六、Tomcat启动过程

解析server.xml

1. Catalina catalina = new Catalina(); // 没做其他事情
2. catalina.setAwait(true);
3. 以下步骤是解析servler.xml
4. StandardServer server = new StandardServer(); // 没做其他事情
5. catalina.setServer(server);
6. server.addLifecycleListener(...);
7. StandardService service = new StandardService(); // 没做其他事情
8. server.addService(service);
9. Connector connector = new Connector(); // 会根据配置初始化protocolHandler
    a. endpoint = new JIoEndpoint(); // 初始化Endpoint， JioEndpoint中会setMaxConnections(0);
    b. cHandler = new Http11ConnectionHandler(this); //
    c. ((JIoEndpoint) endpoint).setHandler(cHandler); // endpoint对应的连接处理器
10. service.addConnector(connector);
11. Engine engine = new StandardEngine(); // pipeline.setBasic(new StandardEngineValve());
12. service.setContainer(engine);
13. Host host = new StandardHost(); // pipeline.setBasic(new StandardHostValve());
14. engine.addChild(host);
15. Context context = new StandardContext(); // pipeline.setBasic(new StandardContextValve());
16. host.addChild(context);

17. engine.setParentClassLoader(Catalina.class.getClassLoader()); // 实际调用的是ContainerBase.setParentClassLoader方法，设置属性parentClassLoader为shareClassLoader
18. server.setCatalina(catalina);
19. server.init(); // 开始初始化
20. catalina.start(); // 开始启动

总结

解析server.xml最主要的作用就是

1. 把server.xml中定义的节点都生成对应的java对象，比如在解析某一个Host节点时就会对应生成一个StandardHost对象
2. 把server.xml中定义的节点的层级关系解析出来，比如StandardContext对象.addChild(StandardHost对象)
3. 设置每个容器的pipeline的基础Valve

初始化

Tomcat初始化主要做了以下事情：

1. 将StandardServer实例注册到JMX
2. 将StringCache实例注册到JMX
3. 将StandardService实例注册到JMX
4. container.init(); // 对StandardEngine进行初始化
    a. 初始化startStopExecutor线程池，用来启动子容器的
5. connector.init(); // 对Connector进行初始化
    a. adapter = new CoyoteAdapter(this);
    b. protocolHandler.setAdapter(adapter);
    c. protocolHandler.init(); // 初始化协议处理器
        i. endpoint.init(); // 初始化协议处理器对应的endpoint，默认在初始化的时候就会bind
            1. endpoint.bind()
                a. serverSocketFactory = new DefaultServerSocketFactory(this);
                b. serverSocket = serverSocketFactory.createSocket(getPort(), getBacklog(), getAddress());
    d. mapperListener.init(); // 没做什么其他的

总结

初始化做得事情比较少，最重要的可能就是endpoint的bind的了

启动

1. catalina.start()
2. getServer().start();
    a. fireLifecycleEvent(CONFIGURE_START_EVENT, null);
    b. services[i].start();
        i. container.start(); // 启动StandardEngine
            1. results.add(startStopExecutor.submit(new StartChild(children[i]))); // 每个Childrean容器（StandardHost）用单独的线程启动
                a. results.add(startStopExecutor.submit(new StartChild(children[i]))); // 每个Childrean容器（StandardContext）用单独的线程启动
                    i. 以下为一个应用的启动过程
                    ii. 生成一个WebappLoader
                    iii. 启动WebappLoader
                        1. 生成WebappClassLoader
                        2. 将/WEB-INF/classes和/WEB-INF/lib目录作为loaderRepositories，后面应用如果加载类就从这两个目录加载
                    iv. fireLifecycleEvent(Lifecycle.CONFIGURE_START_EVENT, null);
                        1. 解析web.xml文件
                            a. 创建WebXml对象
                            b. 解析web.xml文件内容设置WebXml对象属性
                                i. WebXML对象有以下几个主要属性
                                ii. Map<String,ServletDef> servlets
                                iii. Map<String,String> servletMappings
                                iv. Map<String,FilterDef> filters
                                v. Set<FilterMap> filterMaps
                            c. 收集ServletContainerInitializers
                            d. 将WebXML对象中的信息配置到Context对象中
                                i. context.addFilterDef(filter);
                                ii. context.addFilterMap(filterMap);
                                iii. context.addApplicationListener(listener);
                                iv. 遍历每个ServletDef，生成一个Wrapper，context.addChild(wrapper);
                    v. 调用ServletContainerInitializers
                b. 上面会启动在server.xml中定义的Context，接下来会启动webapp文件夹下面的Context，是通过HostConfig触发的，调用HostConfig的start()
                    i. deployApps();
                        1. deployDescriptors(configBase, configBase.list()); // 描述符部署
                        2. deployWARs(appBase, filteredAppPaths); // war包部署
                        3. deployDirectories(appBase, filteredAppPaths); // 文件夹部署

                a. 生成Context对象
                b. context.setName(cn.getName());
                c. context.setPath(cn.getPath());
                d. host.addChild(context); // 这里会启动context，启动Context就会执行和上面类似的步骤
        2. threadStart(); // 启动一个background线程
    ii. executor.start(); // 启动线程池，如果用的默认连接池，这里不会启动
    iii. connector.start(); // 启动请求连接器
        1. protocolHandler.start(); // 启动接收连接
            a. endpoint.start(); // 启动Endpoint
                i. 如果没有配置Executor，就创建一个默认的Executor
                ii. 初始化connectionLimitLatch
                iii. 如果是NIO，则运行Poller线程
                iv. 运行Acceptor线程
        2. mapperListener.start();
            a. 主要初始化Mapper对象，Mapper对象的结构层级如下
                i. Mapper中有属性Host[] hosts
                ii. Host中有属性ContextList contextList
                iii. ContextList中有属性Context[] contexts
                iv. Context中有属性ContextVersion[] versions
                v. ContextVersion中有如下属性
                    1. Wrapper[] exactWrappers，保存需要根据Servlet名字精确匹配的Wrapper
                    2. Wrapper[] wildcardWrappers，保存需要根据Servlet名字匹配以("/*")结尾的Wrapper
                    3. Wrapper[] extensionWrappers，保存需要根据Servlet名字匹配以("*.")开始的Wrapper
                    4. Wrapper中有如下两个属性
                        a. name，Wrapper的名字
                        b. object，真实的Wrapper的对象
3. catalina.await(); // 使用ServerSocket来监听shutdown命令来阻塞
4. catalina.stop(); // 如果阻塞被解开，那么开始停止流程

总结

启动做的事情就比较多了，主要分为以下几大步骤

启动容器

启动容器主要是部署应用，部署应用分为两部分：

1. 部署server.xml中定义的Context
2. 部署webapp文件夹下的Context

部署一个应用主要分为以下步骤

1. 生成Context对象，server.xml中定义的Context在解析server.xml时就已经生成了，webapp文件夹下的是在部署之前生成的
2. 为每个应用生成一个WebappClassLoader
3. 解析web.xml
4. 设置Context对象中的属性，比如有哪些Wrapper

启动Connector

主要是：

1. 启动Endpoint开始接收请求
2. 构造Mapper对象，用来处理请求时，快速解析出当前请求对应哪个Context，哪个Wrapper