 IO 操作 API）。
- 阻塞 ： ATM 排队取款，你只能等待（使用阻塞 IO 时，Java 调用会一直阻塞到读写完成才返回）。
- 非阻塞 ： 柜台取款，取个号，然后坐在椅子上做其它事，等号广播会通知你办理，没到号你就不能去，你可以不断问大堂经理排到了没有，大堂经理如果说还没到你就不能去（使用非阻塞 IO 时，如果不能读写 Java 调用会马上返回，当 IO 事件分发器会通知可读写时再继续进行读写，不断循环直到读写完成）。

2、Java 对 BIO、NIO、AIO 的支持：

- Java BIO ： 同步并阻塞，服务器实现模式为一个连接一个线程，即客户端有连接请求时服务器端就需要启动一个线程进行处理，如果这个连接不做任何事情会造成不必要的线程开销，当然可以通过线程池机制改善。
- Java NIO ： 同步非阻塞，服务器实现模式为一个请求一个线程，即客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有 I/O 请求时才启动一个线程进行处理。
- Java AIO（NIO.2） ： 异步非阻塞，服务器实现模式为一个有效请求一个线程，客户端的 I/O 请求都是由 OS 先完成了再通知服务器应用去启动线程进行处理

3、BIO、NIO、AIO 适用场景分析:

- **BIO**

BIO 方式适用于连接数目比较小且固定的架构，这种方式对服务器资源要求比较高，并发局限于应用中，JDK1.4 以前的唯一选择，但程序直观简单易理解，性能非常低下，没有经过任何优化处理和支持。

- NIO

Nio(new I/O),是 java SE1.4 及后续版本提供的一种新的 I/O 操作方式(即 java.nio 包及其子包)。Java nio 是一个基于缓冲区，并能提供非阻塞 I/O 操作的 java API，因此 nio 也被看成是 non-blocking I/O 的缩写。它拥有比传统 I/O 操作(bio)更好的并发运行性能。适用于连接数目多且连接比较短（轻操作）的架构，比如聊天服务器，并发局限于应用中。

- APR

安装起来最困难，但是从操作系统级别来解决异步的 IO 问题，大幅度的提高了性能

tomcat bio、nio、apr 软件测试版本（10分钟压测结果）							
tomcat默认首页							
tomcat 模式	并发数	样本数	平均响应时间(ms)	吞吐量(s)	偏移	错误率(%)	KB/sec
bio	200	620140	114	1033	194	0	11479
bio	300	596220	174	990	505	0.01	10997
bio	500	460704	645	764	1994	0.65	8443
bio	700	440252	947	730	2914	1.65	8002
bio	1000	255481	2350	422	5020	5.87	4471
bio	1300	185352	4227	305	6767	12.56	3049
bio	1500	253813	3583	413	6565	11.23	4173
nio	200	840157	121	1396	68	0	15512
nio	300	805874	60	1340	145	0	14889
nio	500	817107	235	1354	205	0.01	15041
nio	700	817243	265	1356	283	0	15062
nio	1000	736392	262	1218	505	0	13526
nio	1300	738334	250	1217	567	0	13519
nio	1500	726248	252	1200	663	0	13330
apr	200	882166	94	1463	73	0	16247
apr	300	711858	139	1175	268	0	13056
apr	500	701720	155	1158	410	0.03	12862
apr	700	896486	127	1482	297	0	16465
apr	1000	709752	207	1169	689	0	12981
apr	1300	629729	385	1037	1145	0	11518
apr	1500	627767	390	1028	1253	0.01	11419

Tomcat8 默认运行模式为 NIO

```
[root@localhost ~]# tail -1 /usr/local/tomcat8/logs/catalina.out
08-Apr-2019 11:45:01.589 信息 [main] org.apache.coyote.AbstractProtocol.start Starting
ProtocolHandler ["http-nio-8080"]
```

```
[root@localhost ~]# vim /usr/local/tomcat8/conf/server.xml
    <Connector port="8080" protocol="org.apache.coyote.http11.Http11NioProtocol"
               connectionTimeout="20000"
               redirectPort="8443"
               compression="on"
               compressionMinSize="50"
               noCompressionUserAgents="gozilla, traviata"
compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain" />
[root@localhost ~]# /usr/local/tomcat8/bin/shutdown.sh
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
[root@localhost ~]# tail -f /usr/local/tomcat8/logs/catalina.out
08-Apr-2019 11:56:02.441 信息 [main] org.apache.coyote.AbstractProtocol.start Starting
ProtocolHandler ["http-nio-8080"]
```

模式三 Apr：

```
[root@localhost ~]# rpm -e apr --nodeps
[root@localhost ~]# yum -y install apr    apr-devel
[root@localhost ~]# cp /usr/local/tomcat8/bin/tomcat-native.tar.gz /root/
[root@localhost ~]# tar xf tomcat-native.tar.gz
```

```
[root@localhost ~]# cd tomcat-native-1.2.12-src/native/
[root@localhost native]# ./configure --with-apr=/usr/bin/apr-1-config --with-java-
home=/usr/local/java && make && make install
----------------------------------------------------------------------
Libraries have been installed in:
    /usr/local/apr/lib

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the `-LLIBDIR'
flag during linking and do at least one of the following:
    - add LIBDIR to the `LD_LIBRARY_PATH' environment variable
      during execution
    - add LIBDIR to the `LD_RUN_PATH' environment variable
      during linking
    - use the `-Wl,-rpath -Wl,LIBDIR' linker flag
    - have your system administrator add LIBDIR to `/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------

[root@localhost tomcat-native-1.2.17-src]# cd
[root@localhost native]# vim /usr/local/tomcat8/bin/catalina.sh    ##末尾添加，设置环境变
量
CATALINA_OPTS="-Djava.library.path=/usr/local/apr/lib"
[root@localhost native]# vim /usr/local/tomcat8/conf/server.xml
69        <Connector port="8080" protocol="org.apache.coyote.http11.Http11AprProtocol"
[root@localhost ~]# vim /etc/profile                              ##在最后一行添加
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/apr/lib
[root@localhost native]# /usr/local/tomcat8/bin/shutdown.sh
[root@localhost native]# /usr/local/tomcat8/bin/startup.sh
[root@localhost ~]# netstat -utpln |grep java
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN
2748/java
tcp        0      0 127.0.0.1:8005          0.0.0.0:*               LISTEN
2748/java
[root@localhost native]# tail -3 /usr/local/tomcat8/logs/catalina.out
08-Apr-2019 12:20:20.455 信息 [main] org.apache.coyote.AbstractProtocol.start Starting
ProtocolHandler ["ajp-nio-8009"]
```

解决重启 tomcat 服务后，8005 端口延迟启动的问题；

```
[root@localhost ~]# vi /usr/local/java/jre/lib/security/java.security
117 securerandom.source=file:/dev/urandom
```

Tomcat 线程池

在 tomcat 服务中每一个用户请求都是一个线程，所以可以使用线程池（也叫连接器）来提高性能。

线程池是什么？

线程池是一种多线程处理形式，处理过程中将任务添加到队列，然后创建线程后自动启动这些任务，线程池线程都是后台线程。每个线程都使用默认的堆栈大小。

它由线程池管理器，工作线程，任务接口，任务队列组成。

在什么情况下使用线程池？

单个任务处理的时间短

将需处理的任务的数量大

有什么好处？

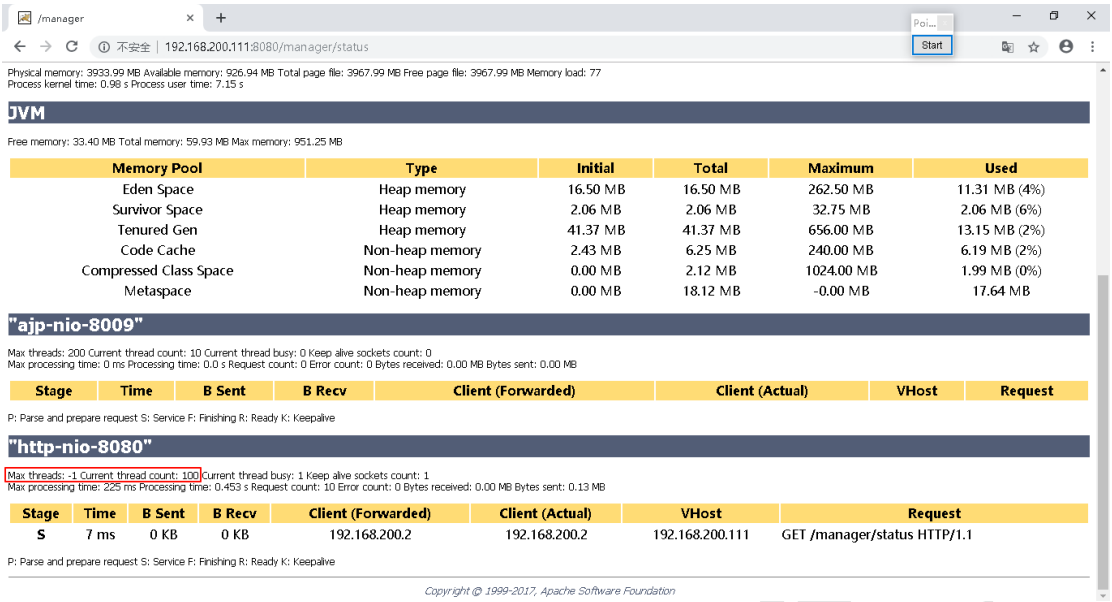
1.减少在创建和销毁线程上所花的时间以及系统资源的开销

2.如不使用线程池，有可能造成系统创建大量线程而导致消耗完系统内存以及”过度切换”。

开启并且使用

```
[root@localhost ~]# vim /usr/local/tomcat8/conf/server.xml
57      <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
58            maxThreads="500" minSpareThreads="100" prestartminSpareThreads="true"/>

69      <Connector executor="tomcatThreadPool"    port="8080"
protocol="org.apache.coyote.http11.Http11NioProtocol"
70                    connectionTimeout="20000"
71                    redirectPort="8443" />
```

更改完重启服务效果



3.3.2.参数说明

maxThreads(最大线程数)	默认值是 200(可适当调整) 如果配置了一个 Executor，则该属性的任何值集将被正确记录，但是它将被显示为-1
minSpareThreads(最小活跃线程数)	默认是 25(调整活跃线程数的时候必须开启下面的参数)
prestartminSpareThreads(是否在启动时就生成 minSpareThreads 个线程)	默认是 flase，改为 true 则开启
MaxQueueSize(最大的等待队列数，超过则请求拒绝)	基本是无上限，假如你超过最大线程数，就可以给你设置 100 的等待队列数

3.3.3. 最佳实践

```
<!--The connectors can use a shared executor, you can define one or more named thread pools-->
<!---->
<Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
    maxThreads="800" minSpareThreads="100" maxQueueSize="100" prestartminSpareThreads="true"/>
```

Tomcat 连接器

Connector 是 Tomcat 接收请求的入口，每个 Connector 有说自己专属的监听端口
Connector 有两种:HTTP Connector 和 AJP Connector

3.4.1.通用属性(高亮是重点)

属性	描述
enableLookups	设置为 true，如果你想调用 request.getRemoteHost()，以便返回远程客户的实际主机名执行 DNS 查找。设置为 false 跳过 DNS 查找并以字符串形式返回 IP 地址（从而提高性能）。默认情况下，DNS

	查找被禁用。
maxPostSize	POST 的最大大小（以字节为单位）将由容器 FORM URL 参数解析处理。可以通过将此属性设置为小于或等于 0 的值来禁用该限制。如果未指定，则此属性设置为 2097152（2 兆字节）。 比如你向 tomcat 提交文件的时候大于 2 兆是不能提交的
port	此**连接器** 将在其上创建服务器套接字并等待传入连接的 TCP 端口号。您的操作系统将只允许一个服务器应用程序侦听特定 IP 地址上的特定端口号。如果使用特殊值 0（零），则 Tomcat 将随机选择一个空闲端口用于此连接器。这通常仅适用于嵌入式和测试应用程序。
protocol	设置协议以处理传入流量。默认值是 HTTP/1.1 使用自动切换机制来选择阻塞的基于 Java 的连接器或基于 APR / native 的连接器。如果 PATH（Windows）或 LD_LIBRARY_PATH（在大多数 unix 系统上）环境变量包含 Tomcat 本机库，则将使用 APR /本机连接器。如果找不到本机库，将使用阻塞的基于 Java 的连接器。请注意，APR /本机连接器对 HTTPS 的设置与 Java 连接器不同。 要使用显式协议而不是依赖于上述自动切换机制，可以使用以下值： org.apache.coyote.http11.Http11Protocol- 阻止 Java 连接器 org.apache.coyote.http11.Http11NioProtocol- 非阻塞 Java 连接器 org.apache.coyote.http11.Http11AprProtocol- APR /本机连接器。 也可以使用定制实现。 看看我们的连接器比较图表。对于 http 和 https，两个 Java 连接器的配置完全相同。 有关 APR 连接器和 APR 特定 SSL 设置的更多信息，请访问 APR 文档 指定运行模式
URIEncoding	这指定在%xx 解码 URL 之后用于解码 URI 字节的字符编码。如果未指定，将使用 ISO-8859-1。 解决乱码的

标准实施

除了上面列出的常见 Connector 属性之外，标准 HTTP 连接器（BIO，NIO 和 APR / native）都支持以下属性。

属性	描述
acceptCount	当所有可能的请求处理线程都在使用时，传入连接请求的最大队列长度。队列已满时收到的任何请求都将被拒绝。默认值为 100。
acceptorThreadCount	用于接受连接的线程数。在多 CPU 机器上增加此值，尽管您真的不需要超过 2。此外，对于大量非保持连接的连

	接，您可能还希望增加此值。默认值是 1。
compression	所述**连接器**可在试图节省服务器的带宽使用 HTTP / 1.1 GZIP 压缩。参数的可接受值是“off”（禁用压缩），“on”（允许压缩，导致文本数据被压缩），“force”（在所有情况下强制压缩）或数字整数值（这是等效于“on”，但指定压缩输出之前的最小数据量）。如果内容长度未知且压缩设置为“on”或更具攻击性，则输出也将被压缩。如果未指定，则将此属性设置为“off”。 *注意*：在使用压缩（节省带宽）和使用 sendfile 功能（节省 CPU 周期）之间需要权衡。如果连接器支持 sendfile 功能，例如 NIO 连接器，则使用 sendfile 将优先于压缩。症状是静态文件大于 48 Kb 将被解压缩。您可以通过设置 useSendfile 连接器的属性来关闭 sendfile，如下所述，或者更改默认 或 Web 应用程序中 DefaultServlet 配置中的 sendfile 使用率阈值 。 conf/web.xmlweb.xml
connectionUploadTimeout	指定在数据上载过程中使用的超时（以毫秒为单位）。这仅在 **disableUploadTimeout** 设置**为时**生效 false。
disableUploadTimeout	此标志允许 servlet 容器在数据上载期间使用不同的，通常更长的连接超时。如果未指定，则将此属性设置为 true 禁用此较长超时。
executor	对 Executor 元素中的名称的引用。如果设置了此属性，并且存在指定的执行程序，则连接器将使用执行程序，并且将忽略所有其他线程属性。请注意，如果未为连接器指定共享执行程序，则连接器将使用专用的内部执行程序来提供线程池。
maxConnections	服务器在任何给定时间接受和处理的最大连接数。达到此数量后，服务器将接受但不处理另一个连接。此附加连接将被阻止，直到正在处理的连接数低于**maxConnections**，此时服务器将再次开始接受和处理新连接。请注意，一旦达到限制，操作系统仍可以根据 acceptCount 设置接受连接。默认值因连接器类型而异。对于 BIO，默认值是 **maxThreads** 的值，除非是 Executor 在这种情况下，默认值将是执行程序中 maxThreads 的值。对于 NIO，默认为 10000。对于 APR / native，默认为 8192。请注意，对于 Windows 上的 APR / native，配置的值将减小到 1024 的最大倍数，小于或等于 maxConnections。这是出于性能原因而完成的。 如果设置为值-1，则禁用 maxConnections 功能并且不计算连接。
maxThreads	此 **Connector** 要创建的最大请求处理线程数，因此确定可以处理的最大并发请求数。如果未指定，则此属性设

	置为 200.如果执行程序与此连接器关联，则忽略此属性，因为连接器将使用执行程序而不是内部线程池执行任务。
minSpareThreads	最小线程数始终保持运行。如果未指定，10 则使用默认值。
SSLEnabled	使用此属性可在连接器上启用 SSL 通信。要在连接器上启用 SSL 握手/加密/解密，请将此值设置为 true。默认值为 false。转换此值时，true 您还需要设置 scheme 和 secure 属性以将正确值 request.getScheme()和 request.isSecure()值传递给 servlet 有关详细信息，请参阅 SSL 支持。

3.4.4.最佳实践

```
-->
<Connector executor="tomcatThreadPool" port="8080" protocol="org.apache.coyote.http11.Http11NioProtocol"
           connectionTimeout="20000"
           redirectPort="8443"
           enableLookups="false"
           maxPostSize="10485760"
           URIEncoding="UTF-8"
           acceptCount="100"
           acceptorThreadCount="2"
           disableUploadTimeout="true"
           maxConnections="10000"
           SSLEnabled="false"/>
<!-- A "Connector" using the shared thread pool-->
<!--
```

禁用 Tomcat AJP 连接器

AJP(Apache JServer Protocol)
AJPV13 协议是面向包的。WEB 服务器和 servlet 容器通过 TCP 链接来交互；为啦节省 SOCKET 创建的昂贵代价，WEB 服务器会尝试维护一个永久 TCP 链接到 servlet 容器，并且在多个请求和响应周期过程会重用链接。
我们一般是使用 Nginx+tomcat 的架构，所以用不着 AJP 协议，所以把 AJP 连接器。

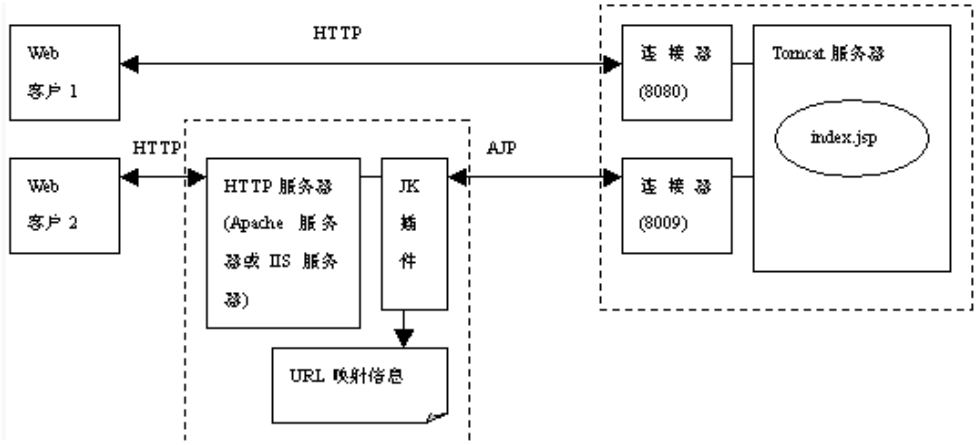


图22-1 Web客户访问Tomcat服务器上的JSP组件的两种方式

```
[root@localhost ~]# vim /usr/local/tomcat8/conf/server.xml
116         <!--
117         <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
118         -->
[root@localhost ~]# /usr/local/tomcat8/bin/shutdown.sh
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
[root@localhost ~]# netstat -lnpt | grep :8009
```

Tomat 热部署与热加载

项目开发过程中，经常要改动 tomcat 的 JSP 文件，但是又不想从新启动服务 ，因为服务器从新启动服务需要花费很长时间，但又想直接获得 debug 结果。

有两种方式：热部署和热加载

热加载：在 server.xml -> context 属性中设置 reloadable="true"

```
<Context docBase="/web/crushlinux" path="" reloadable="true" />
```

热部署：在 server.xml -> context 属性中设置 autoDeploy="true"

```
<Context docBase="/web/crushlinux" path="" autoDeploy="true" />
```

两者区别：

热加载：服务器会监听 class 文件改变，包括 web-inf/class,wen-inf/lib,web-inf/web.xml 等文件，若发生更改，则局部进行加载，不清空 session ，不释放内存。开发中用的多，但是要考虑内存溢出的情况。

热部署： 整个项目重新部署，包括你重新打上.war 文件。 会清空 session ，释放内存。项目打包的时候用的多。

Tomcat JVM 参数优化

JVM 参数的优化

适当调整 tomcat 的运行 jvm 参数可以提升整体性能

JVM 内存模型

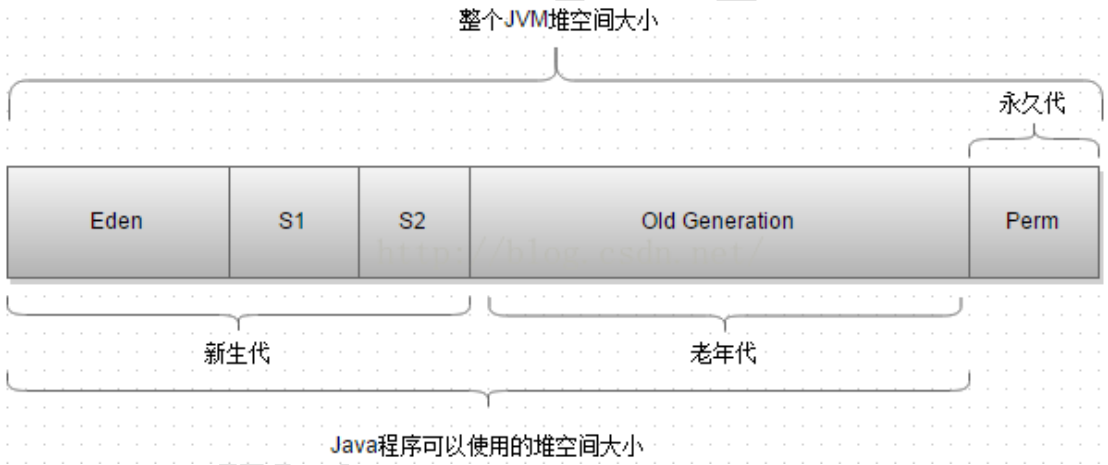
1. Java 栈

Java 栈与每一个线程关联的，jvm 在创建每个线程的时候，会分配一定的栈空间给线程使用，它主要用来存储线程执行过程中的局部变量，和方法的返回值，以及方法调用上下文，栈空间随着线程的终止而释放。

2. Java 堆

Java 中堆是由所有的线程共享的一块内存区域，堆用来保存各种 java 对象，比如数组，线程对象等。

Java 堆的分区



Young 新生代

新生成的对象都是放在新生代的。新生代的目标就是尽可能快速的回收掉那些生命周期短的对象 。

新生代分为三个区：一个 Eden 区，两个 Survivor 区（一般情况）。大部分对象是在 Eden 区中生成，某一个时刻只有其中一个是被使用的，当 Eden 区满了，GC 就会将存活的对象移到空闲的 survivor 区间中，根据 jvm 的策略。在经过几次垃圾收集后，任然存活于 survivor 的对象将被移动到 old generation 区间。

(新生成的对象都在新生代，java 的 GC 会进行垃圾回收你这个对象，但是它发现你这个内存对象还在应用，根据几次的垃圾回收，任然还有些对象没有被回收的话就给他放到老年代里，放到老年代里不一定不会被 GC 回收)。

Old Generation 老年代

Oldgeneration 区主要保存生命周期长的对象，一般是一些老的对象，当一些对象在 Young 复制转移到一定次数以后，内存对象就会被转移到 oldgeneration 区，一般如果系统

中用了 application 级别的缓存，缓存中的对象往往会被转移到这一区间，放到老年代里不一定不会被 GC 回收。

Perm 永久代

Perm 主要保存 class,method,filed 对象，并不会被 GC 回收。

Virtual 区：

最大内存和初始内存的差值，就是 virtual 区，年轻代，老年代，永久代上都有虚拟区(virtual)

设置区的大小

Jvm 提供了相应的参数来对内存大小进行配置，jvm 中堆被分为了 3 个大的区间，同时 jvm 也提供了一些选项对 young,oidgeneration 的大小进行控制

Total heap

- -Xms:指定了 jvm 初始启动以后初始化内存
- -Xmx:指定 jvm 堆的最大内存，在 jvm 启动以后，会分配-Xmx 参数指定大小的内存给 JVM,但是不一定全都使用，jvm 会根据-Xms 参数来调节真正用于 jvm 的内存
- Xmx-Xms=virtual 大小(之差就是三个 virtual 空间大小)

年轻代：

- -XX:NewRatio=8 意味着 oidgeneration 和 young 的比值 8:1,这样 eden+2*survivr=1/9 堆内存
- -XX:SurvivorRatio=32 意味着 eden 和一个 survivor 的比值是 32：1,这样一个 Survivorj 就占 young 区的 1/34
- -Xmn 参数设置了年轻代的大小

永久代：

- -XX:PermSize=16M -XX:MaxPermSize=64M

Thread Stack

- -XX:Xss=128k

常用参数

参数说明：

1. file.encoding 默认文件编码
2. -Xmx1024m 设置 JVM 最大可用内存为 1024MB
3. -Xms1024m 设置 JVM 最小内存为 1024m，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
4. -XX:NewSize 设置年轻代
5. XX:MaxNewSize 设置最大的年轻代大小
6. -XX:PermSize 设置永久代大小
7. -XX:MaxPermSize 设置最大永久代大小

8. -XX:NewRatio=4:设置年轻代(包括 Eden 和两个 Survivor 区)与终身代的比值(除去永久代)。设置为 4，则年轻代与终身代所占比值为 1：4，年轻代占整个堆栈的 1/5
9. -XX:MaxTenuringThreshold=0： 设置垃圾最大年龄，默认为 15，如果设置为 0 的话，则年轻代对象不经过 Survivor 区，直接进去老年代，对于老年代比较多的应用，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，增加再年轻代即被回收的概论。
10. -XX:+DisableExplicitGC 这个将会忽略手动调用 GC 的代码使得 System.gc()的调用就会变成一个空调用，完全不会出发任何 GC

在 tomcat 中设置 JVM 参数

Tomcat 性能取决于内存大小

- 上策：优化代码
  该项需要开发经验足够丰富，对开发人员要求较高
- 中策：jvm 优化机制 垃圾回收机制 把不需要的内存回收
  优化 jvm--优化垃圾回收策略
- 下策：加足够大的内存
  该项的资金投入较大
- 下下策：每天 0 点定时重启 tomcat
  使用较为广泛

优化 catalina.sh 配置文件。在 catalina.sh 配置文件中添加以下代码：

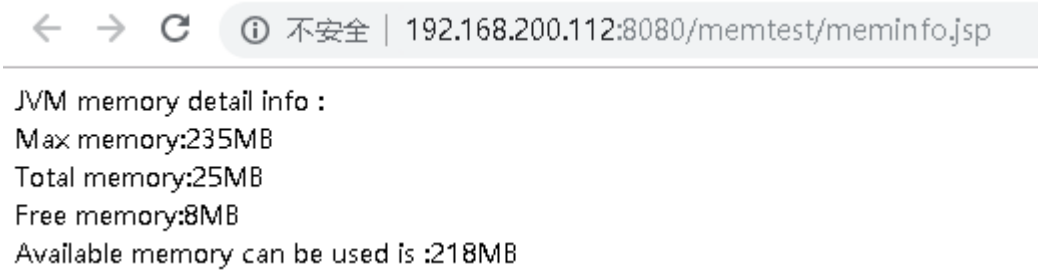
tomcat 分配 1G 内存模板

```
[root@localhost ~]# vim /usr/local/tomcat8/bin/catalina.sh
JAVA_OPTS="-Djava.awt.headless=true -Dfile.encoding=UTF-8 -server -Xms1024m -Xmx1024m
-XX:NewSize=512m -XX:MaxNewSize=512m -XX:PermSize=512m -XX:MaxPermSize=512m"
```

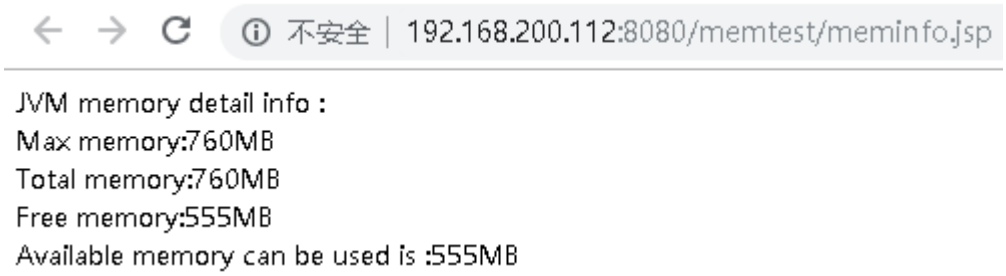
重启服务

```
[root@localhost ~]# /usr/local/tomcat8/bin/shutdown.sh
[root@localhost ~]# /usr/local/tomcat8/bin/startup.sh
```

修改之前



修改之后


不安全 | 192.168.200.112:8080/memtest/meminfo.jsp
JVM memory detail info :
Max memory:760MB
Total memory:760MB
Free memory:555MB
Available memory can be used is :555MB