

Tomcat

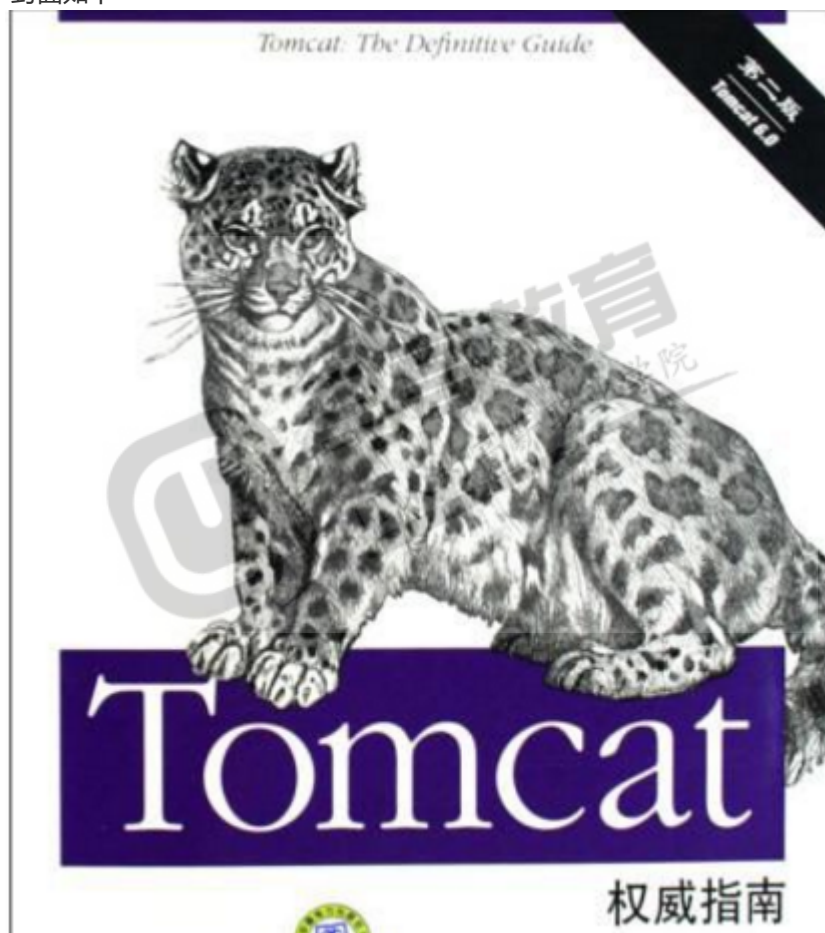
历史

起始于SUN的一个Servlet的参考实现项目Java Web Server，作者是James Duncan Davidson，后将项目贡献给了ASF。和ASF现有的项目合并，并开源成为顶级项目，官网<http://tomcat.apache.org/>。

Tomcat仅仅实现了Java EE规范的与Servlet、JSP相关的类库，是JavaEE不完整实现。

著名图书出版商O'Reilly约稿该项目成员，Davidson希望使用一个公猫作为封面，但是公猫已经被另一本书使用，书出版后封面是一只雪豹。

《Tomcat权威指南》封面如下



1999年发布初始版本是Tomcat 3.0，实现了Servlet 2.2和JSP1.1规范。

Tomcat 4.x发布时，内建了Catalina（Servlet容器）和Jasper（JSP engine）等。

商用的有IBM WebSphere、Oracle WebLogic（原属于BEA公司）、Oracle Oc4j、Glassfish、JBoss等。

开源实现有Tomcat、Jetty、Resin。

安装

可以使用Centos7 yum源自带的安装。yum源中是Tomcat 7.0版本。安装完通过浏览器可以观察一下首页。

```
# yum install tomcat tomcat-admin-webapps tomcat-webapps
# systemctl start tomcat.service
# ss -tanl
LISTEN      0          100        :::8009
LISTEN      0          100        :::8080
```

采用Apache官网下载，下载8.x.x

```
# tar xf apache-tomcat-8.5.42.tar.gz -C /usr/local
# cd /usr/local
# ln -sv apache-tomcat-8.5.42/ tomcat
"tomcat" -> "apache-tomcat-8.5.42/"

# cd tomcat
# cd bin
# ./catalina.sh --help
# ./catalina.sh version
# ./catalina.sh start
# ss -tanlp
# ./catalina.sh stop

# ./startup.sh
# ./shutdown.sh
```

useradd -r java 建立系统账号

上例中，启动身份是root，如果使用普通用户启动可以使用

```
# useradd -r java
# chown -R java.java ./*
# su - java -c '/usr/local/tomcat/bin/catalina.sh start'
# ps -aux | grep tomcat
```

目录结构

目录	说明
bin	服务启动、停止等相关
conf	配置文件
lib	库目录
logs	日志目录
webapps	应用程序，应用部署目录
work	jsp编译后的结果文件

配置文件

文件名	说明
server.xml	主配置文件
web.xml	每个webapp只有“部署”后才能被访问，它的部署方式通常由web.xml进行定义，其存放位置为WEB-INF/目录中；此文件为所有的webapps提供默认部署相关的配置
context.xml	每个webapp都可以专用的配置文件，它通常由专用的配置文件context.xml来定义，其存放位置为WEB-INF/目录中；此文件为所有的webapps提供默认配置
tomcat-users.xml	用户认证的账号和密码文件
catalina.policy	当使用-security选项启动tomcat时，用于为tomcat设置安全策略
catalina.properties	Java属性的定义文件，用于设定类加载器路径，以及一些与JVM调优相关参数
logging.properties	日志系统相关的配置。log4j

组件分类

顶级组件

Server，代表整个Tomcat容器

服务类组件

Service，组织Engine和Connector，里面只能包含一个Engine

连接器组件

Connector，有HTTP、HTTPS、AJP协议的连接器

容器类

Engine、Host、Context都是容器类组件，可以嵌入其它组件，内部配置如何运行应用程序。

内嵌类

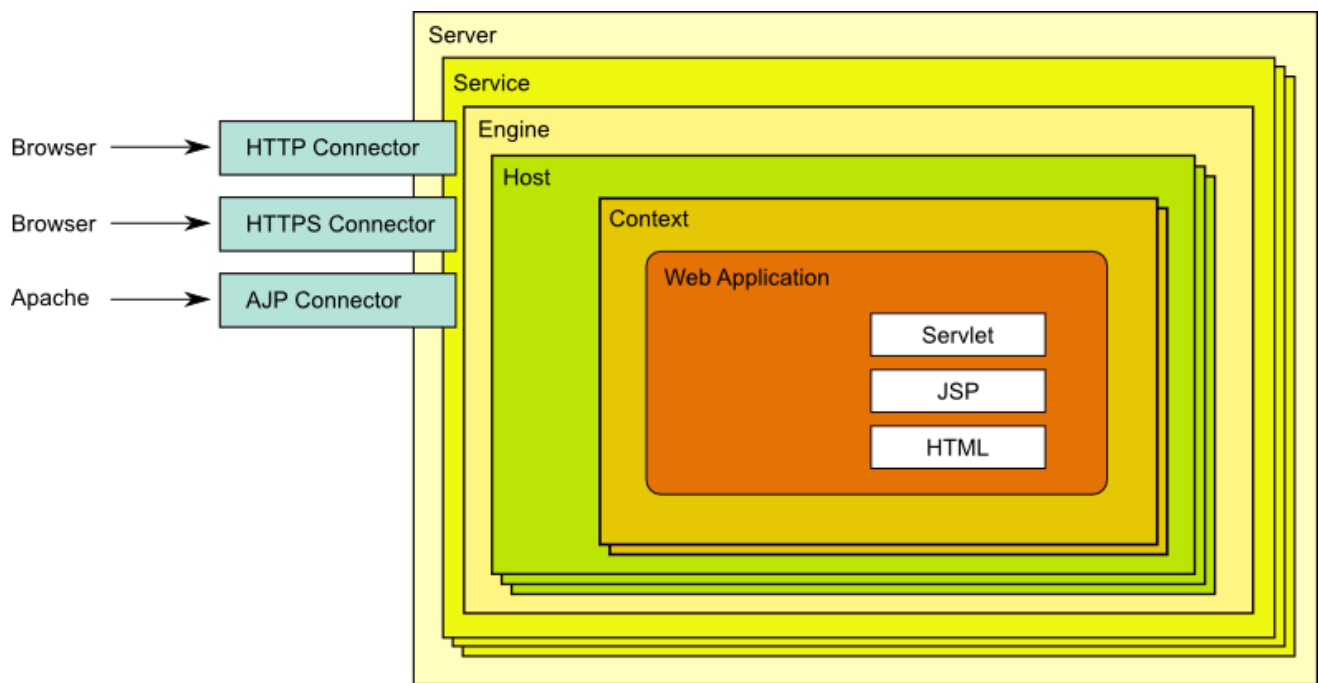
可以内嵌到其他组件内，valve、logger、realm、loader、manager等。以logger举例，在不同容器组件内定义。

集群类组件

listener、cluster

Tomcat内部组成

由上述组件就构成了Tomcat，如下图



名称	说明
Server	Tomcat运行的进程实例
Connector	负责客户端的HTTP、HTTPS、AJP等协议的连接。一个Connector只属于某一个Engine
Service	用来组织Engine和Connector的关系
Engine	响应并处理用户请求。一个引擎上可以绑定多个Connector
Host	虚拟主机
Context	应用的上下文，配置路径映射path => directory

AJP (Apache Jserv protocol) 是一种基于TCP的二进制通讯协议。

核心组件

- Tomcat启动一个Server进程。可以启动多个Server，但一般只启动一个
- 创建一个Service提供服务。可以创建多个Service，但一般也只创建一个
 - 每个Service中，是Engine和其连接器Connector的关联配置
- 可以为这个Server提供多个连接器Connector，这些Connector使用了不同的协议，绑定了不同的端口。其作用就是处理来自客户端的不同的连接请求或响应
- Service内部还定义了Engine，引擎才是真正的处理请求的入口，其内部定义多个虚拟主机Host
 - Engine对请求头做了分析，将请求发送给相应的虚拟主机
 - 如果没有匹配，数据就发往Engine上的defaultHost缺省虚拟主机
 - Engine上的缺省虚拟主机可以修改
- Host定义虚拟主机，虚拟主机有name名称，通过名称匹配
- Context定义应用程序单独的路径映射和配置

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

举例：

假设来自客户的请求为：<http://localhost:8080/test/index.jsp>

- 浏览器端的请求被发送到服务端端口8080，Tomcat进程监听在此端口上。通过侦听的HTTP/1.1 Connector获得此请求。
- Connector把该请求交给它所在的Service的Engine来处理，并等待Engine的响应
- Engine获得请求localhost:8080/test/index.jsp，匹配它所有虚拟主机Host。
- 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
- localhost Host获得请求/test/index.jsp，匹配它所拥有的所有Context
- Host匹配到路径为/test的Context
- path=/test的Context获得请求/index.jsp，在它的mapping table中寻找对应的servlet
- Context匹配到URL PATTERN为 *.jsp 的servlet，对应于JspServlet类构造HttpServletRequest对象和HttpServletResponse对象，作为参数调用JspServlet的doGet或doPost方法。
- Context把执行完了之后的HttpServletResponse对象返回给Host
- Host把HttpServletResponse对象返回给Engine
- Engine把HttpServletResponse对象返回给Connector
- Connector把HttpServletResponse对象返回给浏览器端

应用部署

根目录

Tomcat中默认网站根目录是CATALINA_BASE/webapps/

在Tomcat中部署主站应用程序和其他应用程序，和之前WEB服务程序不同。

nginx

假设在nginx中部署2个网站应用eshop、bbs，假设网站根目录是/var/www/html，那么部署可以是这样的。

eshop解压缩所有文件放到/var/www/html/目录下。

bbs的文件放在/var/www/html/bbs下。

Tomcat

Tomcat中默认网站根目录是CATALINA_BASE/webapps/

在Tomcat的webapps目录中，有个非常特殊的目录ROOT，它就是网站默认根目录。

将eshop解压后的文件放到这个ROOT中。

bbs解压后文件都放在CATALINA_BASE/webapps/bbs目录下。

每一个虚拟主机的目录都可以使用appBase配置自己的站点目录，里面都可以使用ROOT目录作为主站目录。

JSP WebApp目录结构

- 主页配置：一般指定为index.jsp或index.html
- WEB-INF/：当前WebApp的私有资源路径，通常存储当前应用使用的web.xml和context.xml配置文件
- META-INF/：类似于WEB-INF
- classes/：类文件，当前webapp需要的类
- lib/：当前应用依赖的jar包

实验

默认情况下，/usr/local/tomcat/webapps/ROOT/下添加一个index.html文件，观察访问到了什么？

将/usr/local/tomcat/conf/web.xml中的下面 `<welcome-file-list>` 标签 内容（默认页），复制到/usr/local/tomcat/webapps/ROOT/WEB-INF/web.xml中，如下

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1"
  metadata-complete="true">

  <display-name>Welcome to Tomcat</display-name>
  <description>
    Welcome to Tomcat
  </description>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

配置修改后，观察首页变化

webapp归档格式

- .war：WebApp打包
- .jar：EJB类打包文件
- .rar：资源适配器类打包文件

- .ear: 企业级WebApp打包

传统，应用开发测试后，通常打包为war格式，这种文件部署到了Tomcat的webapps下，还可以自动展开。

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

部署Deploy

- 部署：将webapp的源文件放置到目标目录，通过web.xml和context.xml文件中配置的路径就可以访问该webapp，通过类加载器加载其特有的类和依赖的类到JVM上。
 - 自动部署Auto Deploy：Tomcat发现多了这个应用就把它加载并启动起来
 - 手动部署
 - 冷部署：将webapp放到指定目录，才去启动Tomcat
 - 热部署：Tomcat服务不停止，需要依赖工具manager、ant脚本、tcd (tomcat client deployer) 等
- 反部署undeploy：停止webapp的运行，并从JVM上清除已经加载的类，从Tomcat实例上卸载掉webapp
- 启动start：是webapp能够访问
- 停止stop：webapp不能访问，不能提供服务，但是JVM并不清除它

实验

1、添加一个文件，test.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>jsp例子</title>
</head>
<body>
后面的内容是服务器端动态生成字符串，最后拼接在一起
<%
out.println("hello jsp");
%>
</body>
</html>
```

先把test.jsp放到ROOT下去，试试看，访问 `http://YourIP:8080/test.jsp`。

立即可以看到，这是通过路径映射找到相应的test.jsp后，转换成test_jsp.java，在编译成test_jsp.class。

/usr/local/tomcat/work/Catalina/localhost/ROOT/org/apache/jsp下有转换后的文件。

2、添加一个应用

模拟部署一个应用

```
# cd
```

常见开发项目目录组成

```
# mkdir projects/myapp/{WEB-INF,classes,lib} -pv
```

```
mkdir: 已创建目录 "projects"
```

```
mkdir: 已创建目录 "projects/myapp"
```

```
mkdir: 已创建目录 "projects/myapp/WEB-INF"
```

```
mkdir: 已创建目录 "projects/myapp/classes"
```

```
mkdir: 已创建目录 "projects/myapp/lib"
```

常见应用首页，内容就用上面的test.jsp

```
# vi projects/myapp/index.jsp
```

手动复制项目目录到webapps目录下去

```
# cp -r projects/myapp/ /usr/local/tomcat/webapps/
```

使用http://YourIP:8080/myapp/访问试试看

配置详解

server.xml

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

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

8005是Tomcat的管理端口，默认监听在127.0.0.1上。SHUTDOWN这个字符串接收到后就会关闭此Server。


```
# telnet 127.0.0.1 8005
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
SHUTDOWN
```

这个管理功能建议禁用，改shutdown为一串猜不出的字符串。

```
<Server port="8005" shutdown="44ba3c71d57f494992641b258b965f28">
```

```
<GlobalNamingResources>
  <!-- Editable user database that can also be used by
        UserDatabaseRealm to authenticate users
  -->
  <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>
```

用户认证，配置文件是conf/tomcat-users.xml。

打开tomcat-users.xml，我们需要一个角色manager-gui。

```
<tomcat-users xmlns="http://tomcat.apache.org/xml"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
              version="1.0">
  <role rolename="manager-gui"/>
  <user username="wayne" password="wayne" roles="manager-gui"/>
</tomcat-users>
```

Tomcat启动加载后，这些内容是常驻内存的。如果配置了新的用户，需要重启Tomcat。

访问manager的时候告诉403，提示中告诉去manager的context.xml中修改

403 Access Denied

You are not authorized to view this page.

By default the Manager is only accessible from a browser running on the same machine as Tomcat. If you wish to modify this restriction, you'll need to edit the Manager's `context.xml` file.

If you have already configured the Manager application to allow access and you have used your browser's back button, used a saved book-mark or similar then you may have triggered the cross-site request forgery (CSRF) protection that has been enabled for the HTML interface of the Manager application. You will need to reset this protection by returning to the [main Manager page](#). Once you return to this page, you will be able to continue using the Manager application's HTML interface normally. If you continue to see this access denied message, check that you have the necessary permissions to access this application.

If you have not changed any configuration files, please examine the file `conf/tomcat-users.xml` in your installation. That file must contain the credentials to let you use this webapp.

For example, to add the `manager-gui` role to a user named `tomcat` with a password of `secret`, add the following to the config file listed above.

```
<role rolename="manager-gui"/>
<user username="tomcat" password="secret" roles="manager-gui"/>
```

Note that for Tomcat 7 onwards, the roles required to use the manager application were changed from the single `manager` role to the following four roles. You will need to assign the role(s) required for the functionality you wish to access.

- `manager-gui` - allows access to the HTML GUI and the status pages
- `manager-script` - allows access to the text interface and the status pages
- `manager-jmx` - allows access to the JMX proxy and the status pages
- `manager-status` - allows access to the status pages only

The HTML interface is protected against CSRF but the text and JMX interfaces are not. To maintain the CSRF protection:

- Users with the `manager-gui` role should not be granted either the `manager-script` or `manager-jmx` roles.
- If the text or jmx interfaces are accessed through a browser (e.g. for testing since these interfaces are intended for tools not humans) then the browser must be closed afterwards to terminate the session.

For more information - please see the [Manager App How-To](#).

文件路径/usr/local/tomcat/webapps/manager/META-INF/context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context antiResourceLocking="false" privileged="true" >
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
    allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
  <Manager sessionAttributeValueClassNameFilter="java\.lang\.(?:Boolean|Integer|Long|Number|string)|org\.apache\.catalina\.filters\.CsrfPreventionFilter\$LruCache(?:\$1)?|java\.util\.(?:Linked)?HashMap"/>
</Context>
```

看正则表达式就知道是本地访问了，由于当前访问地址是192.168.x.x，可以修改正则则为

```
allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1|192\.168\.\d+\.\d+"
```

再次测试，成功。

```
<Service name="Catalina">
```

一般情况下，一个Server实例配置一个Service，name属性相当于该Service的ID。

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

连接器配置。

redirectPort，如果访问HTTPS协议，自动转向这个连接器。但大多数时候，Tomcat并不会开启HTTPS，因为Tomcat往往部署在内部，HTTPS性能较差。

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

引擎配置。

defaultHost指向内部定义某虚拟主机。缺省虚拟主机可以改动，默认localhost。

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

虚拟主机配置。

name必须是主机名，用主机名来匹配。

appBase，当期主机的网页根目录，相对于CATALINA_HOME，也可以使用绝对路径

unpackWARs是否自动解压war格式

autoDeploy 热部署，自动加载并运行应用

虚拟主机配置实验

尝试再配置一个虚拟主机，并将myapp部署到/data/webapps目录下

```
<Host name="node1.magedu.com" appBase="/data/webapps/" unpackWARs="True" autoDeploy="false" />
```

常见虚拟主机根目录

```
# mkdir /data/webapps -pv
```

```
mkdir: 已创建目录 "/data"
```

```
mkdir: 已创建目录 "/data/webapps"
```

```
# cp -r ~/projects/myapp/ /data/webapps/ROOT
```

```
# pwd
```

```
/usr/local/tomcat
```

```
# bin/shutdown.sh
```

```
# bin/startup.sh
```

刚才在虚拟主机中主机名定义node1.magedu.com，所以需要主机在本机手动配置一个**域名解析**。

如果是windows，修改在C:\Windows\System32\drivers\etc下的hosts文件，需要管理员权限。

使用<http://node1.magedu.com:8080/>访问试试看。

Context配置

Context作用：

- 路径映射
- 应用独立配置，例如单独配置应用日志、单独配置应用访问控制

```
<Context path="/test" docBase="/data/test" reloadable="" />
```

path指的是访问的路径

docBase, 可以是绝对路径, 也可以是相对路径 (相对于Host的appBase)

reloadable, true表示如果WEB-INF/classes或META-INF/lib目录下.class文件有改动, 就会将WEB应用重新加载。生成环境中, 会使用false来禁用。

将~/projects/myapp/下面的项目文件复制到/data/下

```
# cp -r ~/projects/myapp /data/myappv1
# cd /data
# ln -sv myappv1 test
```

可以修改一下index.jsp好区别一下。

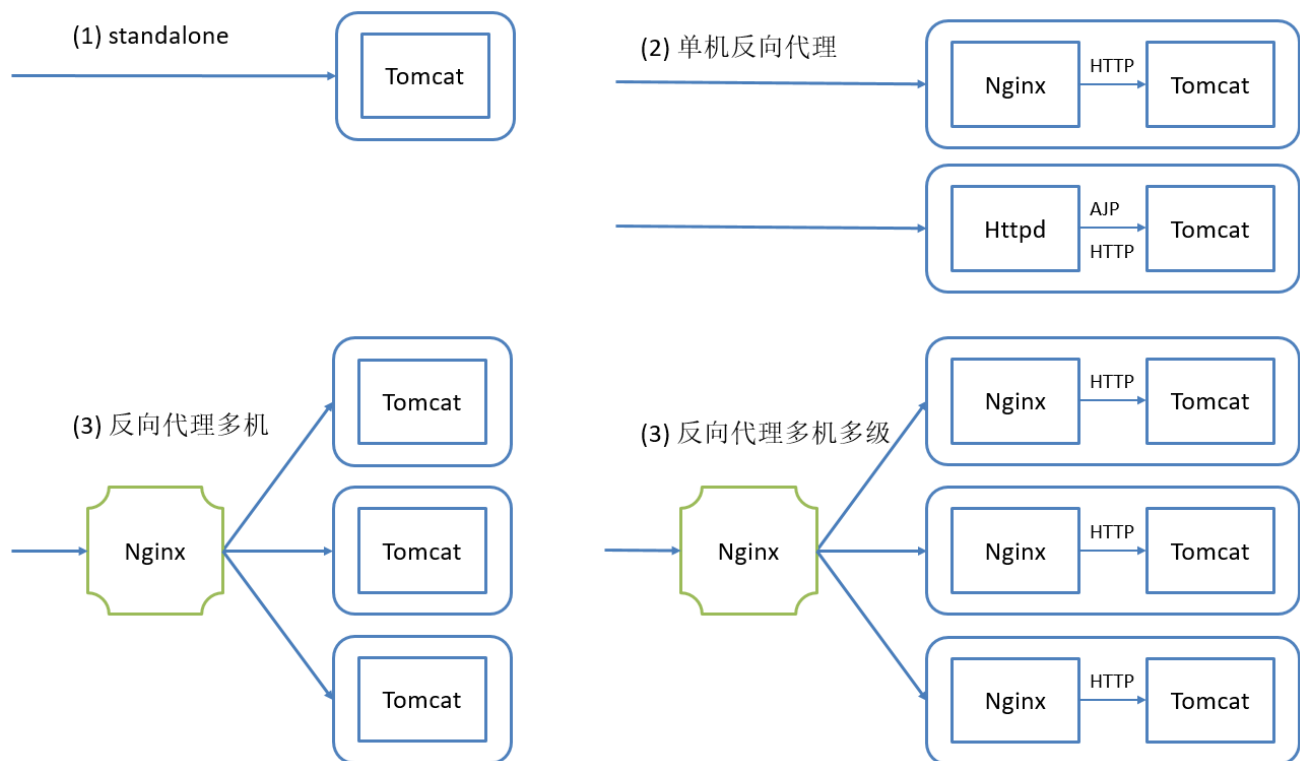
Tomcat的配置文件server.xml中修改如下

```
<Host name="node1.magedu.com" appBase="/data/webapps"
      unpackWARs="true" autoDeploy="true" >
  <Context path="/test" docBase="/data/test" reloadable="" />
</Host>
```

使用<http://node1.magedu.com:8080/test/>

注意: 这里特别使用了软链接, 原因就是以后版本升级, 需要将软链接指向myappv2, 重启Tomcat。如果新版上线后, 出现问题, 重新修改软链接到上一个版本的目录, 并重启, 就可以实现回滚。

常见部署方式



- standalone模式，Tomcat单独运行，直接接受用户的请求，不推荐。
- 反向代理，单机运行，提供了一个Nginx作为反向代理，可以做到静态有nginx提供响应，动态jsp代理给Tomcat
 - LNMT: Linux + Nginx + MySQL + Tomcat
 - LAMT: Linux + Apache (Httpd) + MySQL + Tomcat
- 前置一台Nginx，给多台Tomcat实例做反向代理和负载均衡调度，Tomcat上部署的纯动态页面更适合
 - LNMT: Linux + Nginx + MySQL + Tomcat
- 多级代理
 - LNNMT: Linux + Nginx + Nginx + MySQL + Tomcat

Nginx和Tomcat实践

nginx安装

从epel源安装nginx

```
# wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo

# yum install nginx -y
# cd /etc/nginx
# vim nginx.conf

# nginx -t
```

全部反向代理测试

```
# 全部反向代理测试
location / {
    # proxy_pass http://127.0.0.1:8080; # 不管什么请求，都会访问后面的localhost虚拟主机
    proxy_pass http://node1.magedu.com:8080; # 修改服务器的/etc/hosts
}
```

<http://192.168.142.151/>或者<http://node1.magedu.com/>全部代理给了自定义的虚拟主机

动静分离代理

```
location / {
    root /data/webapps/ROOT;
    index index.html;
}

location ~* \.jsp$ {
    proxy_pass http://node1.magedu.com:8080; # /etc/hosts
}
```

在/data/webapps/ROOT目录下增加一个index.html。

<http://192.168.142.140/>和<http://192.168.142.140/index.jsp>测试一下看看。

但是实际上Tomcat不太适合做动静分离，它的管理程序的图片不好做动静分离部署。

应用管理

```
# 全部反向代理
location / {
    proxy_pass http://127.0.0.1:8080; # 不管什么请求，都会访问后面的localhost虚拟主机
}
```

点击Tomcat首页的右上角的“Manager App”按钮，弹出登录对话框。

管理界面

Applications 应用程序管理，可以启动、停止、重加载、反部署、清理过期session

Deploy 可以热部署，也可以部署war文件。

Deploy	
Deploy directory or WAR file located on server	
Context Path (required):	<input type="text" value="/test1"/>
XML Configuration file URL:	<input type="text"/>
WAR or Directory URL:	<input type="text" value="/data/myapp"/>
<input type="button" value="Deploy"/>	

Host Manager虚拟主机管理

配置如下

```
<tomcat-users xmlns="http://tomcat.apache.org/xml"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
               version="1.0">
  <role rolename="manager-gui"/>
  <role rolename="admin-gui" />
  <user username="wayne" password="wayne" roles="manager-gui,admin-gui"/>
</tomcat-users>
```

重启Tomcat，点击“Host Manager”按钮

可以新增虚拟主机。

httpd和Tomcat实践

```
# yum install httpd -y
# httpd -M
# vim /etc/httpd/conf.d/
```

httpd配置

proxy_http_module模块代理配置

```
<VirtualHost *:80>
    ServerName      node1.magedu.com
    ProxyRequests    Off
    ProxyVia         On
    ProxyPreserveHost On
    ProxyPass        / http://127.0.0.1:8080/
    ProxyPassReverse / http://127.0.0.1:8080/
</VirtualHost>
```

ProxyRequests: Off关闭正向代理。

ProxyPass: 反向代理指令

ProxyPassReverse: 保留代理的response头不重写（个别除外）

ProxyPreserveHost: On开启。让代理保留原请求的Host首部

ProxyVia: On开启。代理的请求响应时提供一个response的via首部

```
# vim /etc/httpd/conf.d/http-tomcat.conf

# httpd -t
# systemctl start httpd

# /usr/local/tomcat/bin/startup.sh
```

<http://192.168.142.140/>

<http://node1.magedu.com/>

<http://node1.magedu.com/index.jsp>

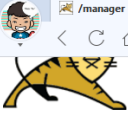
以上3个URL看到了不同的页面，说明 ProxyPreserveHost On 起了作用。

设置 ProxyPreserveHost Off 再看效果，说明什么？

proxy_ajp_module模块代理配置


```
<VirtualHost *:80>
    ServerName      node1.magedu.com
    ProxyRequests    Off
    ProxyVia         On
    ProxyPreserveHost On
    ProxyPass        / ajp://127.0.0.1:8009/
</VirtualHost>
```

查看Server Status可以看到确实使用的是ajp连接了。



/manager

http://192.168.142.140/manager/status



Server Status

Manager

List Applications

HTML Manager Help

Manager Help

Complete Server Status

Server Information

Tomcat Version	JVM Version	JVM Vendor	OS Name	OS Version	OS Architecture	Hostname	IP Address
Apache Tomcat/8.5.42	1.8.0_191-b12	Oracle Corporation	Linux	3.10.0-957.el7.x86_64	amd64	nodex	192.168.142.140

JVM

Free Memory: 9.07 MB Total Memory: 28.78 MB Max Memory: 386.68 MB

Memory Pool	Type	Initial	Total	Maximum	Used
Eden Space	Heap memory	7.00 MB	8.06 MB	106.68 MB	5.24 MB (4%)
Survivor Space	Heap memory	0.81 MB	0.93 MB	13.31 MB	0.93 MB (7%)
Tenured Gen	Heap memory	17.37 MB	19.78 MB	266.68 MB	13.52 MB (5%)
Code Cache	Non-heap memory	2.43 MB	6.37 MB	240.00 MB	6.33 MB (2%)
Compressed Class Space	Non-heap memory	0.00 MB	2.62 MB	1024.00 MB	2.44 MB (0%)
Metaspace	Non-heap memory	0.00 MB	23.87 MB	-0.00 MB	23.48 MB

"ajp-nio-8009"

Max threads: 200 Current thread count: 10 Current thread busy: 1 Keep alive sockets count: 1
Max processing time: 720 ms Processing time: 1.109 s Request count: 9 Error count: 2 Bytes received: 0.00 MB Bytes sent: 0.09 MB

Stage	Time	B Sent	B Recv	Client (Forwarded)	Client (Actual)	VHost	Request
S	58 ms	0 KB	0 KB	192.168.142.1	192.168.142.1	192.168.142.140	GET /manager/status HTTP/1.1

P: Parse and prepare request S: Service F: Finishing R: Ready K: Keepalive

"http-nio-8080"

Max threads: 200 Current thread count: 10 Current thread busy: 0 Keep alive sockets count: 0
Max processing time: 0 ms Processing time: 0.0 s Request count: 0 Error count: 0 Bytes received: 0.00 MB Bytes sent: 0.00 MB

Stage	Time	B Sent	B Recv	Client (Forwarded)	Client (Actual)	VHost	Request
-------	------	--------	--------	--------------------	-----------------	-------	---------

P: Parse and prepare request S: Service F: Finishing R: Ready K: Keepalive

网站架构不是一天建成的，都是演化来的。不是最新最时髦的架构，而是最合适的，能驾驭的了、成本可控的架构。