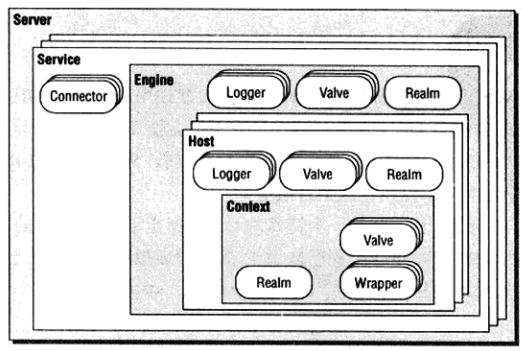
深入剖析tomcat

# tomcat构造



server：即服务器，每个tomcat程序启动后，就是一个server。

　　service：这是一种抽象的服务，通常是在日志或者管理时使用这样一个概念。它把连接器和处理引擎结合在一起。

　　connector：用于处理连接和并发，通常包括两种方式HTTP和AJP。HTTP是用于网页地址栏http这种访问方式；AJP一般用于搭配Apache服务器。

　　engine：处理引擎，所有的请求都是通过处理引擎处理的。

　　host：虚拟主机，用于进行请求的映射处理。每个虚拟主机可以看做独立的请求文件。

　　realm：用于配置安全管理角色，通常读取tomcat-uesrs.xml进行验证。

　　context：上下文，对应于web应用。

# Tomcat-安全性

## 2.1 参考博文

<http://www.cnblogs.com/xing901022/p/4552843.html>

## 2.2 什么是Realm

Realm是tomcat中用来对用户进行身份验证的组件

## 2.3 Realm的不同位置会影响到它作用的范围

1 在<Engine>元素内部 —— Realm将会被所有的虚拟主机上的web应用共享，除非它被<Host>或者<Context>元素内部的Realm元素重写。

2 在<Host>元素内部 —— 这个Realm将会被本地的虚拟主机中的所有的web应用共享，除非被<Context>元素内部的Realm元素重写。

3 在<Context>元素内部 —— 这个Realm元素仅仅被该Context指定的应用使用。

## 2.4 Realm获取用户信息方式

目前tomcat支持多种Realm管理方式，即支持多种方式来读取用户信息进行验证。参考如下：

1 JDBCRealm 用户授权信息存储于某个关系型数据库中，通过JDBC驱动获取信息验证

2 DataSourceRealm 用户授权信息存储于关于型数据中，通过JNDI配置JDBC数据源的方式获取信息验证

3 JNDIRealm 用户授权信息存储在基于LDAP的目录服务的服务器中，通过JNDI驱动获取并验证

4 UserDatabaseRealm 默认的配置方式，信息存储于XML文档中 conf/tomcat-users.xml

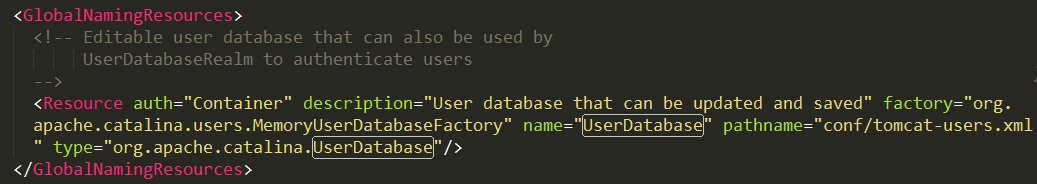
5 MemoryRealm 用户信息存储于内存的集合中，对象集合的数据来源于xml文档 conf/tomcat-users.xml

6 JAASRealm 通过JAAS框架访问授权信息

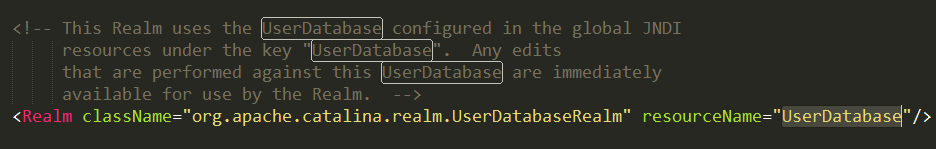
## 2.5 配置过程

**1 在server.xml中配置realm访问方式**

参考下默认的配置server.xml中，可以看到默认情况下使用的就是UserDatabaseRealm的方式：

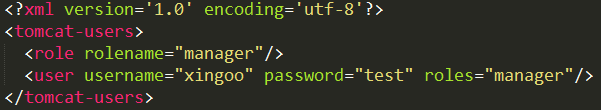


上图中的代码配置了UserDatabase的目录文件，为conf/tomcat-users.xml



上图中的代码配置使用的Realm方式。

**2 在tomcat-users.xml中配置用户密码以及分配角色**



上面是tomcat-users.xml中的配置内容。

**3 在应用的web.xml中配置其访问角色以及安全限制的内容**

关于Realm域的使用，一般都是用来管理一些安全性要求很高的应用，最常见的就是manager应用。

　　manager应用用于在不停止tomcat的情况下部署或者停止某些应用，处于安全考虑，默认情况下时不能访问manager应用的，因此需要现在tomcat-users.xml中添加用户以及相应的角色，才能访问。

参考下tomcat中manager应用的角色配置，观察其web.xml配置文件，可以找到下面这段：

<security-constraint>

<web-resource-collection>

<web-resource-name>HTMLManger and Manager command</web-resource-name>

<url-pattern>/jmxproxy/\*</url-pattern>

<url-pattern>/html/\*</url-pattern>

<url-pattern>/list</url-pattern>

<url-pattern>/sessions</url-pattern>

<url-pattern>/start</url-pattern>

<url-pattern>/stop</url-pattern>

<url-pattern>/install</url-pattern>

<url-pattern>/remove</url-pattern>

<url-pattern>/deploy</url-pattern>

<url-pattern>/undeploy</url-pattern>

<url-pattern>/reload</url-pattern>

<url-pattern>/save</url-pattern>

<url-pattern>/serverinfo</url-pattern>

<url-pattern>/status/\*</url-pattern>

<url-pattern>/roles</url-pattern>

<url-pattern>/resources</url-pattern>

</web-resource-collection>

<auth-constraint>

<!-- NOTE: This role is not present in the default users file -->

<role-name>manager</role-name>

</auth-constraint>

</security-constraint>

<!-- Define the Login Configuration for this Application -->

<login-config>

<auth-method>BASIC</auth-method>

<realm-name>Tomcat Manager Application</realm-name>

</login-config>

<!-- Security roles referenced by this web application -->

<security-role>

<description>

The role that is required to log in to the Manager Application

</description>

<role-name>manager</role-name>

</security-role>

其中role-name就定义了可以访问的角色。

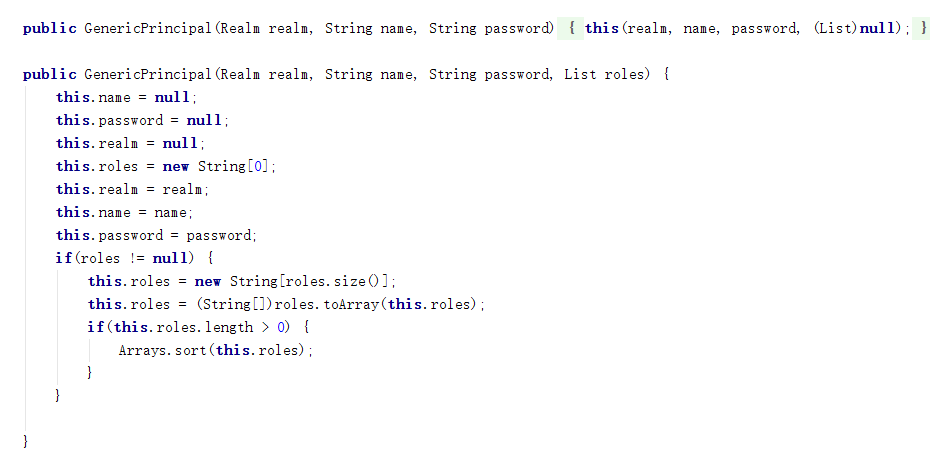
　　其他内容中上面定义了限制访问的资源，下面的Login-config比较重要。

　　它定义了验证的方式，BASIC就是基本的弹出对话框输入用户名密码。还是DIGEST方式，这种方式会对网络中的传输信息进行加密，更安全。

## 2.6 GenericPrincipal

主体对象是java.security.Principal接口的实例。

GenericPrncipal实例必须有一个用户名和密码对，此外，该用户名和密码对所对应的角色列表是可选的。



## 2.7 LoginConfig

登录配置是final型的org.apache.catalina.delopy.LoginConfig类的实例，封装了领域对象名和所要使用的身份验证方法。

可以调用LoginConfig实例的getRealmName()方法获取领域对象的名字，并调用其getAuthName()获取所用的身份验证方法的名字。

获取的身份验证方法的名字必须是以下名字之一：BASIC、DIGEST、FORM或CLIENT-CERT。

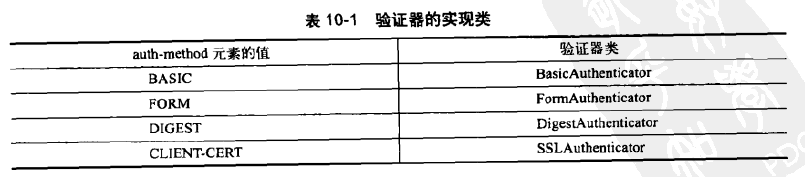
如果使用基于表单的身份验证方法，LoginConfig实例还需要在loginPage属性和errorPage属性中分别存储字符串形式的登录页面和错误页面的URL。

## 2.8 Authenticator接口

验证器是org.apache.catalina.Authenticator接口的实例。Authenticator接口本身并没有声明方法，只是起到了一个标记的作用。

Catalina提供了Authenticator接口的一个基本实现，org.apache.catalina.authenticator.AuthenticatorBase类。AuthenticatorBase还扩展了org.apache.catalina.valves.ValveBase类。所以，AuthenticatorBase也是一个阀。

使用AuthenticatorBase类的哪个子类作为Context实例中的验证阀依赖于部署描述器中的auto-method元素的值。



若没有设置auth-method元素，则LoginConfig对象的auth-method属性的值默认为NONE，这时会使用NonLoginAuthtenticator进行安全验证。