# 官网

http://tomcat.apache.org/

# 基本概念

Tomcat是Apache 软件基金会的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。由于有了Sun 的参与和支持，最新的Servlet 和JSP 规范总是能在Tomcat 中得到体现。因为Tomcat 技术先进、性能稳定，而且免费，因而深受Java 爱好者的喜爱并得到了部分软件开发商的认可，成为目前比较流行的Web 应用服务器。

Tomcat 服务器是一个免费开放源代码的Web 应用服务器，属于轻量级应用[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8)，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试JSP 程序的首选。对于一个初学者来说，可以这样认为当在一台机器上配置好Apache 服务器，可利用它响应HTML页面的访问请求。实际上Tomcat是Apache 服务器的扩展，但运行时它是独立运行的，所以当你运行tomcat 时，它实际上作为一个与Apache 独立的进程单独运行的。

当配置正确时，Apache 为HTML页面服务，而Tomcat 实际上运行JSP 页面和Servlet。另外Tomcat和[IIS](https://baike.baidu.com/item/IIS)等Web服务器一样，具有处理HTML页面的功能，另外它还是一个Servlet和JSP容器，独立的Servlet容器是Tomcat的默认模式，不过Tomcat处理静态[HTML](https://baike.baidu.com/item/HTML)的能力不如Apache服务器。

tomcat默认以java –client方式运行。

# 版本差异

## Tomcat6.x

建立在Tomcat5.5的改进和实现Servlet 2.5和 JSP 2.1规范, 除此之外，它还包括以下改进

（1）内存使用优化

（2）高级IO功能

（3）重构聚类

## Tomcat7.x

建立在Tomcat6.0.x的改进和实现Servlet 3.0， JSP 2.2，EL 2.2和 WebSocket的1.1规格。除此之外，它还包括以下改进：

（1）Web应用程序内存泄露检测和预防

（2）提高了Manager和Host Manager应用程序的安全性

（3）通用CSRF保护

（4）支持直接在Web应用程序中包含外部内容

（5）重构（连接器，生命周期）和大量的内部代码清理

## Tomcat8.x

建立在Tomcat7.x基础上并实施Servlet 3.1，JSP 2.3，EL 3.0 和WebSocket的1.1规格。除此之外，还包括以下重大改进：

单个公共资源实现来替换早期版本中提供的多个资源扩展特性

Apache Tomcat 8.5.x支持相同的Servlet，JSP，EL和WebSocket规范的版本， 除此之外，它也实现了JASPIC 1.1规范。还有在许多领域显著变化引擎下，从而提高了性能、稳定性和总拥有成本。

## Tomcat9.x

建立在Tomcat 8.0.x和实现了目前草案的Servlet 4.0规范，也将执行 JSP 2.4、EL 3.1，目前对WebSocket的1.2 和JASPIC 1.1 规范工作的一次更新上这些规范为Java EE 8除此之外启动时，它包括以下显著改进：

（1）添加对HTTP / 2的支持（需要APR /本地库）

（2）添加对TLS虚拟主机的支持

（3）添加了对使用JSSE连接器（NIO和NIO2）使用OpenSSL for TLS支持的支持

**Alpha**版本可能含有大量的规范和/或显著的bug需要未经测试/缺少的功能，并且预计不会稳定地任何时间运行。

**Beta**版本中可能含有一些未经测试的功能和/或一些相对较小的错误。Beta版本预计不会稳定运行。

**Stable**的版本可能包含少量相对较小的错误。稳定的释放用于生产使用，预计可以稳定运行长时间。

# Tomcat目录结构

## bin

存放启动和关闭tomcat脚本

## conf

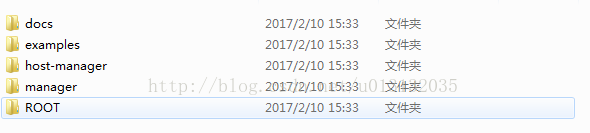
包含不同的配置文件，server.xml和web.xml是tomcat的主要配置文件。

## work

存放jsp编译后产生的class文件。

## webapps

存放应用程序示例，以后你要部署的应用程序也要放到此目录下。webapps目录下的内容如下图所示：tomcat默认部署路径。目录用来存放应用程序，当Tomcat启动时会加载webapps目录下的应用程序。当然可以以war包、Jar包、普通文件夹三种形式发布应用。你也可以把应用程序存放在电脑磁盘的任意一个位置，但是要在配置文件中配置好，使其映射好。



默认包含五个web应用。

## temp

存放Tomcat运行时所产生的临时文件

## logs

存放日志文件

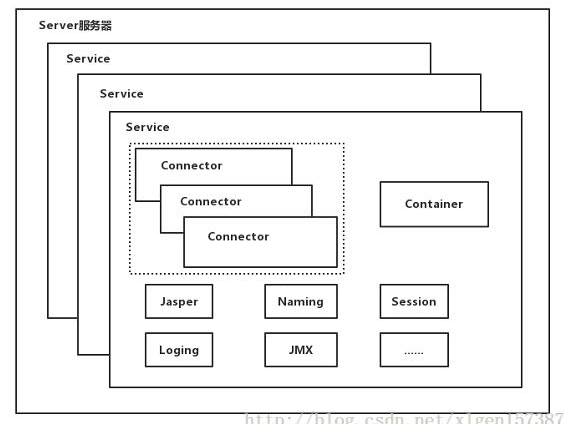
## lib

主要存放tomcat所需的jar文件。

# Tomcat顶层架构

## 基本说明

顶层架构图如下：



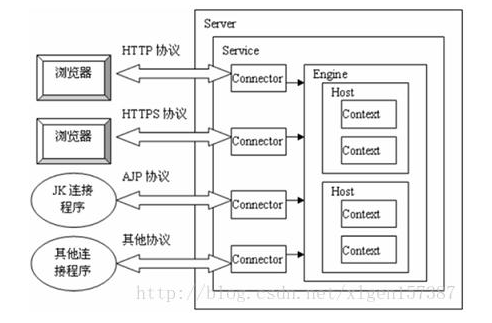
Tomcat中最顶层的容器是Server，代表着整个服务器，从上图中可以看出，一个Server可以包含至少一个Service，用于具体提供服务。

Service主要包含两个部分：Connector和Container。从上图中可以看出 Tomcat 的心脏就是这两个组件，他们的作用如下：

1、Connector用于处理连接相关的事情，并提供Socket与Request和Response相关的转化。

2、Container用于封装和管理Servlet，以及具体处理Request请求；

一个Tomcat中只有一个Server，一个Server可以包含多个Service，一个Service只有一个Container，但是可以有多个Connectors，这是因为一个服务可以有多个连接，如同时提供Http和Https链接，也可以提供向相同协议不同端口的连接,示意图如下（Engine、Host、Context下边会说到）

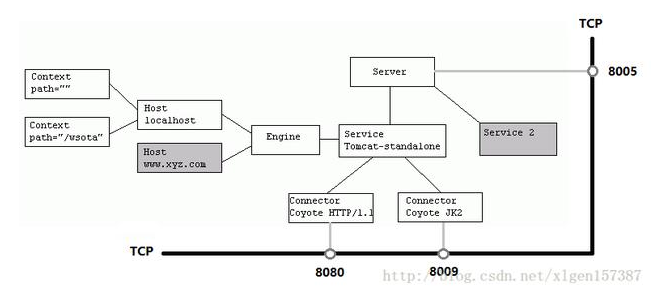


多个 Connector 和一个 Container 就形成了一个 Service，有了 Service 就可以对外提供服务了，但是 Service 还要一个生存的环境，必须要有人能够给她生命、掌握其生死大权，那就非 Server 莫属了！所以整个 Tomcat 的生命周期由 Server 控制。

另外，上述的包含关系或者说是父子关系，都可以在tomcat的conf目录下的server.xml配置文件中看出，下图是删除了注释内容之后的一个完整的server.xml配置文件（Tomcat版本为8.0）



上边的配置文件，还可以通过下边的一张结构图更清楚的理解：



Server标签设置的端口号为8005，shutdown=”SHUTDOWN” ，表示在8005端口监听“SHUTDOWN”命令，如果接收到了就会关闭Tomcat。一个Server有一个Service，当然还可以进行配置，一个Service有多个，Service左边的内容都属于Container的，Service下边是Connector。

总结如下：

（1）Tomcat中只有一个Server，一个Server可以有多个Service，一个Service可以有多个Connector和一个Container；

（2）Server掌管着整个Tomcat的生死大权。

（3）Service 是对外提供服务的

（4）Connector用于接受请求并将请求封装成Request和Response来具体处理。

（5）Container用于封装和管理Servlet，以及具体处理request请求；

## Connector和Container的微妙关系

由上述内容我们大致可以知道一个请求发送到Tomcat之后，首先经过Service然后会交给我们的Connector，Connector用于接收请求并将接收的请求封装为Request和Response来具体处理，Request和Response封装完之后再交由Container进行处理，Container处理完请求之后再返回给Connector，最后在由Connector通过Socket将处理的结果返回给客户端，这样整个请求的就处理完了。

Connector最底层使用的是Socket来进行连接的，Request和Response是按照HTTP协议来封装的，所以Connector同时需要实现TCP/IP协议和HTTP协议！Tomcat既然处理请求，那么肯定需要先接收到这个请求，接收请求这个东西我们首先就需要看一下Connector！

## Connector架构分析

Connector用于接受请求并将请求封装成Request和Response，然后交给Container进行处理，Container处理完之后在交给Connector返回给客户端。

因此，我们可以把Connector分为四个方面进行理解

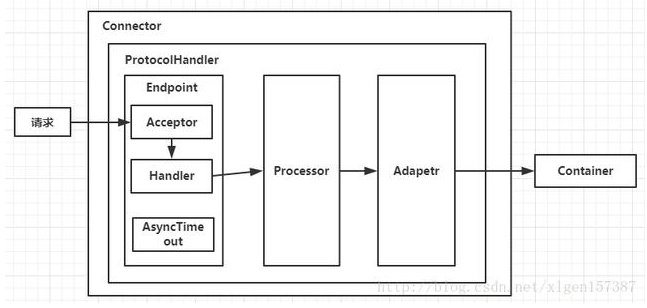
（1）Connector如何接受请求的

（2）如何将请求封装成Request和Response的

（3）封装完之后的Request和Response如何交给Container进行处理的

（4）Container处理完之后如何交给Connector并返回给客户端的

首先看一下Connector的结构图，如下图所示：



Connector就是使用ProtocolHandler来处理请求的，不同的ProtocolHandler代表不同的连接类型，比如：Http11Protocol使用的是普通Socket来连接的，Http11NioProtocol使用的是NioSocket来连接的。

其中ProtocolHandler由包含了三个部件：Endpoint、Processor、Adapter

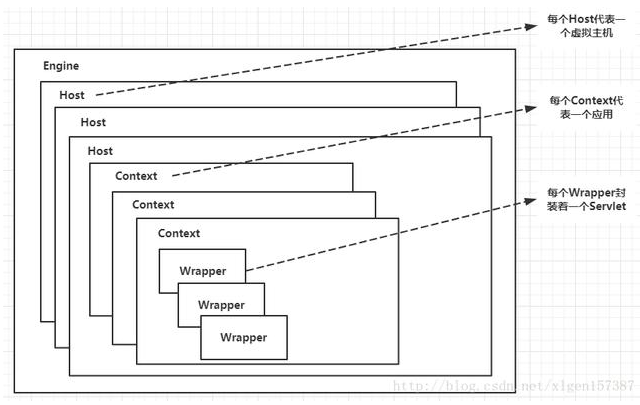
（1）Endpoint用来处理底层Socket的网络连接，Processor用于将Endpoint接收到的Socket封装成Request，Adapter用于将Request交给Container进行具体的处理。

（2）Endpoint由于是处理底层的Socket网络连接，因此Endpoint是用来实现TCP/IP协议的，而Processor用来实现HTTP协议的，Adapter将请求适配到Servlet容器进行具体的处理。

（3）Endpoint的抽象实现AbstractEndpoint里面定义的Acceptor和AsyncTimeout两个内部类和一个Handler接口。Acceptor用于监听请求，AsyncTimeout用于检查异步Request的超时，Handler用于处理接收到的Socket，在内部调用Processor进行处理。

## Container架构分析

Container用于封装和管理Servlet，以及具体处理Request请求，在Connector内部包含了4个子容器，结构图如下：



4个子容器的作用分别是：

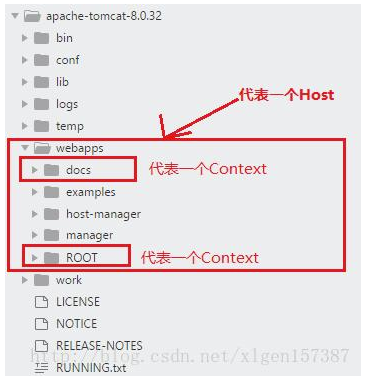
（1）Engine：引擎，用来管理多个站点，一个Service最多只能有一个Engine；

（2）Host：代表一个站点，也可以叫虚拟主机，通过配置Host就可以添加站点；

（3）Context：代表一个应用程序，对应着平时开发的一套程序，或者一个WEB-INF目录以及下面的web.xml文件；

（4）Wrapper：每一Wrapper封装着一个Servlet；

下面找一个Tomcat的文件目录对照一下，如下图所示：



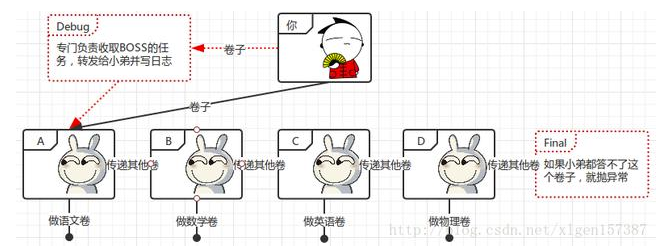
Context和Host的区别是Context表示一个应用，我们的Tomcat中默认的配置下webapps下的每一个文件夹目录都是一个Context，其中ROOT目录中存放着主应用，其他目录存放着子应用，而整个webapps就是一个Host站点。

我们访问应用Context的时候，如果是ROOT下的则直接使用域名就可以访问，例如：www.ledouit.com,如果是Host（webapps）下的其他应用，则可以使用www.ledouit.com/docs进行访问，当然默认指定的根应用（ROOT）是可以进行设定的，只不过Host站点下默认的主营用是ROOT目录下的。

## Container如何处理请求的

Container处理请求是使用Pipeline-Valve管道来处理的！（Valve是阀门之意）

Pipeline-Valve是责任链模式，责任链模式是指在一个请求处理的过程中有很多处理者依次对请求进行处理，每个处理者负责做自己相应的处理，处理完之后将处理后的请求返回，再让下一个处理着继续处理。



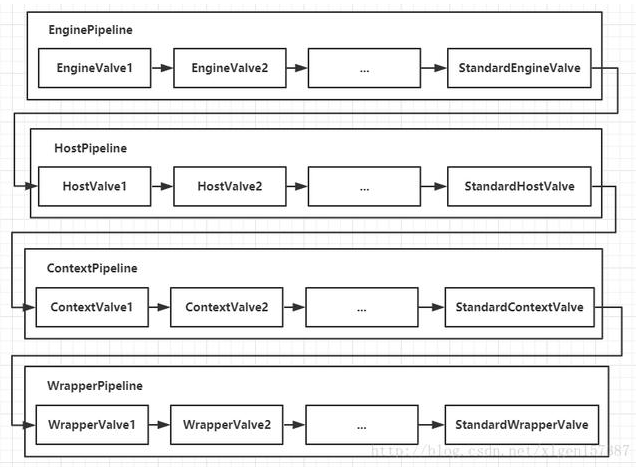
但是！Pipeline-Valve使用的责任链模式和普通的责任链模式有些不同！区别主要有以下两点：

（1）每个Pipeline都有特定的Valve，而且是在管道的最后一个执行，这个Valve叫做BaseValve，BaseValve是不可删除的；

（2）在上层容器的管道的BaseValve中会调用下层容器的管道。

我们知道Container包含四个子容器，而这四个子容器对应的BaseValve分别在：StandardEngineValve、StandardHostValve、StandardContextValve、StandardWrapperValve。

Pipeline的处理流程图如下：



（1）Connector在接收到请求后会首先调用最顶层容器的Pipeline来处理，这里的最顶层容器的Pipeline就是EnginePipeline（Engine的管道）；

（2）在Engine的管道中依次会执行EngineValve1、EngineValve2等等，最后会执行StandardEngineValve，在StandardEngineValve中会调用Host管道，然后再依次执行Host的HostValve1、HostValve2等，最后在执行StandardHostValve，然后再依次调用Context的管道和Wrapper的管道，最后执行到StandardWrapperValve。

（3）当执行到StandardWrapperValve的时候，会在StandardWrapperValve中创建FilterChain，并调用其doFilter方法来处理请求，这个FilterChain包含着我们配置的与请求相匹配的Filter和Servlet，其doFilter方法会依次调用所有的Filter的doFilter方法和Servlet的service方法，这样请求就得到了处理！

（4）当所有的Pipeline-Valve都执行完之后，并且处理完了具体的请求，这个时候就可以将返回的结果交给Connector了，Connector在通过Socket的方式将结果返回给客户端。

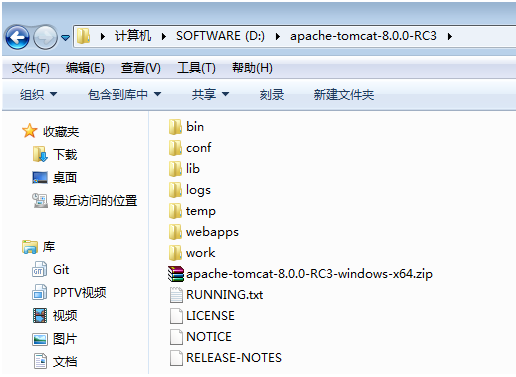
# window系统安装步骤

## 1、下载和安装Java

安装和设置JAVA\_HOME环境变量

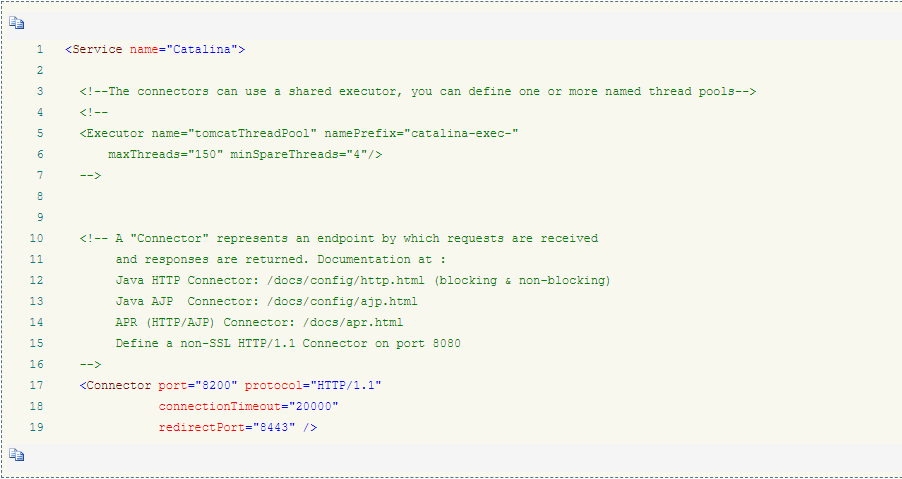
## 2、下载和解压Tomcat

下载地址：<http://tomcat.apache.org/>



## 3、修改端口号

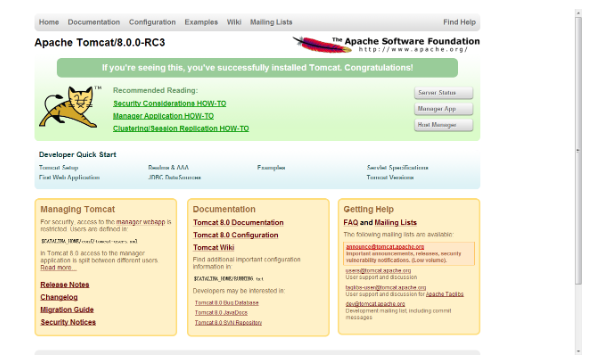
修改D:\apache-tomcat-8.0.0-RC3\conf\server.xml文件



这步可以省略

## 4、测试Tomcat

运行：D:\apache-tomcat-8.0.0-RC3\bin\startup.bat，如果窗口没有立即关闭就代表成功了。



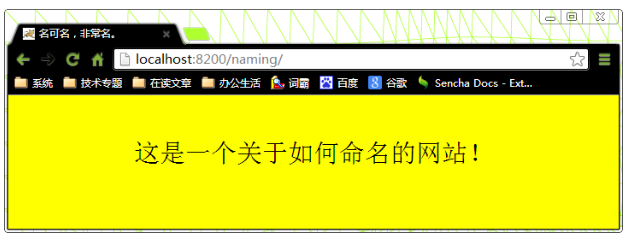
## 5、发布自己的网站

最简单的发布方式是将项目直接拷贝到：D:\apache-tomcat-8.0.0-RC3\webapps，这里介绍另外一种方式。

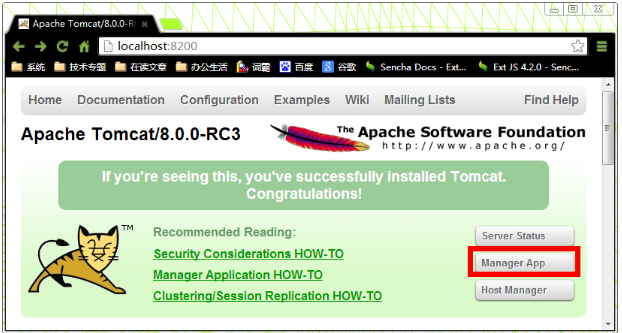
在D:\apache-tomcat-8.0.0-RC3\conf\Catalina\localhost下面建立一个naming.xml文件，内容如下：



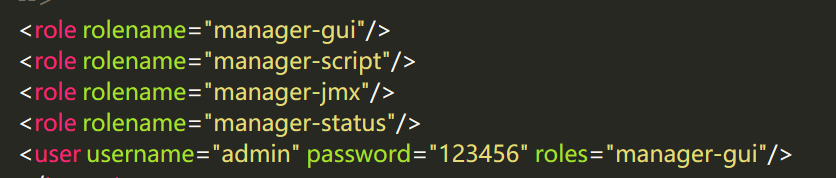
效果：



## 6、管理Web App



当点击 Manager App 的时候，会要求输入用户名和密码，可以在 D:\apache-tomcat-8.0.0-RC3\conf\tomcat-users.xml 中增加用户名和密码，如下图所示：



说明：角色名一定不能修改，按照上面的去写，用户名和密码可以自己去定义。

请注意，对于Tomcat 7以上（包含tomcat7），使用管理器应用程序所需的角色已从单个管理器角色更改为以下四个角色。您需要分配您希望访问的功能所需的角色。

（1）manager-gui - 允许访问HTML GUI和状态页面

（2）manager-script - 允许访问文本接口和状态页面

（3）manager-jmx - 允许访问JMX代理和状态页面

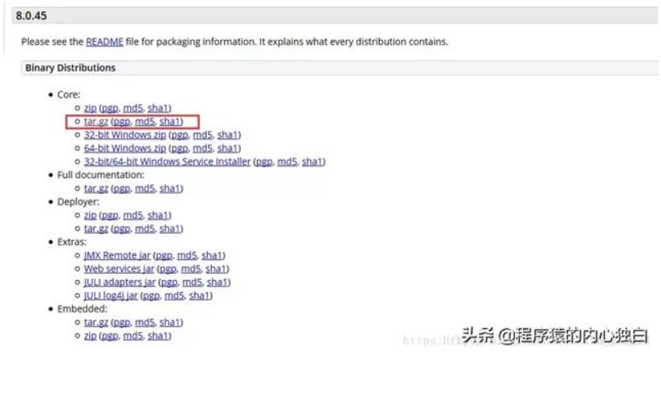
（4）manager-status -仅允许访问状态页面

注意：这种方法只支持访问本地机器的tomcat，不能解决访问远程机器的tomcat。Tomcat 支持 JAAS。

# linux操作系统安装tomcat

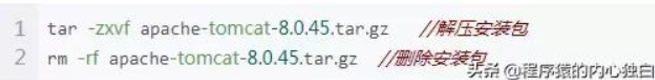
## 1、下载tomcat压缩包

http://tomcat.apache.org/download-80.cgi



选择tar.gz后缀的文件

## 2、把安装包传到CentOS下自己建的目录下，开始解压



tar –zxvf apache-tomcat-8.0.45.tar.gz //解压安装包

rm –rf apache-tomcat-8.0.45.tar.gz //删除安装包

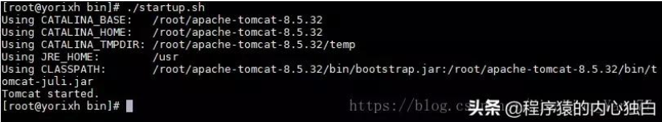
## 3、进入Tomcat的bin目录下

## 4、启动Tomcat

执行如下命令：

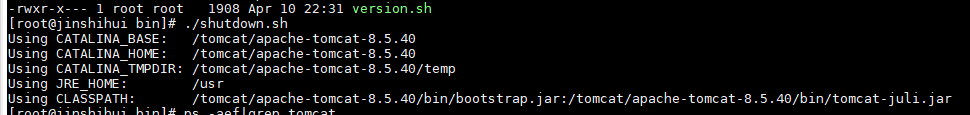
./startup.sh

如下图，出现Tomcat started，表示Tomcat启动成功



## 5、停止tomcat

在Tomcat/bin目录下执行./shutdown.sh 命令



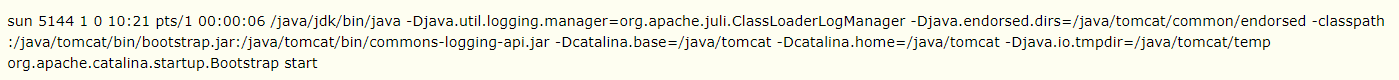
如果无法关闭服务，看是否还有tomcat在运行

ps -ef |grep tomcat

如果有，则kill

kill -9 pid #pid为相应的进程号

例如 ps -ef |grep tomcat 输出如下



则 5144 就为进程号 pid = 5144  
kill -9 5144 就可以彻底杀死tomcat

# 利用Tomcat自带的Manager查看应用日志

## 基本说明

Tomcat Manager 是 Tomcat 的运维管理界面，可以远程部署我们自己的应用、监控应用的运行情况。

## 配置Manager

搭建好Tomcat运行环境之后，点击【Manager App】会报 ：403 Access Denied，页面提示需要添加 Tomcat 管理用户才能访问。



### 修改 tomcat-users.xml

添加Tomcat管理用户，如下所示：

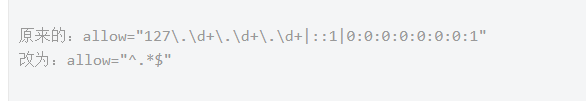
在配置文件末尾增加如下代码：

<user username="admin" password="1q2w3e" roles="manager-gui,manager-script,manager-jmx,manager-status"/>

### 修改 manager 的 context.xml

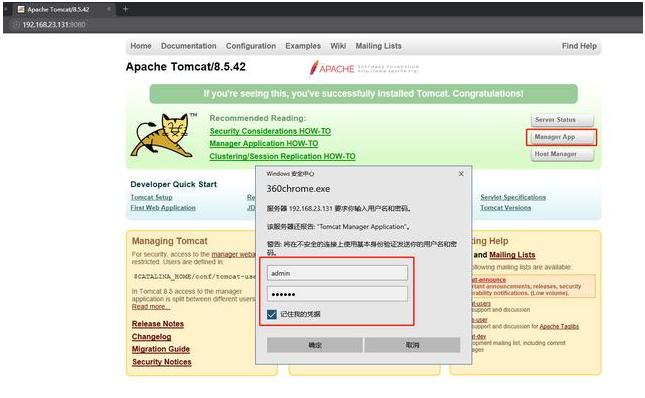
文件在webapps\manager\META-INF这个目录下面。

默认只能本机访问，改为所有IP都可以访问



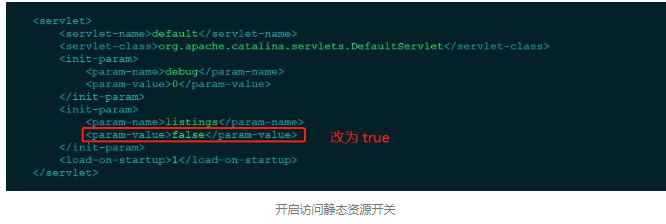
### 重启Tomcat

### 再次访问 Manager，出现登录界面



## 配置日志访问目录

1、修改 tomcat/conf/web.xml，开启访问静态资源开关

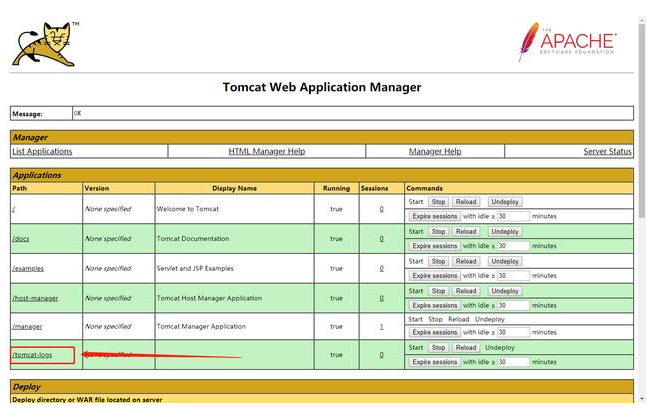


2、修改 tomcat/conf/server.xml，新增 tomcat 日志访问路径



3、重启Tomcat

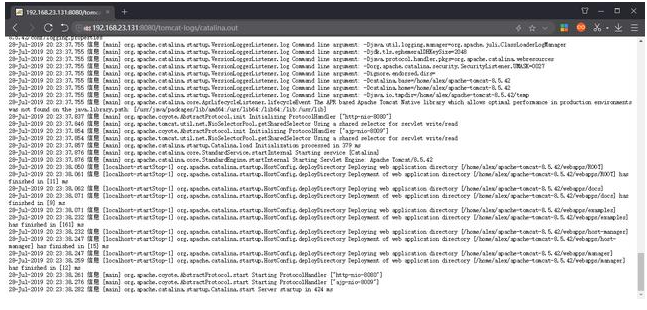
4、查看管理界面，已经有我们的tomcat 日志路径了



点击进入，显示了tomcat/logs 下面的文件



点击 catalina.out 即可查看日志内容。



以上，便实现了通过 tomcat 管理界面获取日志文件的需求。同理，server.xml里也可以设置其它文件地址，实现远程访问的效果。

## 附：使用Linux部署的同学，可能会出现下面两个问题

### ****问题1：8080 端口不能访问****

如果应用启动正常（线程存在、日志没有报错），那基本就是服务器防火墙没有开启 8080端口访问权限。

# 使用 root 用户添加端口访问权限  
/sbin/iptables -I INPUT -p tcp --dport 8080 -j ACCEPT

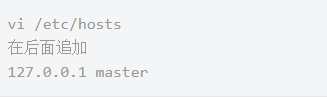
### ****问题2：Manger 界面加载很慢，需要十几秒钟****

解决方法：出现这种问题，一般是我们**重新设置了服务器主机名**，但没有在hosts里配置映射信息。

查看主机名：



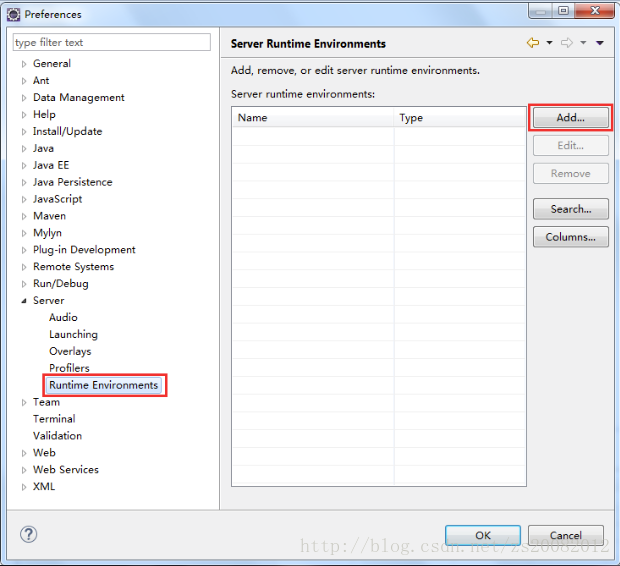
修改hosts



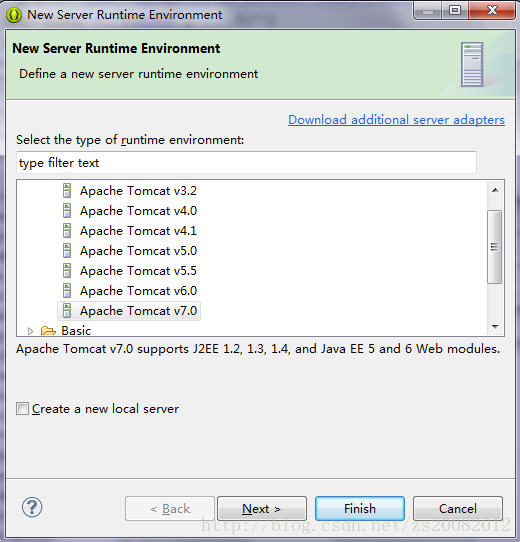
# eclipse配置tomcat

## 基本操作

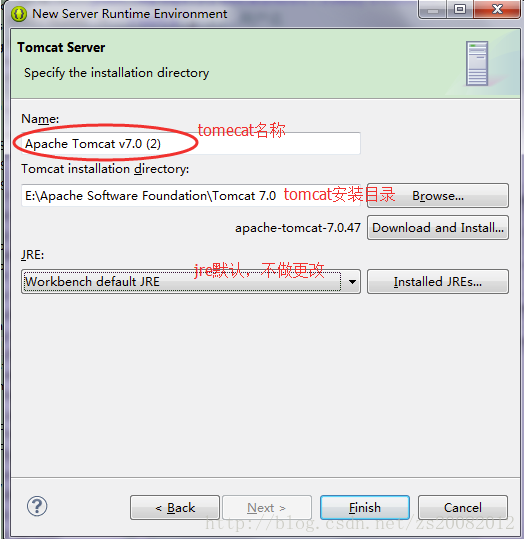
打开Eclipse，单击“Window”菜单，选择下方的“Preferences”，看是否存在“Server”选项，如图：



上图中，点击“add”的添加按钮，弹出如下界面：

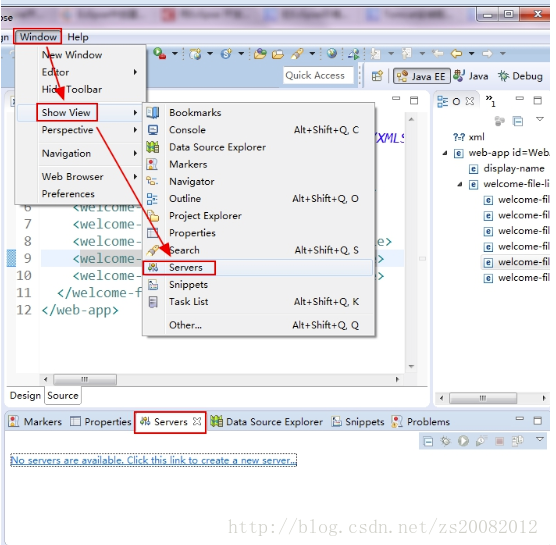


上图中，选择对应的Tomcat版本，点击next，如图

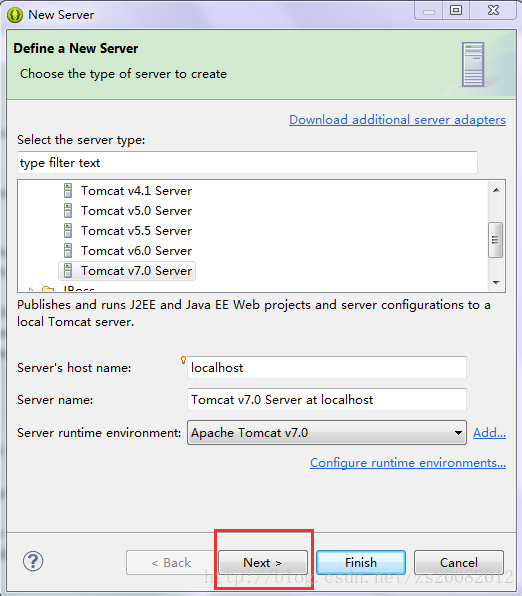


上图中，选择Tomcat的路径以及JRE（使用工作平台默认的，不用做额外添加修改），点击“完成”，配置完毕

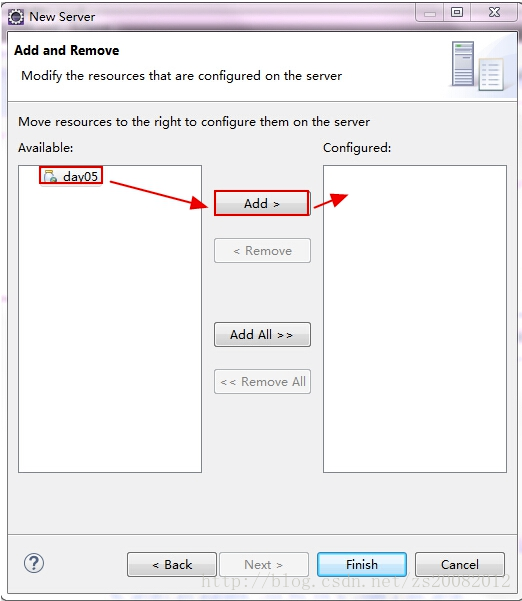
让tomcat服务器显示在控制台上，将web应用部署到tomcat中，Window → Show View → Servers，如图



点击链接No servers are available. Click ths link to create a new server. ，在弹出的对话框中选择Tomcat版本，如图

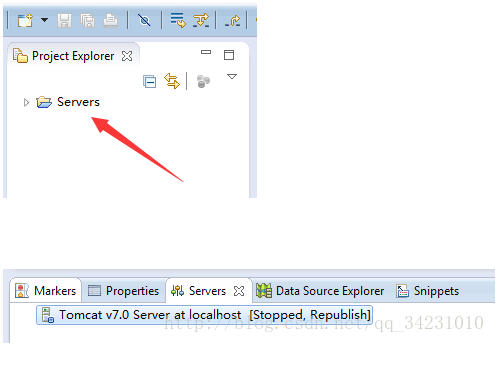


点击“Next”，添加我们的项目 如图

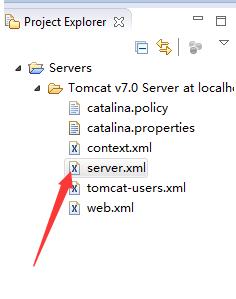


点击finish完成

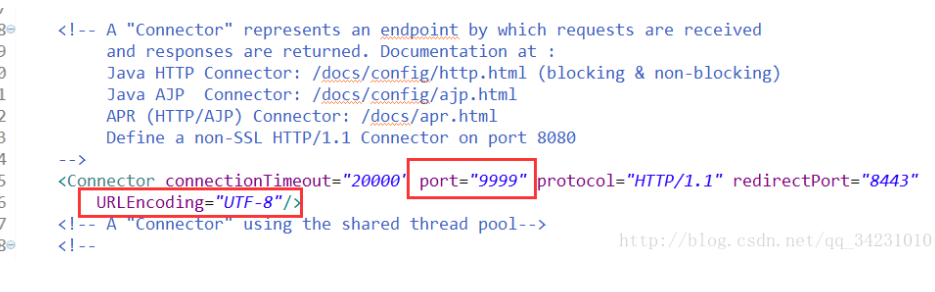
我们就可以在项目栏中看到有一个Servers项目了，而且servers视图中也有我们配置的tomcat的名字。



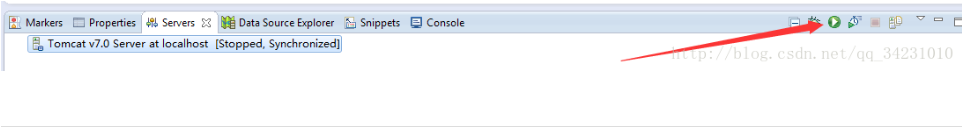
Tomcat的默认端口号为8080，为避免端口号冲突，我们在这里修改下端口号， 并设置下传输方式的默认编码格式，找到并打开server.xml文件。

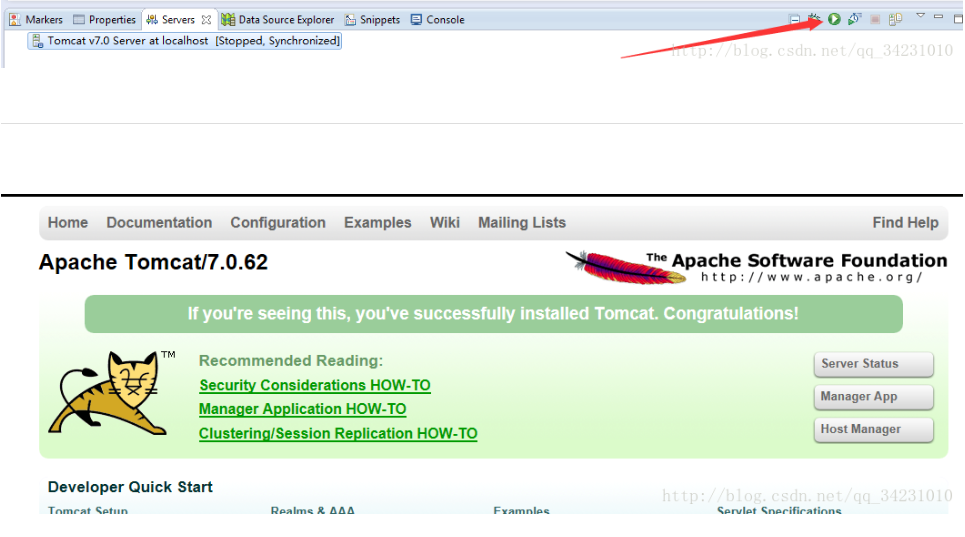


找到图中位置的xml文件，修改端口号并添加URL编码（这里我修改的端口号为9999，默认8080，编码为”UTF-8”）

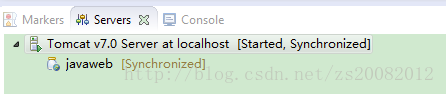


 修改后，保存退出。启动我们的服务器，并打开浏览器，输入”localhost:9999”（9999为端口号，上面自己设置的），能出现tomcat的主页就说明配置成功了

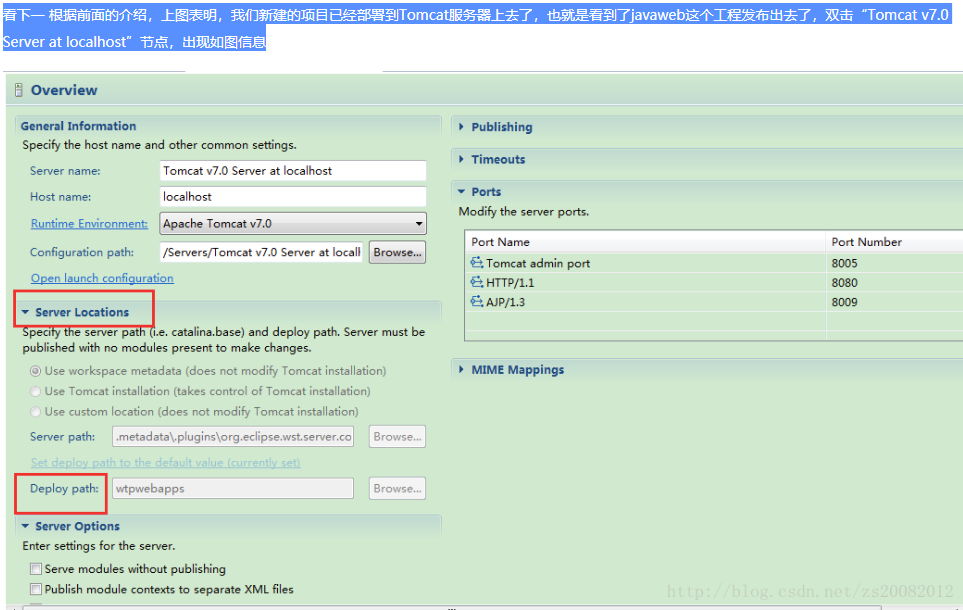




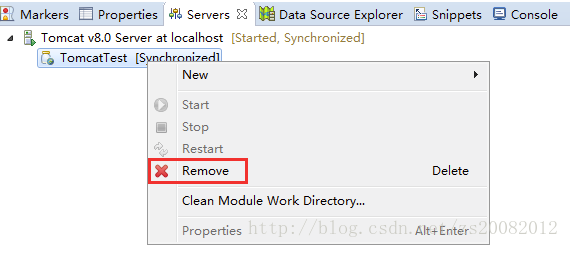
再看下一个细节



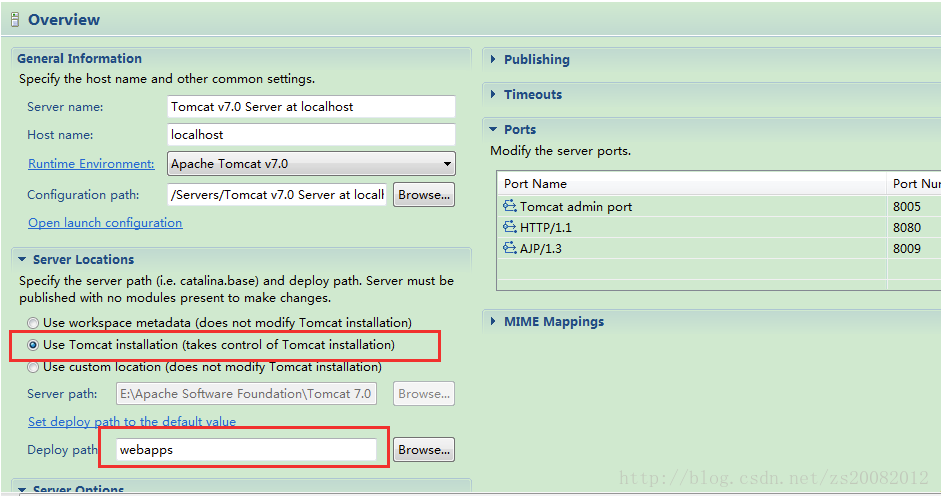
根据前面的介绍，上图表明，我们新建的项目已经部署到Tomcat服务器上了，也就是看到了javaweb这个工程发布出去了，双击“Tomcat v7.0 Server at localhost”节点，出现如图信息：



上图的红框部分表明服务的部署是在eclipse里面（默认是放在工作空间里的.metadata文件夹下）而不是在Tomcat里面。我们来改一下，前提是工程并没有发布到Tomcat中去，那我们先把之前发布的版本删除，然后再删除服务器，重新新建一个服务器。



使用Tomcat的安装目录作为部署的位置，并修改部署的路径Deploy path（建议改成Tomcat的webapps目录下），然后保存即可。



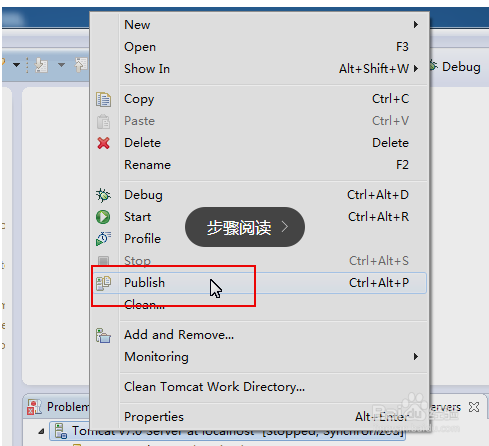
这时重新运行程序。现在来到Tomcat的webapps目录下，发现多了一个javaweb文件夹（即工程文件名），并且文件夹下包含了工程文件中WebContent的内容：



上图说明，说明这才是真正将程序发布到服务器上。

## server视图

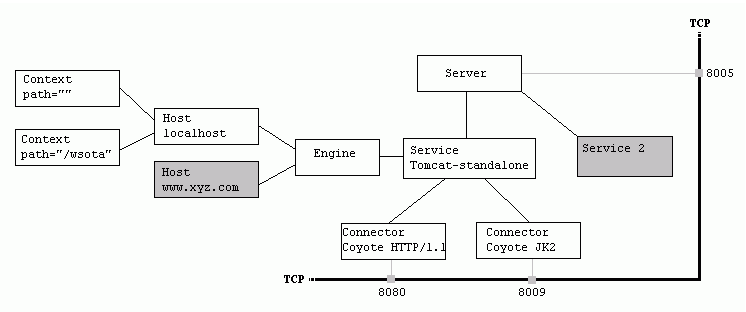
在Server上单击鼠标右键，选择Publish，可以更新项目的发布内容



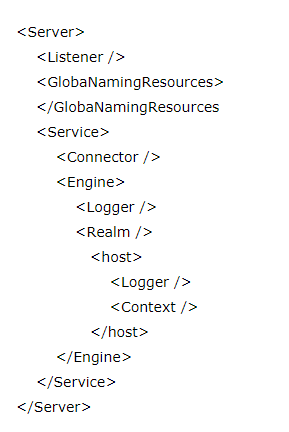
# server.xml文件

## 基本概述

这个文件在conf文件夹下面。Tomcat Server的结构图如下：



这个文件描述了如何启动Tomcat Server



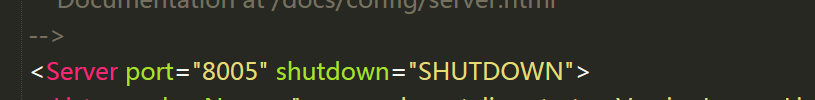
## Server元素

### 属性

port：指定一个端口，这个端口负责监听关闭tomcat的请求

shutdown：指定向端口发送的命令字符串

### 示例



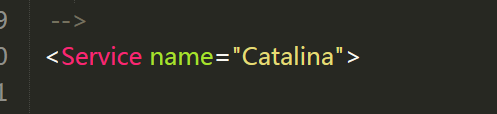
## Service元素

Service元素是一个或多个“Connectors”的集合，它们共享一个“容器”。 “服务”本身不是“容器”，因此您可能无法在此级别定义子组件，例如“Valves”。

### 属性

name：指定service的名字

### 示例



## Connector元素

表示客户端和service之间的连接

### 属性

**port**：指定服务器端要创建的端口号，并在这个端口监听来自客户端的请求

minProcessors：服务器启动时创建的处理请求的线程数

maxProcessors：最大可以创建的处理请求的线程数

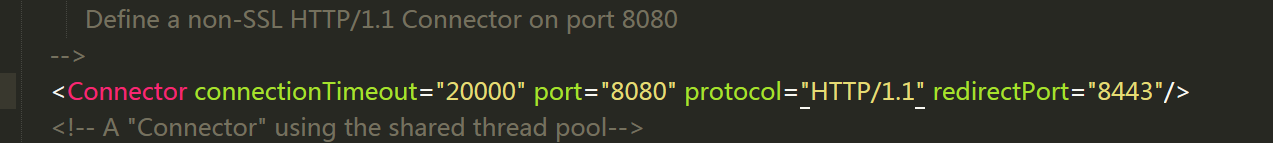
enableLookups：如果为true，则可以通过调用request.getRemoteHost()进行DNS查询来得到远程客户端的实际主机名，若为false则不进行DNS查询，而是返回其ip地址。

**redirectPort**：指定服务器正在处理http请求时收到了一个SSL传输请求后重定向的端口号。

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

**connectionTimeout**：指定超时的时间数(以毫秒为单位)

### 示例



## Engine元素

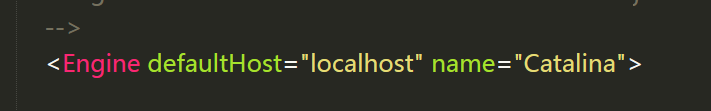
表示指定service中的请求处理机，接收和处理来自Connector的请求。

### 属性

**defaultHost**：指定缺省的处理请求的主机名，它至少与其中的一个host元素的name属性值是一样的

name：

### 示例



## Context元素

表示一个web应用程序，通常为war文件，关于war的具体信息见Servlet规范

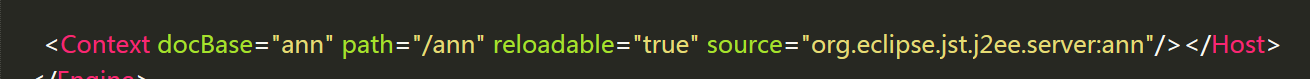
### 属性

**docBase**：应用程序的路径或者是WAR文件存放的路径

**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，我们可以在不重启tomcat的情况下改变应用程序。

### 示例



## Host元素

表示一个虚拟主机

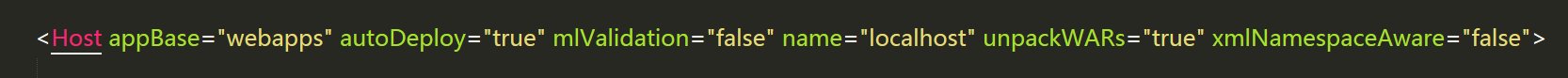
### 属性

name：指定主机名

appBase：应用程序基本目录，即存放应用程序的目录

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。

### 示例



## Logger元素

表示日志，调试和错误信息

### 属性

className：指定logger使用的类名，此类必须实现org.apache.catalina.Logger 接口

prefix：指定log文件的前缀

suffix：指定log文件的后缀

timestamp：如果为true，则log文件名中要加入时间，如下例:localhost\_log.001-10-04.txt

## Realm元素

表示存放用户名、密码及role的数据库

### 属性

className：指定Realm使用的类名，此类必须实现org.apache.catalina.Realm接口

## Value元素

功能和Logger差不多，其prefix和suffix属性解释和Logger 中的一样)

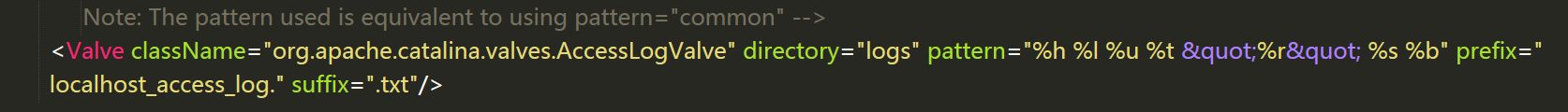
### 属性

className：指定Valve使用的类名，如用org.apache.catalina.valves.AccessLogValve类可以记录应用程序的访问信息

directory：指定log文件存放的位置

pattern：有两个值，common方式记录远程主机名或ip地址，用户名，日期，第一行请求的字符串，HTTP响应代码，发送的字节数。combined方式比common方式记录的值更多。

### 示例



参考资料：

https://www.cnblogs.com/starhu/p/5599773.html#top

# context.xml文件（重点）

## 基本说明

这个文件在conf文件夹下面。首先需要为数据源配置一个JNDI资源。我们的数据源JNDI资源定义在context元素中。在tomcat6版本中，context元素已经从server.xml文件中独立出来了，放在一个context.xml文件中。因为server.xml是不可动态重加载的资源，服务器一旦启动了以后，要修改这个文件，就得重启服务器才能重新加载，而context.xml文件则不然，tomcat服务器会定时去扫描这个文件，一旦发现文件被修改（时间戳改变了），就会自动重新加载这个文件而不需要重启服务器。我们当然推荐把应用需要的JNDI资源配置在context.xml文件中，而不是server.xml文件中。首先将数据库的驱动jar文件拷贝到tomcat6.0\lib文件夹下，假如没有拷贝，运行程序的时候报找不到驱动的异常。

## 配置示例

将下面的代码放到Tomcat 6.0\conf\context.xml中，如：

<Context reloadable="true">

<Resource

name="jdbc/数据库名"

type="javax.sql.DataSource"

maxActive="100"

maxIdle="30"

maxWait="5000"

username="用户名"

password="密码"

driverClassName="数据库的驱动"

url="数据库的连接地址" />

</Context>

解释：

maxActive="最大可以有100名用户连接数据源"

maxIdle="假如没有用户连接，空出30个连接等待用户连接"

maxWait="假如已连接用户5000秒内没有再次连接数据源，则放弃此连接"。

## 各种数据库的配置

说明：name是自定义的

### （1）sql2000

<Resource

name="jdbc/数据库名"

type="javax.sql.DataSource"

maxActive="100"

maxIdle="30"

maxWait="5000"

username="用户名"

password="密码"

driverClassName="com.microsoft.jdbc.sqlserver.SQLServerDriver"

url="jdbc:microsoft:sqlserver://127.0.0.1:1433;DatabaseName=数据库名"

/>

### （2）oracle

<Resource

name="jdbc/数据库名"

type="javax.sql.DataSource"

maxActive="100"

maxIdle="30"

maxWait="5000"

username="用户名"

password="密码"

driverClassName="oracle.jdbc.driver.OracleDriver"

url="jdbc:oracle:thin:@localhost:1521:ora9"

/>

### （3）mysql

<Resource name="jdbc/mysql"

auth="Container"

type="javax.sql.DataSource" //资源类型

driverClassName="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost/数据库名"

username="用户名"

password="密码"

maxActive="100" //最大连接数

maxIdle="30" //最大空闲时间，0为无限制

maxWait="10000"/> //建立连接的的最大等待时间

# 如何优化Tomcat6.0服务器

## 使用Tomcat的线程池

只需要在server.xml中改改配置就可以了。

核心代码：

<Executor name="tomcatThreadPool" namePrefix="catalina-exec-" maxThreads="1000" minSpareThreads="50" maxIdleTime="600000"/>

## 调整Tomcat6响应池

查找tomcat6安装目录下conf目录中的server.xml文件，进行编辑，屏蔽tomcat默认Connector。

<!–

<Connector port=”8080″ protocol=”HTTP/1.1″

connectionTimeout=”20000″

redirectPort=”8443″ />

–>

创建高线程的Connector：

<Connector port="8080" redirectPort="8443"

maxHttpHeaderSize="8192" useBodyEncodingForURI="true"

minProcessors="100" maxProcessors="5000"

maxThreads="5000" minSpareThreads="1000" maxSpareThreads="4000"

enableLookups="false" acceptCount="3500"

compression="on" compressionMinSize="2048"

compressableMimeType="text/html,text/xml,text/javascript,text/css,text/plain"

connectionTimeout="60000" disableUploadTimeout="true" debug="0" URIEncoding="UTF-8"/>

（注：加入响应线程数控制，加入压缩传递模式，调整超时设置，屏蔽调试模式）

## 增加tomcat6启动内存

查找tomcat6安装目录下bin目录中catalina.sh文件，在开始增加如下：

JAVA\_OPTS=” -Xms1400m -Xmx1400m -XX:PermSize=64M -XX:MaxNewSize=256m -XX:MaxPermSize=128m -Djava.awt.headless=true ”