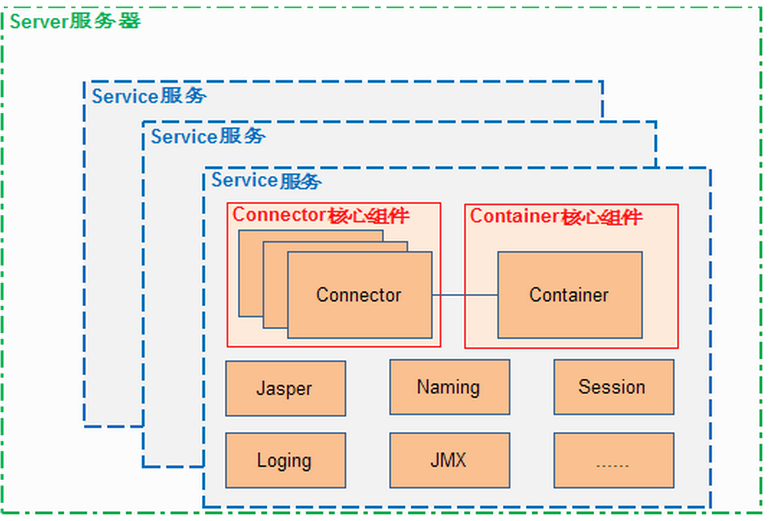
<http://blog.csdn.net/haitao111313/article/details/7717160>

<http://www.ibm.com/developerworks/cn/java/j-lo-tomcat1/>

### 服务启动

Tomcat的整体架构，Tomcat主要有两个组件，连接器和容器，所谓连接器就是一个http请求过来了，连接器负责接收这个请求，然后转发给容器。容器即servlet容器，容器有很多层，分别是Engine，Host，Context，Wrapper。最大的容器Engine，代表一个servlet引擎，接下来是Host，代表一个虚拟机，然后是Context，代表一个应用，Wrapper对应一个servlet。从连接器传过来连接后，容器便会顺序经过上面的容器，最后到达特定的servlet。要说明的是Engine，Host两种容器在不是必须的。实际上一个简单的tomcat只要连接器和容器就可以了，但tomcat的实现为了统一管理连接器和容器等组件，额外添加了服务器组件（server）和服务组件（service），添加这两个东西的原因我个人觉得就是为了方便统一管理连接器和容器等各种组件。一个server可以有多个service，一个service包含多个连接器和一个容器，当然还有一些其他的东西，看下面的图就很容易理解Tomcat的架构了：



一个父组件又可以包含多个子组件，这些被统一管理的组件都实现了Lifecycle接口。只要一个组件启动了，那么他的所有子组件也会跟着启动，比如一个server启动了，它的所有子service都会跟着启动，service启动了，它的所有连接器和容器等子组件也跟着启动了，这样，tomcat要启动，只要启动server就行了，其他的组件都会跟随着启动，那么server是如何启动的？再让我们从头来看．．．

一般启动Tomcat会是运行startup.bat或者startup.sh文件，实际上这两个文件最后会调用org.apache.catalina.startup.Bootstrap类的main方法，这个main方法主要做了两件事情，1：定义和初始化了tomcat自己的类加载器，2:通过反射调用了org.apache.catalina.startup.Catalina的process方法；关键代码如下：

ClassLoader commonLoader = null;

ClassLoader catalinaLoader = null;

ClassLoader sharedLoader = null;

..............................初始化以上三个类加载器

Class startupClass =catalinaLoader.loadClass("org.apache.catalina.startup.Catalina");

Object startupInstance = startupClass.newInstance();

......................................

methodName = "process";//方法名

paramTypes = new Class[1];

paramTypes[0] = args.getClass();

paramValues = new Object[1]

paramValues[0] = args;

method =startupInstance.getClass().getMethod(methodName, paramTypes);

method.invoke(startupInstance, paramValues);//调用process方法

org.apache.catalina.startup.Catalina的process方法很短，就是下面一点东西：

public void process(String args[]) {

setCatalinaHome();

setCatalinaBase();

try {

if (arguments(args))

execute();

} catch (Exception e) {

e.printStackTrace(System.out);

}

}

process的功能也很简单，1：如果catalina.home和catalina.base两个属性没有设置就设置一下，2：参数正确的话就调用execute方法，execute的方法就是简单的调用start方法，其中在判断参数正确的方法arguments中会设置starting标识为true，这样在execute方法中就能调用start方法，start方法是重点，在它里面启动了我们的Tomcat所有的服务，下面是start方法里面一些关键的代码：

protected void start() {

Digester digester = createStartDigester();

File file = configFile();

try {

InputSource is =

new InputSource("file://" + file.getAbsolutePath());

FileInputStream fis = new FileInputStream(file);

is.setByteStream(fis);

digester.push(this);

digester.parse(is);

fis.close();

} catch (Exception e) {

System.out.println("Catalina.start: " + e);

e.printStackTrace(System.out);

System.exit(1);

}

........................

Thread shutdownHook = new CatalinaShutdownHook();

// Start the new server

if (server instanceof Lifecycle) {

try {

server.initialize();

((Lifecycle) server).start();

try {

// Register shutdown hook

Runtime.getRuntime().addShutdownHook(shutdownHook);

} catch (Throwable t) {

// This will fail on JDK 1.2. Ignoring, as Tomcat can run

// fine without the shutdown hook.

}

// Wait for the server to be told to shut down

server.await();

} catch (LifecycleException e) {

System.out.println("Catalina.start: " + e);

e.printStackTrace(System.out);

if (e.getThrowable() != null) {

System.out.println("----- Root Cause -----");

e.getThrowable().printStackTrace(System.out);

}

}

}

}

protected Digester createStartDigester() {

// Initialize the digester

Digester digester = new Digester();

if (debug)

digester.setDebug(999);

digester.setValidating(false);

// Configure the actions we will be using

digester.addObjectCreate("Server",

"org.apache.catalina.core.StandardServer",

"className");//创建一个对象

digester.addSetProperties("Server"); //设置对象的属性

digester.addSetNext("Server",

"setServer",

"org.apache.catalina.Server");//创建对象间的关系

digester.addObjectCreate("Server/GlobalNamingResources",

"org.apache.catalina.deploy.NamingResources");

digester.addSetProperties("Server/GlobalNamingResources");

digester.addSetNext("Server/GlobalNamingResources",

"setGlobalNamingResources",

"org.apache.catalina.deploy.NamingResources");

digester.addObjectCreate("Server/Listener",

null, // MUST be specified in the element

"className");

digester.addSetProperties("Server/Listener");

digester.addSetNext("Server/Listener",

"addLifecycleListener",

"org.apache.catalina.LifecycleListener");

digester.addObjectCreate("Server/Service",

"org.apache.catalina.core.StandardService",

"className");

digester.addSetProperties("Server/Service");

digester.addSetNext("Server/Service",

"addService",

"org.apache.catalina.Service");

digester.addObjectCreate("Server/Service/Listener",

null, // MUST be specified in the element

"className");

digester.addSetProperties("Server/Service/Listener");

digester.addSetNext("Server/Service/Listener",

"addLifecycleListener",

"org.apache.catalina.LifecycleListener");

digester.addObjectCreate("Server/Service/Connector",

"org.apache.catalina.connector.http.HttpConnector",

"className");

digester.addSetProperties("Server/Service/Connector");

digester.addSetNext("Server/Service/Connector",

"addConnector",

"org.apache.catalina.Connector");

digester.addObjectCreate("Server/Service/Connector/Factory",

"org.apache.catalina.net.DefaultServerSocketFactory",

"className");

digester.addSetProperties("Server/Service/Connector/Factory");

digester.addSetNext("Server/Service/Connector/Factory",

"setFactory",

"org.apache.catalina.net.ServerSocketFactory");

digester.addObjectCreate("Server/Service/Connector/Listener",

null, // MUST be specified in the element

"className");

digester.addSetProperties("Server/Service/Connector/Listener");

digester.addSetNext("Server/Service/Connector/Listener",

"addLifecycleListener",

"org.apache.catalina.LifecycleListener");

// Add RuleSets for nested elements

digester.addRuleSet(new NamingRuleSet("Server/GlobalNamingResources/"));

digester.addRuleSet(new EngineRuleSet("Server/Service/"));

digester.addRuleSet(new HostRuleSet("Server/Service/Engine/")); //有容器和StandardService的关系

digester.addRuleSet(new ContextRuleSet("Server/Service/Engine/Default"));

digester.addRuleSet(new NamingRuleSet("Server/Service/Engine/DefaultContext/"));

digester.addRuleSet(new ContextRuleSet("Server/Service/Engine/Host/Default"));

digester.addRuleSet(new NamingRuleSet("Server/Service/Engine/Host/DefaultContext/"));

digester.addRuleSet(new ContextRuleSet("Server/Service/Engine/Host/"));

digester.addRuleSet(new NamingRuleSet("Server/Service/Engine/Host/Context/"));

digester.addRule("Server/Service/Engine",

new SetParentClassLoaderRule(digester,

parentClassLoader));

return (digester);

}

关键是红色标注的代码，Digester是一个外部jar包里面的类，主要的功能就是解析xml里面的元素并把元素生成对象，把元素的属性设置成对象的属性，并形成对象间的父子兄弟等关系。digester.addObjectCreate("Server", "org.apache.catalina.core.StandardServer", "className");//创建一个org.apache.catalina.core.StandardServer对象，实际上这里并没有真正创建出一个对象，而是添加一个模式，只是后面创建的对象是根据这些模式和server.xml来的，所以可以暂时这么理解。真正创建对象是在start方法里面的digester.parse(is),is是server.xml文件的流，digester刚才已经添加了StandardServer和StandardService等服务组件，也添加了StandardServer和StandardService的关系以及StandardService和连接器HttpConnector，容器StandardHost的关系，所以调用digester.parse(is)方法后就会根据模式和server.xml文件来生成对象以及他们之间的相互关系。这样我们便有了服务器组件StandardServer的对象，也有了它的子组件StandardService对象等等，再回到start方法，看下面的代码：

server.initialize();

((Lifecycle) server).start();

既然有了服务器组件的对象，就初始化然后启动就可以了，到此，tomcat就实现了启动服务器组件StandardServer。启动后做的事情就东西比较多，但是还是比较清晰的，StandardServer的start方法关键代码是启动它的子组件StandardService：

// Start our defined Services

synchronized (services) {

for (int i = 0; i < services.length; i++) {

if (services[i] instanceof Lifecycle)

((Lifecycle) services[i]).start();//调用StandardService的start

}

StandardService的start方法跟StandardServer的start方法差不多，是启动它的连接器和容器，上面说了一个Service包含一个容器和多个连接器：

/ Start our defined Container first

if (container != null) {

synchronized (container) {

if (container instanceof Lifecycle) {

((Lifecycle) container).start();//启动容器

}

}

}

// Start our defined Connectors second

synchronized (connectors) {

for (int i = 0; i < connectors.length; i++) {

if (connectors[i] instanceof Lifecycle)

((Lifecycle) connectors[i]).start();//启动连接器

}

}

默认的连接器是HttpConnector，所以会调用HttpConnector的start方法，它的方法如下：

public void start() throws LifecycleException {

// Validate and update our current state

if (started)

throw new LifecycleException

(sm.getString("httpConnector.alreadyStarted"));

threadName = "HttpConnector[" + port + "]";

lifecycle.fireLifecycleEvent(START\_EVENT, null);

started = true;

// Start our background thread

threadStart();//启动一个后台线程，用来处理http请求连接

// Create the specified minimum number of processors

while (curProcessors < minProcessors) {

if ((maxProcessors > 0) && (curProcessors >= maxProcessors))

break;

HttpProcessor processor = newProcessor();//后台处理连接的线程

recycle(processor);

}

}

这里有个两个关键的类：HttpConnector和HttpProcessor，它们都实现了Runnable接口，HttpConnector负责接收http请求，HttpProcessor负责处理由HttpConnector接收到的请求。注意这里HttpProcessor会有很多的实例，最大可以有maxProcessor个，初始化是20个。所以在threadStart方法中会启动一个后台线程来接收http连接，如下：

private void threadStart() {

log(sm.getString("httpConnector.starting"));

thread = new Thread(this, threadName);

thread.setDaemon(true);

thread.start();

}

这样，就会启动HttpConnector后台线程，它的run方法不断循环，主要就是新建一个ServerSocket来监听端口等待连接，主要代码如下：

socket = serverSocket.accept();

..........................

..........................

processor.assign(socket);

serverSocket一直等待连接，得到连接后给HttpProcessor的实例processor来处理，serverSocket则继续循环监听，至于processor具体怎么处理，还有很多要说，这里先不说。

至此，tomcat启动完成了，StandardServer启动后，一直到HttpConnector线程等待监听，就可以处理客户端的http请求了。