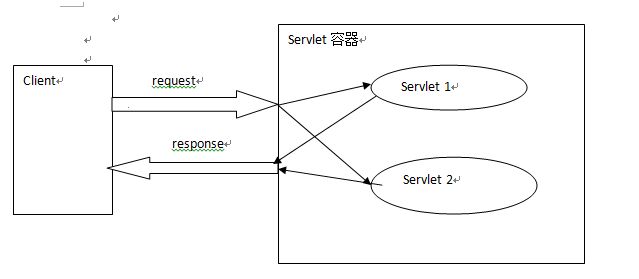
参考：<http://blog.csdn.net/skp127/article/details/52026150>

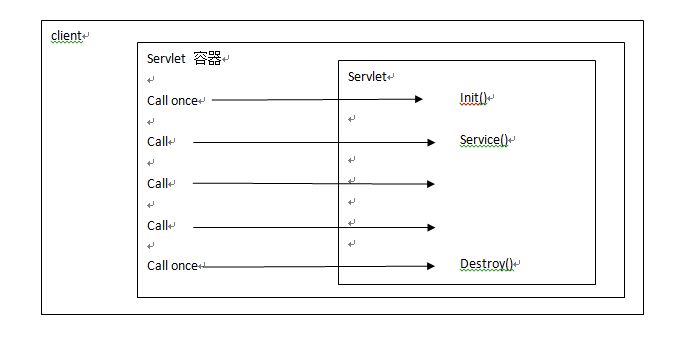
大致的[**架构**](http://lib.csdn.net/base/architecture)是 jsp+tomcat+[**MySQL**](http://lib.csdn.net/base/mysql)，记录tomcat学习一点笔记。

Tomcat是Servlet运行环境（容器）（容器有server启动），每个servlet执行init(),service(),destory()，这个是由java的servlet标准规定的

以下是servlet的作用



Servlet的调用



Tomcat的工作模式3种：独立Servlet，进程内servlet，进程外servlet。

Tomcat是一个基于组件的服务器，他的构建组件都是可以配置的，其中最外层的组件是Catalina Servlet容器（容器的名字可以配置，代表整个servlet容器），其他组件一定要按照一定的格式要求配置在这个顶层的容器中。

　　Tomcat各个组件是在$CATLINA\_HOME/conf/server.xml文件中配置的。

复制代码

<Server> 顶层类元素，可包含多个service  
<Service> 顶层类元素，可包含一个Engine和多个Connector  
<Connector/> 链接类容器，代表通信接口  
<Engine> 容器元素，为Service处理客户请求,含多个Host  
<Host> 容器元素，为Host处理客户请求，含多个Context  
<Context/> 容器元素，为Web应用处理客户请求  
</Host>  
</Engine>  
</Service>  
</Server>

复制代码

一个[**Java**](http://lib.csdn.net/base/java)web应用在Tomcat中与一个Context对应，是一一对应关系。

[**java**](http://lib.csdn.net/base/java)Web应该可以包含如下内容：

* Servlet
* JSP pages
* Java Classes
* static resources(HTML documents, pictures, etc.)
* Description Documents of Web  Applications.

**1 - Tomcat Server的组成部分**

**1.1 – Server --容器**

A Server element represents the entire Catalina servlet [**Container**](http://lib.csdn.net/base/docker). (Singleton，单进程，也是一个tomcat服务器只有一个Catalina)

**1.2 – Service--容器内的服务提供方**

A Service element represents the combination of one or more Connector components that share a single Engine  
Service是这样一个集合：它由一个或者多个Connector组成，以及一个Engine，负责处理所有Connector所获得的客户请求

**1.3 – Connector--连接器**

一个Connector将在某个指定端口上侦听客户请求，并将获得的请求交给Engine来处理，从Engine处获得回应并返回客户  
TOMCAT有两个典型的Connector，一个直接侦听来自browser的http请求，一个侦听来自其它WebServer的请求  
Coyote Http/1.1 Connector 在端口8080处侦听来自客户browser的http请求  
Coyote JK2 Connector 在端口8009处侦听来自其它WebServer(Apache)的servlet/jsp代理请求

**1.4 - Engine**

The Engine element represents the entire request processing machinery associated with a particular Service  
It receives and processes all requests from one or more Connectors  
and returns the completed response to the Connector for ultimate transmission back to the client

一个engine接收和处理从一个或者多个连接器的所有的请求，并且将请求的结果返回给连接器并最终返回给客户端。

Engine下可以配置多个虚拟主机Virtual Host，每个虚拟主机都有一个域名  
当Engine获得一个请求时，它把该请求匹配到某个Host上，然后把该请求交给该Host来处理  
Engine有一个默认虚拟主机，当请求无法匹配到任何一个Host上的时候，将交给该默认Host来处理

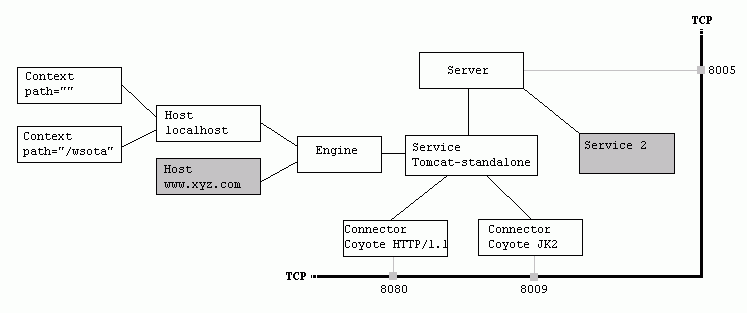
**1.5 - Host**

代表一个Virtual Host，虚拟主机，每个虚拟主机和某个网络域名Domain Name相匹配  
每个虚拟主机下都可以部署(deploy)一个或者多个Web App，每个Web App对应于一个Context，有一个Context path  
当Host获得一个请求时，将把该请求匹配到某个Context上，然后把该请求交给该Context来处理  
匹配的方法是“最长匹配”，所以一个path=""的Context将成为该Host的默认Context  
所有无法和其它Context的路径名匹配的请求都将最终和该默认Context匹配

**1.6 - Context**

一个Context对应于一个Web Application，一个Web Application由一个或者多个Servlet组成  
Context在创建的时候将根据配置文件$CATALINA\_HOME/conf/web.xml和$WEBAPP\_HOME/WEB-INF/web.xml载入Servlet类  
当Context获得请求时，将在自己的映射表(mapping table)中寻找相匹配的Servlet类  
如果找到，则执行该类，获得请求的回应，并返回

**2 - Tomcat Server的结构图**



**3 - 配置文件$CATALINA\_HOME/conf/server.xml的说明**

该文件描述了如何启动Tomcat Server，该配置文件配置的是整个servlet容器，即tomcat

**[html]** [view plain](http://blog.csdn.net/jiandanfeng2/article/details/7342667) [copy](http://blog.csdn.net/jiandanfeng2/article/details/7342667)

1. <!----------------------------------------------------------------->
2. <!-- 启动Server
3. 在端口8005处等待关闭命令
4. 如果接受到"SHUTDOWN"字符串则关闭服务器
5. telnet localhost 8005
6. SHUTDOWN
7. --**>**
8. **<Server** port="8005" shutdown="SHUTDOWN" debug="0"**>**
10. <!-- Listener ???
11. 目前没有看到这里
12. --**>**
14. **<Listener** className="org.apache.catalina.mbeans.ServerLifecycleListener" debug="0"**/>**
15. **<Listener** className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" debug="0"**/>**
17. <!-- Global JNDI resources ???
18. 目前没有看到这里，先略去
19. --**>**
21. **<GlobalNamingResources>**
22. ... ... ... ...
23. **</GlobalNamingResources>**
25. <!-- Tomcat的Standalone Service
26. Service是一组Connector的集合
27. 它们共用一个Engine来处理所有Connector收到的请求
28. --**>**
30. **<Service** name="Tomcat-Standalone"**>**
32. <!-- Coyote HTTP/1.1 Connector
33. className : 该Connector的实现类是org.apache.coyote.tomcat4.CoyoteConnector
34. port : 在端口号8080处侦听来自客户browser的HTTP1.1请求
35. minProcessors : 该Connector先创建5个线程等待客户请求，每个请求由一个线程负责
36. maxProcessors : 当现有的线程不够服务客户请求时，若线程总数不足75个，则创建新线程来处理请求
37. acceptCount : 当现有线程已经达到最大数75时，为客户请求排队
38. 当队列中请求数超过100时，后来的请求返回Connection refused错误
39. redirectport : 当客户请求是https时，把该请求转发到端口8443去
40. 其它属性略
41. --**>**
43. **<Connector** className="org.apache.coyote.tomcat4.CoyoteConnector"
44. port="8080"
45. minProcessors="5" maxProcessors="75" acceptCount="100"
46. enableLookups="true"
47. redirectPort="8443"
48. debug="0"
49. connectionTimeout="20000"
50. useURIValidationHack="false"
51. disableUploadTimeout="true" **/>**

54. <!-- Engine用来处理Connector收到的Http请求
55. 它将匹配请求和自己的虚拟主机，并把请求转交给对应的Host来处理
56. 默认虚拟主机是localhost
57. --**>**
59. **<Engine** name="Standalone" defaultHost="localhost" debug="0"**>**

62. <!-- 日志类，目前没有看到，略去先 -->
64. **<Logger** className="org.apache.catalina.logger.FileLogger" ...**/>**
66. <!-- Realm，目前没有看到，略去先 -->
68. **<Realm** className="org.apache.catalina.realm.UserDatabaseRealm" ...**/>**

71. <!-- 虚拟主机localhost
72. appBase : 该虚拟主机的根目录是webapps/
73. 它将匹配请求和自己的Context的路径，并把请求转交给对应的Context来处理
74. --**>**
76. **<Host** name="localhost" debug="0" appBase="webapps" unpackWARs="true" autoDeploy="true"**>**
78. <!-- 日志类，目前没有看到，略去先 -->
80. **<Logger** className="org.apache.catalina.logger.FileLogger" ...**/>**

83. <!-- Context，对应于一个Web App
84. path : 该Context的路径名是""，故该Context是该Host的默认Context
85. docBase : 该Context的根目录是webapps/mycontext/  ,这里的路径是
86. 相对于host的appBase的
87. --**>**
89. **<Context** path="" docBase="mycontext" debug="0"**/>**

92. <!-- 另外一个Context，路径名是/wsota -->
94. **<Context** path="/wsota" docBase="wsotaProject" debug="0"**/>**

97. **</Host>**
99. **</Engine>**
101. **</Service>**
103. **</Server>**

106. <!----------------------------------------------------------------------------------------------->

**4 - Context的部署配置文件web.xml的说明**

一个Context对应于一个Web App，这两个是一一对应的，即一个context对应多个servlet，每个Web App是由一个或者多个servlet组成的  
当一个Web App被初始化的时候，它将用自己的ClassLoader对象载入“部署配置文件web.xml”中定义的每个servlet类  
它首先载入在$CATALINA\_HOME/conf/web.xml中部署的servlet类  
然后载入在自己的Web App根目录下的WEB-INF/web.xml中部署的servlet类（me:自己在web.xml中定义的能够覆盖外部默认的web.xml）  
web.xml文件主要包含两部分：servlet类定义和servlet映射定义  
每个被载入的servlet类都有一个名字，且被填入该Context的映射表(mapping table)中（me：这个context类似路由映射表），和某种URL PATTERN对应  
当该Context获得请求时，将查询mapping table，找到被请求的servlet，并执行以获得请求回应

分析一下所有的Context共享的web.xml文件，在其中定义的servlet被所有的Web App载入

**[html]** [view plain](http://blog.csdn.net/jiandanfeng2/article/details/7342667) [copy](http://blog.csdn.net/jiandanfeng2/article/details/7342667)

1. <!----------------------------------------------------------------------------------------------->

4. **<web-app>**
6. <!-- 概述：
7. 该文件是所有的WEB APP共用的部署配置文件，
8. 每当一个WEB APP被DEPLOY，该文件都将先被处理，然后才是WEB APP自己的/WEB-INF/web.xml
9. --**>**
11. <!--  +-------------------------+  -->
12. <!--  |    servlet类定义部分    |  -->
13. <!--  +-------------------------+  -->

16. <!-- DefaultServlet
17. 当用户的HTTP请求无法匹配任何一个servlet的时候，该servlet被执行
18. URL PATTERN MAPPING : /
19. --**>**
21. **<servlet>**
22. **<servlet-name>**default**</servlet-name>**
23. **<servlet-class>**
24. org.apache.catalina.servlets.DefaultServlet
25. **</servlet-class>**
26. **<init-param>**
27. **<param-name>**debug**</param-name>**
28. **<param-value>**0**</param-value>**
29. **</init-param>**
30. **<init-param>**
31. **<param-name>**listings**</param-name>**
32. **<param-value>**true**</param-value>**
33. **</init-param>**
34. **<load-on-startup>**1**</load-on-startup>**
35. **</servlet>**

38. <!-- InvokerServlet
39. 处理一个WEB APP中的匿名servlet
40. 当一个servlet被编写并编译放入/WEB-INF/classes/中，却没有在/WEB-INF/web.xml中定义的时候
41. 该servlet被调用，把匿名servlet映射成/servlet/ClassName的形式
42. URL PATTERN MAPPING : /servlet/\*
43. --**>**
45. **<servlet>**
46. **<servlet-name>**invoker**</servlet-name>**
47. **<servlet-class>**
48. org.apache.catalina.servlets.InvokerServlet
49. **</servlet-class>**
50. **<init-param>**
51. **<param-name>**debug**</param-name>**
52. **<param-value>**0**</param-value>**
53. **</init-param>**
54. **<load-on-startup>**2**</load-on-startup>**
55. **</servlet>**

58. <!-- JspServlet
59. 当请求的是一个JSP页面的时候（\*.jsp）该servlet被调用
60. 它是一个JSP编译器，将请求的JSP页面编译成为servlet再执行
61. URL PATTERN MAPPING : \*.jsp
62. --**>**
64. **<servlet>**
65. **<servlet-name>**jsp**</servlet-name>**
66. **<servlet-class>**org.apache.jasper.servlet.JspServlet**</servlet-class>**
67. **<init-param>**
68. **<param-name>**logVerbosityLevel**</param-name>**
69. **<param-value>**WARNING**</param-value>**
70. **</init-param>**
71. **<load-on-startup>**3**</load-on-startup>**
72. **</servlet>**


76. <!--  +---------------------------+  -->
77. <!--  |    servlet映射定义部分    |  -->
78. <!--  +---------------------------+  -->

81. **<servlet-mapping>**
82. **<servlet-name>**default**</servlet-name>**
83. **<url-pattern>**/**</url-pattern>**
84. **</servlet-mapping>**
86. **<servlet-mapping>**
87. **<servlet-name>**invoker**</servlet-name>**
88. **<url-pattern>**/servlet/\***</url-pattern>**
89. **</servlet-mapping>**
91. **<servlet-mapping>**
92. **<servlet-name>**jsp**</servlet-name>**
93. **<url-pattern>**\*.jsp**</url-pattern>**
94. **</servlet-mapping>**

97. <!--  +------------------------+  -->
98. <!--  |    其它部分，略去先    |  -->
99. <!--  +------------------------+  -->
101. ... ... ... ...
103. **</web-app>**

106. <!----------------------------------------------------------------------------------------------->

**5 - Tomcat Server处理一个http请求的过程**

假设来自客户的请求为：  
http://localhost:8080/wsota/wsota\_index.jsp

1) 请求被发送到本机端口8080，被在那里侦听的Coyote HTTP/1.1 Connector（http连接器）获得  
2) Connector把该请求交给它所在的Service的Engine来处理，并等待来自Engine的回应  
3) Engine获得请求localhost/wsota/wsota\_index.jsp，匹配它所拥有的所有虚拟主机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）  
5) localhost Host获得请求/wsota/wsota\_index.jsp，匹配它所拥有的所有Context  
6) Host匹配到路径为/wsota的Context（如果匹配不到就把该请求交给路径名为""的Context去处理）  
7) path="/wsota"的Context获得请求/wsota\_index.jsp，在它的mapping table中寻找对应的servlet  
8) Context匹配到URL PATTERN为\*.jsp的servlet，对应于JspServlet类  
9) 构造HttpServletRequest对象和HttpServletResponse对象，作为参数调用JspServlet的doGet或doPost方法  
10)Context把执行完了之后的HttpServletResponse对象返回给Host  
11)Host把HttpServletResponse对象返回给Engine  
12)Engine把HttpServletResponse对象返回给Connector  
13)Connector把HttpServletResponse对象返回给客户browser

client

server

service

connector

engine

Host

context

Servlet