1. 安装：

下载地址：<https://tomcat.apache.org/>

下载压缩包解压即可，Src为源码

Tomcat要运行必须得安装jdk环境

1. Tomcat目录结构

　bin：目录存放一些启动运行Tomcat的可执行程序和相关内容。

　conf：存放关于Tomcat服务器的全局配置。

   lib： 目录存放Tomcat运行或者站点运行所需的jar包，所有在此Tomcat上的站点共享这些jar包。

   logs：存放日志文件

   temp: 存放临时文件

　wabapps：目录是默认的站点根目录，可以更改。

   work：目录用于在服务器运行时过度资源，简单来说，就是存储jsp、servlet翻译、编译后的结果。

1. 启动与停止

Windows: 双击bin/startup.bat启动， 双击bin/shutdown.bat停止

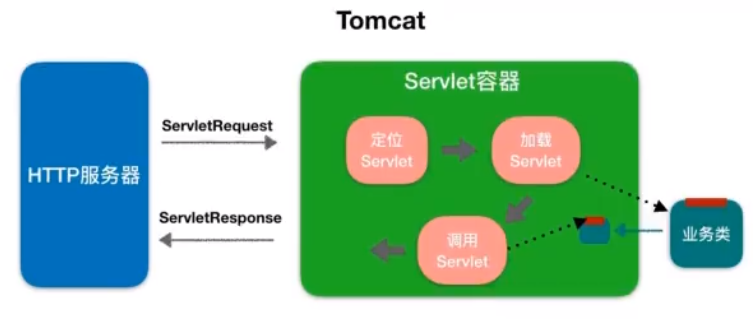
Linux：运行bin/startup.sh启动， 运行bin/shutdown.sh停止

启动之后产生的进程



# Tomcat 核心：

## Servlet容器工作流程



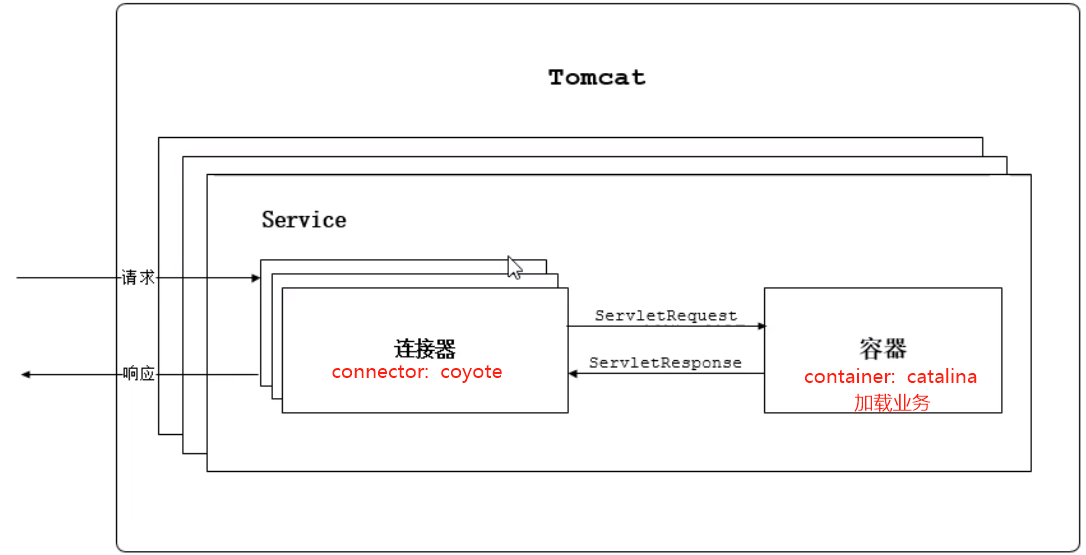
## Tomcat整体架构

我们知道如果要设计一个系统，首先是要了解需求，我们已经了解了 Tomcat要实现两个核心功能

1）处理 Socke连接，负责网络字节流与 Request和 Response对象的转化。

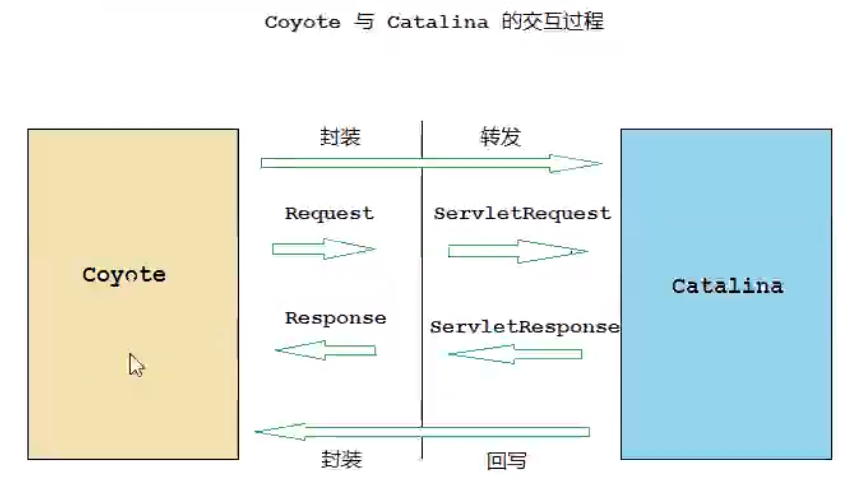
2）加载和管理servlet，以及具体处理 Request请求。

Tomcat设计了两个核心组件**连接器组件Connector（coyote实现）**和**容器组件container（catalina实现）**来分别做这两件事情。连接器（ Connector）负责对外交流，容器（ Container）负责内部处理。



## 连接器：coyote

* coyote负责协议的解析
* Coyote是 Tomcat的连接器框架的名称，是 Tomcat服务器提供的供客户端访问的外部接口。客户端通过c.yote与服务器建立连接、发送请求并接受响应
* Coyote封装了底层的网络通信（ Socket请求及响应处理），为catalina容器提供了统一的接口，使catalina容器与具体的请求协议及工操作方式完全解耦。
* Coyote将 Socket输入转换封装为 Request对象，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。
* Coyote作为独立的模块，只负责具体协议和IO的相关操作，与servlet规范实现没有直接关系，因此即便是 Request和Response对象也并未实现 Servlet规范对应的接口，而是在catalina中将他们进一步封装为ServletRequest和ServletResponse

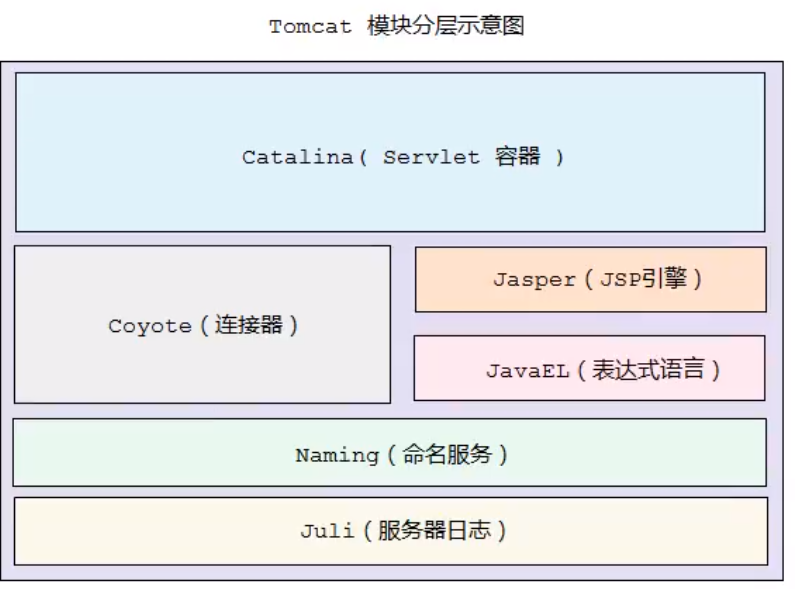
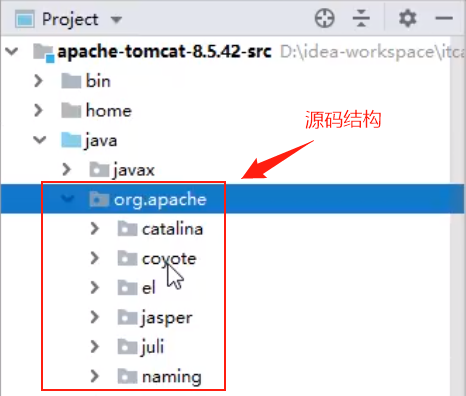


## 容器：catalina

Tomcat是一个由一系列可配置的组件构成的Web容器，而catalina是 Tomcat的servlet容器。

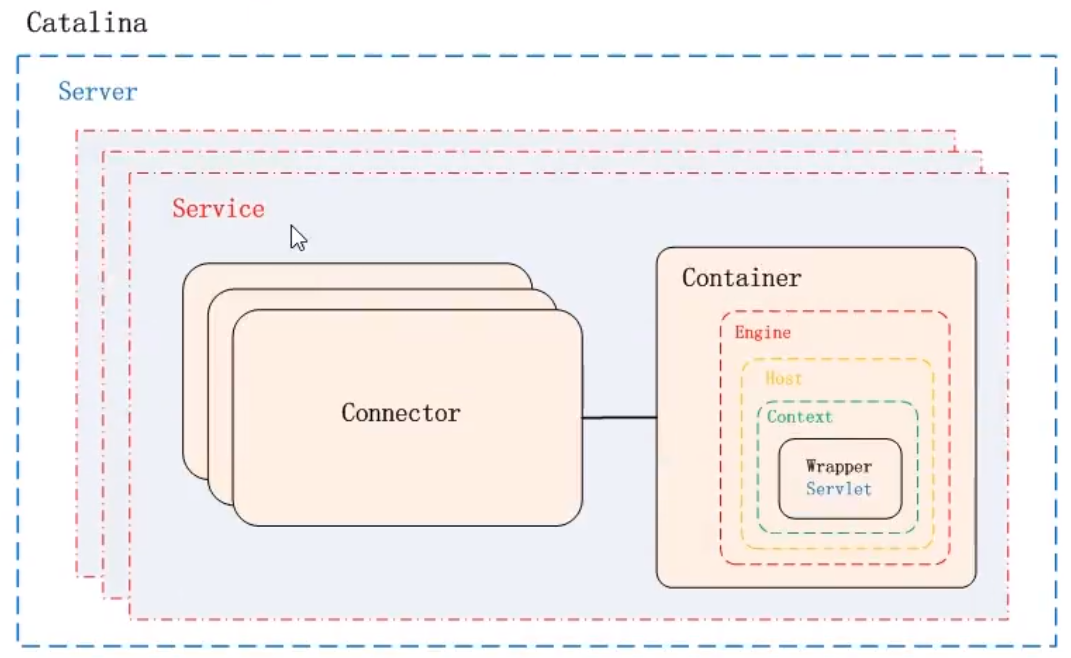
catalina是 Servlet容器实现，后续涉及到的安全、会话、集群、管理等servlet容器

架构的各个方面。它通过松耦合的式集成 Coyote，以完成按照请求协议进行数据读写。同时，它还包括我们的启动入口、shell程序等

Tomcat本质上就是一款servlet容器，因此catalina才是 Tomcat的核心，其他模块都是为catalina提供支撑的。比如通过 Coyote模块提供链接通信，Jasper模块提供Jsp引擎， Naming提供JNDI服务，Juli提供日志服务。

catalina结构



如上图所示，catalina负责管理 server，而 Server表示着整个服务器。 Server下面有多个服务 Service，每个服务都包含着多个连接器组件 Connector（ Coyote实现）和一个容器组件 Container。在Tomcat启动的时候，会初始化一个catalina的实例。



## Container结构

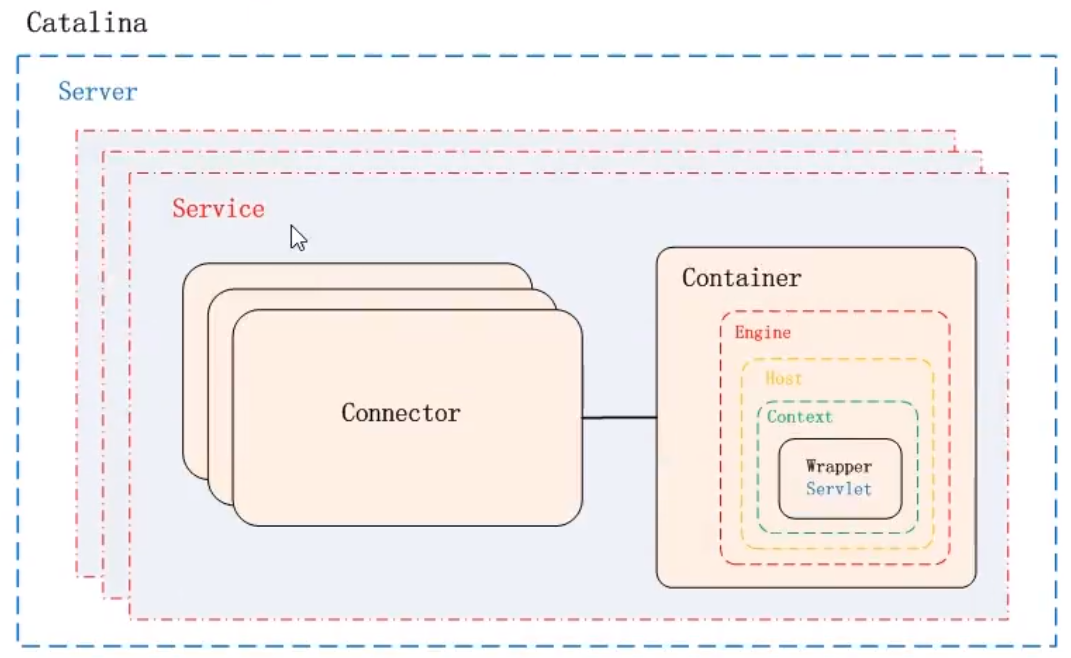
Tomcat设计了4种容器，分别是 Engine(引擎)、Host(主机)、Context(上下文应用)和Wrapper(具体的servlet)。这4种容器不是平行关系，而是父子关系。， Tomcat通过一种分层的架构，使得 Servlet容器具有很好的灵活性。



各个组件的含义

* Engine(引擎)：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 ngine，但是一个引擎可包含多个Host(多个虚拟主机)
* 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,可以包含多个servlet

<Server>



* 一个Server里面可含多个Service,
* 一个Service里面可含多个Connector（连接器）、含一个Engine(引擎)
* 一个Engine里面可含多个Host
* 一个Host里面可含多个Context
* 一个Context表示一个web应用，
* 一个Web应用包含多个wrapper(servlet)

<Service>

<Connector/>

<Connector/>

<Engine>

<Host>

<Context></Context>

<Context></Context>

</Host>

<Host>

<Context></Context>

</Host>

</Engine>

</Service>

<Service>

<Connector/>

<Connector/>

<Engine >

<Host>

<Context></Context>

</Host>

<Host>

<Context></Context>

</Host>

</Engine>

</Service>

</Server>

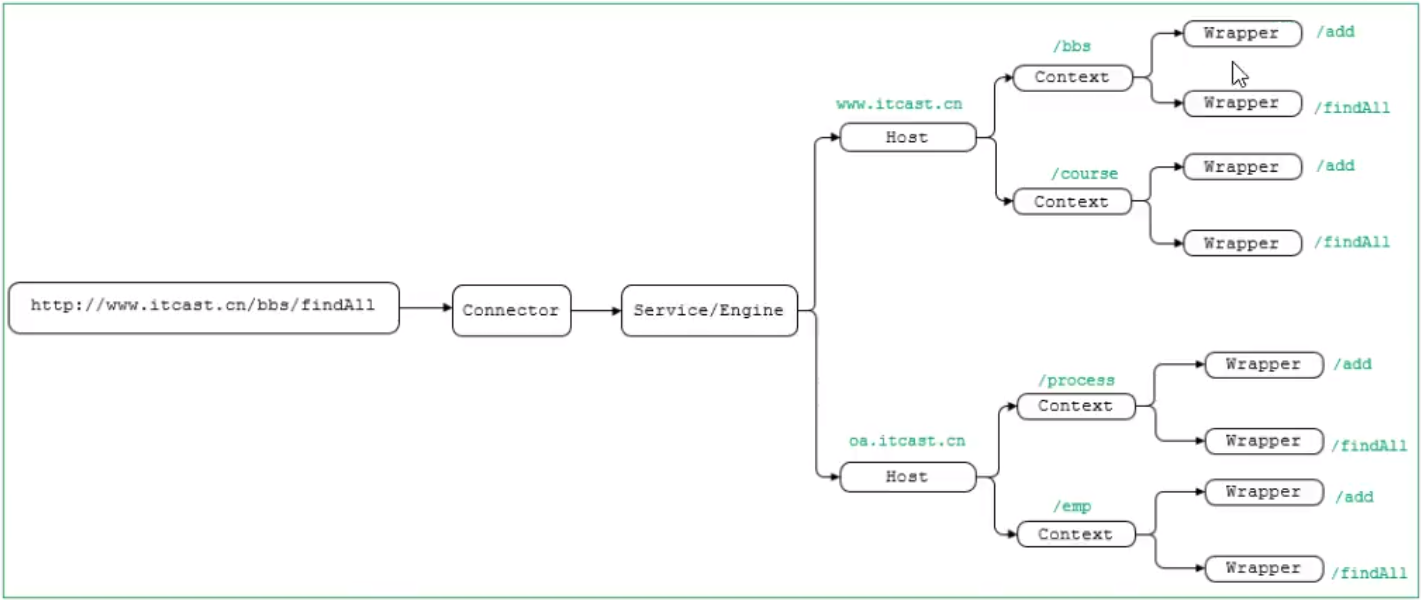
## Tomcat请求流程Mapper组件：

这么多层次的容器，Tomcat是怎么确定每一个请求应该由哪个wrapper容器里的 Servlet来处理的呢？答案是， Tomcat是用 Mapper组件来完成这个任务的。Mapper组件的功能就是将用户请求的URL定位到一个servlet。

Mapper组件工作原理是：保存了web应用的配置信息，其实就是容器组件与访问路径的映射关系，比如Hst容器里配置的域名、 Context容器里的web应用路径，以及 Wrapper容器里 Servlet映射的路径，你可以想象这些配置信息就是一个多层次的Map

当一个请求到来时， Mapper组件通过解析请求URL里的域名和路径，再到自己保存的Map里去查找，就能定位到一个servlet。请你注意，一个请求URL最后只会定位到一个 Wrapper容器，也就是一个serv1et。

下面的示意图中，就描述了当用户请求链接http://www.itcast.cn/bbs/findalt之后，是如何找到最终处理业务逻辑的servlet



Tomcat目录结构

　bin：目录存放一些启动运行Tomcat的可执行程序和相关内容。

　conf：存放关于Tomcat服务器的全局配置。

   lib： 目录存放Tomcat运行或者站点运行所需的jar包，所有在此Tomcat上的站点共享这些jar包。

   logs：存放日志文件

   temp: 存放临时文件

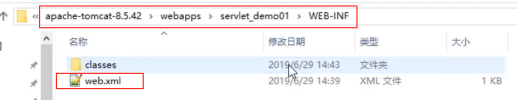
　wabapps：目录是默认的站点根目录，可以更改。

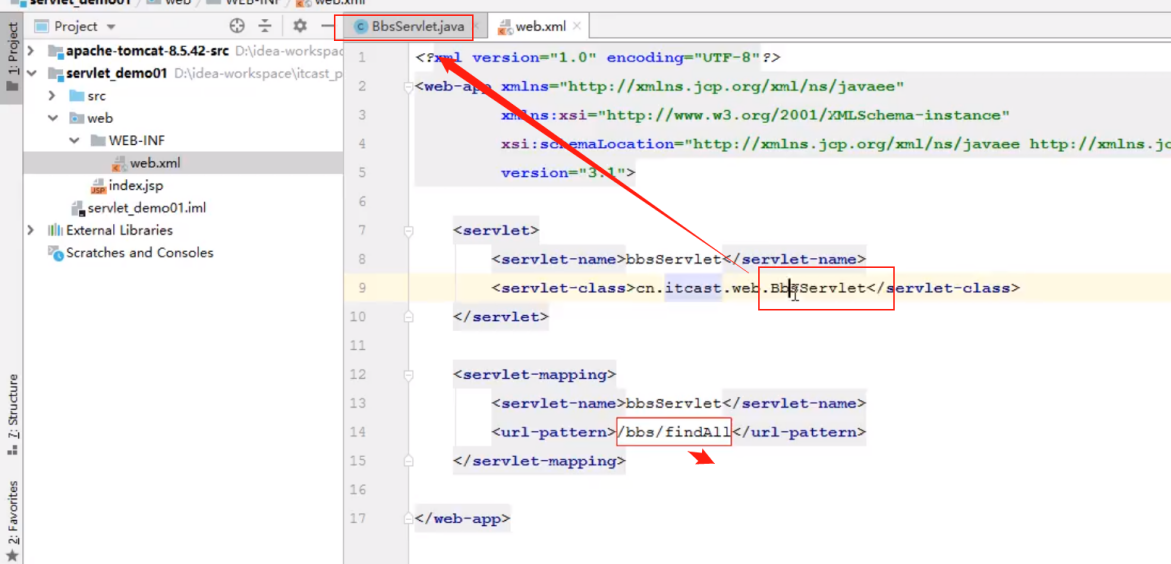
   work：目录用于在服务器运行时过度资源，简单来说，就是存储jsp、servlet翻译、编译后的结果。

## Tomcat 访问定位1：

将写好的java 工程servlet\_demo01放入tomcat的webapps目录中截图_20200306025743

文件夹Servlet\_demo01中文件内容：截图_20200306025633

文件夹WEB-INF中文件内容：

Web.xml文件内容：

Tomcat 访问定位：

<Service name=catalina>

<connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000' redirector="8443" />

<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

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

<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 &quot;%r&quot; %s &b" />

</Host>

</Engine>

</Service>

访问http://localhost:8080/servlet\_demo01/bbs/findAll:

1. 根据访问地址localhost寻找服务器主机，本案例为本主机
2. 根据端口8080找到service 和connector（连接器）,
3. 根据域名localhost寻找Engine中配置了name=”localhost”的Host配置
4. 该Host中定义了appBase=”webapps”的参数，那么该地址的应用就去webapps目录中去寻找
5. 访问http://localhost:8080/servlet\_demo01即就是访问tomcat/webapps/servlet\_demo01
   1. 前面已经将写好的java 工程servlet\_demo01放入tomcat的webapps目录中
   2. servlet\_demo01为java开发好的项目
6. 访问http://localhost:8080/servlet\_demo01/bbs/findALL，即就是访问

tomcat/webapps/servlet\_demo01/web/WEB-INF/web.xml文件中的java类，

1. Java类中再写了别的功能与页面

## Jasper

**Jasper简介:**对于基于JSP的web应用来说，我们可以直接在JSP页面中编写Java代码，添加第三方的标签库，以及使用EL表达式。但是无论经过何种形式的处理，最终输出到客户端的都是标准的HTML页面（包含js,css..），并不包含任何的java相关的语法。也就是说，我们可以把JSP看做是一种运行在服务端的脚本。那么服务器是如何将JSP页面转换为HTML页面的呢？

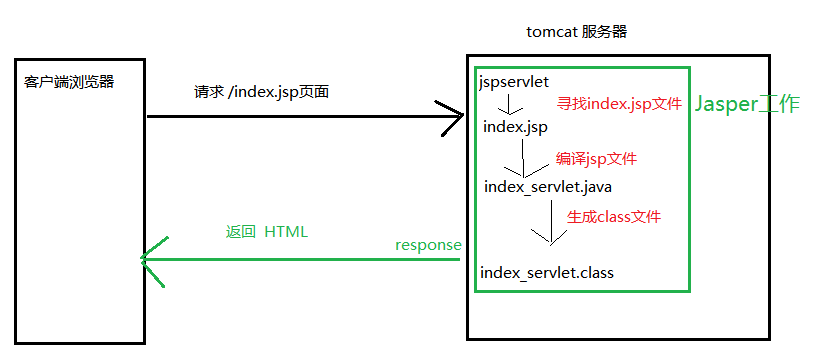
**Jasper模块：**是Tomcat的JSP核心引擎，我们知道JSP本质上是一个Servlet。Tomcat使用 Jasper对JSP语法进行解析，生成servlet并生成class字节码，用户在进行访问JSP时，会访问servlet，最终将访问的结果直接响应在浏览器端。另外，在运行的时候，Jasper还会检测JSP文件是否修改，如果修改，则会重新编译JSP文件。

**JSP（全称JavaServer Pages）：**是由Sun Microsystems公司主导创建的一种动态网页技术标准。JSP部署于网络服务器上，可以响应客户端发送的请求，并根据请求内容动态地生成HTML、XML或其他格式文档的Web网页，然后返回给请求者。JSP技术以Java语言作为脚本语言，为用户的HTTP请求提供服务，并能与服务器上的其它Java程序共同处理复杂的业务需求。

JSP将Java代码和特定变动内容嵌入到静态的页面中，实现以静态页面为模板，动态生成其中的部分内容。JSP引入了被称为“JSP动作”的XML标签，用来调用内建功能。另外，可以创建JSP标签库，然后像使用标准HTML或XML标签一样使用它们。标签库能增强功能和服务器性能，而且不受跨平台问题的限制。JSP文件在运行时会被其编译器转换成更原始的Servlet代码。JSP编译器可以把JSP文件编译成用Java代码写的Servlet，然后再由Java编译器来编译成能快速执行的二进制机器码，也可以直接编译成二进制码。

**EL（Expression Language）表达式：**是为了使JSP写起来更加简单。表达式语言的灵感来自于 ECMAScript 和 XPath 表达式语言，它提供了在 JSP 中简化表达式的方法，让Jsp的代码更加简化。

**Jasper作用**：



**运行时编译：**

Tomcat并不会在启动web应用的时候自动编译JsP文件，而是在客户端第一次请求时，才编译需要访问的Jsp文件。

**预编译**

除了运行时编译，我们还可以直接在Web应用启动时，一次性将Web应用中的所有的JSP页面一次性编译完成。在这种情况下，Web应用运行过程中，便可以不必再进行实时编译，而是直接调用JSP页面对应的 Servlet完成请求处理，从而提升系统性能

Tomcat提供了一个shell程序JspC，用于支持JSP预编译，而且在 tomcat的安装目录下提供了一个cata1ina- tasks.xml文件声明了tomcat支持的Ant任务，因此，我们很容易使用Ant来执行Jsp预编译。（要想使用这种方式，必须得确保在此之前已经下载并安装了Apache Ant）。

**编译：详细JSP编译源码解析了解就好，未深入研究**

编译结果文件所在地：

1. 如果在 tomcat/conf/web,xml中配置了参数 scratchdir，则Jsp编译后的结果，就存储在该目录下

<init-param>

<param-name>scratchdir</param-name>

<param-value>D:/tmp/isp/</param-value>

</init-param>

1. 如果没有配置该选项，则会将编译后的结果，存储在tomcat安装目录下的 work/catalina（ Engine名称）/localhost（Host名称）/ Context名称。假设项目名称为 jsp\_demo\_01，默认的目录为：work/catalina/localhost/ jsp\_demo\_01 （work目录就是存储jsp、servlet翻译、编译后的结果）
2. 如果使用的是IDEA开发工具集成Tomcat访问web工程中的Jsp，编译后的结果，存放在

C:\Users\Administrator\.IntelliJIdea2019.1\system\tomcat\\_project\_tomcat\work\Catalina\localhost\jsp\_demo\_01\_war\_exploded\org\apache\jsp

# Tomcat 服务器配置：

Tomcat服务器的配置主要集中于 tomcat/conf下的 catalina. policy、catalina.oroperties、context.xml、 server.xml、tomcat- users.xml、web.xml文件。

重点介绍： server.xml、tomcat- users.xml、web.xml三个文件

## server. xml文件配置

server.xml是 tomcat服务器的核心配置文件，包含了 Tomcat的 Servlet容器（catalina）的所有配置。由于配置的属性特别多，我们在这里主要讲解其中的部分更要配置

Server是server.xml的根元素，用于创建一个server实例，

默认使用的实现类是:org. apache.catalina.core. StandardServer。

Server内嵌的子元素为 Listener、GlobalNamingResources（全局命名服务）、Service（可多个）。

-----------------------Server.xml文件配置详情---------------------------------------

<?xml version='1.0' encoding='utf-8'?>

...

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

<!-- 8005:关闭服务器监听的端口、shutdown:关闭服务器的指令字符串。-->

### Listener配置详解

<Listener className="org.apache.catalina.startup.VersionLoggerListener" />

<!--用于以日志形式输出服务器、操作系统、JVM的版本信息-->

<Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />

<!--用于加载（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-->

<Listener className="org.apache.catalina.core.JasperListener" />

<Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />

<！--用于避免JRE内存泄漏问题-->

<Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />

<！--用户加载（服务器启动）和销毁（服务器停止）全局命名服务->

<Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

<！--用于在 Context停止时重建 Executor池中的线程，以避免ThreadLocal相关的内存泄漏->

### GlobalNamingResources配置详解

<GlobalNamingResources>

<！--中定义了全局命名服务，一般不会动-->

<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" />

<！--此处定义了用户角色与权限文件-->

</GlobalNamingResources>

### Service配置详解

<Service name="Catalina">

<Connector />

<Connector />

<Engine>

<Host>

<Context> </Context>

<Context> </Context>

</Host>

<Host>

<Context></Context>

</Host>

</Engine>

</Service>

Service：该元素用于创建 service实例，

默认使用org.apache.catalina. Core. Standardservice，默认情况下Tomcat仅指定了Service的名称，值为" catalina"。

Service可以内嵌的元素为：Listener、Executor、Connector、Engine。

Listener：用于为 service添加生命周期监听器，

Executor：用于配置 service共享线程池，

Connector：用于配置 Service包含的链接器，

Engine：用于配置 Service中链接器对应的servlet容器引擎。

#### Listener配置详解

#### Executor线程池配置详解

<!--The connectors can use a shared executor, you can define one or more named thread pools-->

<!--

<Executor name="tomcatThreadPool" namePrefix="catalina-exec-"

maxThreads="150" minSpareThreads="4"/>

-->

默认情况下，Service并未添加共享线程池配置，多个connector使用的是自己的线程池，如果添加配置添加一个线程池，各个connector使用的是这个共享的线程池

可以在< Service>下添加如下配置添加线程池

<Executor name="tomcathreacpoolt" --自定义名字

namePrefix="catalina-exec-" --线程池名字前缀

maxThreads="1000" --最大创建多少线程

minSpareThreads="100" --活跃线程数（核心线程），JDK会自动销毁线程，设置这个之后，这100个不会被销毁

maxIdleTime="60000" --最大空闲时长后销毁线程

maxQueue size="Integer. MAX\_ VALUE" --最大线程排队数量，默认为Int的最大值，一般不修改，否则会有请求不会被处理发生

prestartminspareThreads="false" --默认false即不启动，启动线程池时是否启动minspareThreads的线程池（核心线程）

threadPriority="5" --线程池中线程优先级，默认值为5，值从1到10。

className="org. apache. catalina. core. standardThreadExecutor"/>

可用JDK中jconsole.exe工具查看线程，目录：JDK/bin/jconsole.exe

#### Connector连接器配置详解

Connector用于创建链接器实例。默认情况下 server.xml配置了两个链接器，一个支持HTTP协议，一个支持AJP协议。

<Connector port="8080" protocol="HTTP/1.1" --支持http协议的connector:

connectionTimeout="20000" redirectPort="8443" URIEncoding='utf-8'/>

<Connector port="6006" protocol="AJP/1.3" redirectPort="6143" /> --支持AJP协议的connector

属性说明:

1）port：端口号,Connector用于创建服务端soocket并进行监听，以等待客户端请求链接。

2）protocol：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 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）。如果不希望采用上述自动切换的机制，而是明确指定协议，可以使用以下值。

Http协议

1、org.apache.coyote.http11.Http11Nioprotocol，非阻塞式vavaNIo链接器

2、org.apache.coyote.http11.Http11Nio2protocol，非阻塞式AVANIO2链接器

3、org.apache.coyote.http11.Http11AprProtocol, ARP链接器

AJP协议

1、org. Apache. coyote.ajp.AjpNioProtoco1，非阻塞式 Java NIo链接器

2、org. apache. coyote.ajp.AjpNio2Protoco1，非阻塞式 JAVA NIO2链接器

3、org. apache. coyote.ajp.AjpAprProtoco1，ApR链接器

3） connectionTimeout:Connector接收链接后的等待超时时间，单位为毫秒。-1表示不超时。20000为20秒

4） redirector：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，需要SSL传输，catalina自动将请求重定向到指定的端口，例如重定向到：8443

5）executor：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。

6） URIEncoding：用于指定编码UR的字符编码，Tomcat8.x版本默认的编码为UTF-8,Tomcat7.x版本默认ISO-8859-1。

<!-- A "Connector" represents an endpoint by which requests are received

and responses are returned. Documentation at :

Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)

Java AJP Connector: /docs/config/ajp.html

APR (HTTP/AJP) Connector: /docs/apr.html

Define a non-SSL HTTP/1.1 Connector on port 8080

-->

<Connector port="8080" protocol="HTTP/1.1"

connectionTimeout="20000" redirectPort="8443" URIEncoding='utf-8'/>

<!-- Define an AJP 1.3 Connector on port 8009 -->

<Connector port="6006" protocol="AJP/1.3" redirectPort="6143" />

<!-- An Engine represents the entry point (within Catalina) that processes

every request. The Engine implementation for Tomcat stand alone

analyzes the HTTP headers included with the request, and passes them

on to the appropriate Host (virtual host).

Documentation at /docs/config/engine.html -->

#### Engine引擎配置详解：

Engine作为servlet引擎的顶级元素，内部可以嵌入：cluster、Listener、Realm、Host和Valve

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

<Cluster />

<Realm />

<Host />

<Valve />

</Engine>

属性说明:

1）name：用于指定Engine的名称，默认为catalina。该名称会影响部分 Tomcat的存储路径（如临时文件）。

2）defaultHost：默认虚拟主机名，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

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

......

</Engine>

**例如：**

<Engine name="Catalina" defaultHost="www.test1.com">

<Realm className="org.apache.catalina.realm.LockOutRealm">

<Realm className="org.apache.catalina.realm.UserDatabaseRealm"

resourceName="UserDatabase"/>

</Realm>

<Host name="www.test1.com" appBase="webapps1" unpackWARs="true" autoDeploy="true">

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log" suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b" />

</Host>

<Host name="www.test2.com" appBase="webapps2" unpackWARs="true" autoDeploy="true">

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log" suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b" />

</Host>

</Engine>

**浏览器访问localhost:8080就是访问www.test1.com中的webapps1里面的内容**

**浏览器访问www.test1.com 就是访问www.test1.com中的webapps1里面的内容**

**浏览器访问www.test2.com就是访问www.test2.com中的webapps2里面的内容**

##### Cluster

<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>

-->

<!-- Use the LockOutRealm to prevent attempts to guess user passwords

via a brute-force attack -->

##### Realm

<Realm className="org.apache.catalina.realm.LockOutRealm">

<!-- This Realm uses the UserDatabase configured in the global JNDI

resources under the key "UserDatabase". Any edits

that are performed against this UserDatabase are immediately

available for use by the Realm. -->

<Realm className="org.apache.catalina.realm.UserDatabaseRealm"

resourceName="UserDatabase"/>

</Realm>

##### Host

Host元素用于配置一个虚拟主机，它支持以下嵌入元素：Alias、cluster、Listener、Valve、Realm、Context。如果在Engine下配置了Realm，那么此配置将在当前Engine下的所有Host中共享。同样，如果在Host中配置Realm，则在当前Host下的所有 Context中共享。

Context中的Realm优先级>Host的Realm优先级> Engine中的Realm优先级。

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

...

</Host>

属性说明:

1）name：当前Host通用的网络名称，必须与DNS服务器上的注册信息一致。Engine中包含的Host必须存在一个名称与 Engine的defaultHost设置一致。

2）appEase：当前Host的应用基础目录，当前Host上部署的Web应用均在该目录下（可以是绝对目录或相对路径），默认为webapps

3） unpackWARs：设置为true，Host在启动时会将appBase目录下war包解压为目录放置在同级目录。设置为false， Host将直接从war文件启动。

4）autoDeploy：控制tomcat是否在运行时定期检测并自动部署新增或变更的web应用。

5）多个虚拟主机配置：

<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 &quot;%r&quot; %s %b" />

<Context reloadable="false" debug="0" docBase="E:\FileRepository\file" path="/file"/>

</Host>

<Host name="www.a.com" appBase="webapps1" unpackWARs="true" autoDeploy="true">

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log." suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b" />

</Host>

<Host name="www.b.com" appBase="webapps2" unpackWARs="true" autoDeploy="true">

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log." suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b" />

</Host>

###### Context

Host 中的子标签 Context用于配置一个Web应用，默认的配置如下

<Context docBase="myApp" path="/myApp">

...

</Context>

属性说明：

1）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中appEase的相对路径。

2）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：http://localhost:8080/myApp

它支持的内嵌元素为：CookieProcessor, Loader, Manager, Realm, Resources, WatchedResource, JarScanner, Valve。

<hosTname=www.tomcatcomappbase=webapPs"unpackwars=true"autodeploy=true>

234567

<Context docBase="D：\servlet project03" path="/myApp"></Context>

<Valve className=org. apache. catalina. valves. AccessLogValve directory= logs

prefix="localhost access log" suffix=".txt

pattern="h 41 u it &quoti rsquoti is b"/>

</Engine>

</Service>

</Server>

## tomcat-users.xml文件配置

该配置文件中，主要配置的是 Tomcat的用户，角色等信息，

用来控制 Tomcat中 manager,host- manager的访问权限

## web.xml文件配置（web应用配置）

分

### Tomcat中的web.xml文件配置

在tomcat安装包中的conf/web.xml文件

-----------------------web.xml文件配置详情---------------------------------------

<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

version="3.0">

<servlet>

<servlet-name>default</servlet-name> ---默认sevlet配置

<servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>

<init-param>

<param-name>debug</param-name>

<param-value>0</param-value>

</init-param>

<init-param>

<param-name>listings</param-name>

<param-value>false</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

....

<servlet>

<servlet-name>jsp</servlet-name> ---默认jsp servlet的配置

<servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>

<init-param>

<param-name>fork</param-name>

<param-value>false</param-value>

</init-param>

<init-param>

<param-name>xpoweredBy</param-name>

<param-value>false</param-value>

</init-param>

<load-on-startup>3</load-on-startup>

</servlet>

.......

<servlet-mapping>

<servlet-name>default</servlet-name> --默认servlet拦截路径

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

</servlet-mapping>

....

<!-- The mappings for the JSP servlet -->

<servlet-mapping>

<servlet-name>jsp</servlet-name> --- jsp servlet的拦截路径

<url-pattern>\*.jsp</url-pattern>

<url-pattern>\*.jspx</url-pattern>

</servlet-mapping>

.....

<session-config>

<session-timeout>30</session-timeout> ---默认的seesion 超时时间30分钟

</session-config>

........

<mime-mapping> ----很多的mime-mapping映射，太多不一一列举

<extension>123</extension>

<mime-type>application/vnd.lotus-1-2-3</mime-type>

</mime-mapping>

<mime-mapping>

<extension>3dml</extension>

<mime-type>text/vnd.in3d.3dml</mime-type>

</mime-mapping>

........

<welcome-file-list>

<welcome-file>index.html</welcome-file> ---默认的欢迎页面文件

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

### 工程项目中的web.xml配置

在java开发出来的包中的WEB-INF/web.xml文件

#### Servletcontext初始化参数

我们可以通过< context- param>添加 Servletcontext初始化参数，它配置了一个键值对，这样我们可以在应用程序中使用javax. serv1et.serv1 context, getinitparameter（）方法获取参数。

配置< context- param>的参数，可在servlet中引用

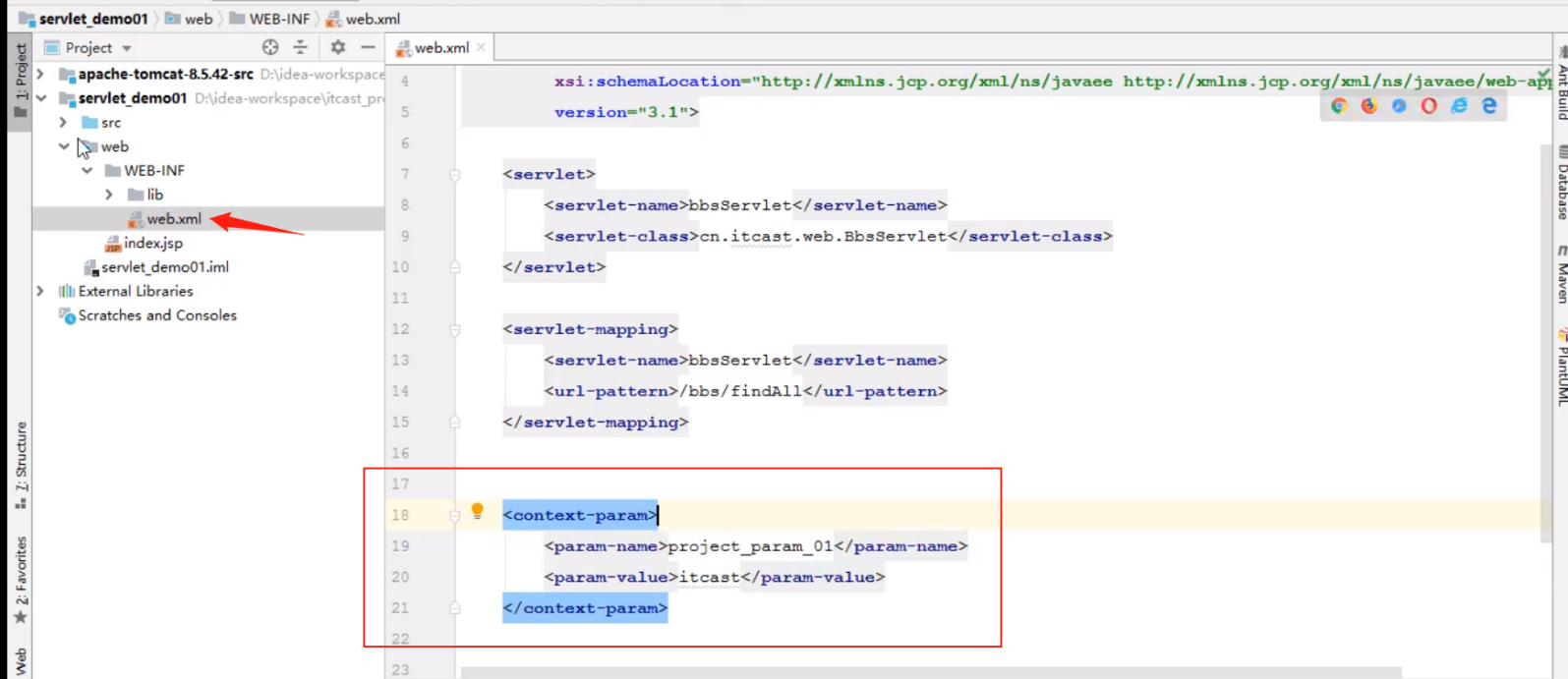
<context-paran>

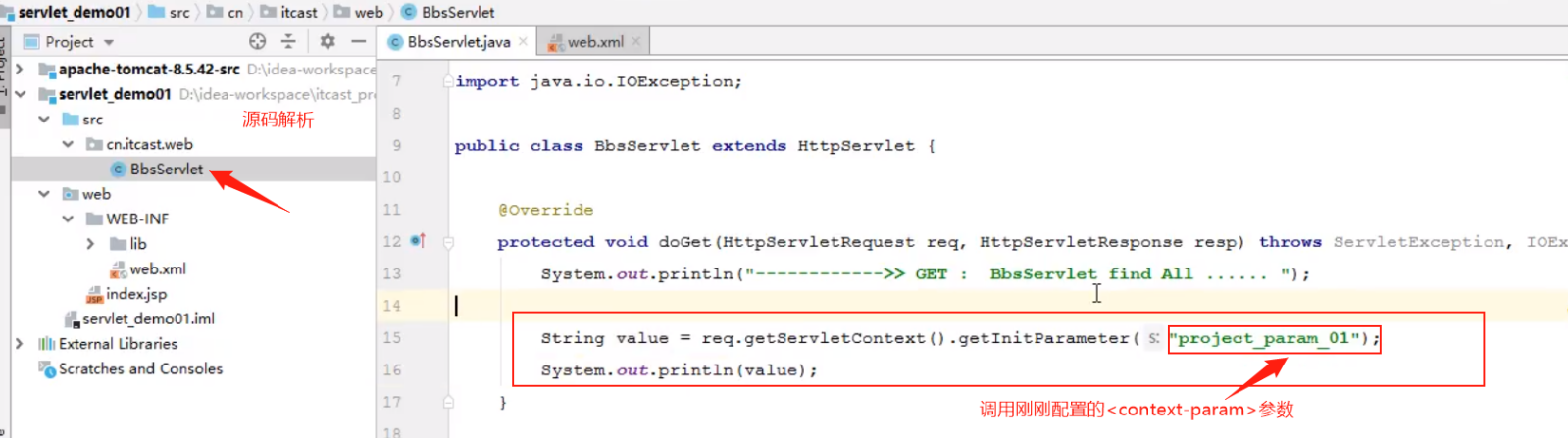
<param-name>project\_param 01</param-name>

<param-value>itcast</param-value>

</context-param>

例如：





#### 会话session-config配置

< session- config >用于配置web应用会话，包括超时时间、cookie配置以及会话追踪模式。

它将覆盖 server.xml和context.xml中的配置。

<session-config>

<session- timeout>30</ Besson- timeouT> --session有效期30分钟

<cookie-contig>

<name>JESSIONID</name> --cookie名字

<domain>www.itcast.cn</domain> --cookie域名

<path>/</path> --cookie路径

<comment>Session Cookie</comment> --注释信息

<http-only>true</http-only> --cookie对象只能通过http访问，不能通过js获取，提高安全性

<secure>false</secure> --如果为true此cookie只能通过HTTPS连接传递到服务器，而HTTP连接则不会传递该信息。注意是从浏览器传递到服务器，服务器端的Cookie对象不受此项影响

<max-age>3600</max-age> --以秒为单位表示 cookie的生存期，默认为-1表示是会话 Cookie，浏览器关闭时就会消失

</cookie-config>

<tracking-mode>COOKIE</tracking-mode> --用于配置会话追踪模式， Servlet3.0版本中支持的追踪模式：COOKIE、URL、SSL

</session-config>

支持的追踪模式：

* A. COOKIE：通过 Http Cookie追踪会话是最常用的会话追踪机制，而且servlet规范也要求所有的 Servlet规范都需要支持Cookie追踪
* B.URL:URL重写是最基本的会话追踪机制。当客户端不支持 Cookie时，可以采用URL重写的方式。当采用URL追踪模式时，请求路径需要包含会话标识信息，servlet容器会根据路径中的会话标识设置请求的会话信息。如http://www.myserver.com/user/index.html;jessionid=1234567890.
* C.SSL：对于ssl请求，通过s会话标识确定请求会话标识。

#### servlet的配置

servlet的配置主要是两部分， servlet和servlet- mapping：

<servlet>

<servlet-name>myservlet</servlet-name>

<servlet-class>cn. itcast web. MyServlet</servlet-class>

<init-param>

<param-name>fileName</param-name>

<param-value>init. conf</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

<enabled>true</enabled>

</servlet>

<servlet-mapping>

<servlet-name>myServlet</servlet-name> ---

<url-pattern>\*.do</url-pattern> ---请求路径1

<url-pattern>/myservet/\*</url-pattern> ---请求路径2，可多个

</servlet-mapping>

配置说明：

* servlet-name：指定servlet的名称，该属性在web.xml中唯一
* servlet-class：用于指定servelet类名
* init- param：用于指定servlet的初始化参数，在应用中可以通过 HttpServlet.getInitParameter获取
* load-on-startup：用于控制在web应用启动时，servelet的加载顺序，值小于0，web应用启动时，不加载该servlet，第一次访问时加载
* enabled：true / false。 若为fase，表示该servlet不处理任何请求

#### Listener配置

Listen用于监听servlet中的事件，例如 context、request、session对象的创建、修改、删除，并触发响应事件。Listener是观察者模式的实现，在servlet中主要用于对 context、request、session对象的生命周期进行监控。在servlet2.5规范中共定义了8种Listener。在启动时，servletContextListener的执行顺序与web.xm1中的配置顺序一致，停止时执行顺序相反。

<listener>

<listener-class>org. Springframework.web.context. ContextLoaderListener</listener-class>

</listener>

#### Filter配置(过滤器)

Filter用于配置web应用过滤器，用来过滤资源请求及响应，拦截用户验证访问权限。经常用于认证、日志、加密、数据转换等操作，配置如下:

<filter>

<filter-name>myFilter</filter-name>

<filter-class>cn. Itcast.web.MyFilter</filter-class>

<async-supported>true</async-supported>

<init-param>

<param-name>language</param-name>

<param-value>CN</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>myFilter</filter-name>

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

</filter-mapping>

配置说明

* filter-name：用于指走过滤器名称，在web.xml中，过滤器名称必须唯一
* filter- class：过滤器的全限定类名，该类必须实现 Filter接口
* async- supported：该过滤器是否支持异步
* init- param：用于配置 Filter的初始化参数,可以配置多个,可以通过 Filterconfig. getInitParameter获取
* url- pattern：指定该过滤器需要拦截的URL

#### 欢迎页面配置

welcome-file-list用于指定we应用的欢迎文件列表。在tomcat/config/web.xml中有配置，也可以在自己的web应用中配置，请求的顺序，从上到下。

<welcome-file-list>

<welcome-file>index. html</wel come-file>

<welcome-file>index. htm</welcome -file>

<welcome-file>index. isp</welcome-file>

</weIcome-file-list>

自定义欢迎页面：



#### 错误页面配置

Web应用中难免出现网络、BUG等问题，如果将这些问题直接暴露给用户是不合适的，

1. 用户体验性差
2. 报错给用户系统不安全，详细的类调用和使用什么开发技术都暴露

解决办法：系统如果出现错误，统一跳转到一个优美的错误页面

error-page用于配置Web应用访问异常时定向到的页面，支持HTTP响应码和异常类两种形式。

<error-page>

<error-code>404</error-code> --响应码404的跳转指定页面配置

<location>/404. html</location>

</error-page>

<error-page>

<error-code>500</error-code> --响应码500的跳转指定页面配置

<location>/500. html</location>

</error-page>

<error-page>

<exception-type>java. lang.Exception</exception-type> --根据异常信息跳转到指定页面

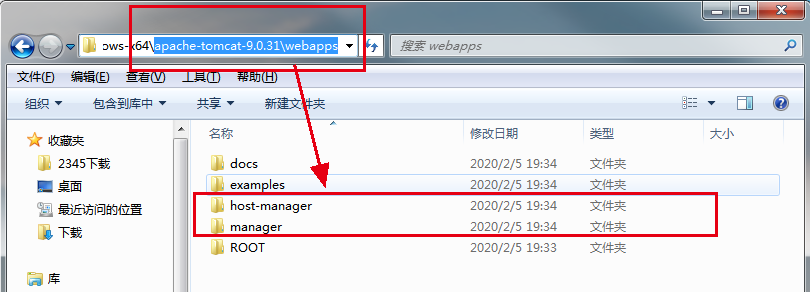
<location>/error. jsp</location>

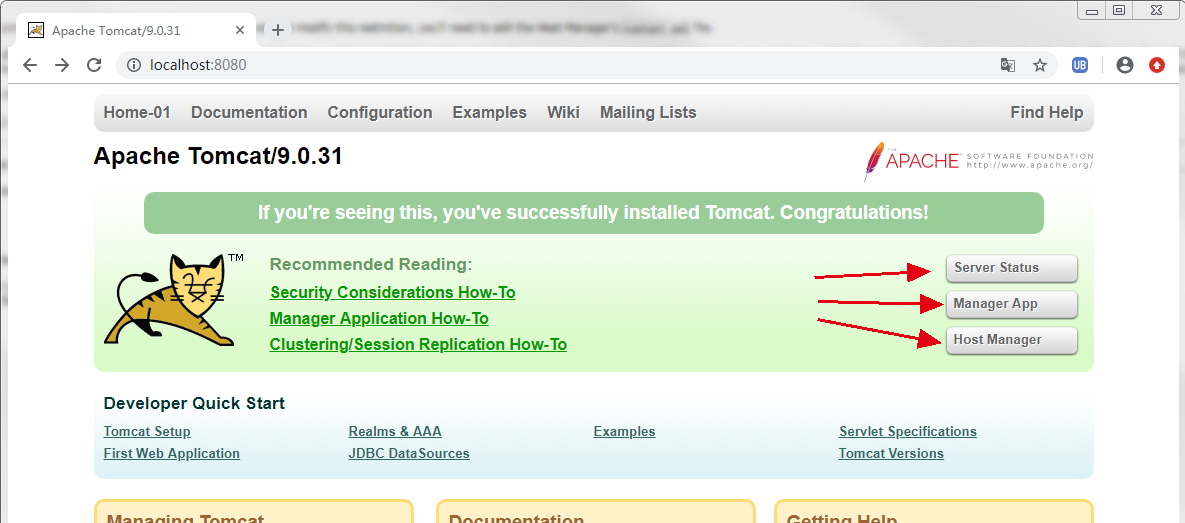
</error-page>

## Tomcat管理配置

从早期的 Tomcat版本开始，就提供了web版的管理控制台，他们是两个独立的Web应用，位于 webapps目录下。Tomcat提供的管理应用有用于管理的Host的host- manage.和用于管理Web应用的 manager。

host- manage：管理虚拟主机 manage：管理web应用





### Server-Status、Manager APP

Server-Status、Manager APP两个按钮和manager相关，具体角色名为

* manager-gui ------- allows access to the HTML GUI and the status pages

管理器gui-允许访问HTML gui和状态页

* manager-script ------ allows access to the text interface and the status pages

管理器脚本-允许访问文本界面和状态页

* manager-jmx -------- allows access to the JMX proxy and the status pages

管理器jmx-允许访问jmx代理和状态页

* manager-status ------- allows access to the status pages only

管理器状态-仅允许访问状态页

### host-manager

* Tomcat启动之后，通过http://localhost:8080/host-manager/html访问该Web应用
* host-manager默认添加了访问权限控制，当打开网址时，需要输入用户名和密码（conf/ tomcat- users.xm中配置）。
* 要想访问该页面，需要在conf/ tomcat- users,xm1中配置，并分配对应的角色
* tomcat- users,xml中配置：admin-gui：用于控制页面访问权限
* tomcat- users,xml中配置：admin- script：用于控制以简单文本的形式进行访问
* 第三个按钮和admin相关，具体角色名为
* admin-gui - allows access to the HTML GUI and the status pages

管理图形用户界面-允许访问HTML图形用户界面和状态页

* admin-script - allows access to the text interface and the status pages

管理脚本-允许访问文本界面和状态页

tomcat- users.xml原配置如下：

<!--

<role rolename="tomcat"/>

<role rolename="role1"/>

<user username="tomcat" password="<must-be-changed>" roles="tomcat"/>

<user username="both" password="<must-be-changed>" roles="tomcat,role1"/>

<user username="role1" password="<must-be-changed>" roles="role1"/>

-->

### 添加案例配置如下：

<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="admin-gui"/>

<role rolename="manager-gui"/>

<user username="role1" password="123456" roles="admin-gui,manager-gui"/>

</tomcat-users>

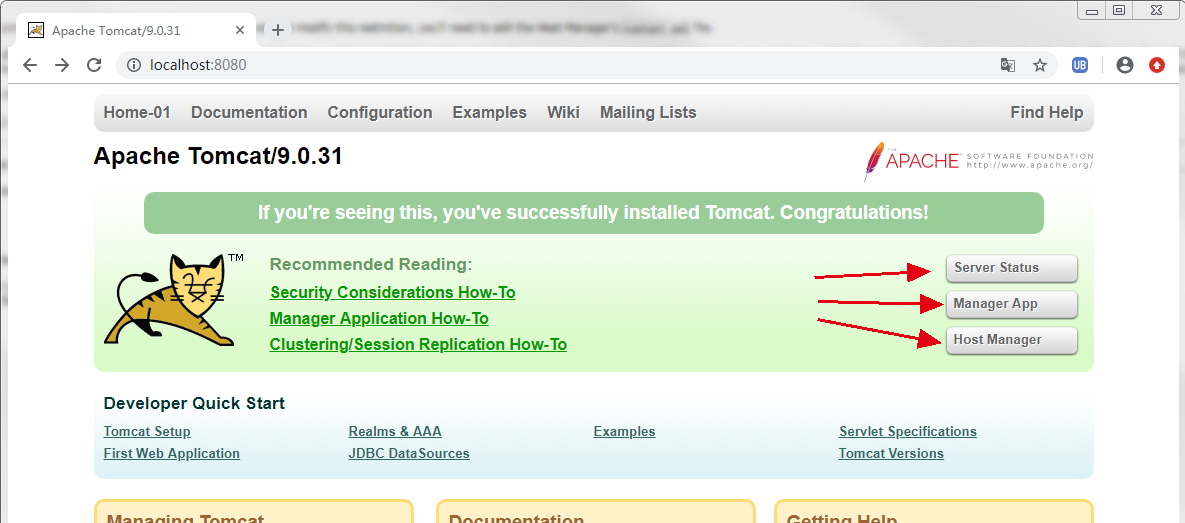
该案例配置了两个角色：

admin-gui：拥有该角色用户允许HTML页面

manager-gui：拥有该角色用户允许访问文本接口

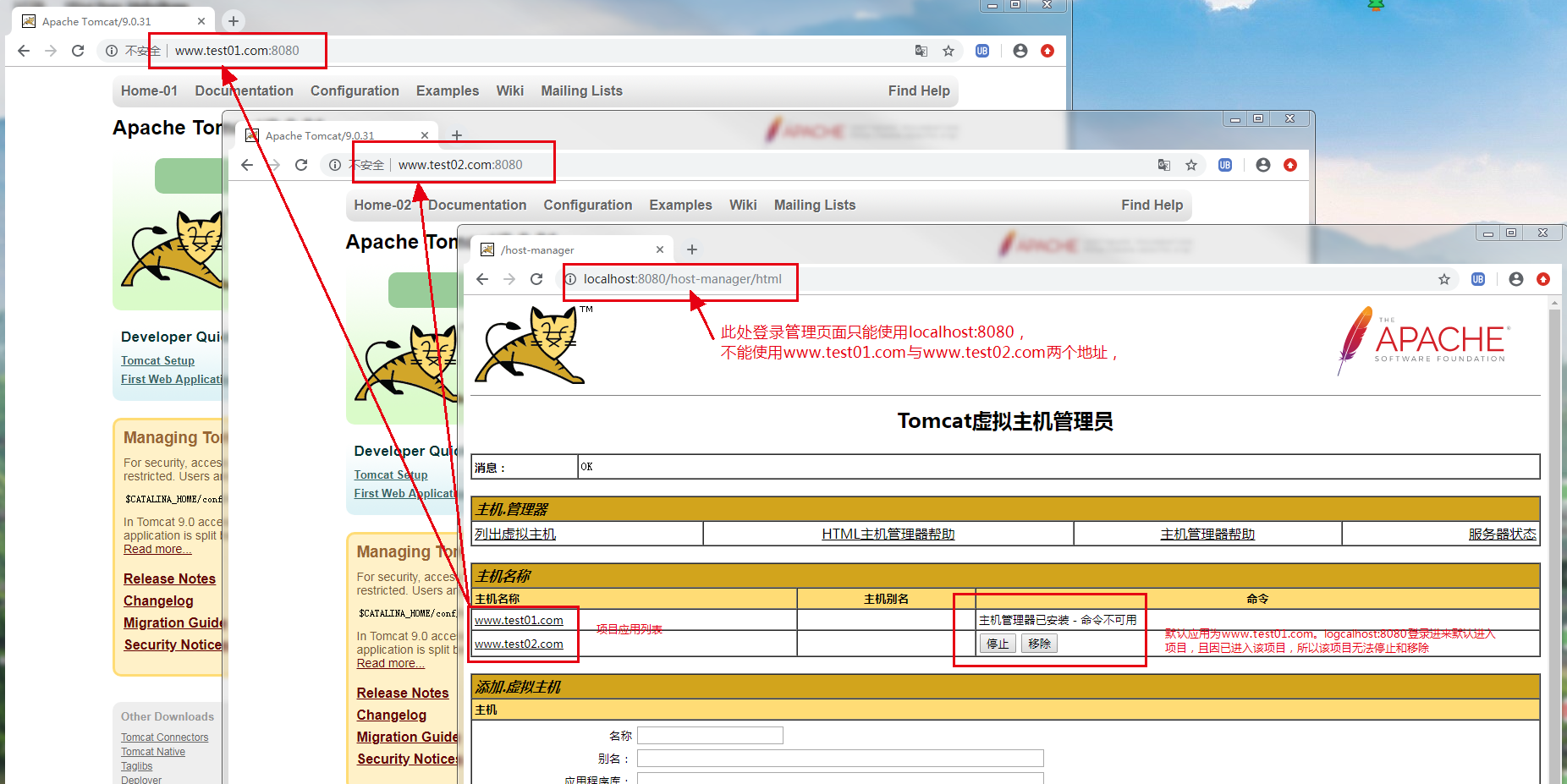
配置一个用户：role1，登录密码为：123456，拥有admin-gui和manager-gui两个角色

访问：localhost:8080点击Server-Status、Manager APP、HostManager三个按钮可



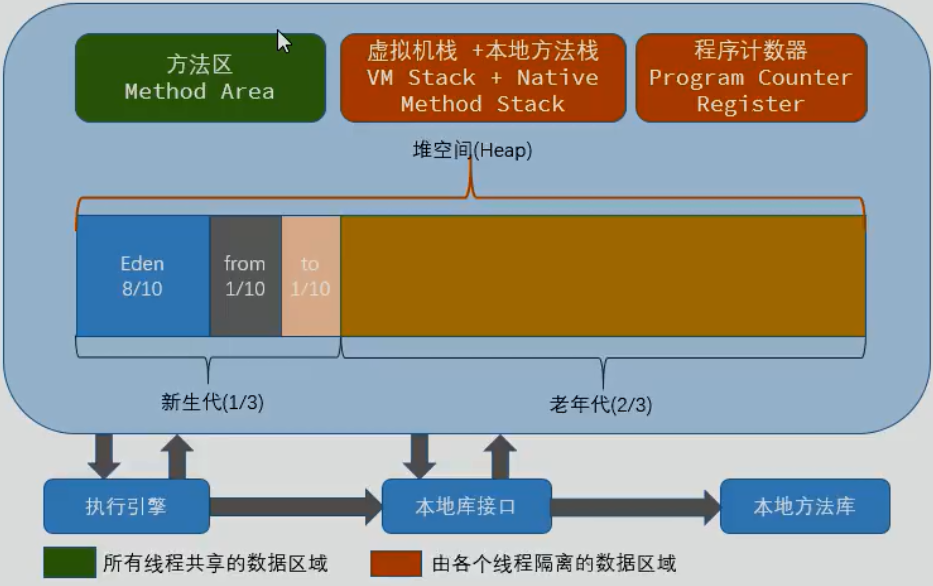
登录验证：

部署了两个应用项目www.test01.com、www.test02.com Engine的defultHost=”www.test01.com”



# JVM配置：

## JVM内存模型图



堆内存中 Eden称为伊甸园区 from与to都称为幸存区

## JVM配置选项

**winndows平台**（bin\catalina.bat）

set JAVA\_OPTS=-server -Xms2048m -Xmx2048m -XX:MetaspaceSize=256m -XX:MaxMetaspaceSize=256m -XX:SurvivorRatio=8

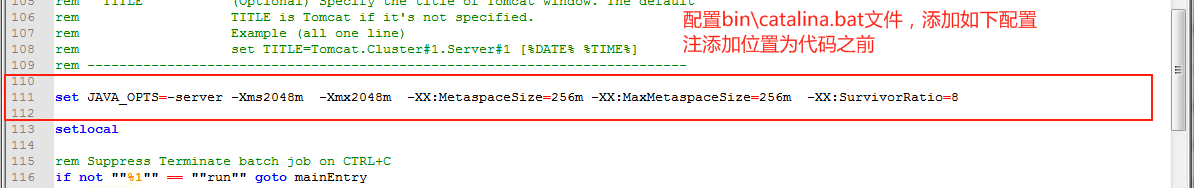
**linux平台**（bin/catalina.sh）

JAVA\_OPTS=”-server -Xms2048m -Xmx2048m -XX:MetaspaceSize=256m -XX:MaxMetaspaceSize=256m -XX:SurvivorRatio=8”

* -server：表示以服务端运行
* -Xms：堆内存的初始大小
* -Xmx：堆内存的最大大小，-Xms与-XmX一般设置一样不会开辟新的内存空间，节省资源消耗
* -Xmn：新生代的内存大小，官方建议是整个堆得3/8
* -XX:Metaspacesize元空间内存初始大小，在JDK1.8版本之前配置为-xx:Permsize（永久代）
* -XX:MaxMetaspaceSize元空间内存最大大小，在JDK1.8版本之前配置为-XX:Maxpermsize（永久代）
* -XX:Initialcodecachesize 代码缓存区大小

-XX:Reservedcodecachesize

* -XX:NewRatio设置新生代和老年代的相对大小比例。这种方式的优点是新生代大小会随着整个堆大小动态扩展。如-XX:NewRatio=3指定老年代/新生代为3/1。老年代占堆大小的3/4，新生代占1/4。
* -XX:SurvivorRatio指定伊甸园区（Eden）与幸存区大小比例。如-xx:SurvivorRatlo=8表示伊甸园区（Eden）是幸存区To大小的8倍（也是幸存区From的8倍）。所以，伊甸园区（Eden）占新生代大小的8/10，幸存区From和幸存区To每个占新生代的1/10。注意，两个幸存区永远是一样大的。（可查看上面“JVM内存模型图”）

**配置案例：**

set JAVA\_OPTS=-server -Xms2048m -Xmx2048m -XX:MetaspaceSize=256m -XX:MaxMetaspaceSize=256m -XX:SurvivorRatio=8



此图可看出：

伊甸园区PS Eden Space 大小为546.5M 68M x 8 = 544M

老年代PS Old Gen：大小为：1365.5M 1365.5 + 546.8 + 68 + 68 = 2048

幸存区PS Survivor Space：大小为68M，这里是From与To都是68M

# Tomcat集群

负载均衡

session共享

session复制

SSO单点登录

# Tomcat安全

## 配置安全

1. 删除 webapps目录下的所有文件，禁用 tomcat管理界面；
2. 注释或删除 tomcat- users.xml文件内的所有用户权限；
3. 更改关闭 tomcat指令或禁用；

tomcat的 server,xml中定义了可以直接关闭 Tomcat实例的管理端口（默认8005）。可以通过 telnet连接上该端口之后，输入SHUTDOWN（此为默认关闭指令）即可关闭 Tomcat实例（注意，此时虽然实例关闭了，但是进程还是存在的）。由于默认关闭 Tomcat的端口和指令都很简单。默认端口为8005，指令为 SHUTDOWN。

方案一更改端口号和指令：

<Server port="8456" shutdown="itcast shut">

方案二禁用8005端口：

<Server port="-1" shutdown="SHUTDOWN">

1. 定义错误页面在 webapps/R目录下定义错误页面404.htm1,500.htm1然后在 tomcat/conf/web.xm中进行配置，配置错误页面

<error-page>

<error-code>404</error-code>

<location>/404. html</location>

</error-page>

<error-page>

<error-code>500</error-code>

<location>/500.html</location>

</error-page>

这样配置之后，用户在访问资源时出现404,500这样的异常，就能看到我们自定义的错误页面，而不会看到异常的堆栈信息，提高了用户体验，也保障了服务的安全性

## 应用安全

在大部分的web应用中，特别是一些后台应用系统，都会实现自己的安全管理模块（权限模块），用于控制应用系统的安全访问，基本包含两个部分：认证（登录/单点登录）和授权（功能权限、数据权限）两个部分。

对于当前的业务系统，可以自己做一套适用于自己业务系统的权限模块，

也有很多的应用系统直接使用一些功能完善的安全框架，将其集成到我们的web应用中，如：SpringSecurity、 Apache Shiro等

## 传输安全

在HTTP的基础上加入SSL/TLS来进行数据加密，保护交换数据不被泄露、窃取。

SSL和TLS是用于网络通信安全的加密协议，它允许客户端和服务器之间通过安全链接通信。SSL协议的3个特性：

1. 保密：通过ss链接传输的数据时加密的
2. 鉴别：通信双方的身份鉴别，通常是可选的，单至少有一方需要验证
3. 完整性：传输数据的完整性检查

从性能角度考虑，加解密是一项计算昂贵的处理，因为尽量不要将整个web应用采用SSL链接，实际部署过程中，选择有必要进行安全加密的页面（存在敏感信息传输的页面）采用SSL通信

HTTPS和HTTP的区别主要为以下四点

1）HTPS协议需要到证书颁发机构CA申请SSL证书，然后与域名进行绑定，HTTP不用申请证书；

2）HTTP是超文本传输协议，属于应用层信息传输，HTTPS则是具有SSL加密传安全性传输协议，对数据的传输进行加密，相当于HTP的升级

3）HTTP和HTTPS使用的是完全不同的连接方式，用的端口也不一样，前者是8080，后者是8443

4）HTTP的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比HTTP协议安全。

HTTPS协议优势

1. 提高网站排名，有利于SEO(搜索引擎优化)。谷歌已经公开声明两个网站在搜索结果方面相同，如果一个网站启用了SSL，它可能会获得略高于没有SSL网站的等级，而且百度也表明对安装了SSL的网站表示友好。因此，网站上的内容中启用SSL都有明显的SEO优势

2）隐私信息加密，防止流量劫持。特别是涉及到隐私信息的网站，互联网大型的数据泄露的事件频频发生，网站进行信息加密势在必行。

3）浏览器受信任。自从各大主流浏览器大力支持HTTPS协议之后，访问HTTP的网站都会提示不安全的警告信息。

### Tomcat配置HTTPS：

1、本地生成秘钥库文件:

命令：keytool -genkey -alias tomcat -keyalg RSA -keystore tomcatkey.keystore

keytool：keytool命令在JDK环境中就有，如果正确的安装了JDK，配置了环境变量即可

-genkey：生成证书

-alias tomcat：别名为tomcat

-keyalg RSA：加密算法为RSA

-keystore tomcatkey.keystore：生成的秘钥文件，会在当前文件夹中出现—个秘钥库文件：tomcatkey.κeystore



2、将秘钥文件复制到tomcat/conf目录下

3、配置 tomcat/conf/server.xml

<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"

maxThreads="150" schema="https" secure="true" SSLEnabled="true">

<SSLHostConfig certificateVerification="false">

<Certificate certificateKeystoreFile="conf/tomcatkey.keystore" certificateKeystorePassword="itcast" type="RSA" />

</SSLHostConfig>

</Connector>

<Certificate certificateKeystoreFile="D:/DevelopProgramFile/apache-tomcat-85.42-windows-x64/ apache-tomcat-8.5.42/conf/tomcatkey.keystore" certificateKeystorePassword="itcast" type="RSA"/>

</SSLHostConfig>

</Connector>

配置说明：

port="8443" 端口号

protocol="org.apache.coyote.http11.Http11NioProtocol" 使用Nio模型

maxThreads="150" 线程数

schema="https" 使用的协议

secure="true" 是否安全

SSLEnabled="true" SSL证书是否开启

Certificate certificateKeystoreFile="conf/tomcatkey.keystore" 证书文件地址

certificateKeystorePassword="itcast" 证书访问密码

type="RSA" 加密算法