




























tomcat结构介绍

1.tomcat目录结构:

1.1 bin目录:

名称	修改日期	类型	大小
 bootstrap.jar	2020/6/30 22:50	Executable Jar File	36 KB
 catalina.bat	2020/6/30 22:50	Windows 批处理...	17 KB
 catalina.sh	2020/6/30 22:50	Shell Script	25 KB
 catalina-tasks.xml	2020/6/30 22:50	XML 文档	2 KB
 ciphers.bat	2020/6/30 22:50	Windows 批处理...	3 KB
 ciphers.sh	2020/6/30 22:50	Shell Script	2 KB
 commons-daemon.jar	2020/6/30 22:50	Executable Jar File	25 KB
 configtest.bat	2020/6/30 22:50	Windows 批处理...	2 KB
 configtest.sh	2020/6/30 22:50	Shell Script	2 KB
 daemon.sh	2020/6/30 22:50	Shell Script	9 KB
 digest.bat	2020/6/30 22:50	Windows 批处理...	3 KB
 digest.sh	2020/6/30 22:50	Shell Script	2 KB
 service.bat	2020/6/30 22:50	Windows 批处理...	9 KB
 setclasspath.bat	2020/6/30 22:50	Windows 批处理...	4 KB
 setclasspath.sh	2020/6/30 22:50	Shell Script	4 KB
 shutdown.bat	2020/6/30 22:50	Windows 批处理...	2 KB
 shutdown.sh	2020/6/30 22:50	Shell Script	2 KB
 startup.bat	2020/6/30 22:50	Windows 批处理...	2 KB
 startup.sh	2020/6/30 22:50	Shell Script	2 KB
 tcnative-1.dll	2020/6/30 22:50	应用程序扩展	2,541 KB
 tomcat8.exe	2020/6/30 22:50	应用程序	122 KB
 tomcat8w.exe	2020/6/30 22:50	应用程序	119 KB
 tomcat-juli.jar	2020/6/30 22:50	Executable Jar File	51 KB
 tool-wrapper.bat	2020/6/30 22:50	Windows 批处理...	5 KB
 tool-wrapper.sh	2020/6/30 22:50	Shell Script	6 KB
 version.bat	2020/6/30 22:50	Windows 批处理...	2 KB
 version.sh	2020/6/30 22:50	Shell Script	2 KB

1.2 conf目录: 图中logging.properties说明有误, 该文件应该是关于日志的配置

Catalina	2020/7/8 11:35	文件夹	
catalina.policy	2020/6/30 22:50	POLICY 文件	14 KB
catalina.properties	2020/6/30 22:50	PROPERTIES 文件	8 KB
context.xml	2020/6/30 22:50	XML 文档	2 KB
jaspic-providers.xml	2020/6/30 22:50	XML 文档	2 KB
jaspic-providers.xsd	2020/6/30 22:50	XSD 文件	3 KB
logging.properties	2020/7/11 16:51	PROPERTIES 文件	4 KB
server.xml	2020/6/30 22:50	XML 文档	8 KB
tomcat-users.xml	2020/6/30 22:50	XML 文档	3 KB
tomcat-users.xsd	2020/6/30 22:50	XSD 文件	3 KB
web.xml	2020/6/30 22:50	XML 文档	173 KB

存放日志信息

端口定义在这里配置

定义tomcat用户、角色以及角色用于的功能等

和项目的web.xml一样，它是全局的，项目中的web.xml相同配置会覆盖它

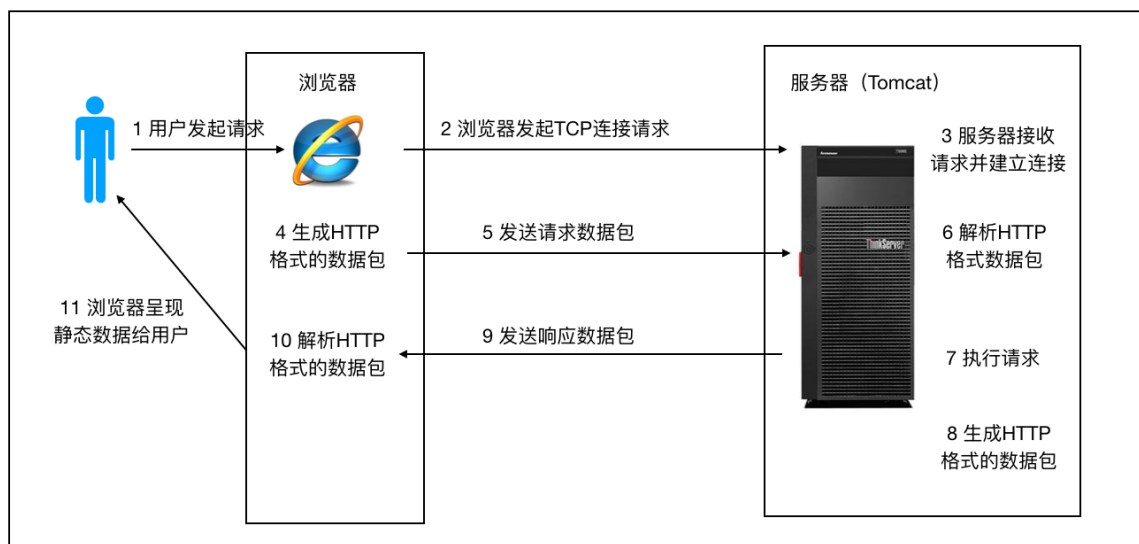
1.3 work：存放过程文件，jsp编译、运行时会产生一些过程文件就存放在这里。

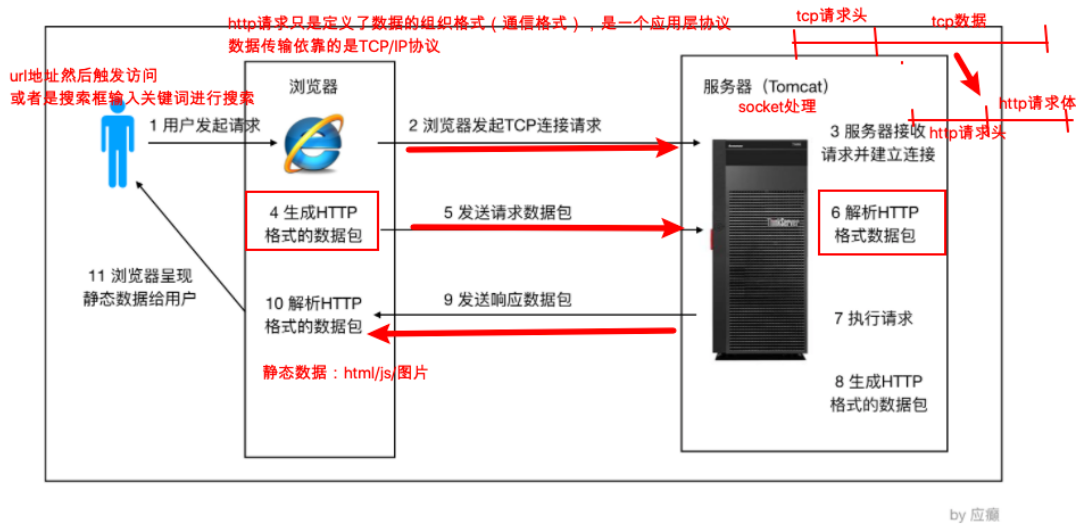
那么redirectPort属性的作用是什么呢？

当用户用http请求某个资源，而该资源本身又被设置了必须要https方式访问，此时Tomcat会自动重定向到这个redirectPort设置的https端口。

2.浏览器访问服务器流程：

http请求的处理过程：





上图中http请求只是定义了数据的通信格式，是应用层协议。真正数据传输依靠的是TCP/IP协议，期间经历三次握手。第6步tcp包括请求头和数据，http请求头和请求体就在tcp数据里。

http无状态、无连接?

无状态:

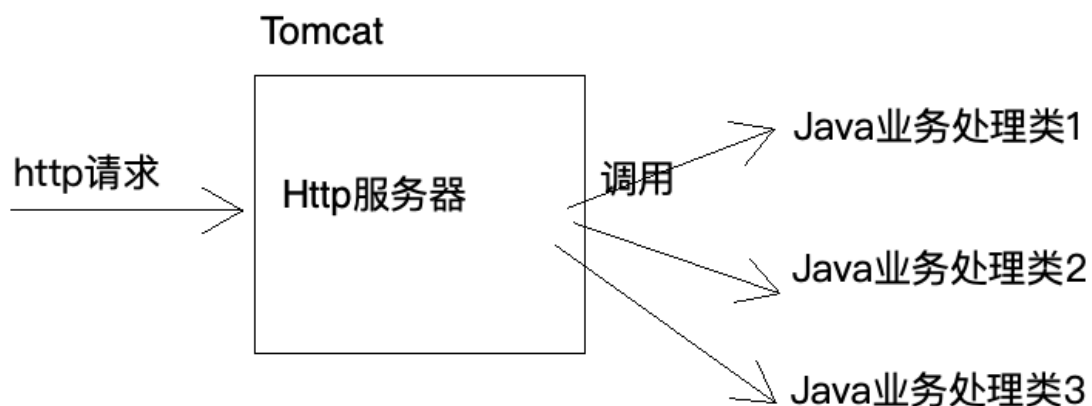
无连接:

session和cookie的优缺点

<https://www.cnblogs.com/lingyejun/p/9282169.html>

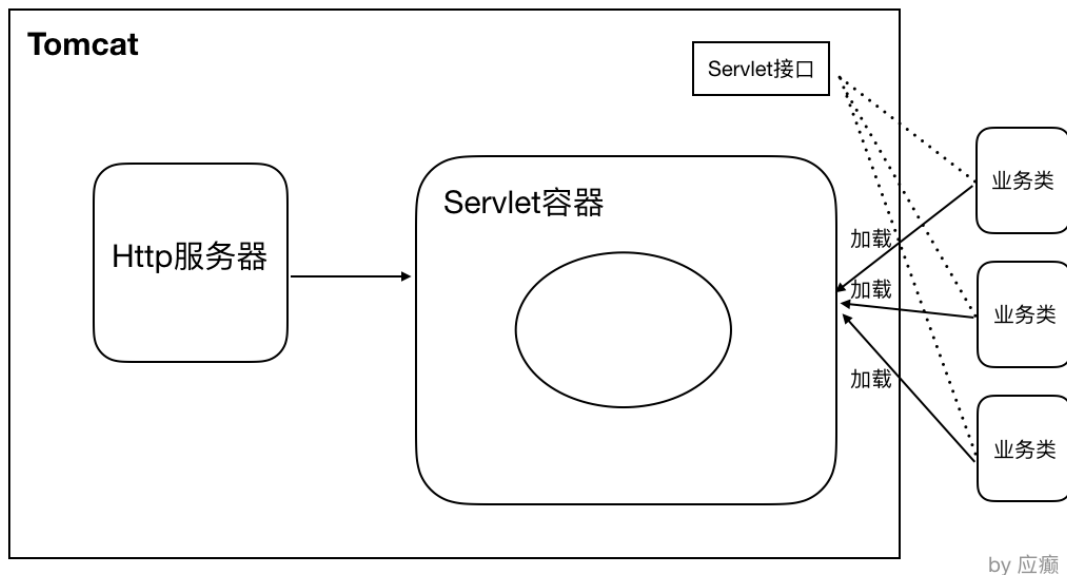
3. Tomcat 系统总体架构:

3.1 Tomcat 请求处理大致过程



如果Http服务器 (tomcat) 直接调用业务处理类完成业务处理的话存在一定问题: tomcat和业务类耦合在一起了

如果上图tomcat直接调用java业务处理类会出现耦合问题，所以Servlet容器出现解决该问题。



Tomcat的两个重要身份

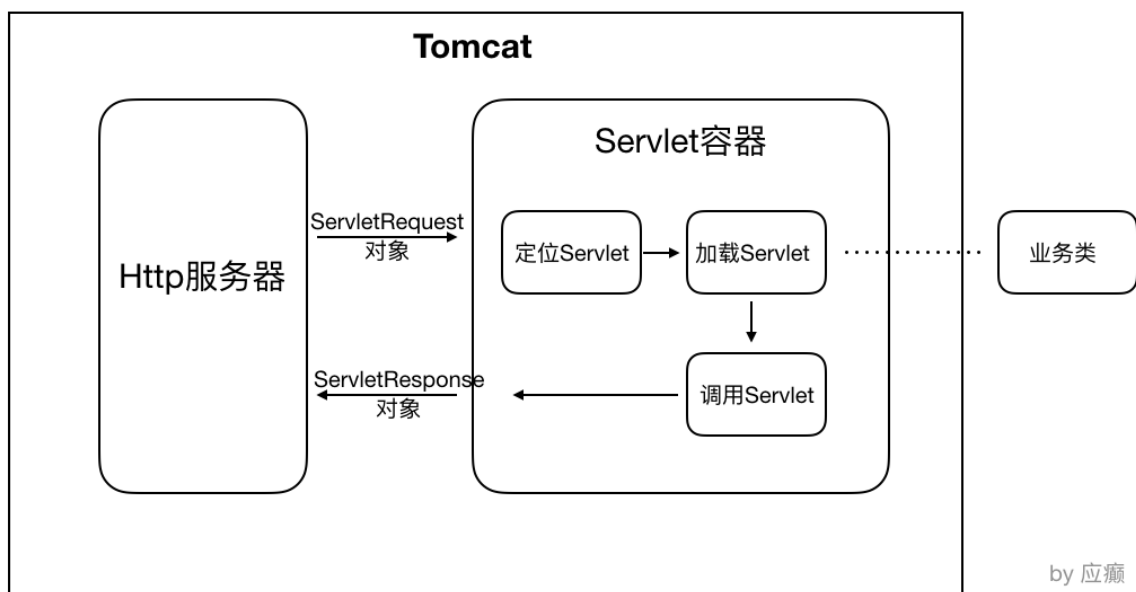
- 1) http服务器
- 2) Tomcat是一个Servlet容器（按照Servlet规范的要求去实现了Servlet容器）

Servlet规范：Servlet 容器通过Servlet接口调用业务类。Servlet接口和Servlet容器这一整套内容叫作Servlet规范。

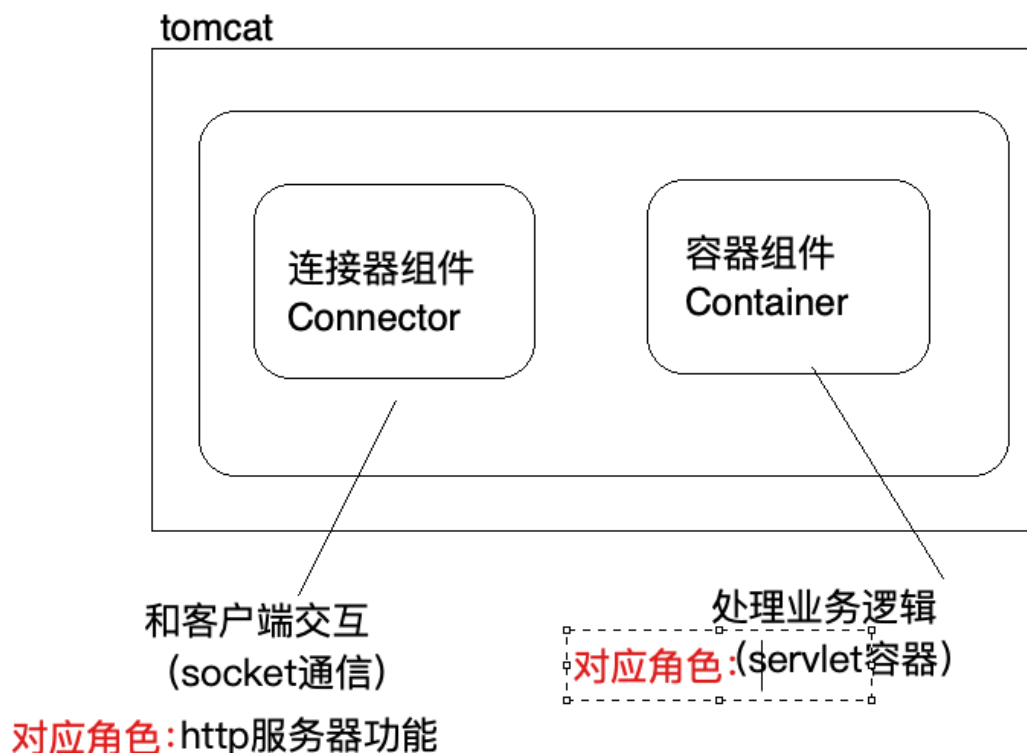
3.2 Tomcat Servlet容器处理流程

当用户请求某个URL资源时

- 1) HTTP服务器会把请求信息使用ServletRequest对象封装起来
- 2) 进一步去调用Servlet容器中某个具体的Servlet
- 3) 在 2) 中，Servlet容器拿到请求后，根据URL和Servlet的映射关系，找到相应的Servlet
- 4) 如果Servlet还没有被加载，就用反射机制创建这个Servlet，并调用Servlet的init方法来完成初始化
- 5) 接着调用这个具体Servlet的service方法来处理请求，请求处理结果使用ServletResponse对象封装
- 6) 把ServletResponse对象返回给HTTP服务器，HTTP服务器会把响应发送给客户端



3.3 Tomcat 系统总体架构



Tomcat 设计了两个核心组件连接器（Connector）和容器（Container）来完成 Tomcat 的两大核心功能。

连接器，负责对外交流： 处理Socket连接，负责网络字节流与Request和Response对象的转化；

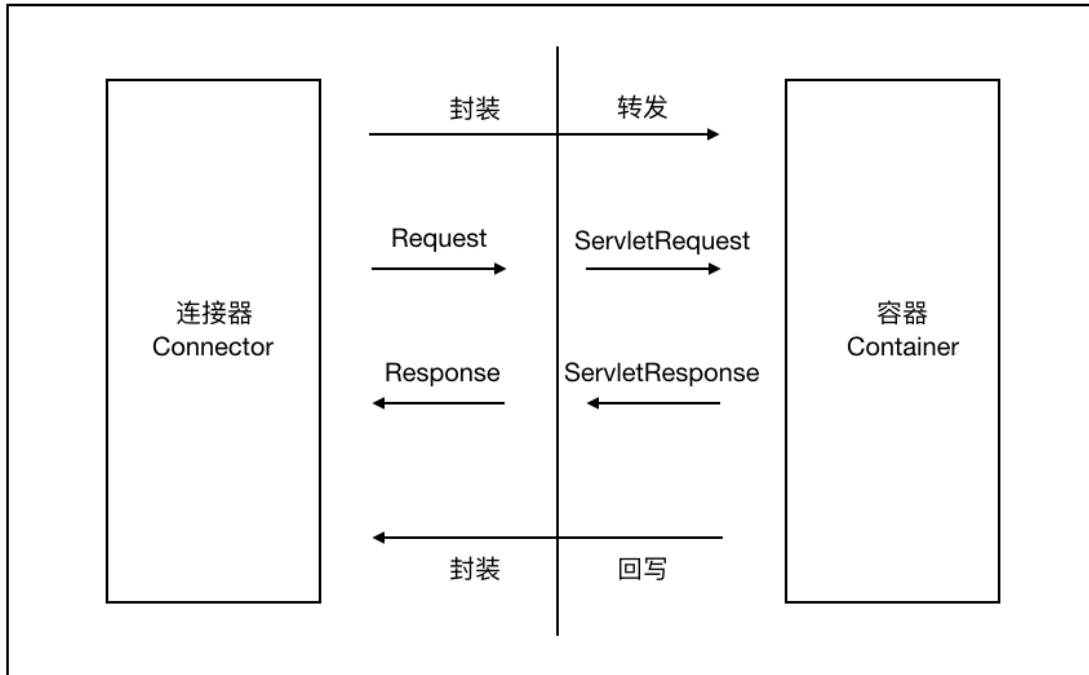
容器，负责内部处理： 加载和管理Servlet，以及具体处理Request请求；

4. Tomcat 连接器组件 Coyote

4.1 Coyote 简介

Coyote 是Tomcat 中连接器的组件名称，**是对外的接口**。客户端通过Coyote与服务器建立连接、发送请求并接受响应。

- (1) Coyote 封装了底层的网络通信（就是封装了Socket 请求及响应处理）
- (2) Coyote 使Catalina 容器（容器组件）与具体的请求协议及IO操作方式完全解耦（意思是具体的请求协议及IO操作方式交给Coyote）
- (3) Coyote 将Socket 输入转换封装为 Request 对象，进一步封装后交由Catalina 容器进行处理，处理请求完成后, Catalina 通过Coyote 提供的Response 对象将结果写入输出流（Request 转为 ServletRequest 是由Coyote 的adapter负责的）
- (4) Coyote 负责的是具体协议（应用层）和IO（传输层）相关内容



by 应癡

Tomcat Coyote 支持的 IO模型与协议

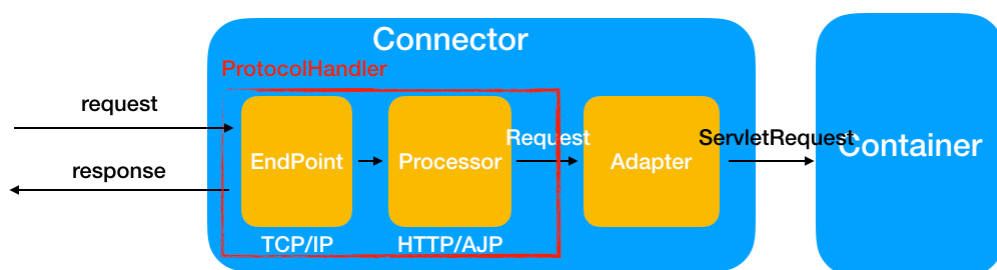
Tomcat支持多种应用层协议和I/O模型，如下：

	应用层协议	描述
应用层	HTTP/1.1	这是大部分Web应用采用的访问协议。
	AJP	用于和WX集成（如Apache），以实现静态资源的优化以及集群部署。当前支持AJP/1.3。
	HTTP/2	HTTP 2.0大幅度的提升了Web性能。下一代HTTP协议，自8.5以及9.0版本之后支持。
	IO模型	描述
传输层	NIO	非阻塞I/O，采用Java NIO类库实现。
	NIO2	异步I/O，采用JDK 7最新的NIO2类库实现。
	APR	采用Apache可移植运行库实现，是C/C++编写的本地库。如果选择该方案，需要单独安装APR库

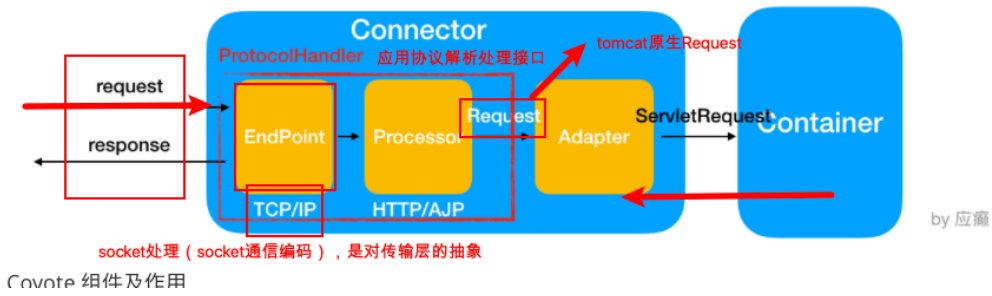
默认协议是HTTP/1.1 ,默认IO模型是NIO。

在 8.0 之前，Tomcat 默认采用的I/O方式为 BIO，之后改为 NIO。无论 NIO、NIO2 还是 APR，在性能方面均优于以往的BIO。如果采用APR，甚至可以达到 Apache HTTP Server 的影响性能。

4.2 Coyote 的内部组件及流程



by 应癡

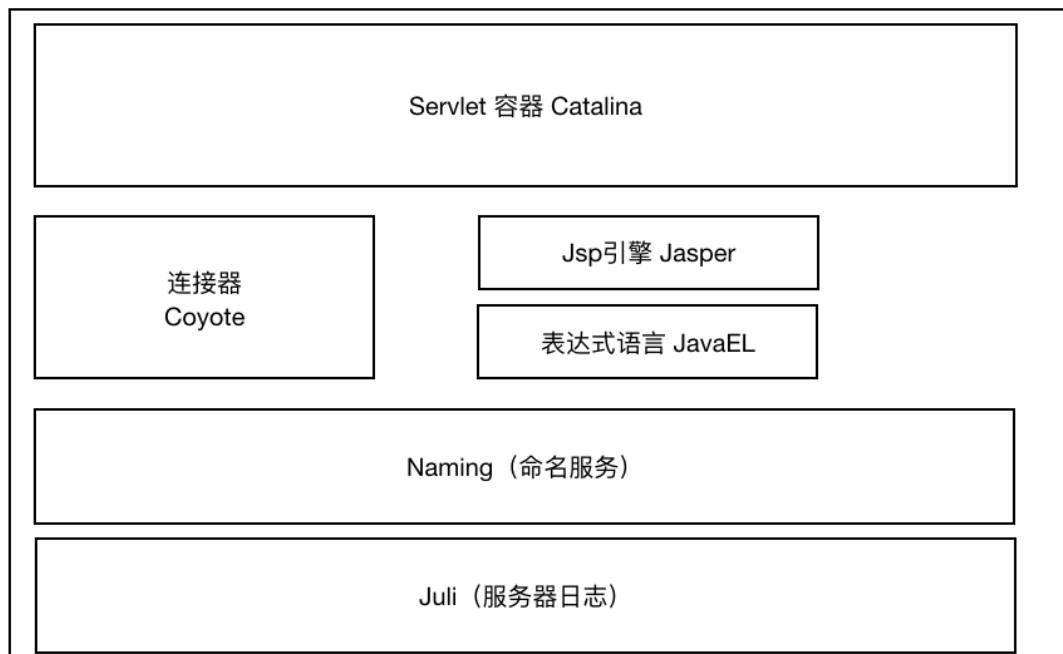


组件	作用描述
EndPoint	通信监听的接口，是具体Socket接收和发送处理器，是对传输层的抽象，因此EndPoint用来实现TCP/IP协议的
Processor	协议处理接口，用来实现HTTP协议，接收来自EndPoint的Socket，读取字节流解析成Tomcat Request和Response。
ProtocolHandler	协议接口，针对协议解析处理，Tomcat 按照协议和I/O 提供了6个实现类。
Adapter	将tomcat的原生Request转成ServletRequest，再调用容器

5.Tomcat Servlet 容器 Catalina

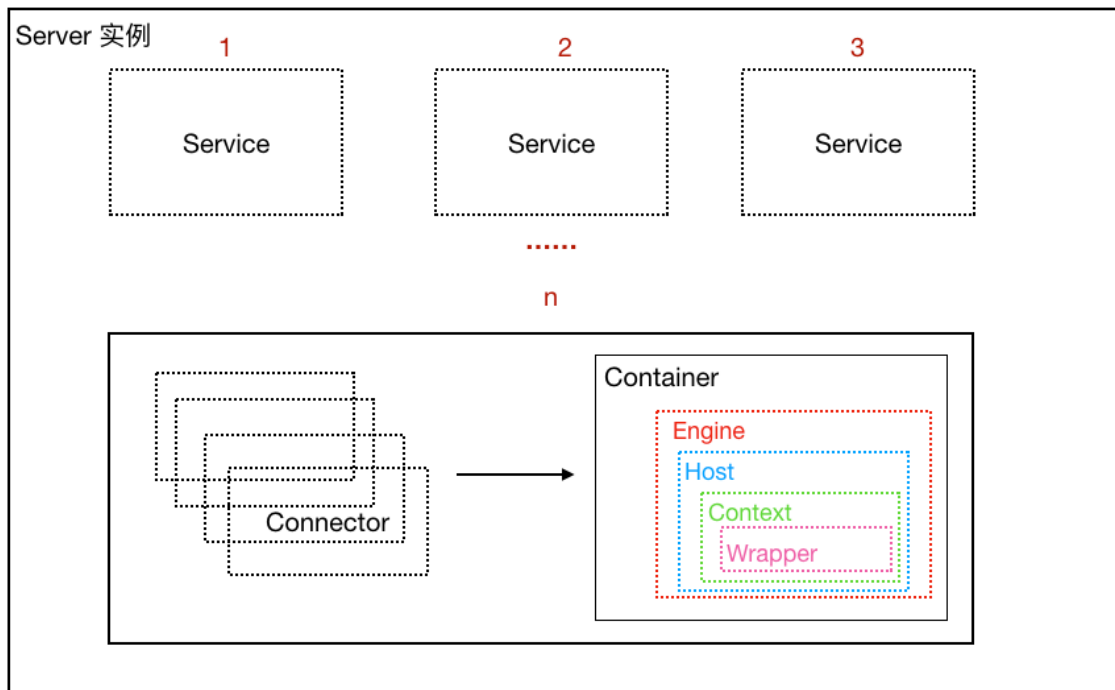
Catalina 是 Tomcat 的核心，其他模块是为Catalina提供支撑的。

tomcat模块分层结构：



by 应癡

Catalina结构：



by 应癡

可以认为整个Tomcat就是一个Catalina实例，Tomcat启动的时候会初始化这个实例，Catalina实例通过加载server.xml完成其他实例的创建，创建并管理一个Server，Server创建并管理多个服务(Service)，每个服务又可以有多个Connector和一个Container。

- **Catalina**：负责解析Tomcat的配置文件（server.xml），以此来创建服务器Server组件并进行管理。
- **Server**：负责组装并启动Servlet引擎,Tomcat连接器。Server通过实现Lifecycle接口，提供了一种优雅的启动和关闭整个系统的方式
- **Service**：它将若干个Connector组件绑定到一个Container。
- **Container**：容器，负责处理用户的servlet请求，并返回对象给web用户的模块。

|-**Engine**：表示整个Catalina的Servlet引擎，用来管理多个虚拟站点，一个Service最多只能有一个Engine，但是一个引擎可包含多个Host。

|-**Host**：代表一个虚拟主机，或者说一个站点。可以给Tomcat配置多个虚拟主机地址，而一个虚拟主机下可包含多个Context。

|-**Context**：表示一个Web应用程序，一个Web应用可包含多个Wrapper

|-**Wrapper**：表示一个Servlet，Wrapper作为容器中的最底层，不能包含子容器

上述组件的配置其实就体现在conf/server.xml中。

6. Tomcat 服务器核心配置详解

- Tomcat 作为服务器的配置，主要是 server.xml 文件的配置；
- server.xml中包含了 Servlet容器的相关配置，即 Catalina 的配置；

```

1 <!--port: 关闭服务器的监听端口 shutdown: 关闭服务器的指令字符串-->
2 <Server port="8005" shutdown="SHUTDOWN">
3     <!-- 以日志形式输出服务器、操作系统、JVM的版本信息 -->
4     <Listener className="org.apache.catalina.startup.VersionLoggerListener" />

```



```

5      <!-- 加载（服务器启动） 和 销毁（服务器停止） APR。 如果找不到APR库， 则会输出日
志， 并不影响 Tomcat启动 -->
6      <Listener className="org.apache.catalina.core.AprLifecycleListener"
SSLEngine="on" />
7      <!-- 避免JRE内存泄漏问题 -->
8      <Listener
className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
9      <!-- 加载（服务器启动） 和 销毁（服务器停止） 全局命名服务 -->
10     <Listener
className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
11     <!-- 在Context停止时重建 Executor 池中的线程， 以避免ThreadLocal 相关的内存泄漏 -
-->
12     <Listener
className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
13     <!--GlobalNamingResources 中定义了全局命名服务-->
14     <GlobalNamingResources>
15     <Resource name="UserDatabase" auth="Container"
16     type="org.apache.catalina.UserDatabase"
17     description="User database that can be updated and saved"
18     factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
19     pathname="conf/tomcat-users.xml" />
20     </GlobalNamingResources>
21
22     <!--
23     该标签用于创建 Service 实例，默认使用 org.apache.catalina.core.StandardService。
24     默认情况下，Tomcat 仅指定了Service 的名称， 值为 "Catalina"。
25     Service 子标签为： Listener、Executor、Connector、Engine，
26     其中：
27     Listener 用于为Service添加生命周期监听器，
28     Executor 用于配置Service 共享线程池，
29     Connector 用于配置Service 包含的链接器，
30     Engine 用于配置Service中链接器对应的Servlet 容器引擎
31     -->
32     <Service name="Catalina">
33         <!--
34         默认情况下，Service 并未添加共享线程池配置。 如果我们想添加一个线程池，可以在
35         <Service> 下添加如下配置：
36         name: 线程池名称，用于 Connector中指定
37         namePrefix: 所创建的每个线程的名称前缀，一个单独的线程名称为
38         namePrefix+threadNumber
39         maxThreads: 池中最大线程数
40         minSpareThreads: 活跃线程数，也就是核心池线程数，这些线程不会被销毁，会一直存在
41         maxIdleTime: 线程空闲时间，超过该时间后，空闲线程会被销毁，默认值为6000（1分钟），
单位
42         毫秒
43         maxQueueSize: 在被执行前最大线程排队数目，默认为Int的最大值，也就是广义的无限。除非
特
44         殊情况，这个值 不需要更改，否则会有请求不会被处理的情况发生
45         prestartminSpareThreads: 启动线程池时是否启动 minSpareThreads部分线程。默认值为
46         false，即不启动
47         threadPriority: 线程池中线程优先级，默认值为5，值从1到10
48         className: 线程池实现类，未指定情况下，默认实现类为
49         org.apache.catalina.core.StandardThreadExecutor。如果想使用自定义线程池首先需要
实现
50         org.apache.catalina.Executor接口
51         -->
52         <Executor name="commonThreadPool"
53         namePrefix="thread-exec-"
54         maxThreads="200"
55         minSpareThreads="100"
56         maxIdleTime="60000"

```

```

57     maxQueueSize="Integer.MAX_VALUE"
58     prestartminSpareThreads="false"
59     threadPriority="5"
60     className="org.apache.catalina.core.StandardThreadExecutor"/>
61
62     <!--
63     port:
64     端口号, Connector 用于创建服务端Socket 并进行监听, 以等待客户端请求链接。如果该属
性设置
65     为0, Tomcat将会随机选择一个可用的端口号给当前Connector 使用
66     protocol:
67     当前Connector 支持的访问协议。默认为 HTTP/1.1 , 并采用自动切换机制选择一个基于
JAVA
68     NIO 的链接器或者基于本地APR的链接器(根据本地是否含有Tomcat的本地库判定)
69     connectionTimeout:
70     Connector 接收链接后的等待超时时间, 单位为 毫秒。 -1 表示不超时。
71     redirectPort:
72     当前Connector 不支持SSL请求, 接收到了一个请求, 并且也符合security-constraint 约
束,
73     需要SSL传输, Catalina自动将请求重定向到指定的端口。
74     executor:
75     指定共享线程池的名称, 也可以通过maxThreads、minSpareThreads 等属性配置内部线程
池。
76     URIEncoding:
77     用于指定编码URI的字符编码, Tomcat8.x版本默认的编码为 UTF-8 , Tomcat7.x版本默认为
ISO-
78     8859-1
79     -->
80     <!--org.apache.coyote.http11.Http11NioProtocol , 非阻塞式 Java NIO 链接器-->
81     <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"
82     redirectPort="8443" />
83     <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
84
85     <!--可以使用共享线程池-->
86     <Connector port="8080"
87     protocol="HTTP/1.1"
88     executor="commonThreadPool"
89     maxThreads="1000"
90     minSpareThreads="100"
91     acceptCount="1000"
92     maxConnections="1000"
93     connectionTimeout="20000"
94     compression="on"
95     compressionMinSize="2048"
96     disableUploadTimeout="true"
97     redirectPort="8443"
98     URIEncoding="UTF-8" />
99     <!--
100     name: 用于指定Engine 的名称, 默认为Catalina
101     defaultHost: 默认使用的虚拟主机名称, 当客户端请求指向的主机无效时, 将交由默认的虚
拟主机处
理, 默认为localhost
102
103     -->
104     <Engine name="Catalina" defaultHost="localhost">
105         <!-- Host 标签用于配置一个虚拟主机 -->
106         <Host name="localhost" appBase="webapps" unpackWARs="true"
autoDeploy="true">
107             <!--
108             docBase: Web应用目录或者War包的部署路径。可以是绝对路径, 也可以是相对于
Host appBase的
109             相对路径。

```

```
110         path: Web应用的Context 路径。如果我们Host名为localhost， 则该web应用访问
    的根路径为：
111         http://localhost:8080/web3。
112         -->
113         <Context docBase="/Users/yingdian/web_demo" path="/web3"></Context>
114     </Host>
115 </Engine>
116 </Service>
117 </Server>
118
119 配置多个Host可以实现不同路径进入不同的webapp文件：比如www.abc.com:8080进入webapp2的文件，
    localhost:8080进入webapp的文件
120
121 配置多个Context可以实现不同端口后缀不一样进入不同文件，比如www.abc.com:8080/root进入a
    文件，www.abc.com:8080/root1 进入b文件。
```

service的作用是统一管理connector和container，一个service可以包括多个connector和一个container。而server的作用是，管理所有的service，**一个server可以包括多个service**。server负责管理所有service的**生命周期**，这样就管理了所有的connector和container，以及connector和container的所有内部组件。**这样就不需要单独对connector和container单独进行开启或关闭了。**