**Tomcat配置文件(Server.xml)介绍**

Tomcat隶属于Apache基金会，是开源的轻量级Web应用服务器，使用非常广泛。server.xml是Tomcat中最重要的配置文件，**server.xml的每一个元素都对应了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。因此，学习server.xml文件的配置，对于了解和使用Tomcat至关重要。

说明：由于server.xml文件中元素与Tomcat中组件的对应关系，后文中为了描述方便，“元素”和“组件”的使用不严格区分。

1. **一个server.xml配置实例**

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

<Listener className="org.apache.catalina.startup.VersionLoggerListener" />

<Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />

<Listener className="org.apache.catalina.core.JasperListener" />

<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="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="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>

1. **server.xml文档的元素分类和整体结构**
2. **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>

该结构中只给出了Tomcat的核心组件，除了核心组件外，Tomcat还有一些其他组件，下面介绍一下组件的分类。

1. **元素分类：**

server.xml文件中的元素可以分为以下4类：

（1）顶层元素：<Server>和<Service>

<Server>元素是整个配置文件的根元素，<Service>元素则代表一个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>

容器的功能是处理Connector接收进来的请求，并产生相应的响应。Engine、Host和Context都是容器，但它们不是平行的关系，而是父子关系：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。

（4）内嵌组件：可以内嵌到容器中的组件。实际上，**Server、Service、Connector、Engine、Host和Context是最重要的最核心的Tomcat组件**，其他组件都可以归为内嵌组件。

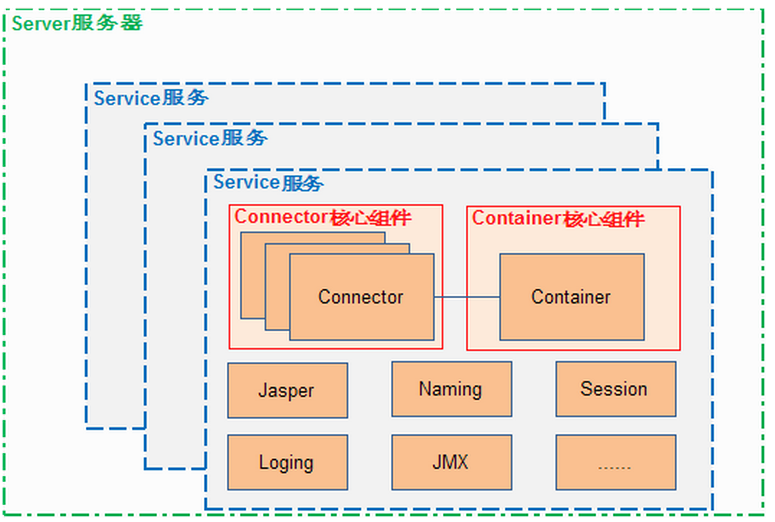
**三、核心组件**

1. **Server**

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

在第一部分的例子中，在最外层有一个<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。

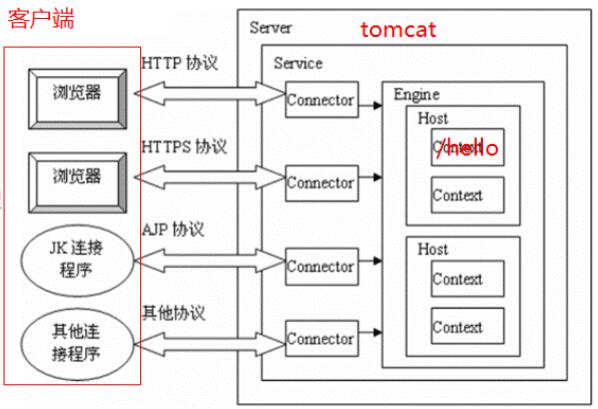
Server的主要任务，就是提供一个接口让客户端能够访问到这个Service集合，同时维护它所包含的所有的Service的声明周期，包括如何初始化、如何结束服务、如何找到客户端要访问的Service。



**2、Service**

Service的作用，是在Connector和Engine外面包了一层，把它们组装在一起，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。

在第一部分的例子中，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，后文会有介绍。



**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。在第一部分的例子中，Service包含两个Connector：

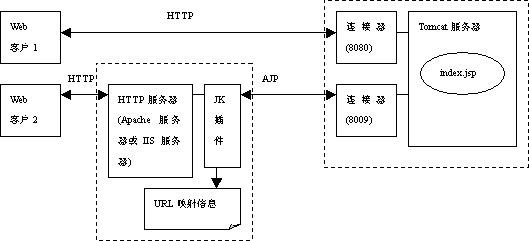
1 <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />

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

（1）通过配置第1个Connector，客户端可以通过8080端口号使用http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，很少将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，因此一般仍使用8080端口。

（2）通过配置第2个Connector，客户端可以通过8009端口号使用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HTTP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，但是对静态资源的处理速度较慢，不如Apache和IIS等HTTP服务器；因此常常将Tomcat与Apache等集成，前者作Servlet容器，后者处理静态资源，而AJP协议便负责Tomcat和Apache的连接。Tomcat与Apache等集成的原理如下图([图片来源](http://blog.163.com/cmdbat@126/blog/static/170292123201311301419411/))：



1. **Engine**

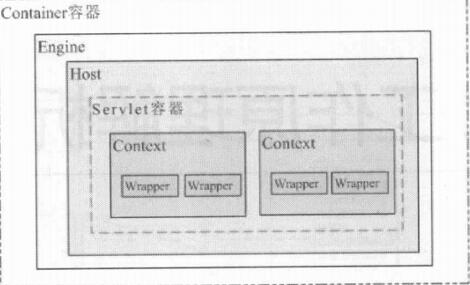
**Engine组件在Service组件中有且只有一个；Engine是Service组件中的请求处理组件。**Engine组件从一个或多个Connector中接收请求并处理，并将完成的响应返回给Connector，最终传递给客户端。

前面已经提到过，Engine、Host和Context都是容器，但它们不是平行的关系，而是父子关系：Engine包含Host，Host包含Context。

在第一部分的例子中，Engine的配置语句如下：

1 <Engine name="Catalina" defaultHost="localhost">

其中，name属性用于日志和错误信息，在整个Server中应该唯一。defaultHost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。



1. **Host**

**（1）Engine与Host**

Host是Engine的子容器。Engine组件中可以内嵌1个或多个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组件的defaultHost属性相匹配。

**（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、启动和结束每个Web应用。

Host组件代表的虚拟主机，对应了服务器中一个网络名实体(如”www.test.com”，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，寻找名称匹配的主机。如果没有匹配，请求将发送至默认主机。因此默认主机不需要是在DNS服务器中注册的网络名，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。

**（3）Host的配置**

在第一部分的例子中，Host的配置如下：

1 <Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">

下面对其中配置的属性进行说明：

name属性指定虚拟主机的主机名，一个Engine中有且仅有一个Host组件的name属性与Engine组件的defaultHost属性相匹配；一般情况下，主机名需要是在DNS服务器中注册的网络名，但是Engine指定的defaultHost不需要，原因在前面已经说明。

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

Host的autoDeploy和appBase属性，与Host内Web应用的自动部署有关；此外，本例中没有出现的xmlBase和deployOnStartup属性，也与Web应用的自动部署有关；将在下一节(Context)中介绍。

1. **Context**

**（1）Context的作用**

**Context元素代表在特定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，它们指代的都是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中可以定义任意多的Context元素。

在第一部分的例子中，可以看到server.xml配置文件中并没有出现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。下面介绍一下Tomcat自动部署Web应用的机制。

**（2）Web应用自动部署**

**Host的配置**

要开启Web应用的自动部署，需要配置所在的虚拟主机；配置的方式就是前面提到的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应用的更新时，会触发应用的部署(或重新部署)。二者的主要区别在于，deployOnStartup为true时，Tomcat在启动时检查Web应用，且检测到的所有Web应用视作新应用；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。除此之外，二者的处理相似。

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元素的appBase和xmlBase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>/<host\_name>，例如第一部分的例子中，主机localhost的xmlBase的默认值是$TOMCAT\_HOME/conf/Catalina/localhost。

**检查Web应用更新**

一个Web应用可能包括以下文件：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：

A、扫描虚拟主机指定的xmlBase下的XML配置文件

B、扫描虚拟主机指定的appBase下的WAR文件

C、扫描虚拟主机指定的appBase下的应用目录

**<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应用使用的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docBase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，指定了反而会造成问题。

path指定了访问该Web应用的上下文路径，当请求到来时，Tomcat根据Web应用的 path属性与URI的匹配程度来选择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，**在自动部署场景下(配置文件位于xmlBase中)，不能指定path属性，path属性由配置文件的文件名、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应用时，发现了xmlBase目录下的app1.xml，或appBase目录下的app1.WAR或app1应用目录，则该Web应用的path属性是”app1”。如果名称不是app1而是ROOT，则该Web应用是虚拟主机默认的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，会触发Web应用的重新加载。在开发环境下，reloadable设置为true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。

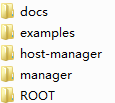
下面来看自动部署时，xmlBase下的XML配置文件app1.xml的例子：

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

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”；由于是在开发环境下，因此reloadable设置为true，便于开发调试。

**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目录下有如下文件夹：



当我们启动Tomcat后，可以使用http://localhost:8080/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来访问docs应用，同理我们可以访问examples/host-manager/manager这几个Web应用。

**（3）server.xml中静态部署Web应用**

**除了自动部署，我们也可以在server.xml中通过<context>元素静态部署Web应用。**静态部署与自动部署是可以共存的。在实际应用中，并不推荐使用静态部署，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。而自动部署可以在Tomcat运行时通过定期的扫描来实现，不需要重启服务器。

server.xml中使用Context元素配置Web应用，Context元素应该位于Host元素中。举例如下：

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

docBase：静态部署时，docBase可以在appBase目录下，也可以不在；本例中，docBase不在appBase目录下。

path：静态部署时，可以显式指定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属性的用法与自动部署时相同。

**四、核心组件的关联**

**1、整体关系**

核心组件之间的整体关系，在上一部分有所介绍，这里总结一下：

Server元素在最顶层，代表整个Tomcat容器；一个Server元素中可以有一个或多个Service元素。

Service在Connector和Engine外面包了一层，把它们组装在一起，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；Connector接收请求，Engine处理请求。

Engine、Host和Context都是容器，且 Engine包含Host，Host包含Context。每个Host组件代表Engine中的一个虚拟主机；每个Context组件代表在特定Host上运行的一个Web应用。

**2、如何确定请求由谁处理？**

当请求被发送到Tomcat所在的主机时，如何确定最终哪个Web应用来处理该请求呢？

**（1）根据协议和端口号选定Service和Engine**

Service中的Connector组件可以接收特定端口的请求，因此，当Tomcat启动时，Service组件就会监听特定的端口。在第一部分的例子中，Catalina这个Service监听了8080端口（基于HTTP协议）和8009端口（基于AJP协议）。当请求进来时，Tomcat便可以根据协议和端口号选定处理请求的Service；Service一旦选定，Engine也就确定。

通过在Server中配置多个Service，可以实现通过不同的端口号来访问同一台机器上部署的不同应用。

**（2）根据域名或IP地址选定Host**

Service确定后，Tomcat在Service中寻找名称与域名/IP地址匹配的Host处理该请求。如果没有找到，则使用Engine中指定的defaultHost来处理该请求。在第一部分的例子中，由于只有一个Host（name属性为localhost），因此该Service/Engine的所有请求都交给该Host处理。

（3）根据URI选定Context/Web应用

这一点在Context一节有详细的说明：Tomcat根据应用的 path属性与URI的匹配程度来选择Web应用处理相应请求，这里不再赘述。

（4）举例

以请求http://localhost:8080/app1/index.html为例，首先通过协议和端口号（http和8080）选定Service；然后通过主机名（localhost）选定Host；然后通过uri（/app1/index.html）选定Web应用。

**3、如何配置多个服务**

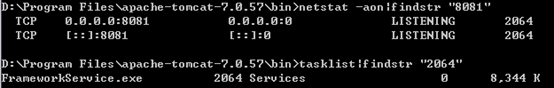
通过在Server中配置多个Service服务，可以实现通过不同的端口号来访问同一台机器上部署的不同Web应用。

在server.xml中配置多服务的方法非常简单，分为以下几步：

（1）复制<Service>元素，放在当前<Service>后面。

（2）修改端口号：根据需要监听的端口号修改<Connector>元素的port属性；必须确保该端口没有被其他进程占用，否则Tomcat启动时会报错，而无法通过该端口访问Web应用。

以Win7为例，可以用如下方法找出某个端口是否被其他进程占用：netstat -aon|findstr "8081"发现8081端口被PID为2064的进程占用，tasklist |findstr "2064"发现该进程为FrameworkService.exe(这是McAfee杀毒软件的进程)。



（3）修改Service和Engine的name属性

（4）修改Host的appBase属性（如webapps2）

（5）Web应用仍然使用自动部署

（6）将要部署的Web应用(WAR包或应用目录)拷贝到新的appBase下。

以第一部分的server.xml为例，多个Service的配置如下：

[复制代码](javascript:void(0);)

1 <?xml version='1.0' encoding='utf-8'?>

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

3 <Listener className="org.apache.catalina.startup.VersionLoggerListener" />

4 <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />

5 <Listener className="org.apache.catalina.core.JasperListener" />

6 <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />

7 <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />

8 <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

9

10 <GlobalNamingResources>

11 <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" />

12 </GlobalNamingResources>

13

14 <Service name="Catalina">

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

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

17 <Engine name="Catalina" defaultHost="localhost">

18 <Realm className="org.apache.catalina.realm.LockOutRealm">

19 <Realm className="org.apache.catalina.realm.UserDatabaseRealm"

20 resourceName="UserDatabase"/>

21 </Realm>

22

23 <Host name="localhost" appBase="/opt/project/webapps" unpackWARs="true" autoDeploy="true">

24 <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" />

25 </Host>

26 </Engine>

27 </Service>

28

29 <Service name="Catalina2">

30 <Connector port="8084" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />

31 <Connector port="8010" protocol="AJP/1.3" redirectPort="8443" />

32 <Engine name="Catalina2" defaultHost="localhost">

33 <Realm className="org.apache.catalina.realm.LockOutRealm">

34 <Realm className="org.apache.catalina.realm.UserDatabaseRealm"

35 resourceName="UserDatabase"/>

36 </Realm>

37

38 <Host name="localhost" appBase="/opt/project/webapps2" unpackWARs="true" autoDeploy="true">

39 <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" />

40 </Host>

41 </Engine>

42 </Service>

43 </Server>

[复制代码](javascript:void(0);)

再将原webapps下的docs目录拷贝到webapps2中，则通过如下两个接口都可以访问docs应用：

http://localhost:8080/docs/

http://localhost:8084/docs/

**五、其他组件**

除核心组件外，server.xml中还可以配置很多其他组件。下面只介绍第一部分例子中出现的组件，如果要了解更多内容，可以查看[Tomcat官方文档](http://tomcat.apache.org/tomcat-8.0-doc/config/index.html)。

**1、Listener**

[复制代码](javascript:void(0);)

1 <Listener className="org.apache.catalina.startup.VersionLoggerListener" />

2 <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />

3 <Listener className="org.apache.catalina.core.JasperListener" />

4 <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />

5 <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />

6 <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

[复制代码](javascript:void(0);)

Listener(即监听器)定义的组件，可以在特定事件发生时执行特定的操作；被监听的事件通常是Tomcat的启动和停止。

监听器可以在Server、Engine、Host或Context中，本例中的监听器都是在Server中。实际上，本例中定义的6个监听器，都只能存在于Server组件中。监听器不允许内嵌其他组件。

监听器需要配置的最重要的属性是className，该属性规定了监听器的具体实现类，该类必须实现了org.apache.catalina.LifecycleListener接口。

下面依次介绍例子中配置的监听器：

* VersionLoggerListener：当Tomcat启动时，该监听器记录Tomcat、Java和操作系统的信息。该监听器必须是配置的第一个监听器。
* AprLifecycleListener：Tomcat启动时，检查APR库，如果存在则加载。APR，即Apache Portable Runtime，是Apache可移植运行库，可以实现高可扩展性、高性能，以及与本地服务器技术更好的集成。
* JasperListener：在Web应用启动之前初始化Jasper，Jasper是JSP引擎，把JVM不认识的JSP文件解析成java文件，然后编译成class文件供JVM使用。
* JreMemoryLeakPreventionListener：与类加载器导致的内存泄露有关。
* GlobalResourcesLifecycleListener：通过该监听器，初始化< GlobalNamingResources>标签中定义的全局JNDI资源；如果没有该监听器，任何全局资源都不能使用。< GlobalNamingResources>将在后文介绍。
* ThreadLocalLeakPreventionListener：当Web应用因thread-local导致的内存泄露而要停止时，该监听器会触发线程池中线程的更新。当线程执行完任务被收回线程池时，活跃线程会一个一个的更新。只有当Web应用(即Context元素)的renewThreadsWhenStoppingContext属性设置为true时，该监听器才有效。

**2、GlobalNamingResources与Realm**

第一部分的例子中，Engine组件下定义了Realm组件：

1 <Realm className="org.apache.catalina.realm.LockOutRealm">

2 <Realm className="org.apache.catalina.realm.UserDatabaseRealm"

3 resourceName="UserDatabase"/>

4 </Realm>

Realm，可以把它理解成“域”；**Realm提供了一种用户密码与web应用的映射关系，从而达到角色安全管理的作用。**在本例中，Realm的配置使用name为UserDatabase的资源实现。而该资源在Server元素中使用GlobalNamingResources配置：

1 <GlobalNamingResources>

2 <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" />

3 </GlobalNamingResources>

GlobalNamingResources元素定义了全局资源，通过配置可以看出，该配置是通过读取$TOMCAT\_HOME/ conf/tomcat-users.xml实现的。

关于Tomcat域管理的更多内容，可以参考：[Realm域管理](http://www.cnblogs.com/xing901022/p/4552843.html)

**3、Valve**

在第一部分的例子中，Host元素内定义了Valve组件：

1 <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" />

单词Valve的意思是“阀门”，在Tomcat中代表了请求处理流水线上的一个组件；Valve可以与Tomcat的容器(Engine、Host或Context)关联。

不同的Valve有不同的特性，下面介绍一下本例中出现的AccessLogValve。

AccessLogValve的作用是通过日志记录其所在的容器中处理的所有请求，在本例中，Valve放在Host下，便可以记录该Host处理的所有请求。**AccessLogValve记录的日志就是访问日志，每天的请求会写到一个日志文件里。**AccessLogValve可以与Engine、Host或Context关联；在本例中，只有一个Engine，Engine下只有一个Host，Host下只有一个Context，因此AccessLogValve放在三个容器下的作用其实是类似的。

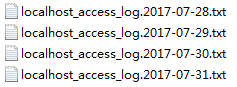
本例的AccessLogValve属性的配置，使用的是默认的配置；下面介绍AccessLogValve中各个属性的作用：

（1）className：规定了Valve的类型，是最重要的属性；本例中，通过该属性规定了这是一个AccessLogValve。

（2）directory：指定日志存储的位置，本例中，日志存储在$TOMCAT\_HOME/logs目录下。

（3）prefix：指定了日志文件的前缀。

（4）suffix：指定了日志文件的后缀。通过directory、prefix和suffix的配置，在$TOMCAT\_HOME/logs目录下，可以看到如下所示的日志文件。



（5）pattern：指定记录日志的格式，本例中各项的含义如下：

* %h：远程主机名或IP地址；如果有nginx等反向代理服务器进行请求分发，该主机名/IP地址代表的是nginx，否则代表的是客户端。后面远程的含义与之类似，不再解释。
* %l：远程逻辑用户名，一律是”-”，可以忽略。
* %u：授权的远程用户名，如果没有，则是”-”。
* %t：访问的时间。
* %r：请求的第一行，即请求方法(get/post等)、uri、及协议。
* %s：响应状态，200,404等等。
* %b：响应的数据量，不包括请求头，如果为0，则是””-。

例如，下面是访问日志中的一条记录

https://images2017.cnblogs.com/blog/1174710/201708/1174710-20170804130221459-772676907.png

pattern的配置中，除了上述各项，还有一个非常常用的选项是%D，含义是请求处理的时间(单位是毫秒)，对于统计分析请求的处理速度帮助很大。

**开发人员可以充分利用访问日志，来分析问题、优化应用。**例如，分析访问日志中各个接口被访问的比例，不仅可以为需求和运营人员提供数据支持，还可以使自己的优化有的放矢；分析访问日志中各个请求的响应状态码，可以知道服务器请求的成功率，并找出有问题的请求；分析访问日志中各个请求的响应时间，可以找出慢请求，并根据需要进行响应时间的优化。