catalina脚本启动时设置mainclass为org.apache.catalina.startup.Bootstrap

其main方法中，如果BootSrap的单例daemon不存在，则new一个并执行init，若已存在，设置当前线程的Classloader为daemon的catalinaLoader；然后根据传入的参数——args判断是start、startd、stop、stopd、configtest命令的哪一种对daemon执行对应的操作（实际上是通过反射调用catalinaDaemon对应的方法——"start(Class),stop(Class),stopServer(Class)等等"）。

Bootstrap.init执行机制：

1. initClassLoaders()
   1. create commonLoader——common （代表加载配置中的common.loader属性）
      1. 加载配置中的common.loader获得value
      2. 对value的${catalina.home}和${catalina.base}用相应的file实例path替换，其它的${...}属性用系统属性表的属性替换
      3. 对value分隔成单个string，得到多个路径
      4. 迭代每一个string，各创建一个repository，对于\*.jar结尾的类型GLOB，.jar结尾的类型JAR，其余的类型DIR，
      5. 调用ClassLoaderFactory的createClassLoader，传入List-repository、父classLoder
         1. 迭代每一个repository，将不同类型的repository根据其location转换成URL（代表实际路径）并加入到一个URL的set集
         2. 创建一个URLClassLoader，参数为URL[]、ClassLoder(parent)
   2. create catalinaLoader——server 父loader- commonLoader
   3. create sharedLoader——shared父loader- commonLoader
2. 设置当前线程的Classloader为catalinaLoader
3. 通过反射创造类“org.apache.catalina.startup.Catalina”的实例，并调用“setParentClassLoader(java.lang.ClassLoader cl)”方法，传入参数为sharedLoader
4. 将Object类型的catalinaDaemon引用设置为上一步创造的对象。

ps:

init之后建立了三种classLoader，当类加载时其余两种的都会先委托common加载器去加载，若加载不到才到自己的路径下面去寻找

注：

CatalinaProperties可以用来加载配置属性，可以在jvm启动时用-Dcatalina.config指定文件，如不指定或者指定了不存在的，使用默认配置${catalinaBasePath}/conf/catalina.properties

默认配置中定义了，common.loader="${catalina.base}/lib","${catalina.base}/lib/\*.jar","${catalina.home}/lib","${catalina.home}/lib/\*.jar"

其他loader都为空字符串

Catalina.start启动机制

1. 如果没有Server实例，则调用load方法创建
   1. load方法实质就是按照 {catalinaBase}/conf/server.xml、classpath\*:/conf/server.xml、classpath\*:/conf/server-embed.xml顺序去找到一个xml配置，创建一个Digester解析
   2. 修改System的out和err流
   3. 调用getServer().init方法
   4. 打印初始化时间信息 Initialization processed in xxx ms
2. 检查：若扔没有Server实例，打印异常信息
3. 调用Server实例的start方法。
4. 打印启动时间信息Server startup in xxx ms
5. hook等等

注意：

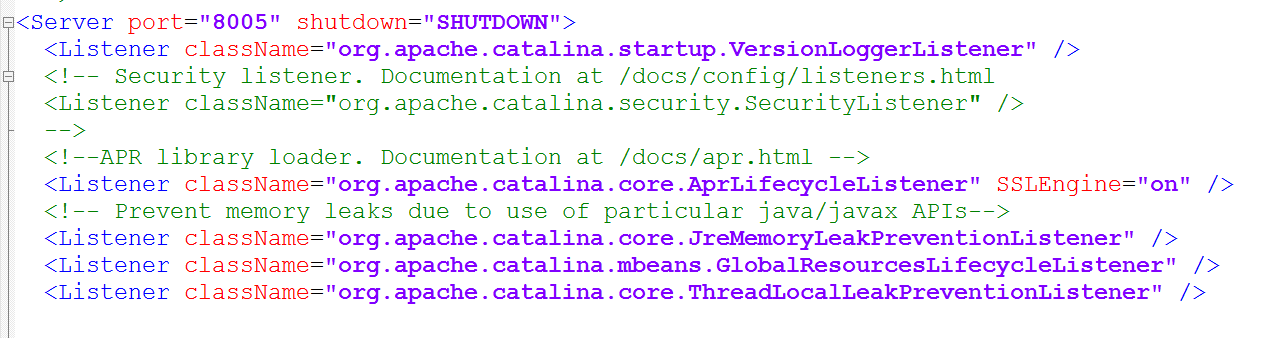
Digester创建过程中对自己进行了一系列配置，如下

Server->org.apache.catalina.core.StandardServer

Server/Service->org.apache.catalina.core.StandardService

Server/Service/Executor->org.apache.catalina.core.StandardThreadExecutor

对于listener实际类，必须在xml中声明



整的结构

Server(默认使用StandardServer，内部一个Service[])

1. Service... (默认使用StandardService，内部一个Engine, 一个Connector[])
   1. Connector...
   2. Engine (默认使用StandardEngine)
      1. Pipeline(默认使用StandardPipeline,basic设为StandardEngineValve)
      2. Host (默认使用StandardHost)
         1. Pipeline(默认使用StandardPipeline,basic设为StandardHostValve)
         2. Valve (默认使用)

接口总结:

Server:g&s port,g&s Catalina, g&s File-catalinaBase/Home, add,find,remove Service

Service:g&s container(Engine), g&s Server,g domain, find,remove Connector, add,find,remove Executor

Host:g&s xmlBase, g&s appBase,g&s autoDeploy,g&s autoDeployOnStartup,g&s deployIgnore,g&s createDirs,g&s undeployOldVersions, add,find,remove alias[]

tomcat启动jmx

set JAVA\_OPTS=%JAVA\_OPTS% -Dcom.sun.management.jmxremote.port=1090 -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false

windows系统下写到catalina.bat的第二行。1090为端口号

(linux为JAVA\_OPTS=-Dcom.sun.management.jmxremote.port=1090 -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false)

然后在VisualVM中先添加远程主机，再添加jmx连接即可。

方式二：

在

找到Execute The Requested Command

加入

CATALINA\_OPTS="$CATALINA\_OPTS -Djava.rmi.server.hostname=192.168.9.205 -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=1090 -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false"

无换行

重启

可以输入netstat -an | grep 1090 查看端口是否开启

若开启则有下图



然后连接



若连接成功，可以看到

192.168.9.205:1090