tomcat简介

Tomcat是servlet/jsp容器，为Servlet/Jsp提供运行环境。Servlet是一种拓展web服务器（动态页面）的组件规范（JCP）。http服务器如Apache是处理静态页面，Tomcat生成动态页面。

# Tomcat目录结构

backup：系统重要配置文件的备份。

bin : 存放启动和关闭tomcat脚本

conf : 包含所有的配置文件。如：server.xml，web.xml 等

lib：存放tomcat服务的核心类库（jar包），也放第三方扩展包。

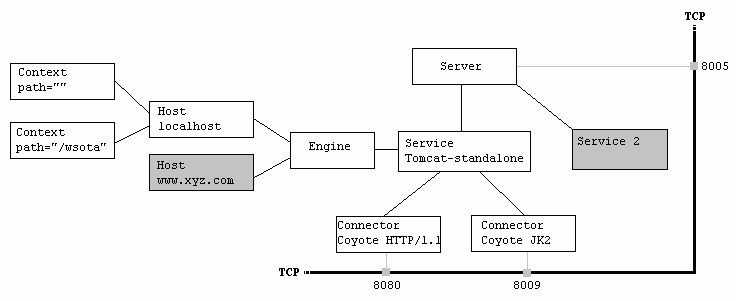
logs : 存放日志文件 。

temp：保存web运行过程中生成的临时文件。

work : 存放jsp编译后产生的class文件 。删除后系统启动会重新创建。

webapp: 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 。

# server.xml配置简介



## server

是整个配置文件的根元素。表示整个Catalina容器。属性：

1. className：实现了org.apache.catalina.Server接口的类名，标准实现类是org.apache.catalina.core.StandardServer类。
2. Port：Tomcat服务器监听用于关闭Tomcat服务器的命令（必须）
3. Shutdown：发送到端口上用于关闭Tomcat服务器的命令。

## service

name 指定service的名字

## Connector (表示客户端和service之间的连接):

### HTTP连接器

allowTrace：是否允许HTTP的TRACE方法，默认为false

emptySessionPath：如果设置为true，用户的所有路径都将设置为/，默认为false。

enableLookups：调用request、getRemoteHost()执行DNS查询，以返回远程主机的主机名，如果设置为false，则直接返回IP地址。

maxPostSize：指定POST方式请求的最大量，没有指定默认为2097152。

protocol：值必须为HTTP1.1，如果使用AJP处理器，该值必须为AJP/1.3

proxyName：如这个连接器正在一个代理配置中被使用，指定这个属性，在request.getServerName()时返回

redirectPort：如连接器不支持SSL请求，如收到SSL请求，Catalina容器将会自动重定向指定的端口号，让其进行处理。

scheme：设置协议的名字，在request.getScheme()时返回，SSL连接器设为"https"，默认为"http"

secure：在SSL连接器可将其设置为true，默认为false

URIEncoding：用于解码URL的字符编码，没有指定默认值为ISO-8859-1

useBodyEncodingForURI：主要用于Tomcat4.1.x中，指示是否使用在contentType中指定的编码来取代URIEncoding，用于解码URI查询参数，默认为false

xpoweredBy：为true时，Tomcat使用规范建议的报头表明支持Servlet的规范版本，默认为false

acceptCount：当所有的可能处理的线程都正在使用时，在队列中排队请求的最大数目。当队列已满，任何接收到的请求都会被拒绝，默认值为10

bufferSize：设由连接器创建输入流缓冲区的大小，以字节为单位。默认情况下，缓存区大的大小为2048字节

compressableMimeType：MIME的列表，默认以逗号分隔。默认值是text/html，text/xml，text/plain

compression：指定是否对响应的数据进行压缩。off：表示禁止压缩、on：表示允许压缩（文本将被压缩）、force：表示所有情况下都进行压缩，默认值为off

connectionTimeout：设置连接的超时值，以毫秒为单位。默认值为60000=60秒

disableUploadTimeOut：允许Servlet容器，正在执行使用一个较长的连接超时值，以使Servlet有较长的时间来完成它的执行，默认值为false

maxHttpHeaderSize：HTTP请求和响应头的最大量，以字节为单位，默认值为4096字节

maxKeepAliveRequest：服务器关闭之前，客户端发送的流水线最大数目。默认值为100

maxSpareThreads：允许存在空闲线程的最大数目，默认值为50

minSpareThreads：设当连接器第一次启协创建线程的数目，确保至少有这么多的空闲线程可用。默认值为4

port：服务端套接字监听的TCP端口号，默认值为8080（必须）

socketBuffer：设Socket输出缓冲区的大小（以字节为单位），-1表示禁止缓冲，默认值为9000字节

toNoDelay：为true时，可以提高性能。默认值为true

threadPriority：设JVM中请求处理线程优先级。默认值为NORMAL-PRIORITY

### AJP连接器

用于将Apache与Tomcat集成在一起，当Apache接收到动态内容请求时，通过在配置中指定的端口号将请求发送给在此端口号上监听的AJP连接器组件。属性：

* backlog：当所有可能的请求处理线程都在使用时，队列中排队的请求最大数目。默认为10，当队列已满，任何请求都将被拒绝
* maxSpareThread：允许存在空闲线程的最大数目，默认值为50
* maxThread：最大线程数，默认值为200
* minSpareThreads：设当连接器第一次启动时创建线程的数目，确保至少有这么多的空闲线程可用，默认值为4
* port：服务端套接字的TCP端口号，默认值为8089（必须）
* topNoDelay：为true时，可以提高性能，默认值为true
* setTimeout：超时值

## Engine

为特定的Service处理所有的请示。每个Service只能包含一个Engine元素，它负责接收和处理此Service所有的连接器收到的请求，向连接发回响应，并最终显示在客户端。<Engine>至少有一个<Host>元素，必须至少有一个<Host>属性的名字与defaultHost指定的名字相匹配。属性：

* className：实现org.apache.catalina.Engine接口，默认实现类为org.apache.catalina.core.StandardEngine类
* defaultHost：默认主机名，值必须与<Service>的name值相匹配
* name：指定Engine的逻辑名字（必须）
* jvmRoute：在负载匀衡中使用的标识符，必须唯一

## Realm

(表示存放用户名，密码及role的数据库) className 指定Realm使用的类名，此类必须实现org.apache.catalina.Realm接口

## Host

表示一个虚拟主机，为特定的虚拟主机处理所有请求。属性：

* appBase：设定应用程序的基目录，绝对路径或相对于%CATALINA\_HOME%的路径名
* autoDeploy：指示Tomcat运行时，如有新的WEB程序加开appBase指定的目录下，是否为自动布署，默认值为true
* className：实现了org.apache.catalina.Host接口的类，标准实现类为org.apache.catalina.core.StandardHost类
* deployOnStartup：Tomcat启动时，是否自动部署appBase属性指定目录下所有的WEB应用程序，默认值为true
* name：虚拟主机的网络名（必须）

标准Host实现类org.apahce.catalina.core.StandardHost支持的附加属性：

* deployXML：为false将不会解析WEB应用程序内部的context.xml，默认值为true
* unPackWARs：虚拟主机指定临时读写使用的目录的路径名，不设，Tomcat会在%CATALINA\_HOME%/work目录下提供一个合适的目录。

## context

一个WEB应用程序，处理当前WEB应用程序的所有请求，每一个<Context>必须使用唯一的上下文路径。属性：

* className：实现了org.apache.catalina.Context接口的类，标准实现类org.apache.catalina.core.StandardContext类
* cookies：是否将Cookie应用于Session，默认值为true
* crossContext：是否允许跨域访问，为true时，在程序内调用ServletContext.getContext()方法将返回一个虚拟主机上其它web程序的请求调度器。默认值为false，调 径用getContext()返回为null
* docBase：绝对路径或相对于Host的appBase 属性的相对路径
* privileged：为true，允许Web应用程序使用容器的Servlet
* path：指定上下文路径。一个虚拟主机中，上下文路径必须唯一
* reloadable：为true，Tomcat运行时，如果WEB-INF/classes和WEB-INF/lib目录中有改变，Tomcat会自动重新加载该WEB应用程序。虽方便，但开销也大，默认值为false，我们在调用可以打开，发布后再关闭。
* cacheMaxSize：静态资源缓存最大值，以KB为单位，默认值为10240KB
* cachingAllowed：是否允许静态资源缓存，默认为true
* caseSensitive：默认为true,资源文件名大小写敏感，如果为false大小写不敏感
* unpackWAR：默认为true
* workDir：为WEB应用程序内部的Servlet指定临时读写的目录路径名。如没有设置，则Tomcat会在%CATALINA\_HOME%/work目录下提供一个合适的目录

# Tomcat用户角色管理配置

~/conf/tomcat-users.xml文件,关于用户角色、管理员的信息都在这个配置文件中。在默认情况下，Tomcat Manager是处于禁用状态的。准确地说，Tomcat Manager需要以用户角色进行登录并授权才能使用相应的功能，不过Tomcat并没有配置任何默认的用户，因此需要我们进行相应的用户配置之后才能使用Tomcat Manager。以下是Tomcat Manager 4种角色的大致介绍：

* manager-gui：允许访问html接口(即URL路径为/manager/html/\*)
* manager-script：允许访问纯文本接口(即URL路径为/manager/text/\*)
* manager-jmx：允许访问JMX代理接口(URL路径/manager/jmxproxy/\*)
* manager-status：允许访问Tomcat只读状态页面(即URL路径为/manager/status/\*)

注意：manager-gui、manager-script、manager-jmx均具备manager-status的权限。

|  |
| --- |
| <role rolename="manager-gui"/>  <role rolename="manager-script" />  <role rolename="manager"/>  <role rolename="admin"/>  <user username="tomcat" password="p@$$w0rd" roles="manager-gui,manager-script,manager,admin"/> |

# Tomcat部署方式

如果开启了Tomcat的reloadable=true,那么每当相关文件改变时，Tomcat会停止web app并释放内存,然后重新加载web app.这实在是个耗时的工程。

方式一：将web应用的war包拷贝到tomcat的webapps下，系统自动会将应用部署到tomcat。因为service.xml的host元素中配置了， tomcat的安装目录~/webapps中的应用。在tomcat启动或者运行的时候自动部署该目录的应用。

方式二：登陆tomcat管理控制台：http://localhost:8080/ ，输入用户名和密码后便可管理应用并动态发布。

Context Path(option)：中输入应用名，这代表你的应用的访问地址。

XML Configration file URL：指定一个xml文件。比如我们在F:\下建立一个cfg.xml文件，内容如下： <Context reloadable="false" />其中docBase不用写了，因为在下一个文本框中填入。

WAR or Directory URL：输入应用目录（或者war包），然后点击Deploy按钮，上面就可以看到了web应用程序。

注意：如果部署.war文件还有更加简单的方式，下面还有个Select WAR file uploae点击浏览选择.war文件，然后点击Deploy也可以。

# Tomcat数据源的配置

.驱动包复制到tomcat的lib目录下面

配置局部数据源（/META-INF/context.xml或者tomcat的context.xml）。

* name：指定数据源在容器中的JNDI名字。
* driverClassName：指定连接数据库的驱动
* usr,username,password
* maxActive：指定连接数据库最大活动连接数
* maxIdle：指定数据池中最大的连接空闲数
* maxWait：指定数据池中最大等待获取连接的客户端数

|  |
| --- |
| <Resource  name="jdbc/test"  auth="Container"  type="javax.sql.DataSource"  maxActive="100" maxIdle="30" maxWait="10000"  username="root" password="123"  driverClassName="com.mysql.jdbc.Driver"  url="jdbc:mysql://localhost:3306/db\_test"  /> |
| import java.sql.Connection;  import javax.naming.Context;  import javax.naming.InitialContext;  import javax.sql.DataSource;  Connection conn = null;  try {  // 初始化查找命名空间  Context ctx = new InitialContext();  // 找到DataSource  DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/test");  conn = ds.getConnection();  System.out.println(conn);  } catch (Exception e) {  e.printStackTrace();  } |