Tomcat(一) Tomcat是什么：Tomcat与Java技术 Tomcat与Web应用 以及Tomcat基本框架及相关配置

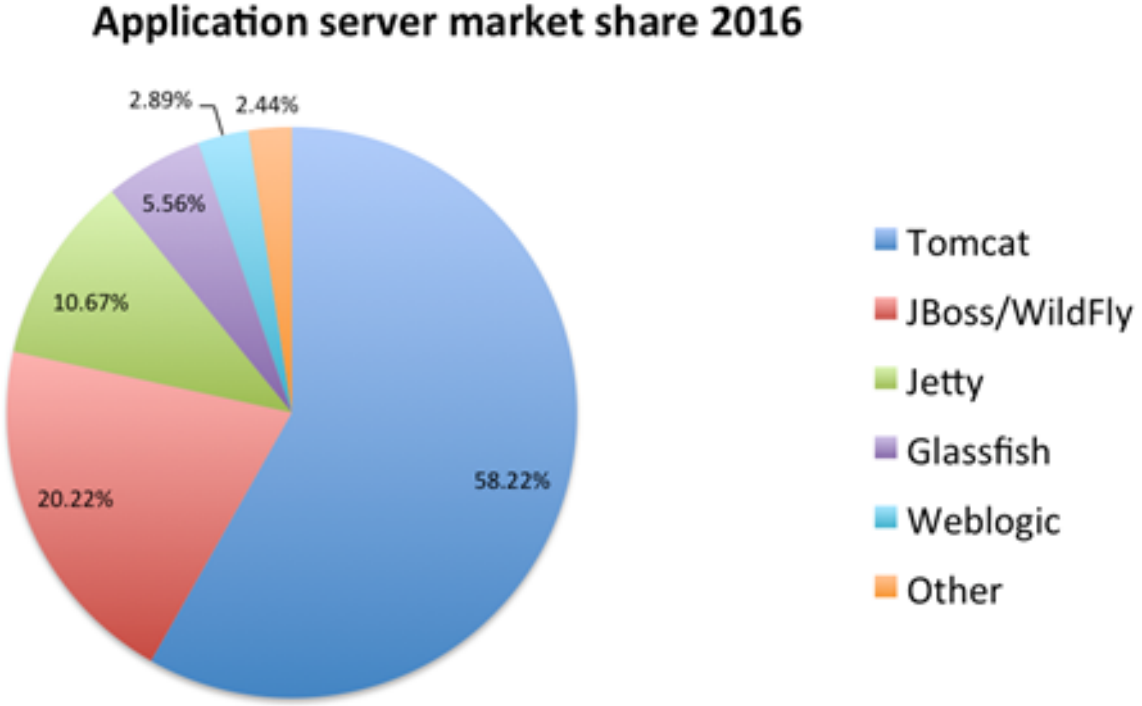
Tomcat(一) Tomcat是什么：

Tomcat与Java技术 Tomcat与Web应用 以及 Tomcat基本框架及相关配置

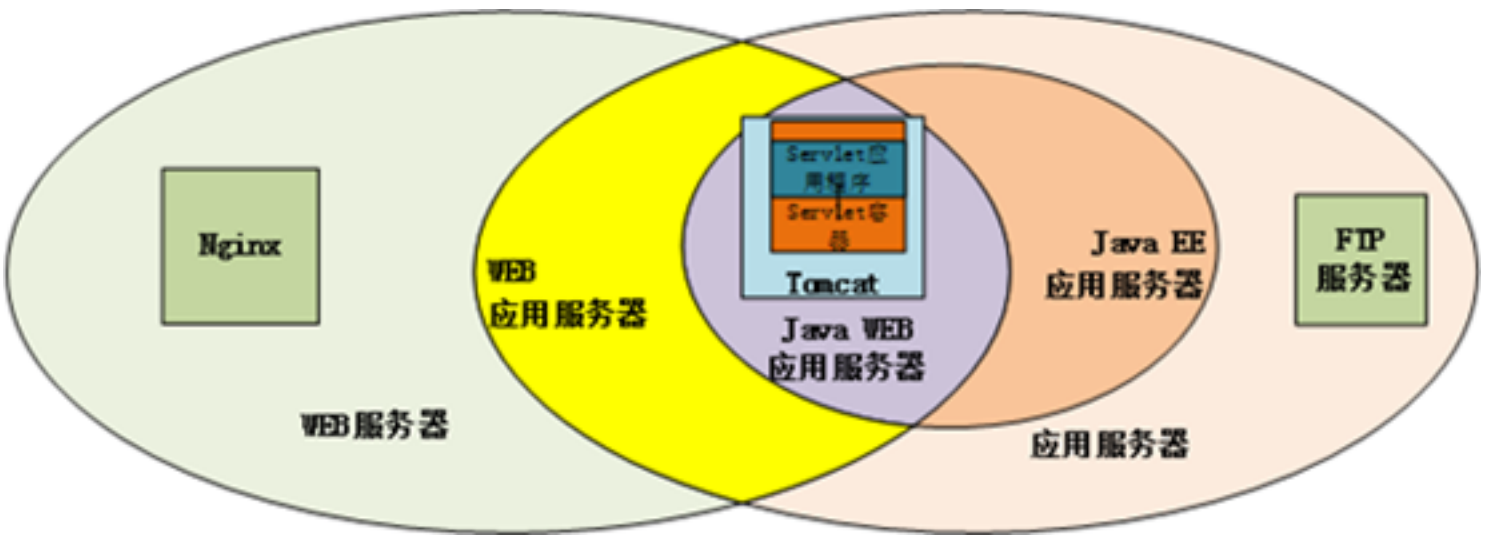
> Tomcat是一个被广泛使用的Java WEB应用服务器，我们有必要对它有足够的了解。下面将认识Tomcat：
>
> > 1、先来了解Tomcat与Java技术的关系、以及在WEB中的应用场景；
> >
> > 2、再来了解Tomcat的安装目录结构、Tomcat配置文件、Tomcat部署Web应用程序的目录结构；
> >
> > 3、最后了解Tomcat基本架构、以及Tomcat架构各组件的一些配置。

1、Tomcat是什么

> Apache Tomcat是由Apache Software Foundation（ASF）开发的一个开源Java WEB应用服务器。
>
> 类似功能的还有：Jetty、Resin、Websphere、weblogic、JBoss、Glassfish、GonAS等，它们的市场占有率如下，可以看到Tomcat是最受欢迎的Java WEB应用服务器：



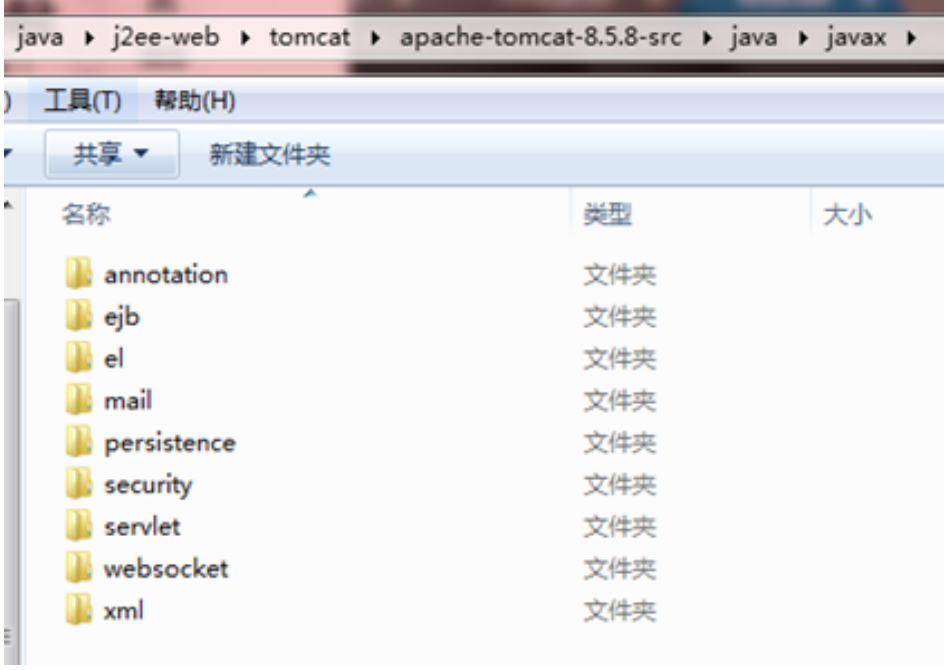
> Tomcat在技术实现上所处的位置如下：



> 下面我们来了解下Tomcat与这些技术之间的关系。

1-1、Tomcat与Java

> 1、Tomcat与Java SE
>
> Tomcat是用Java语言编写的，需要运行在Java虚拟机上，所以一般需要先安装JDK，以提供运行环境。
>
> 2、Tomcat与Java EE
>
> Tomcat实现了几个Java EE规范，包括Java Servlet、Java Server Pages（JSP），Java Expression Language和Java WebSocket等，这些是都下载Tomcat安装包默认提供的，可以在源码中看到相关Java EE 规范API源码引用，如下：



> 此外，官网还提供了另外一些Java EE规范的实现，如JMX Remote、Web services，要使用的话需要另外下载，放到Tomat安装目录/lib中，官网相关下载如下：

- Extras:
  - JMX Remote jar (pgp, md5, sha1)
  - Web services jar (pgp, md5, sha1)

> 可以说Tomcat是一个不完整的Java EE应用服务器。
>
> 更多关于Java SE、Java EE等java技术信息请参考：《Java平台体系：组成结构 运行机制 JRE/JDK/OpenJDK Java SE/EE/ME Java优点》

1-2、Tomcat与Servlet/编程开发

> Tomcat实现的几个Java EE规范最重的是Servlet，因为实现了Servlet规范，所以Tomcat也是一个Servlet容器，可以运行我们自己编写的Servlet应用程序处理动态请求。
>
> 平时用的Struts2、SpringMVC框架就是基于Servlet，所以我们可以在这些框架的基础上进行快速开发，然后部署到Tomcat中运行。
>
> 另外对于JSP规范实现：可以混合HTML与Java开发在一个文件中（.jsp），这种文件在第一次调用之后会被Tomcat的Jasper组件编译成一个servlet类，在后续的操作中则可以直接使用此类。这种开发方式不灵活，一般少用。

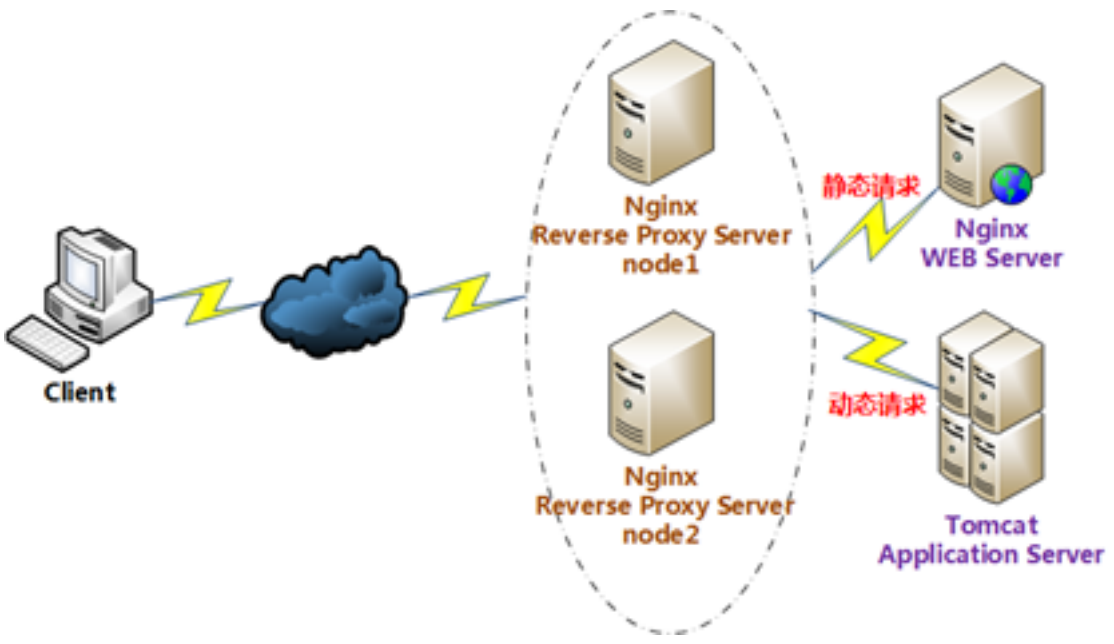
1-3、Tomcat与WEB/HTTP请求

Tomcat的（HTTP类型）Connector组件实现了HTTP请求的解析，Tomcat通过Connector组件接收HTTP请求并解析，然后把解析后的信息交给Servlet处理：

> 对于静态资源（html/js/jpg等）请求，Tomcat提供默认的Servlet来处理并响应；
>
> 对于动态请求，可以映射到自己编写的Servlet应用程序来处理。

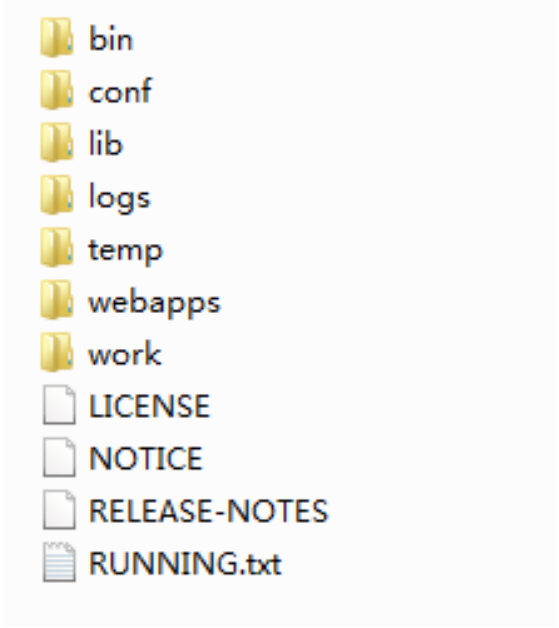
1-4、Tomcat与Nginx/Apache的应用架构

> Nginx、Apache都是目前主流的Web服务器，也可以作为反向代理服务器；它们在处理大量并发的请求连接、连接会话管理和静态内容请求等方面相比Tomcat更具优势。
>
> 所以一般在实际应用中，先是通过Nginx（或Apache）反向代理服务器接收请求，匹配分离动态/静态请求（动静分离）；
>
> 如果是静态请求，则转发到另外的Nginx WEB服务器上，返回静态内容；如果是动态请求，则转发到后面的Tomcat应用服务器，处理动态请求的业务逻辑。
>
> 简单的架构如下：



> 更多信息请参考：《各种容器与服务器的区别与联系：Servlet容器 WEB容器 Java EE容器 应用服务器 WEB服务器 Java EE服务器》

2、Tomcat安装目录



Tomcat安装后根目录如上图，由环境变量$ CATALINA_HOME表示，可以手动设置，可以由/bin/catalina.sh命令脚本自动设置该环境变量，如果使用多个Tomcat实例，需要为每个实例定义$ CATALINA_BASE环境变量。

这些目录说明如下：

bin：启动、关闭和其他脚本，*.sh文件（对于Unix系统）是*.bat文件的功能重复（对于Windows系统）。

conf：配置文件及相关数据文件存放目录，如server.xml,tomcat-users.xml,web.xml；

logs：默认的日志文件存放目录，如访问日志，可以通过server.xml文件配置到其他目录；

lib：Tomcat使用的库文件存放目录，如Servlet规范的API；

temp：临时文件的工作目录，如上传大文件时的缓存数据会存储到这里；

webapps：我们的应用程序部署目录，可以通过server.xml文件配置；

work：Tomcat工作目录，如存放JSP编译后的类文件。

更多相关信息请参考：http://tomcat.apache.org/tomcat-8.5-doc/introduction.html

3、Tomcat配置文件

Tomcat的配置文件默认存放在$CATALINA_HOME/conf目录中，主要有以下几个：

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

context.xml：为部署与此Tomcat实例上的web应用程序提供的默认配置文件，每个webapp都可以使用独有的context.xml,通常放置于webapp目录的META-INF子目录中，常用于定义会话管理器，Realm已经JDBC等。

web.xml：为部署与Tomcat实例上的所有web应用程序提供部署描述符，通常用于为webapp提供默认的servlet定义和基本的MUIME映射表。

tomcat-users.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：当基于-securty选项启动tomcat实例时会读取此配置文件；此文件是JAVA的安全策略配置文件，用于配置访问codebase（代码库）或某些Java类的权限。

catalina.properties：java属性定义文件，设定类加载器路径，安全包列表和一些调整性能的参数信息。

logging.properties：定义日志相关的配置信息，如日志级别、文件路径等。

4、WEB应用部署目录结构

我们的应用程序一般会打包成归档格式（.war），然后放到Tomcat的应用程序部署目录。而webapp有特定的组织格式，是一种层次型目录结构，通常包含了servlet代码文件、HTML/jsp页面文件、类文件、部署描述符文件等等，相关说明如下：

/：web应用程序的根目录，可以存放HTML/JSP页面以及其他客户端浏览器必须可见的其他文件（如js/css/图像文件）。在较大的应用程序中，还可以选择将这些文件划分为子目录层次结构。

/WEB-INF：此webapp的**所有私有资源目录**，用户浏览器不可能访问到的，通常web.xml和context.xml均放置于此目录。

/WEB-INF/web.xml：此webapp的私有的部署描述符，描述组成应用程序的servlet和其他组件（如filter），以及相关初始化参数和容器管理的安全性约束。

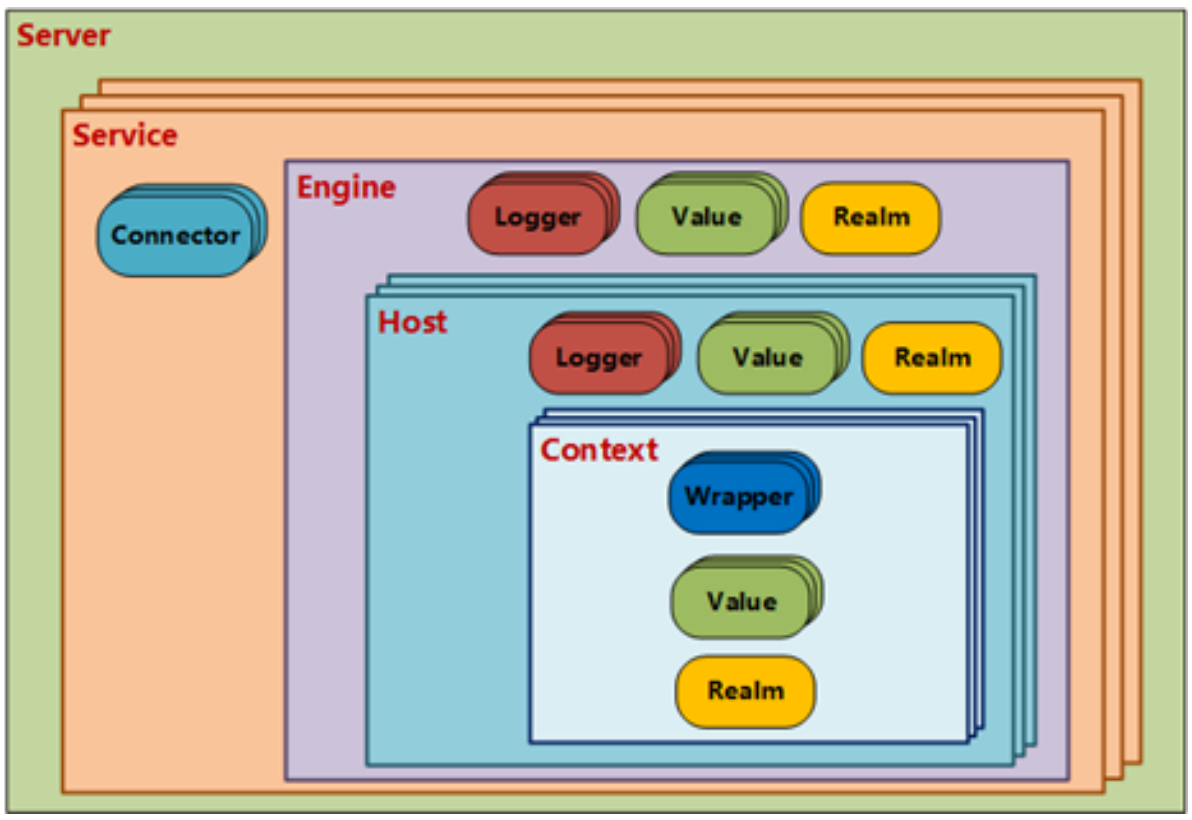
/WEB-INF/classes：此webapp自有的Java程序类文件（.class）及相关资源存放目录。

/WEB-INF/lib：此目录存放webapp自有的JAR文件，其中包含应用程序所需的Java类文件（及相关资源），例如第三方类库或JDBC驱动程序。

更多信息请参考：

> Tomcat《Application Developer's Guide》Deployment
>
> 《Servlet3.1规范（最终版）》第10章

5、Tomcat基本框架及相关配置



如上图，**Tomcat可以按功能划分许多不同的组件，这些组件都可以通过/conf/server.xml文件中可定义和配置**，包括Server, Service, Connector, Engine, Cluster, Host, Alias, Context, Realm, Valve, Manager, Listener, Resources, ResourceEnvRef, WatchedResource, Store, Transaction, Channel, Membership, Transport, Member, ClusterListener等，一般可分为以下四类：

> 1、**顶级组件**：位于配置层次的顶级，并且彼此间有着严格的对应关系，有Server组件、Service组件；
>
> 2、**连接器**：连接客户端（可以是浏览器或Web服务器）请求至Servlet容器，只有Connector组件，
>
> 3、**容器**：表示其功能是处理传入请求的组件，并创建相应的响应。如Engine处理对一个Service的所有请求，Host处理对特定虚拟主机的所有请求，并且Context处理对特定web应用的所有请求；
>
> 4、**被嵌套的组件**：位于一个容器当中，但不能包含其它组件；一些组件可以嵌套在任何Container中，而另一些只能嵌套在Context中。
>
> 如**server.xml默认配置**（删除注释内容）如下：

```xml
<?xml version="1.0" encoding="UTF-8"?>

<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.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 &amp;quot;%r&amp;quot; %s %b" />
      </Host>
    </Engine>
  </Service>
</Server>
```

> 下面来了解下上图中主要的组件。

5-1、Server组件

> Server（服务器）表示Tomcat的一个实例，因此，它必须是/conf /server.xml配置文件中的单个最外层元素，它的属性表示servlet容器的整体特性。通常一个JVM只能包含一个Tomcat实例。
>
> 默认配置表示监听在8005端口以接收shutdown命令，默认仅允许通过本机访问。
>
> 更多Server配置信息请参考：《Apache Tomcat 8 Configuration Reference》 The Server Component

5-2、Service组件

> Service（服务）主要用于关联一个Engine和与此Engine相关的Connector，每个Connector通过一个特定的端口和协议接收请求，并将其转发至关联的Engine进行处理。

因此，Service可以包含一个Engine、以有一个或多个Connector；而一个Server可以包含多个Service组件，但通常情下只为一个Server指派一个Service。通常需要给Service命名，可以方便管理员在日志文件中识别不同Service产生的日志。

如默认配置中server只包含一个名为"Catalina"的service，而service里包含两个Connector，其中一个监听8080端口接收HTTP请求，另一个监听8009端口接收AJP协议的请求。

更多Service配置信息请参考：《Apache Tomcat 8 Configuration Reference》 The Service Component

5-3、Connector组件

如上面所述，Connector（连接器）通过一个特定的端口接收特定协议的客户端请求，并将其转发至关联的Engine进行处理。一个Engine可以配置多个连接器，但这些连接器必须使用不同的端口。

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，连接器类型可以分为两种：

（1）、HTTP连接器

HTTP连接器元素表示支持HTTP／1.1协议的连接器组件，**它能使Tomcat能够作为独立的Web服务器**。此组件的特定实例侦听服务器上特定TCP端口号上的连接，每个转发到相关联的Engine以执行请求处理并创建响应。

HTTP连接器又有三种不同的实现：Java Nio Connector、Java Nio2 Connector、APR/native Connector，它们的对比如下：

	Java Nio Connector NIO	Java Nio2 Connector NIO2	APR/native Connector APR
Classname	`Http11NioProtocol`	`Http11Nio2Protocol`	`Http11AprProtocol`
Tomcat Version	6.x onwards	8.x onwards	5.5.x onwards
Support Polling	YES	YES	YES
Polling Size	`maxConnections`	`maxConnections`	`maxConnections`
Read Request Headers	Non Blocking	Non Blocking	Non Blocking
Read Request Body	Blocking	Blocking	Blocking
Write Response Headers and Body	Blocking	Blocking	Blocking
Wait for next Request	Non Blocking	Non Blocking	Non Blocking
SSL Support	Java SSL or OpenSSL	Java SSL or OpenSSL	OpenSSL
SSL Handshake	Non blocking	Non blocking	Blocking
Max Connections	`maxConnections`	`maxConnections`	`maxConnections`

默认配置文件，定义了一个连接器为protocol="HTTP/1.1" 表示的是使用自动切换机制来选择基于Java NIOConnector或基于APR /Native Connector（需要设置），也可以手动指定，

更多HTTP连接器配置信息请参考：《Apache Tomcat 8 Configuration Reference》 The HTTP Connector

（2）、AJP 1.3连接器

AJP连接器元素表示通过AJP(Apache JServ Protocol)协议与Web连接器通信的连接器组件。

AJP协议是基于二进制的格式在Web服务器和Tomcat之间传输数据，这比HTTPP获得更好的效率，但比较复杂不通用。

通常用于将Tomcat集成到现有Apache服务器中，并且希望Apache处理Web应用程序中包含的静态内容或SSL连接处理的情况，即Apache服务器作为代理服务器。Apache与Tomcat结合可以由mod_jk或mod_proxy模块来实现，但它们的使用范围不同：mod_jk支持apache/1.3,apache/2.0，mod_proxy支持apache/2.2+。

默认配置文件中定义了一个监听8009端口的AJP连接器，其实官方文档说明这种连接器不久后不再支持，一般用得不多，就不再多介绍了。

更多AJP 1.3连接器配置信息请参考：《Apache Tomcat 8 Configuration Reference》The AJP Connector

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（更多请参考前面给出的文档）：

> 1）、address：指定连接器监听的地址，默认为所有地址，即0.0.0.0；
>
> 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；
>
> 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；

5-4、Engine组件

> Engine（引擎）表示与特定Service相关联的整个请求处理机制，即Servlet容器引擎。它接收和处理来自一个或多个连接器的所有请求，并检查每一个请求的HTTP首部信息以辨别此请求应该发往哪个Host或Context，并将完成的响应返回到连接器，以便最终传输回客户端。

一个Engine元素必须嵌套在Service元素内，它可以包含多个host组件，还可以包含Realm、Listener和Valve等子容器。

常用的属性定义：

> 1、**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。
>
> 2、**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。

如默认配置中定义了一个名为"Catalina"的Engine，而Engine里包含一个Hots，并被配置为默认的虚拟主机。

> 更多Engine配置信息请参考：《Apache Tomcat 8 Configuration Reference》 The Engine Component

5-5、Host组件

> Host（虚拟主机）类似于Apache中的虚拟主机，但在Tomcat中只支持基于FQDN的"虚拟主机"。**Host位于Engine容器中用于接收请求并进行相应处理，它是服务器**（例如"www.mycompany.com"）的网络名称与运行Tomcat的特定服务器的关联。
>
> 客户端通常使用主机名来标识他们希望连接的服务器，但要使客户端能够使用其网络名称连接到Tomcat服务器，此名称必须在管理所属的Internet域的域名服务（DNS）服务器中注册。此主机名也包含在HTTP请求标头中，Tomcat从HTTP头中提取主机名，并查找具有匹配名称的主机；如果未找到匹配项，请求将路由到默认主机。
>
> 一个Engine至少要包含一个Host组件，而在Host元素内可以嵌入与此

虚拟主机关联的Web应用程序的Context等元素。

常用属性说明：

1)、**name**：此Host的FQDN虚拟主机名称；

2)、**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$CATALINA_HOME的相对路径；

3)、**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；

4)、**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。

如默认配置中定义了一个主机名为"localhost"的Host，而webapps目录为$ CATALINA_BASE相对的"webapps"，即前面说到的默认目录，也可用绝对路径来配置其他目录。

更多Host配置信息请参考：《Apache Tomcat 8 Configuration Reference》 The Host Component

5-6、Context组件

Context（上下文）表示在特定虚拟主机中运行的Web应用程序，一个Context对应一个Web应用程序，而里面的Wrapper可以理解为一个个Servlet程序。

Context需要根据其定义的上下文路径（path）匹配请求URI的最长前缀（除主机名外）来选择。一旦选择，可以由docBase来找到该上下文将对应的web应用程序部署目录，由目录中web.xml定义的servlet映射选择一个合适的servlet来处理传入的请求。

一个Host可以有多个Context，通常不建议定义在server.xml文件中，而是每一个context定义使用一个单独的XML文件进行，其文件的目

录　　为$CATALINA_HOME/conf/<engine name>/<host name>。

可以看到server.xml中默认没有定义Context，但存在/conf/context.xml，在前面说Tomcat配置文件时曾介绍过，context.xml为部署与此Tomcat实例上所有的web应用程序提供的默认配置文件，删除注释后其内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>

<Context>
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>
</Context>
```

通过它可以找到默认的和各web应用程序提供部署描述符文件web.xml，/conf/web.xml定义了Tomcat提供的默认Servlet处理程序，主要用来处理静态资源请求；而各webapp的web.xml可以定义其他的动态请求url映射到不同Servlet程序处理。

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，docBase的路径名不能与相应的Host中appBase中定义的路径名有包含关系，比如，如果appBase为deploy，而docBase绝不能为deploy-bbs类的名字；

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，此属性不需要定义；

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；

更多Context配置信息请参考：《Apache Tomcat 8 Configuration Reference》 The Context Component

5-7、Realm组件

Realm（领域）表示分配给这些用户的用户名，密码和角色（类似于

Unix组）的"数据库"。一个Realm（领域）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。

Catalina容器（Engine，Host或Context）可以包含不超过一个Realm元素（但自身可以嵌套）。此外，与引擎或主机关联的领域由低级容器自动继承，除非下级容器显式定义了自己的领域。如果没有为引擎配置领域，将自动为引擎配置空领域的实例。

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，Tomcat中实现了多种不同的Realm，如下：

> **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MemoryRealm，因此能够跟标准的MemoryRealm兼容；它通过JNDI实现；
>
> LockOutRealm：提供锁定功能，以便在给定时间段内出现过多的失败认证尝试时提供用户锁定机制；
>
> 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。

可以看到默认配置文件中定义了一个LockOutRealm并嵌套一个UserDatabaseRealm的Realm来通过tomcat-user.xml文件实现用户认证。

更多Realm配置信息请参考：《Apache Tomcat 8 Configuration Reference》 The Realm Component

5-8、Valve组件

Valve（阀门）类似于过滤器，用来拦截请求并在将其转至目标之前进行某种处理操作；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。

Valve常被用来记录客户端请求、客户端IP地址和服务器等信息，这种处理技术通常被称作请求转储(request dumping)。请求转储valve记录请求客户端请求数据包中的HTTP首部信息和cookie信息文件中，响应转储valve则记录响应数据包首部信息和cookie信息至文件中。

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。不同类型的Value具有不同的处理能力，Tomcat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
>
> 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；
>
> 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。

通过属性className定义相关的java实现的类名来选择Value。如默认配置文件中定义了一个AccessLogValve的Value来记录访问日志到文件中。

更多Valve配置信息请参考：《Apache Tomcat 8 Configuration Reference》 The Valve Component

5-9、其他组件

1、Logger

日志记录器(Logger)：用于记录组件内部的状态信息，可被用于除Context之外的任何容器中。日志记录的功能可被继承，因此，一个引擎级别的Logger将会记录引擎内部所有组件相关的信息，除非某内部组件定义了自己的Logger组件（前面介绍的AccessLogValve使用自包含的逻辑来写它的日志文件，以获得更好的效率）。

2、Listener

Listener用于创建和配置LifecycleListener对象，而LifecycleListener通常被开发人员用来创建和删除容器。

3、Loader

Java的动态装载功能是其语言功能强大表现之一，Servlet容器使用此功能在运行时动态装载servlet和它们所依赖的类。Loader可以用于Context中控制java类的加载，即WebApp类加载器。

4、Resources

经常用于实现在Context中指定需要装载的但不在Tomcat本地磁盘上的应用资源，如Java类，HTML页面，JSP文件等。

5、GlobalNamingResources

应用于整个服务器的JNDI映射，此可以避免每个Web应用程序都需要在各自的web.xml创建，这在web应用程序以WAR的形式存在时尤为有用。它通常可以包含三个子元素：Environment、Resource和ResourceEnvRef。

6、WatchedResource

WatchedResource可以用于Context中监视指定的webapp程序文件的改变，并且能够在监视到文件内容发生改变时重新装载此文件。

7、Manager

Manger对象用于实现HTTP会话管理的功能，Tomcat中有5种Manger的实现：

1)、StandardManager

Tomcat6的默认会话管理器，用于非集群环境中对单个处于运行状态的Tomcat实例会话进行管理。当Tomcat关闭时，这些会话相关的数据会被写入磁盘上的一个名叫SESSION.ser的文件，并在Tomcat下次启动时读取此文件。

2)、PersistentManager

当一个会话长时间处于空闲状态时会被写入到swap会话对象，这对于内存资源比较吃紧的应用环境来说比较有用。

3)、DeltaManager

属于ClusterManager，用于Tomcat集群的会话管理器，它通过将改变了会话数据同步给集群中的其它节点实现会话复制。这种实现会将所有会话的改变同步给集群中的每一个节点，也是在集群环境中用得最多的一种实现方式。

但集群节点较多时，会消耗大量的网络资源，一般适用于3、4个节点的集群。

4)、BackupManager

属于ClusterManager，用于Tomcat集群的会话管理器，与DeltaManager不同的是，某节点会话的改变只会同步给集群中的另一个而非所有节点。

5)、SimpleTcpReplicationManager

> Tomcat4时用到的版本，过于老旧了。

8、Stores

PersistentManager必须包含一个Store元素以指定将会话数据存储至何处。这通常有两种实现方式：FileStore和JDBCStore。

9、Cluster

专用于配置Tomcat集群的元素，可用于Engine和Host容器中。在用于Engine容器中时，Engine中的所有Host均支持集群功能。在Cluster元素中，需要直接定义一个Manager元素，这个Manager元素有一个其值为org.apache.catalina.ha.session.DeltaManager或org.apache.catalina.ha.session.BackupManager的className属性。同时，Cluster中还需要分别定义一个Channel和ClusterListener元素。

10、Channel

用于Cluster中给集群中同一组中的节点定义通信"信道"。Channel中需要至少定义Membership、Receiver和Sender三个元素，此外还有一个可选元素Interceptor。

11、Membership

用于Channel中配置同一通信信道上节点集群组中的成员情况，即监控加入当前集群组中的节点并在各节点间传递心跳信息，而且可以在接收不到某成员的心跳信息时将其从集群节点中移除。Tomcat6中Membership的实现是org.apache.catalina.tribes.membership.McastService。

12、Sender

用于Channel中配置"复制信息"的发送器，实现发送需要同步给其它节点的数据至集群中的其它节点。发送器不需要属性的定义，但可以在其内部定义一个Transport元素。

13、Transport

用于Sender内部，配置数据如何发送至集群中的其它节点。Tomcat有

两种Transport的实现：

> 1）、PooledMultiSender
>
> 基于Java阻塞式IO，可以将一次将多个信息并发发送至其它节点，但一次只能传送给一个节点。
>
> 2)、PooledParallelSener
>
> 基于Java非阻塞式IO，即NIO，可以一次发送多个信息至一个或多个节点。

14、Receiver

用于Channel定义某节点如何从其它节点的Sender接收复制数据，Tomcat中实现的接收方式有两种BioReceiver和NioReceiver。

更多组件及配置信息请参考：

> 《Apache Tomcat 8 Configuration Reference》
>
> 《Apache Tomcat User Guide》

> 到这里，我们对Tomcat有了一个基本的认识：了解到Tomcat技术与在Web中的应用、以及Tomcat基本框架及相关配置，后面将更深入了解Tomcat：了解JavaEE Servlet技术、Tomcat中的一些实现细节，而后再来配置Tomcat+nginx+keepalived的动静分离、会话保持的高可用集群……

【参考资料】

1、Apache Tomcat User Guide：http://tomcat.apache.org/tomcat-8.5-doc/index.html

2、Apache Tomcat 8 Configuration Reference：ttp://tomcat.apache.org/tomcat-8.5-doc/config/index.html

3、Servlet3.1规范（最终版）

4、Java平台体系：组成结构 运行机制 JRE/JDK/OpenJDK Java SE/EE/ME

Java优点

5、各种容器与服务器的区别与联系： Servlet容器 WEB容器 Java EE容器 应用服务器 WEB服务器 Java EE服务器

6、《深入分析Java Web技术内幕》

7、《深入剖析tomcat》