昵称：奋斗de程序猿
园龄：10个月
粉丝：2
关注：1
+加关注

搜索
找找看
谷歌搜索

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签
Spring(3)
Tomcat(3)
前端(2)
SpringBoot(2)
PHP(2)
JavaScript(1)
JDK(1)
数据库(1)
网络(1)

随笔档案
2017年5月 (2)
2017年4月 (2)
2017年1月 (12)

阅读排行榜
1. Tomcat原理详解及请求过程(10861)
2. Spring中常见的bean创建异常(8578)
3. AOP的实现原理(1668)
4. Http协议简单解析及web请求过程(502)
5. JDK动态代理实现原理(226)

Powered by 博客园

TOMCAT原理详解及请求过程

Tomcat:

Tomcat是一个JSP/Servlet容器。其作为Servlet容器，有三种工作模式：独立的Servlet容器、进程内的Servlet容器和进程外的Servlet容器。

Tomcat目录：

tomcat
|---bin：存放启动和关闭tomcat脚本
|---conf：存放不同的配置文件（server.xml和web.xml）；
|---doc：存放Tomcat文档；
|---lib/japser/common：存放Tomcat运行需要的库文件（JARS）；
|---logs：存放Tomcat执行时的LOG文件；
|---src：存放Tomcat的源代码；
|---webapps：Tomcat的主要Web发布目录（包括应用程序示例）；
|---work：存放jsp编译后产生的class文件；

Tomcat配置文件：

我们打开con文件夹可以看到Tomcat的配置文件：

server.xml: Tomcat的主配置文件，包含Service, Connector, Engine, Realm, Valve, Hosts主组件的相关配置信息；

web.xml：遵循Servlet规范标准的配置文件，用于配置servlet，并为所有的Web应用程序提供包括MIME映射等默认配置信息；

tomcat-user.xml：Realm认证时用到的相关角色、用户和密码等信息；Tomcat自带的manager默认情况下会用到此文件；在Tomcat中添加/删除用户，为用户　　指定角色等将通过编辑此文件实现；

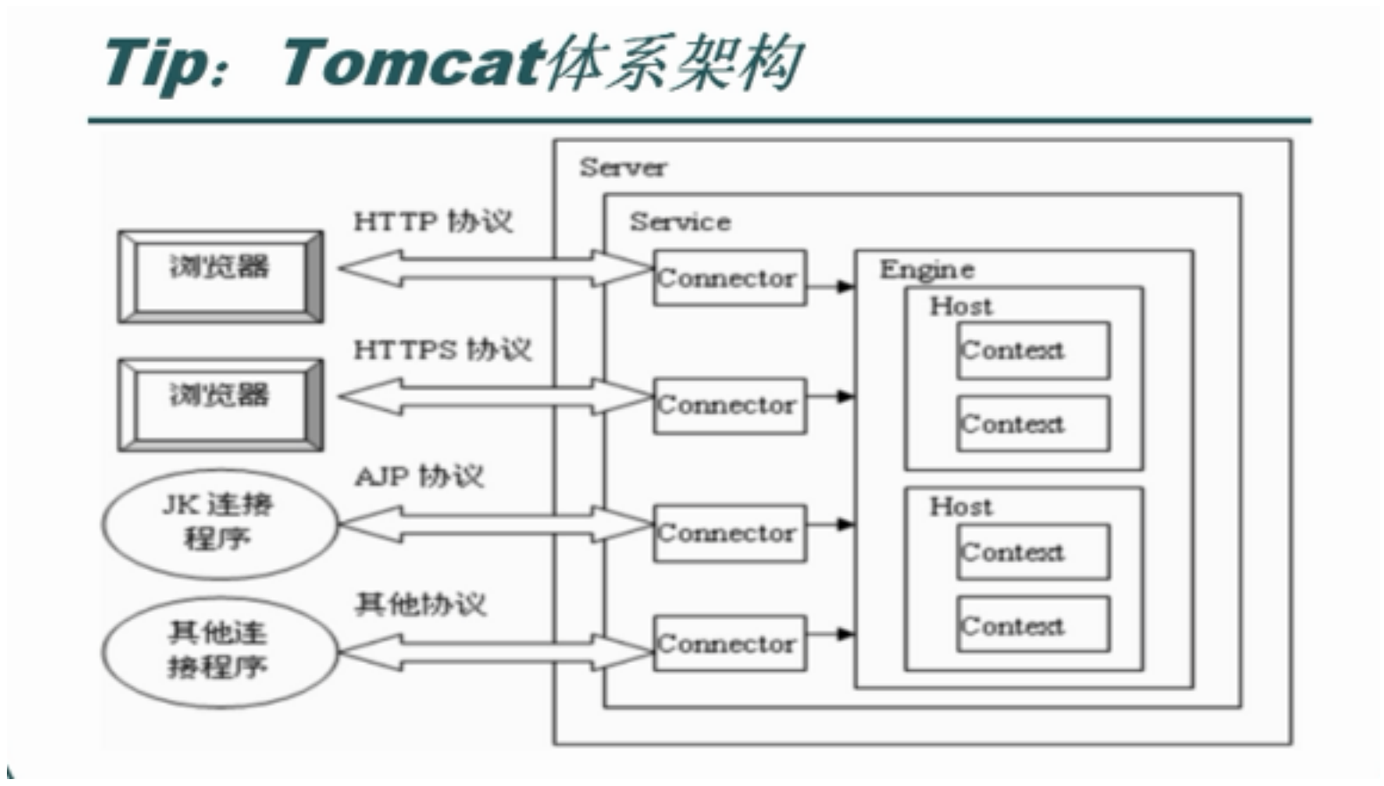
catalina.policy：Java相关的安全策略配置文件，在系统资源级别上提供访问控制的能力；

catalina.properties：Tomcat内部package的定义及访问相关控制，也包括对通过类装载器装载的内容的控制；Tomcat在启动时会事先读取此文件的相关设置；

logging.properties: Tomcat6通过自己内部实现的JAVA日志记录器来记录操作相关的日志，此文件即为日志记录器相关的配置信息，可以用来定义日志记录的组　　件级别以及日志文件的存在位置等；

context.xml：所有host的默认配置信息；

Tomcat架构及常用的组件：



1、Server组件

如上面示例文件中定义的：

<Server port=”8005” shutdown=”SHUTDOWN”>

这会让Tomcat6启动一个server实例（即一个JVM），它监听在8005端口以接收shutdown命令，使用 telnet 连接8005 端口可以直接执行 SHUTDOWN 命令来关闭 Tomcat。各Server的定义不能使用同一个端口，这意味着如果在同一个物理机上启动了多个Server实例，必须配置它们使用不同的端口。这个端口的定义用于为管理员提供一个关闭此实例的便捷途径，因此，管理员可以直接telnet至此端口使用SHUTDOWN命令关闭此实例。不过，基于安全角度的考虑，这通常不允许远程进行。

Server的相关属性：

className: 用于实现此Server容器的完全限定类的名称，默认为org.apache.catalina.core.StandardServer；

port: 接收shutdown指令的端口，默认仅允许通过本机访问，默认为8005；

shutdown：发往此Server用于实现关闭tomcat实例的命令字符串，默认为SHUTDOWN；

2、Service组件：

Service主要用于关联一个引擎和与此引擎相关的连接器，每个连接器通过一个特定的端口和协议接收入站请求交将其转发至关联的引擎进行处理。困此，Service要包含一个引擎、一个或多个连接器。

如上面示例中的定义：

<Service name=”Catalina”>

这定义了一个名为Catalina的Service，此名字也会在产生相关的日志信息时记录在日志文件当中。

Service相关的属性：

className： 用于实现service的类名，一般都是org.apache.catalina.core.StandardService。

name：此服务的名称，默认为Catalina；

3、Connector组件：

进入Tomcat的请求可以根据Tomcat的工作模式分为如下两类：

Tomcat作为应用程序服务器：请求来自于前端的web服务器，这可能是Apache, IIS, Nginx等；

Tomcat作为独立服务器：请求来自于web浏览器；

Tomcat应该考虑工作情形并为相应情形下的请求分别定义好需要的连接器才能正确接收来自于客户端的请求。一个引擎可以有一个或多个连接器，以适应多种请求方式。

定义连接器可以使用多种属性，有些属性也只适用于某特定的连接器类型。一般说来，常见于server.xml中的连接器类型通常有4种：

1) HTTP连接器 2) SSL连接器 3) AJP 1.3连接器 4) proxy连接器

如上面示例server.xml中定义的HTTP连接器：

<Connector port=”8080″ protocol=”HTTP/1.1″
maxThreads=”150″ connectionTimeout=”20000″
redirectPort=”8443″/>

定义连接器时可以配置的属性非常多，但通常定义HTTP连接器时必须定义的属性只有“port“，定义AJP连接器时必须定义的属性只有”protocol”，因为默认的协议为HTTP。以下为常用属性的说明：

```
1) address：指定连接器监听的地址，默认为所有地址，即0.0.0.0； 可以自己指定地，如
2) maxThreads：支持的最大并发连接数，默认为200；
3) port：监听的端口，默认为0；
4) protocol：连接器使用的协议，默认为HTTP/1.1，定义AJP协议时通常为AJP/1.3；
5) redirectPort：如果某连接器支持的协议是HTTP，当接收客户端发来的HTTPS请求时，则转发至此属性定义的端口；
6) connectionTimeout：等待客户端发送请求的超时时间，单位为毫秒，默认为60000，即1分钟；
7) enableLookups：是否通过request.getRemoteHost()进行DNS查询以获取客户端的主机名；默认为true； 进行反解的，可以设
8) acceptCount：设置等待队列的最大长度；通常在tomcat所有处理线程均处于繁忙状态时，新发来的请求将被放置于等待队列中；
```

下面是一个定义了多个属性的SSL连接器：

<Connector port=”8443″
maxThreads=”150″ minSpareThreads=”25″ maxSpareThreads=”75″
enableLookups=”false” acceptCount=”100″ debug=”0″ scheme=”https” secure=”true”
clientAuth=”false” sslProtocol=”TLS” />

4、Engine组件：

Engine是Servlet处理器的一个实例，即servlet引擎，默认为定义在server.xml中的Catalina。Engine需要defaultHost属性来为其定义一个接收所有发往非明确定义虚拟主机的请求的host组件。如前面示例中定义的：

<Engine name=”Catalina” defaultHost=”localhost”>

常用的属性定义：

defaultHost：Tomcat支持基于FQDN的虚拟主机，这些虚拟主机可以通过在Engine容器中定义多个不同的Host组件来实现；但如果此引擎的连接器收到一个发往非非明确定义虚拟主机的请求时则需要将此请求发往一个默认的虚拟主机进行处理，因此，在Engine中定义的多个虚拟主机的主机名称中至少要有一个跟defaultHost定义的主机名称同名；

name：Engine组件的名称，用于日志和错误信息记录时区别不同的引擎；

Engine容器中可以包含Realm、Host、Listener和Valve子容器。

5、Host组件：

位于Engine容器中用于接收请求并进行相应处理的主机或虚拟主机，如前面示例中的定义：

<Host name=”localhost” appBase=”webapps”
unpackWARs=”true” autoDeploy=”true”
xmlValidation=”false” xmlNamespaceAware=”false”>
</Host>

常用属性说明：

```
1) appBase：此Host的webapps目录，即存放非归档的web应用程序的目录或归档后的WAR文件的目录路径；可以使用基于$CATAL
2) autoDeploy：在Tomcat处于运行状态时放置于appBase目录中的应用程序文件是否自动进行deploy；默认为true;
3) unpackWars：在启用此webapps时是否对WAR格式的归档文件先进行展开；默认为true;
```

虚拟主机定义示例：

<Engine name=”Catalina” defaultHost=”localhost”>
<Host name=”localhost” appBase=”webapps”>
<Context path=”” docBase=”ROOT”/>
<Context path=”/bbs” docBase=”/web/bss” #path路径是定义在defaultHost背后的
reloadable=”true” crossContext=”true”/>
</Host>

<Host name=”mail.magedu.com” appBase=”/web/mail”>
<Context path=”” docBase=”ROOT”/>
</Host>
</Engine>

主机别名定义：

如果一个主机有两个或两个以上的主机名，额外的名称均可以以别名的形式进行定义，如下：
<Host name=”www.ttlsa.com” appBase=”webapps” unpackWARs=”true”>
<Alias>feiyu.com</Alias>
</Host>

6、Context组件：

Context在某些意义上类似于apache中的路径别名，一个Context定义用于标识tomcat实例中的一个Web应用程序；如下面的定义：

<!– Tomcat Root Context –>
<Context path=”” docBase=”/web/webapps”/>

<!– buzzin webapp –>
<Context path=”/bbs”

docBase=”/web/threads/bbs”

reloadable=”true”>

</Context>

<!– chat server –>

<Context path=”/chat” docBase=”/web/chat”/>

<!– darian web –>

<Context path=”/darian” docBase=”darian”/>

在Tomcat6中，每一个context定义也可以使用一个单独的XML文件进行，其文件的目录为$CATALINA_HOME/conf//。可以用于Context中的XML元素有Loader，Manager，Realm，Resources和WatchedResource。

常用的属性定义有：

```
1) docBase：相应的Web应用程序的存放位置；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；切记，
2) path：相对于Web服务器根路径而言的URI；如果为空“”，则表示为此webapp的根路径；如果context定义在一个单独的xml文件中
3) reloadable：是否允许重新加载此context相关的Web应用程序的类；默认为false；
```

7、Realm组件：

一个Realm表示一个安全上下文，它是一个授权访问某个给定Context的用户列表和某用户所允许切换的角色相关定义的列表。因此，Realm就像是一个用户和组相关的数据库。定义Realm时惟一必须要提供的属性是classname，它是Realm的多个不同实现，用于表示此Realm认证的用户及角色等认证信息的存放位置。

```
JAASRealm：基于Java Authintication and Authorization Service实现用户认证；
JDBCRealm：通过JDBC访问某关系型数据库表实现用户认证；
JNDIRealm：基于JNDI使用目录服务实现认证信息的获取；
MemoryRealm：查找tomcat-user.xml文件实现用户信息的获取；
UserDatabaseRealm：基于UserDatabase文件(通常是tomcat-user.xml)实现用户认证，它实现是一个完全可更新和持久有效的Me
```

下面是一个常见的使用UserDatabase的配置：

<Realm className=”org.apache.catalina.realm.UserDatabaseRealm”

resourceName=”UserDatabase”/>

下面是一个使用JDBC方式获取用户认证信息的配置：

<Realm className=”org.apache.catalina.realm.JDBCRealm” debug=”99″

driverName=”org.gjt.mm.mysql.Driver”

connectionURL=”jdbc:mysql://localhost/authority”

connectionName=”test” connectionPassword=”test”

userTable=”users” userNameCol=”user_name”

userCredCol=”user_pass”

userRoleTable=”user_roles” roleNameCol=”role_name” />

8、Valve组件：

Valve类似于过滤器，它可以工作于Engine和Host/Context之间、Host和Context之间以及Context和Web应用程序的某资源之间。一个容器内可以建立多个Valve，而且Valve定义的次序也决定了它们生效的次序。Tomcat6中实现了多种不同的Valve：

```
AccessLogValve：访问日志Valve
ExtendedAccessValve：扩展功能的访问日志Valve
JDBCAccessLogValve：通过JDBC将访问日志信息发送到数据库中；
RequestDumperValve：请求转储Valve；
RemoteAddrValve：基于远程地址的访问控制；
RemoteHostValve：基于远程主机名称的访问控制；
SemaphoreValve：用于控制Tomcat主机上任何容器上的并发访问数量；
JvmRouteBinderValve：在配置多个Tomcat为以Apache通过mod_proxy或mod_jk作为前端的集群架构中，当期望停止某节点时，
```

ReplicationValve：专用于Tomcat集群架构中，可以在某个请求的session信息发生更改时触发session数据在各节点间进行复制；

SingleSignOn：将两个或多个需要对用户进行认证webapp在认证用户时连接在一起，即一次认证即可访问所有连接在一起的webapp

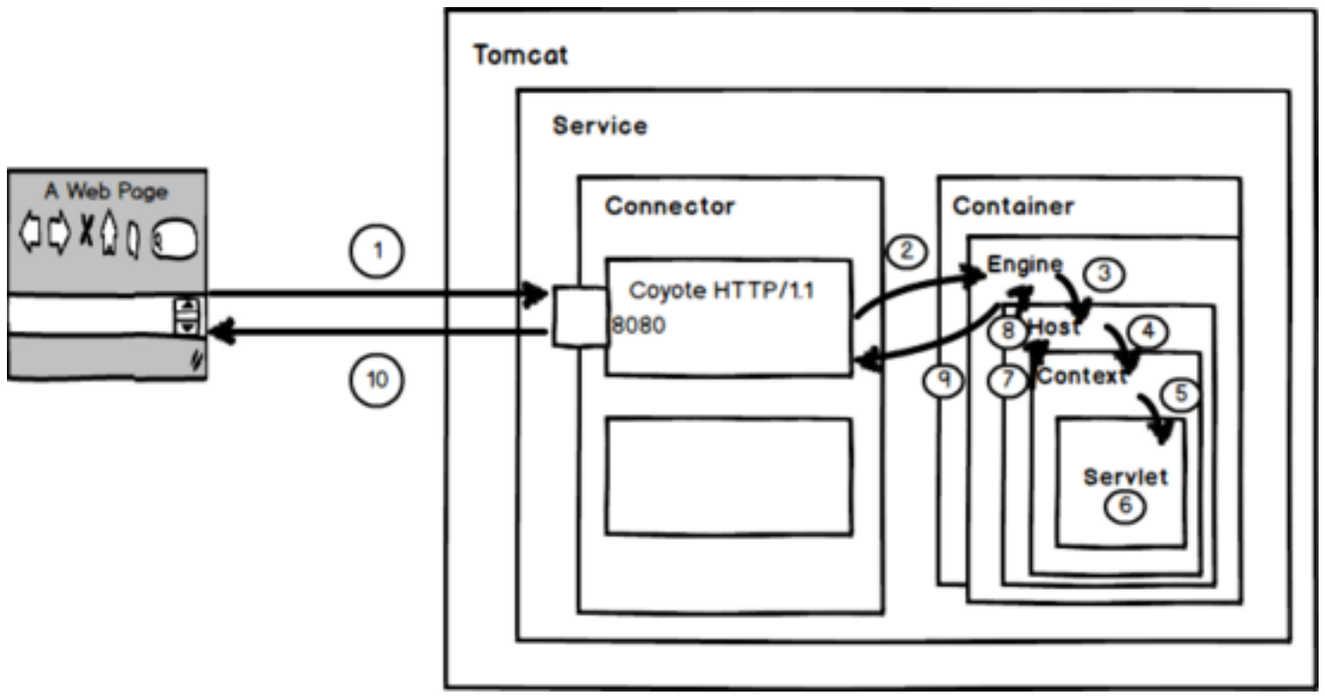
ClusterSingleSingOn：对SingleSignOn的扩展，专用于Tomcat集群当中，需要结合ClusterSingleSignOnListener进行工作；

RemoteHostValve和RemoteAddrValve可以分别用来实现基于主机名称和基于IP地址的访问控制，控制本身可以通过allow或deny来进行定义，这有点类似于Apache的访问控制功能；如下面的Valve则实现了仅允许本机访问/probe：

<Context path=”/probe” docBase=”probe”>
<Valve className=”org.apache.catalina.valves.RemoteAddrValve”
allow=”127\.0\.0\.1″/>
</Context>

Tomcat请求过程：

Tomcat Server处理一个HTTP请求的过程：



描述：

1、用户点击网页内容，请求被发送到本机端口8080，被在那里监听的Coyote HTTP/1.1 Connector获得。 2、Connector把该请求交给它所在的Service的Engine来处理，并等待Engine的回应。 3、Engine获得请求localhost/test/index.jsp，匹配所有的虚拟主机Host。 4、Engine匹配到名为localhost的Host（即使匹配不到也把请求交给该Host处理，因为该Host被定义为该Engine的默认主机），名为localhost的Host获得请求/test/index.jsp，匹配它所拥有的所有的Context。Host匹配到路径为/test的Context（如果匹配不到就把该请求交给路径名为“ ”的Context去处理）。 5、path=“/test”的Context获得请求/index.jsp，在它的mapping table中寻找出对应的Servlet。Context匹配到URL PATTERN为*.jsp的Servlet,对应于JspServlet类。 6、构造HttpServletRequest对象和HttpServletResponse对象，作为参数调用JspServlet的doGet（）或doPost（）.执行业务逻辑、数据存储等程序。 7、Context把执行完之后的HttpServletResponse对象返回给Host。 8、Host把HttpServletResponse对象返回给Engine。 9、Engine把HttpServletResponse对象返回Connector。 10、Connector把HttpServletResponse对象返回给客户Browser。

Tomcat配置虚拟主机：

Tomcat可以通过修改本地虚拟主机和修改本地DNS的方式下，实现访问主流网站：www.baidu.com而跳转到自己配置的界面。

```
<Host name="localhost"  appBase="webapps"
      unpackWARs="true" autoDeploy="true"
      xmlValidation="false" xmlNamespaceAware="false">

    <!-- SingleSignOn valve, share authentication between web applications
         Documentation at: /docs/config/valve.html -->
    <!--
    <Valve className="org.apache.catalina.authenticator.SingleSignOn" />
    -->

    <!-- Access log processes all example.
         Documentation at: /docs/config/valve.html -->
    <!--
    <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
           prefix="localhost_access_log." suffix=".txt" pattern="common" resolveHosts="false"/>
    -->
    <Context path="" docBase="C:\apache-tomcat-6.0.20\webapps\mail"/>

  </Host>


  <Host name="www.sina.com" appBase="c:\sina">
            <Context path="/mail" docBase="c:\sina\mail"/>
  </Host>
```

修改本地dns的步骤：
c盘-windows-system32-driver-etc-host
修改host中的ip地址，完成修改。

标签：Tomcat

奋斗de程序猿
关注 - 1
粉丝 - 2
+加关注

« 上一篇：mysql数据库乱码的问题解决
» 下一篇：Http协议简单解析及web请求过程

发表于 2017-01-09 11:06 奋斗DE程序猿 阅读(10863) 评论(0) 编辑 收藏