Tomcat 架构探索

358 查看 6 回复

(https://www.shiyanlou.com/user/8490) 实验楼管理员 (https://www.shiyanlou.com/vip) 2016-06-30 17:01 技术分享 (https://www.shiyanlou.com/questions/?area_type=sharing) 技术分享 (https://www.shiyanlou.com/questions/?tag=技术分享)

前言

花了一个礼拜的时间阅读了《how tomcat works》，本文基于此书，整理了一下Tomcat 5的基本架构，其实也没什么多复杂的东西，无非是解析Http请求，然后调用相应的Servlet。另推荐看CSAPP的网络编程那一章

全部回复

实验楼管理员 (https://www.shiyanlou.com/user/8490) (https://www.shiyanlou.com/vip)
(https://www.shiyanlou.com/user/8490)

基本架构

Tomcat由两个模块协同合作

- connector
- container

connector 负责解析处理HTTP请求，比如说请求头,查询字符串,请求参数之类的。生成HttpRequest和HttpResponse， 之后交给container，由它负责调用相应的Servlet。

Connector

Tomcat默认的Connector为HttpConnector。作为Connector必须要实现Connector这个接口。

Tomcat启动以后会开启一个线程，做一个死循环，通过ServerSocket来等待请求。一旦得到请求，生成Socket，注意这里HttpConnector并不会自己处理Socket，而是把它交给HttpProcessor。详细看下面代码，这里我只保留了关键代码。

```
public void run() {
        // Loop until we receive a shutdown command
        while (!stopped) {
            Socket socket = null;
            try {
                socket = serverSocket.accept(); //等待链接
            } catch (AccessControlException ace) {
                log("socket accept security exception", ace);
                continue;
            }
            // Hand this socket off to an appropriate processor
            HttpProcessor processor = createProcessor();
            processor.assign(socket);  //这里是立刻返回的
            // The processor will recycle itself when it finishes
        }
    }
```

注意一点，上面的 `processor.assign(socket)` ;是立刻返回的，并不会阻塞在那里等待。因为Tomcat不可能一次只能处理一个请求，所以是异步的，每个processor处理都是一个单独的线程。

HttpProcessor

上面的代码并没有显示调用HttpProcessor的process方法，那这个方法是怎么调用的呢？我们来看一下HttpProcessor的run方法。

```
public void run() {
        // Process requests until we receive a shutdown signal
        while (!stopped) {
            // Wait for the next socket to be assigned
            Socket socket = await();
            if (socket == null)
                continue;
            // Process the request from this socket
            try {
                process(socket);
            } catch (Throwable t) {
                log("process.invoke", t);
            }
            // Finish up this request
            connector.recycle(this);
        }
    }
```

我们发现他是调用await方法来阻塞等待获得socket方法。而之前Connector是调用assign分配的，这是什么原因？

2016-06-30 17:02

实验楼管理员 (https://www.shiyanlou.com/user/8490) (https://www.shiyanlou.com/vip)
(https://www.shiyanlou.com/user/8490)

下面仔细看await和assign方法。这两个方法协同合作，当assign获取socket时会通知await然后返回socket。

```
synchronized void assign(Socket socket) {
    // Wait for the Processor to get the previous Socket
    while (available) {
        try {
            wait();
        } catch (InterruptedException e) {
        }
    }
    // Store the newly available Socket and notify our thread
    this.socket = socket;
    available = true;
    notifyAll();
}
private synchronized Socket await() {
    // Wait for the Connector to provide a new Socket
    while (!available) {
        try {
            wait();
        } catch (InterruptedException e) {
        }
    }
    // Notify the Connector that we have received this Socket
    Socket socket = this.socket;
    available = false;
    notifyAll();
    return (socket);
}
```

默认available为false。

接下来就是剩下的事情就是解析请求，填充HttpRequest和HttpResponse对象，然后交给container负责。

这里我不过多赘述如何解析

```
private void process(Socket socket) {
    //parse
    ....
    connector.getContainer().invoke(request, response);
    ....
}
```

Container

> A Container is an object that can execute requests received from a client, and return responses based on those requests

Container是一个接口，实现了这个接口的类的实例，可以处理接收的请求，调用对应的Servlet。

总共有四类Container，这四个Container之间并不是平行关系，而是父子关系

- Engine - 最顶层的容器，可以包含多个Host
- Host - 代表一个虚拟主机，可以包含多个Context
- Context - 代表一个web应用，也就是ServletContext，可以包含多个Wrappers
- Wrapper - 代表一个Servlet,不能包含别的容器了，这是最底层

2016-06-30 17:02

实验楼管理员 (https://www.shiyanlou.com/user/8490) (https://www.shiyanlou.com/vip)
(https://www.shiyanlou.com/user/8490)

Container的调用

容器好比是一个加工厂，加工接受的request，加工方式和流水线也很像，但又有点区别。这里会用到一个叫做Pipeline的 东西，中文翻译为管道，request就放在管道里顺序加工，进行加工的工具叫做Valve，好比手术刀，Pipeline可添加多个Valve,最后加工的工具称为BaseValve

上面可能讲的比较抽象，接下来我们来看代码。Engine是顶层容器，所以上面invoke，执行的就是Engine的方法。StandardEngine是Engine的默认实现，注意它也同时实现了Pipeline接口，且包含了Pipeline。

它的构造方法同时指定了baseValve,也就是管道最后一个调用的Valve

```
public StandardEngine() {
    super();
    pipeline.setBasic(new StandardEngineValve());
}
```

好，接着我们看invoke,这个方法是继承自ContainerBase。只有一行，之间交给pipeline，进行加工。

```
public void invoke(Request request, Response response)
        throws IOException, ServletException {
        pipeline.invoke(request, response);
}
```

下面是StandardPipeline的invoke实现，也就是默认的pipeline实现。

```
public void invoke(Request request, Response response)
        throws IOException, ServletException {
        // Invoke the first Valve in this pipeline for this request
        (new StandardPipelineValveContext()).invokeNext(request, response);
}
```

也只有一行！调用StandardPipelineValveContext的invokeNext方法，这是一个pipeline的内部类。让我们来看 具体代码

```
public void invokeNext(Request request, Response response)
            throws IOException, ServletException {
            int subscript = stage;
            stage = stage + 1;
            // Invoke the requested Valve for the current request thread
            if (subscript < valves.length) {
                valves[subscript].invoke(request, response, this);  //加工
            } else if ((subscript == valves.length) && (basic != null)) {
                basic.invoke(request, response, this);
            } else {
                throw new ServletException
                    (sm.getString("standardPipeline.noValve"));
            }
}
```

2016-06-30 17:02

实验楼管理员 (https://www.shiyanlou.com/user/8490) (https://www.shiyanlou.com/vip)
(https://www.shiyanlou.com/user/8490)

它调用了pipeline所用的Valve来对request做加工，当Valve执行完，会调用BaseValve,也就是上面的StandardEngineValve，我们再来看看它的invoke方法

```
// Select the Host to be used for this Request
StandardEngine engine = (StandardEngine) getContainer();
Host host = (Host) engine.map(request, true);
if (host == null) {
    ((HttpServletResponse) response.getResponse()).sendError
        (HttpServletResponse.SC_BAD_REQUEST,
            sm.getString("standardEngine.noHost",
                        request.getRequest().getServerName()));
    return;
}
// Ask this Host to process this request
host.invoke(request, response);
```

它通过(Host) engine.map(request, true);获取所对应的Host,然后进入到下一层容器中继续执行。后面的执行顺序 和Engine相同，我不过多赘述

执行顺序小结

经过一长串的invoke终于讲完了第一层容器的执行顺序。估计你们看的有点晕，我这里小结一下。

> Connector -> HttpProcessor.process() -> StandardEngine.invoke() -> StandardPipeline.invoke() -> StandardPipelineValveContext.invokeNext() -> valves.invoke() -> StandardEngineValve.invoke() -> StandardHost.invoke()

到这里位置Engine这一层结束。接下来进行Host，步骤完全一致

> StandardHost.invoke() -> StandardPipeline.invoke() -> StandardPipelineValveContext.invokeNext() -> valves.invoke() -> StandardHostValve.invoke() -> StandardContext.invoke()

然后再进行Context这一层的处理，到最后选择对应的Wrapping执行。

2016-06-30 17:03

实验楼管理员 (https://www.shiyanlou.com/user/8490) (https://www.shiyanlou.com/vip)
(https://www.shiyanlou.com/user/8490)

Wrapper

Wrapper相当于一个Servlet实例，StandardContext会更根据的request来选择对应的Wrapper调用。我们直接来看看 Wrapper的basevalve是如果调用Servlet的service方法的。下面是StandardWrapperValve的invoke方法，我省略了很多， 只看关键。

```
public void invoke(Request request, Response response,
                       ValveContext valveContext)
        throws IOException, ServletException {
        // Allocate a servlet instance to process this request
        if (!unavailable) {
            servlet = wrapper.allocate();
        }
        // Create the filter chain for this request
        ApplicationFilterChain filterChain =
            createFilterChain(request, servlet);
        // Call the filter chain for this request
        // NOTE: This also calls the servlet's service() method
        String jspFile = wrapper.getJspFile();  //是否是jsp
        if (jspFile != null)
            sreq.setAttribute(Globals.JSP_FILE_ATTR, jspFile);
        else
            sreq.removeAttribute(Globals.JSP_FILE_ATTR);
        if ((servlet != null) && (filterChain != null)) {
            filterChain.doFilter(sreq, sres);
        }
        sreq.removeAttribute(Globals.JSP_FILE_ATTR);
    }
```

2016-06-30 17:03

实验楼管理员 (https://www.shiyanlou.com/user/8490) (https://www.shiyanlou.com/vip)
(https://www.shiyanlou.com/user/8490)

首先调用 `wrapper.allocate()` ,这个方法很关键，它会通过反射找到对应servlet的class文件，构造出实例返回给我们。然后创建一个FilterChain，熟悉j2ee的各位应该对这个不陌生把？这就是我们在开发web app时使用的filter。然后就执行doFilter方法了，它又会调用internalDoFilter，我们来看这个方法

```
 private void internalDoFilter(ServletRequest request, ServletResponse response)
        throws IOException, ServletException {
        // Call the next filter if there is one
        if (this.iterator.hasNext()) {
            ApplicationFilterConfig filterConfig =
              (ApplicationFilterConfig) iterator.next();
            Filter filter = null;

            filter = filterConfig.getFilter();
            filter.doFilter(request, response, this);
            return;
        }
        // We fell off the end of the chain -- call the servlet instance
        if ((request instanceof HttpServletRequest) &&
            (response instanceof HttpServletResponse)) {
            servlet.service((HttpServletRequest) request,
                            (HttpServletResponse) response);
        } else {
            servlet.service(request, response);
        }
}
```

终于，在这个方法里看到了service方法，现在你知道在使用filter的时候如果不执行doFilter，service就不会执行的原因了把。

小结

Tomcat的重要过程应该都在这里了，还值得一提的是LifeCycle接口，这里所有类几乎都实现了LifeCycle，Tomcat通过它来统一管理容器的生命流程，大量运用观察者模式。有兴趣的同学可以自己看书

Referance

How Tomcat works

转载自：一派胡言

文章地址：http://threezj.com/2016/06/25/Tomcat 架构探索 (http://threezj.com/2016/06/25/Tomcat 架构探索/)

2016-06-30 17:04

登录 (https://www.shiyanlou.com/login?next=/questions/4453)后回复帖子

我要发帖

标签

课程相关 (https://www.shiyanlou.com/questions/?tag=课程相关) Linux (https://www.shiyanlou.com/questions/?tag=Linux)
Python (https://www.shiyanlou.com/questions/?tag=Python) 实验环境 (https://www.shiyanlou.com/questions/?tag=实验环境)
C/C++ (https://www.shiyanlou.com/questions/?tag=C/C++) 技术分享 (https://www.shiyanlou.com/questions/?tag=技术分享)
课程需求 (https://www.shiyanlou.com/questions/?tag=课程需求) Java (https://www.shiyanlou.com/questions/?tag=Java)
功能建议 (https://www.shiyanlou.com/questions/?tag=功能建议) 其他 (https://www.shiyanlou.com/questions/?tag=其他)
Web (https://www.shiyanlou.com/questions/?tag=Web) Hadoop (https://www.shiyanlou.com/questions/?tag=Hadoop)
NodeJS (https://www.shiyanlou.com/questions/?tag=NodeJS) SQL (https://www.shiyanlou.com/questions/?tag=SQL)
PHP (https://www.shiyanlou.com/questions/?tag=PHP) Shell (https://www.shiyanlou.com/questions/?tag=Shell)
常见问题 (https://www.shiyanlou.com/questions/?tag=常见问题) Git (https://www.shiyanlou.com/questions/?tag=Git)
HTML (https://www.shiyanlou.com/questions/?tag=HTML) 网络 (https://www.shiyanlou.com/questions/?tag=网络)
HTML5 (https://www.shiyanlou.com/questions/?tag=HTML5) 信息安全 (https://www.shiyanlou.com/questions/?tag=信息安全)
Android (https://www.shiyanlou.com/questions/?tag=Android) NoSQL (https://www.shiyanlou.com/questions/?tag=NoSQL)
GO (https://www.shiyanlou.com/questions/?tag=GO) Ruby (https://www.shiyanlou.com/questions/?tag=Ruby)
训练营 (https://www.shiyanlou.com/questions/?tag=训练营) Perl (https://www.shiyanlou.com/questions/?tag=Perl)

(https://www.shiyanlou.com/vip)

相关帖子

Python渗透测试工具合集 (https://www.shiyanlou.com/questions/3972)

Google 和 Baidu 常用的搜索技巧 (https://www.shiyanlou.com/questions/6287)

全栈必备——Mysql性能调优 (https://www.shiyanlou.com/questions/6295)

10个你不一定知道的PHP内置函数 (https://www.shiyanlou.com/questions/6241)

不可错过的 12 款开源的 Ruby on Rails 开发工具 (https://www.shiyanlou.com/questions/6196)

手游如何实现自动化求解答 (https://www.shiyanlou.com/questions/6191)

TCP/IP、Http、Socket的区别 (https://www.shiyanlou.com/questions/2655)

JVM 各区域的用途以及潜在出现异常的示例 (https://www.shiyanlou.com/questions/6117)

12款最佳Linux命令行终端工具 (https://www.shiyanlou.com/questions/6075)

利用Linux系统生成随机密码的10种方法 (https://www.shiyanlou.com/questions/6010)

实验楼
shiyanlou.com

动手做实验，轻松学IT。

(http://weibo.com/shiyanlou2013)

公司

关于我们 (https://www.shiyanlou.com/aboutus)
联系我们 (https://www.shiyanlou.com/contact)
加入我们 (http://www.simplecloud.cn/jobs.html)
技术博客 (https://blog.shiyanlou.com/)

合作

我要投稿 (https://www.shiyanlou.com/contribute)
教师合作 (https://www.shiyanlou.com/labs)
高校合作 (https://www.shiyanlou.com/edu/)
友情链接 (https://www.shiyanlou.com/friends)

服务

实战训练营 (https://www.shiyanlou.com/bootcamp/)
会员服务 (https://www.shiyanlou.com/vip)
实验报告 (https://www.shiyanlou.com/courses/reports)
常见问题 (https://www.shiyanlou.com/questions/?tag=常见问题)
隐私条款 (https://www.shiyanlou.com/privacy)

学习路径

Python学习路径 (https://www.shiyanlou.com/paths/python)
Linux学习路径 (https://www.shiyanlou.com/paths/linuxdev)
大数据学习路径 (https://www.shiyanlou.com/paths/bigdata)
Java学习路径 (https://www.shiyanlou.com/paths/java)
PHP学习路径 (https://www.shiyanlou.com/paths/php)
全部 (https://www.shiyanlou.com/paths/)