# Tomcat-server.xml配置文件详解

# 序

server.xml是Tomcat中最重要的配置文件，**server.xml的每个元素都对应了Tomcat中的一个组件；**通过对xml文件中元素的配置，可以实现对Tomcat中各个组件的控制。

# 1. server.xml的整体结构

|  |
| --- |
| <Server>  <Service>  <Connector />  <Connector />  <Engine>  <Host>  <Context /><!-- 现在常常使用自动部署，不推荐配置Context元素，Context小节有详细说明 -->  </Host>  </Engine>  </Service>  </Server> |

该结构只是给出了Tomcat的核心组件，除了核心组件外，Tomcat还有一些其他组件。下面我们开始介绍这些核心组件。

# 2. 元素分类

**1. 顶层元素：<Server></Server>**

<Server>元素是整个配置文件的根元素，<Server>元素代表**一个Engine元素**以及**一组与之相连的Connector元素**。

**2. 连接器：<Connector>**

<Connector>代表了外部客户端发送请求到特定Service的接口；同时也是外部客户端从特定Service接收响应的接口。

**3. 容器：<Engine><Host><Context>**

容器的功能时处理Connector接收进来的请求，并产生相应的响应。Engine包含Host，Host包含Context。一个Engine组件可以处理Service中的所有请求，一个Host组件可以处理发向一个特定虚拟主机的所有请求，一个Context组件可以处理一个特定Web应用的所有请求。

# 3. 核心组件

## 3.1 Server

Server元素在最顶层，代表整个Tomcat容器，因此他必须是server.xml中唯一一个最外层的元素。**一个Server元素可以有一个或多个Service元素。**

|  |
| --- |
| <Server port="8005" shutdown="SHUTDOWN">  </Server> |

可以看到，最外层有一个<Server>元素，shutdown属性表示关闭Server的指令；port属性表示Server接收shutdown指令的端口号，设置为-1可以禁掉该端口。

Server的主要任务，就是提供一个接口让客户端能够访问到这个Service集合，同时维护它所包含的所有的Service的生命周期，包含如何初始化，如何结束服务，如何找到客户端要访问的Service。

## 3.2 Service

service的作用，是在Connector和Engine外面包一层，把它们组合在一起，对外提供服务。**一个Service可以包含多个Connector，但是只能包含一个Engine**；其中Connector的作用是从客户端接收请求，Engine的作用是处理接收进来的请求。

|  |
| --- |
| <Server port="8005" shutdown="SHUTDOWN">  <Service name="Catalina">  </Service>  </Server> |

如上图，Server中包含一个名称为“Catalina”的Service。实际上，Tomcat可以提供多个Service，不同的Service监听不同的端口。

## 3.3 Connector

Connector的主要功能，是接收连接请求，创建Request和Response对象用于和请求端交换数据；然后分配线程让Engine来处理这个请求，并把产生的Request和Response对象传给Engine。

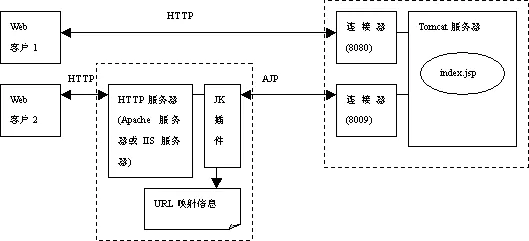
通过配置Connector，可以控制请求Service的协议及端口号。

|  |
| --- |
| <Server port="8005" shutdown="SHUTDOWN">  <Service name="Catalina">  <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />  <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />  </Service>  </Server> |

通过配置第一个Connector，客户端可以通过8080端口号协议访问tomcat。其中，protocol属性规定了请求的协议，port规定了请求的端口号，redirectPort表示当强制要求https而请求是http时，重定向至端口号为8443的Connector，connectionTimeout表示连接的超时时间。

在这个例子中，Tomcat监听Http请求，使用的是8080端口，而不是正式的80端口；实际上，在正式的生产环境中，Tomcat也常常监听8080端口。而不是80端口。这是因为在生产环境中，很少讲Tomcat直接对外开放接收请求，而是在Tomcat和客户端之间加一层代理服务器（如nginx），用于请求的转发、负载均衡、处理静态文件等；通过代理服务器访问Tomcat时，是在局域网中，因为一般仍使用8080端口。

第二个配置Connector，客户端可以通过8009端口使用AJP协议访问Tomcat。AJP协议负责和其他的Http服务器（如Apache）建立连接；在把Tomcat与其他服务器集成时，就需要用到这个**连接器**，之所以使用Tomcat和其他服务器集成，是因为Tomcat可以用作Servlet/JSP容器，但是对静态资源处理速度较慢，不如Apache和IIS等HTTP服务器；因此常常将Tomcat和Apache等集成，前者做Servlet容器，后者处理静态资源，而AJP协议便负责Tomcat与Apache的连接。Tomcat和Apache等集成的原理如下图：



# 4 Engine

Engine组件在Service组件有且只有一个；Engine是service组件中的请求处理组件。Engine组件从一个或多个Connector中接收并处理，并将完成的响应返回给Connector，最终传递给客户端。

前面说到，Engine、Host和Context都是容器，但是它们不是平行关系，而是父子关系：Engine包含Host，Host包含Context。

|  |
| --- |
| <Server port="8005" shutdown="SHUTDOWN">  <Service name="Catalina">  <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">  </Engine>  </Service>  </Server> |

其中name属性用于日志和错误信息，在整个Server中应该是唯一的。defalutHost属性指定了默认的host名称，当发往本机的请求指定的host名称不存在时，一律使用defaultHost指定的host进行处理；**因此defaulthost的值，必须与Engine中的一个Host组件的name属性值匹配。**

# 5. Host

**5.1 Engine和Host**

Host是Engine的子容器。Engine组件中可以内嵌1个或者多个Host组件，**每个Host组件代表Engine中的一个虚拟主机。**Host组件至少有一个，且其中一个的name必须与Engine组件中的defaultHost属性相匹配。

**5.2 Host的作用**

Host虚拟主机的作用，是运行多个Web应用（一个Context代表一个Web应用），并负责安装、展开、启动、结束每个Web应用。

Host组件代表的虚拟主机，对应服务器中一个网络名实体（如"[www.test.com](https://links.jianshu.com/go?to=http://www.test.com" \t "https://www.jianshu.com/p/_blank)"或IP地址"116.25.25.25"）；为了使用户可以通过网络名连接Tomcat服务器，这个名字应该在DNS服务器上注册。

客户端通常使用主机名来标识它们希望连接的服务器，该主机名也会包含在HTTP请求头中，Tomcat从HTTP头中提取出主机名，寻找名字匹配的主机。如果没有匹配，请求会发送至默认的主机。因此默认主机不需要再DNS服务器中注册网络名，因为任何与所有Host名称不匹配的请求，都会路由至默认主机。

**5.3 Host的配置**

在第一部分的例子中，Host配置如下：

|  |
| --- |
| <Server port="8005" shutdown="SHUTDOWN">  <Service name="Catalina">  <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">  <Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">    </Host>  </Engine>  </Service>  </Server> |

name属性指定虚拟主机的主机名，一个Engine有且只有一个Host组件的name属性和Engine组件的defaultHost属性相匹配；一般情况下，主机名需要是在DNS服务器中注册网络名，但是Engine指定的defaultHost不需要。

**unpackWARs指定了是否将代表Web应用的WAR文件解压；如果是true，通过解压后的文件结构运行该Web应用，如果是false，直接使用WAR文件运行Web应用。**

# 6 Context

**6.1 Context的作用**

**Context元素代表在虚拟主机上运行的一个Web应用。**在后文中，提到Context、应用或Web应用，他们都代指Web应用，每个Web应用基于WAR文件，或WAR文件解压后对应的目录（这里称为应用目录）  
Context是Host的子容器，每个Host都可以定义任意多的Context元素。

若是自动部署（即autoDeploy="true"），那么server.xml配置文件中没有Context元素的配置。这是因为Tomcat开启了自动部署，Web应用没有在server.xml中配置静态部署，而是由Tomcat通过特定的规则自动部署。

**6.2 Web的自动部署**

要开启Web应用的自动部署，需要配置所在的虚拟主机；配置的方式就是在配置Host元素的**deployOnStartup和autoDeploy属性**。如果deployOnStartup和autoDeploy设置为true，则tomcat启动自动部署：当检测到新的Web应用或Web应用更新时，会触发应用的部署（或重新部署）。二者的主要区别在于

* deployeOnStartup为true时，Tomcat在启动时检查Web应用，且检测到所有的Web应用都试做新应用；
* autoDeploy为true时，Tomcat在运行时定期检查新的Web应用或Web应用的更新；

通过配置deployOnStartup和autoDeploy可以开启虚拟主机自动部署Web应用；实际上，自动部署依赖于检查是否有新的或更改过的Web应用，而Host元素中的appBase和xml配置设置了检查web应用更新的目录。

其中，appBase属性指定Web应用所在的目录，默认值是webapps，这是一个相对路径，代表Tomcat根目录下的webapps文件夹。

xmlBase属性指定Web应用的XML配置文件所在的目录，默认值为conf/<engine\_name><engine\_name>，例如上面例子中，主机localhost的xmlBase的默认值是$TOMCAT\_HOME/conf/Catalina/localhost。

**6.3 检查WEB应用更新**

一个Web应用可能包含以下文件：xml配置文件，WAR包，以及一个应用目录（该目录包含Web应用的文件结构）；其中XML配置文件位于xmlBase指定的目录，WAR包和应用目录位于appBase指定的目录。

Tomcat按照如下的顺序进行扫描，来检查应用更新：

1. 扫描虚拟主机指定的xmlBase下的xml配置；
2. 扫描虚拟主机指定的appBase下的WAR文件；
3. 扫描虚拟主机指定的appBase下的应用目录；

**6.4 <Context>元素的配置**

Context元素最重要的属性是docBase和path，此外reloadable属性也比较常用。

**docBase指定了该Web应用使用WAR包路径，或应用目录。**需要注意的是：**在自动部署场景下（配置文件位于xmlBase中），docBase不在appBase目录中，才需要指定；如果docBase指定的WAR包或应用目录就在appBase中，则不需要指定。**因为Tomcat会自动扫描appBase中的WAR包和应用目录，制定了反而造成问题。

**path指定了访问该Web应用上下文路径**，当请求到来时，Tomcat根据Web应用的path属性与URL匹配程度来选择Web应用处理相应请求。例如：Web应用app1的path属性是"/app1"，Web应用app2的path属性是"/app2"，那么请求/app1/index.html会交由app1来处理；而请求/app2/index.html会交由app2来处理。**如果一个Context元素的path属性为""，那么这个Context是虚拟主机的默认的Web应用；当请求的uri与所有的path都不匹配时，使用该默认Web应用来处理。**

但是，需要注意的是，**在自动部署场景（配置文件位于xmlBase中），不能指定path属性，path属性由配置的文件的文件名，WAR文件的文件名或应用目录的名称自动推导出来。**如扫描Web应该时，发现xmlBase目录下的app1.xml，或appBase目录下的app1.WAR或app1应用目录，则该Web用于的path属性是"app1"。如果名称不是app1而是ROOT，则该Web应用时虚拟主机默认的Web应用，此时path属性推导为""。

reloadable属性指示tomcat是否在运行时监控在WEB-INF/classes和WEB-INF/lib目录下class文件的改动。如果值为true，那么当class文件改动时，会重新web应用的重新加载。在开发环境下，reloadable设置为ture便于调试；但是在生产环境中设置为true会给服务器带来性能压力，因此reloadable参数的默认值为false。

|  |
| --- |
| <Context docBase="D:\Program Files\app1.war" reloadable="true">  </Context> |

在该例子中，docBase位于Host的appBase目录之外；path属性没有指定，而是根据app1.xml自动推导为"app1"。

**6.5 自动部署举例**

最典型的自动部署，就是当我们安装完Tomcat后，$TOMCAT\_HOME/webapps目录有如下目录：



image.png

当我们启动tomcat后，可以使用[http://localhost:8080/](https://links.jianshu.com/go?to=http://localhost:8080/" \t "https://www.jianshu.com/p/_blank)来访问Tomcat，其实访问的就是ROOT对应的Web应用；我们也可以通过[http://localhost:8080/docs](https://links.jianshu.com/go?to=http://localhost:8080/docs" \t "https://www.jianshu.com/p/_blank)来访问docs应用。

**6.6 server.xml中静态部署Web应用**

**除了自动部署，我们也可以在server.xml通过<Context>元素部署Web应用。**静态部署与自动部署是可以共存的。但是静态部署中server.xml是不可动态重加载的资源，服务器一旦启动，要修改这个文件，就得重启服务器才能重新加载。

server.xml中使用Context元素配置web应用，Context元素位于Host元素中。

|  |
| --- |
| <Server port="8005" shutdown="SHUTDOWN">  <Service name="Catalina">  <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">  <Host name="localhost" appBase="webapp" unpackWARs="true" autoDeploy="true">  <Context path="" docBase="/app/tomcat/webapps/xxx" reloadable="true" workDir="/app/tomcat/work" ></Context>  </Host>  </Engine>  </Service>  </Server> |

* docBase：静态部署时，docBase可以在appBase目录下，也可以不在。
* path：静态部署时，可以显示指定path属性，但是仍然受到了严格的限制：只有当自动部署完全关闭(deployOnStartup和autoDeploy都为false)或docBase不在appBase中时，才可以设置path属性。在本例中，docBase不在appBase中，因此path属性可以设置。
* reloadable：属性用法和自动部署相同。

**appBase和docBase的区别？**

appBase：这个目录下面的子目录将自动被部署为应用，且war文件将被自动解压缩并部署为应用，默认为tomcat下webapps目录。

docBase：指定需要关联的项目自动解压并部署到appBase目录下。项目的名称由path属性决定。

先部署<Context path="" docBase="/app/tomcat/webapps/xxx"> 需要注意，docBase所在的文件或者war包必须存在。否则项目启动找不到对应的目录。此时文件解压到appBase目录下，根据path属性，决定解压后的文件名。

若采用了<Host name="localhost" appBase="webapp" autoDeploy="true">配置，那么appBase目录下的应用目录将会再次部署。  
此时项目是部署了两遍。

解决办法，设置autoDeploy="false"。