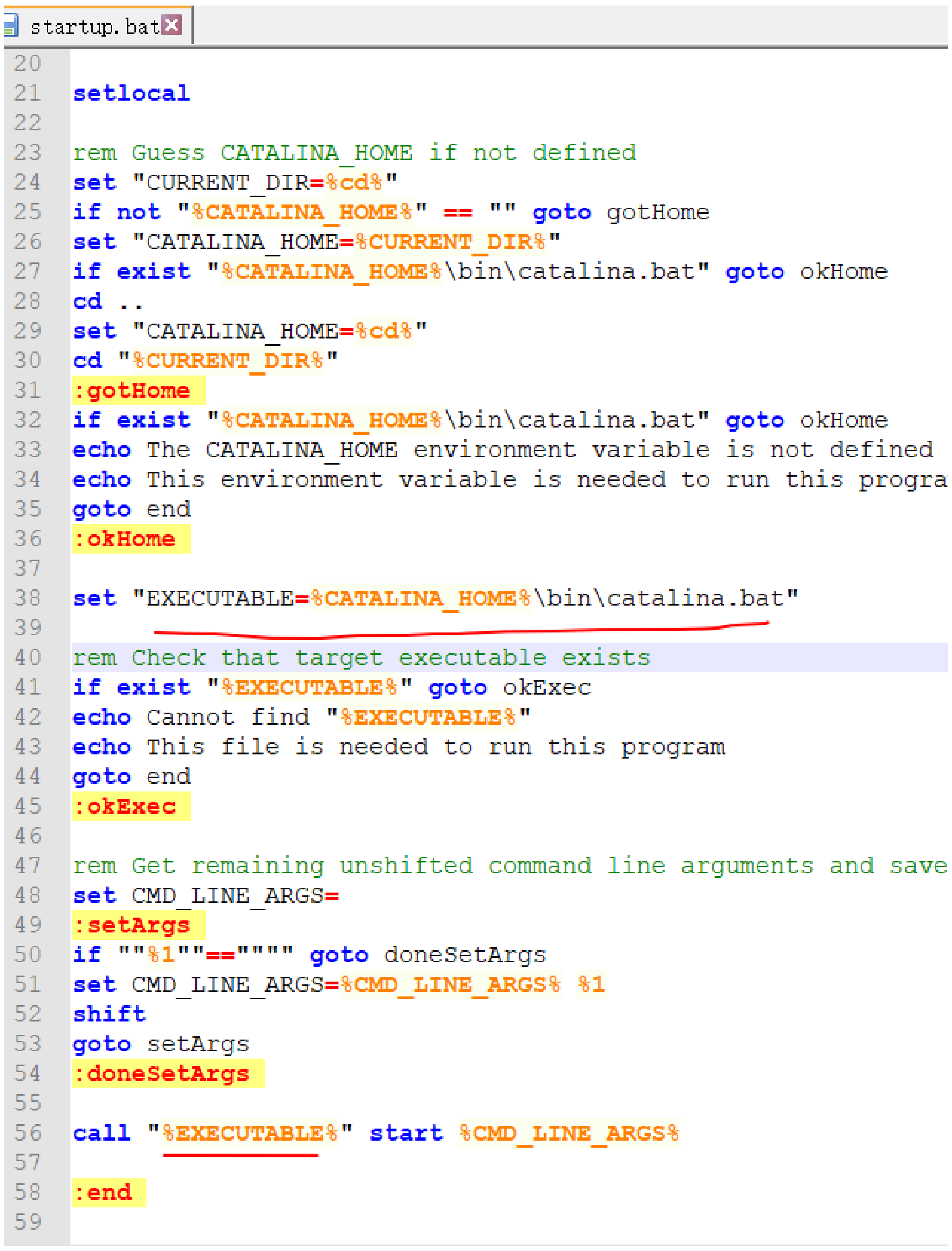
tomcat源码剖析

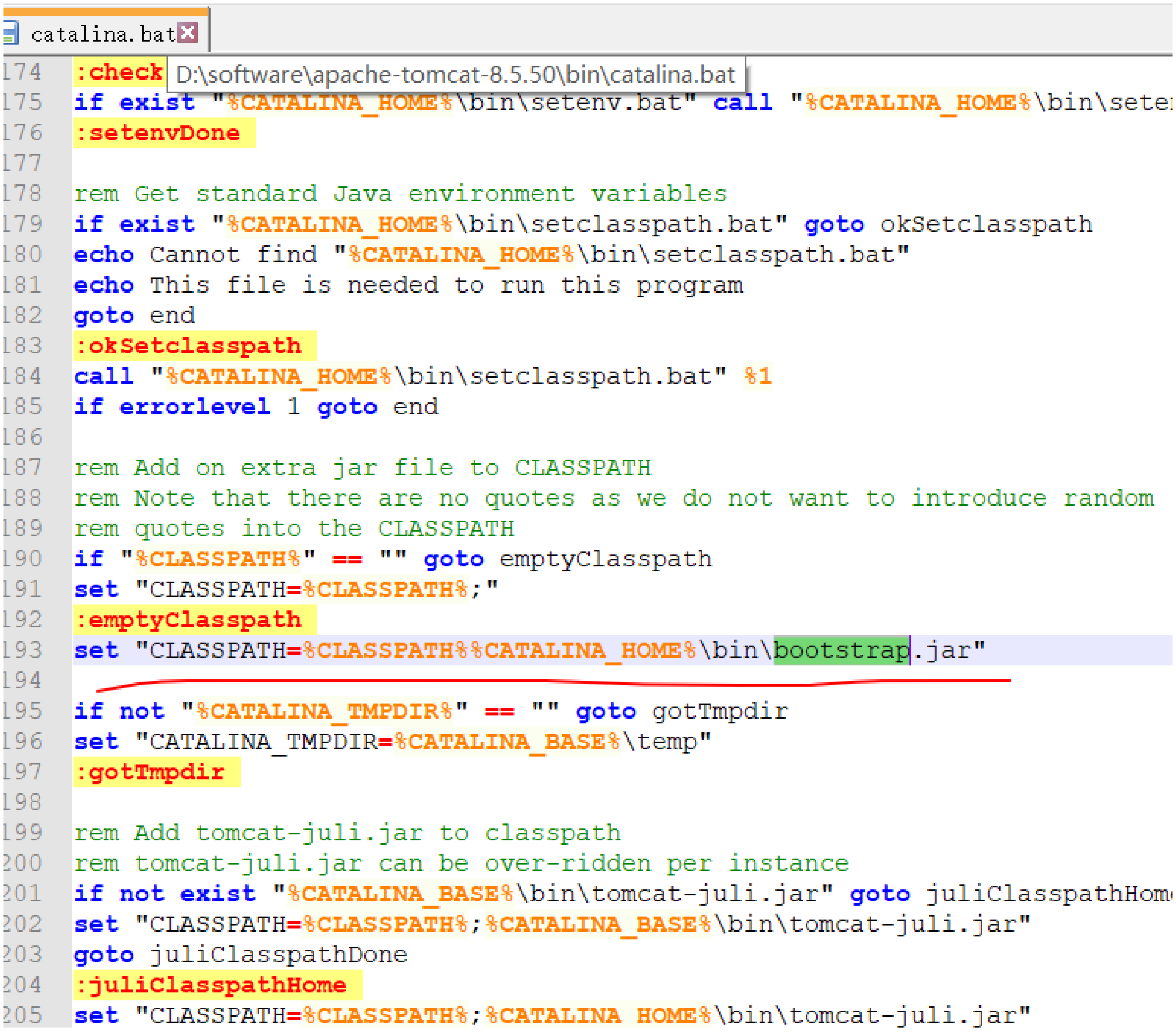
Tomcat启动流程:

tomcat启动器调用catalina.bat


startup.bat

```
setlocal

rem Guess CATALINA_HOME if not defined
set "CURRENT_DIR=%cd%"
if not "%CATALINA_HOME%" == "" goto gotHome
set "CATALINA_HOME=%CURRENT_DIR%"
if exist "%CATALINA_HOME%\bin\catalina.bat" goto okHome
cd ..
set "CATALINA_HOME=%cd%"
cd "%CURRENT_DIR%"
:gotHome
if exist "%CATALINA_HOME%\bin\catalina.bat" goto okHome
echo The CATALINA_HOME environment variable is not defined
echo This environment variable is needed to run this progra
goto end
:okHome

set "EXECUTABLE=%CATALINA_HOME%\bin\catalina.bat"

rem Check that target executable exists
if exist "%EXECUTABLE%" goto okExec
echo Cannot find "%EXECUTABLE%"
echo This file is needed to run this program
goto end
:okExec

rem Get remaining unshifted command line arguments and save
set CMD_LINE_ARGS=
:setArgs
if ""%1""=="""" goto doneSetArgs
set CMD_LINE_ARGS=%CMD_LINE_ARGS% %1
shift
goto setArgs
:doneSetArgs

call "%EXECUTABLE%" start %CMD_LINE_ARGS%

:end

```

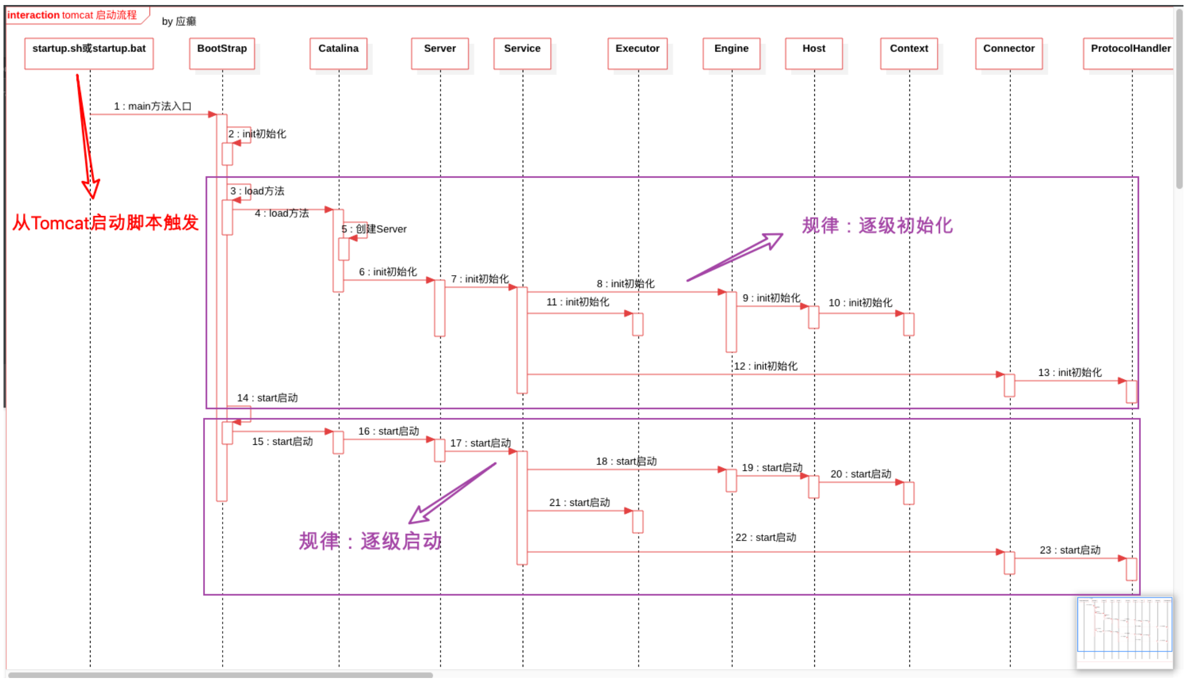
catalina.bat调用bootstrap



catalina.bat

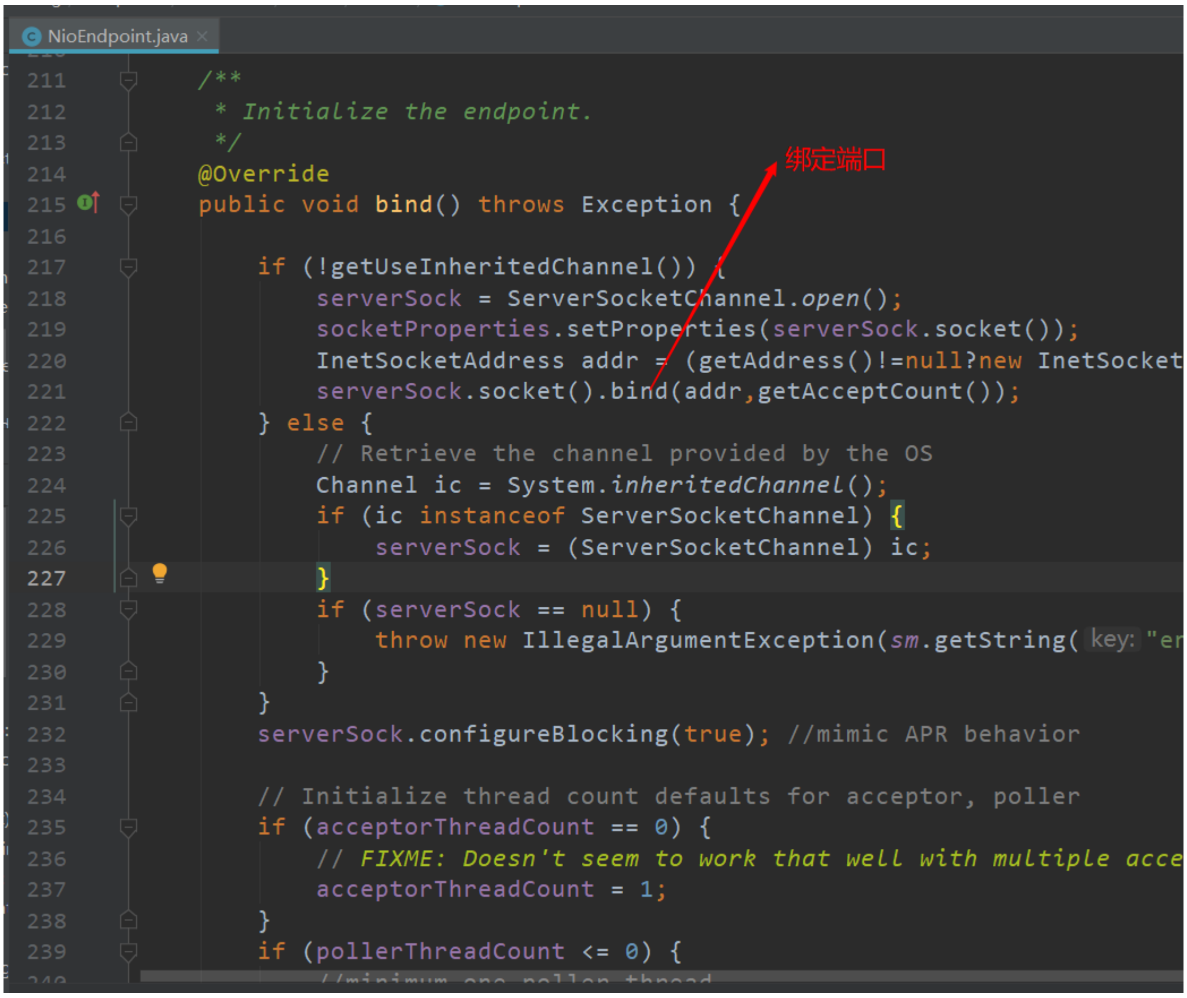
D:\software\apache-tomcat-8.5.50\bin\catalina.bat

```
:check
if exist "%CATALINA_HOME%\bin\setenv.bat" call "%CATALINA_HOME%\bin\sete
:setenvDone

rem Get standard Java environment variables
if exist "%CATALINA_HOME%\bin\setclasspath.bat" goto okSetclasspath
echo Cannot find "%CATALINA_HOME%\bin\setclasspath.bat"
echo This file is needed to run this program
goto end
:okSetclasspath
call "%CATALINA_HOME%\bin\setclasspath.bat" %1
if errorlevel 1 goto end

rem Add on extra jar file to CLASSPATH
rem Note that there are no quotes as we do not want to introduce random
rem quotes into the CLASSPATH
if "%CLASSPATH%" == "" goto emptyClasspath
set "CLASSPATH=%CLASSPATH%;"
:emptyClasspath
set "CLASSPATH=%CLASSPATH%%CATALINA_HOME%\bin\bootstrap.jar"

if not "%CATALINA_TMPDIR%" == "" goto gotTmpdir
set "CATALINA_TMPDIR=%CATALINA_BASE%\temp"
:gotTmpdir

rem Add tomcat-juli.jar to classpath
rem tomcat-juli.jar can be over-ridden per instance
if not exist "%CATALINA_BASE%\bin\tomcat-juli.jar" goto juliClasspathHom
set "CLASSPATH=%CLASSPATH%;%CATALINA_BASE%\bin\tomcat-juli.jar"
goto juliClasspathDone
:juliClasspathHome
set "CLASSPATH=%CLASSPATH%;%CATALINA_HOME%\bin\tomcat-juli.jar"
```



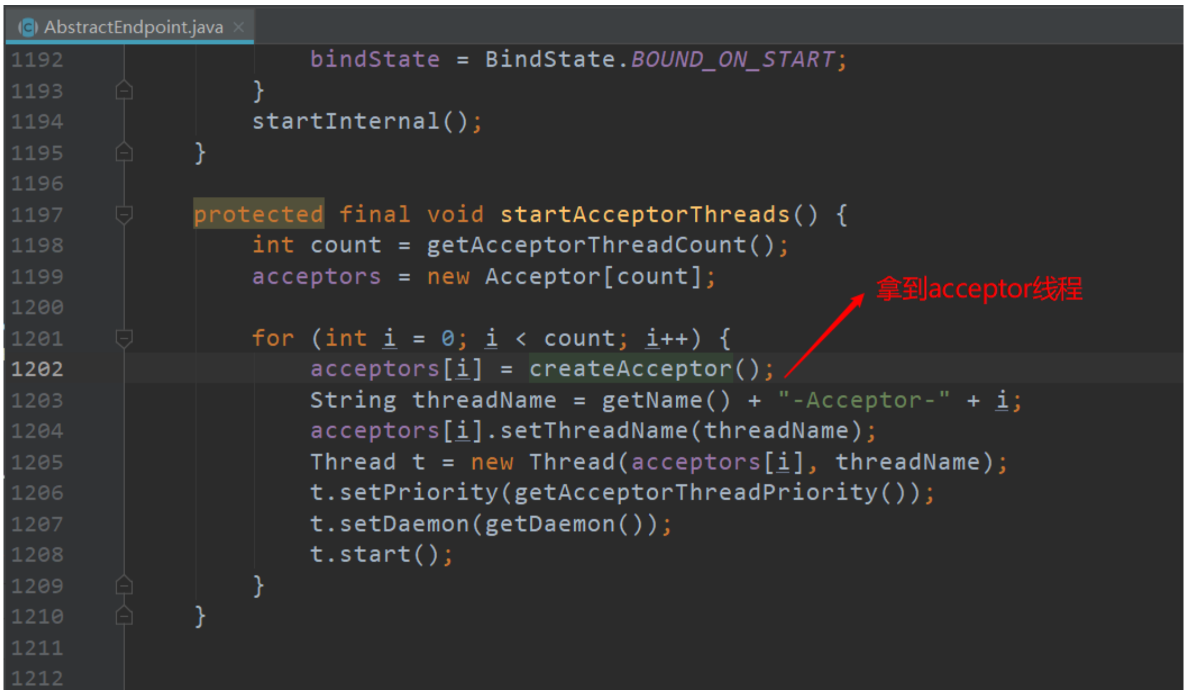
- **Executor：**共享线程池。
- **protocolhandler：**进行socket处理
- 2：创建Catalina实例
- 4：Catalina的load( )调用**createStartDigester( )：xml配置文件的解析器：如server.xml**
- 6：调用LifecycleBase的initInternal( )使用**模板方法模式。**（后面也有用到该模式）

- 12：Connector类使用Adapter adapter = new CoyoteAdapter(this);将Request转为ServletRequest
- 13：protocolhandler实现类AbstractProtocol调用endpoint.init( )完成socket通信。（其中bind( )实现采用i/o模型是Nio，如图1.1）
- 23：protocolhandler启动start方法时调用startAcceptorThreads( )（如图2.1.2）,启动Acceptor线程（如图2.1.3）



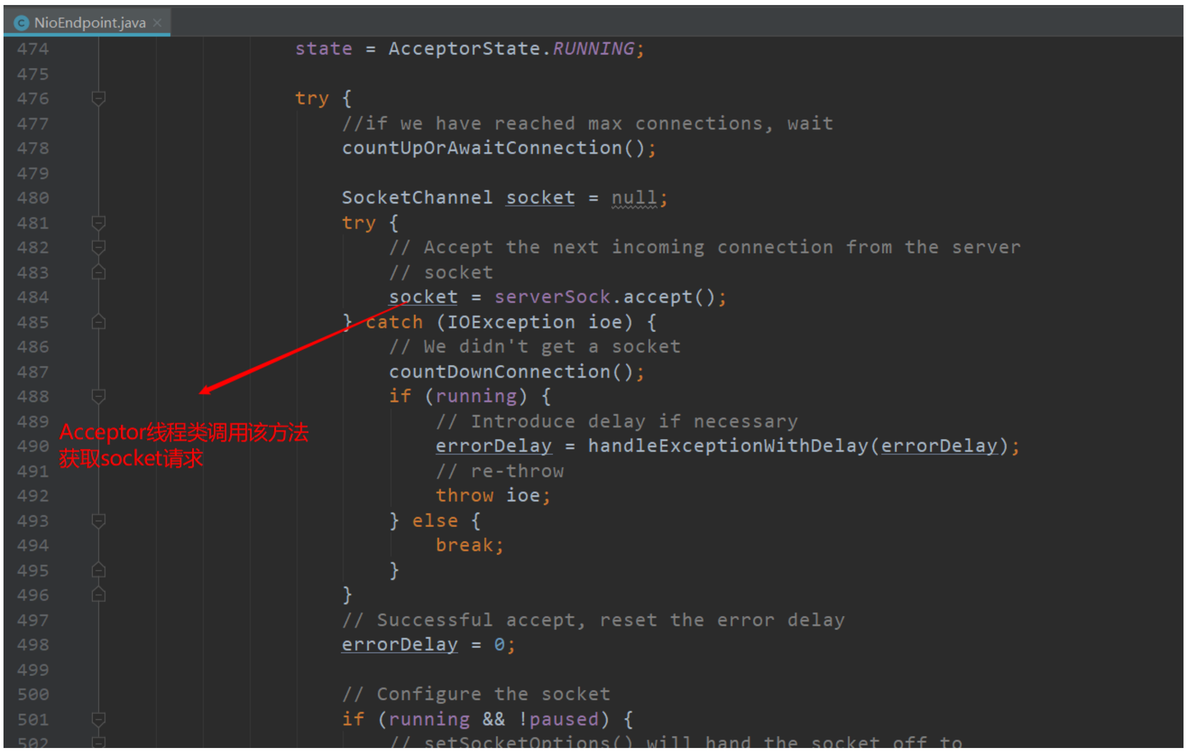
```
    /**
     * Initialize the endpoint.
     */
    @Override
    public void bind() throws Exception {

        if (!getUseInheritedChannel()) {
            serverSock = ServerSocketChannel.open();
            socketProperties.setProperties(serverSock.socket());
            InetSocketAddress addr = (getAddress()!=null?new InetSocket
            serverSock.socket().bind(addr,getAcceptCount());
        } else {
            // Retrieve the channel provided by the OS
            Channel ic = System.inheritedChannel();
            if (ic instanceof ServerSocketChannel) {
                serverSock = (ServerSocketChannel) ic;
            }
            if (serverSock == null) {
                throw new IllegalArgumentException(sm.getString( key: "er
            }
        }
        serverSock.configureBlocking(true); //mimic APR behavior

        // Initialize thread count defaults for acceptor, poller
        if (acceptorThreadCount == 0) {
            // FIXME: Doesn't seem to work that well with multiple acce
            acceptorThreadCount = 1;
        }
        if (pollerThreadCount <= 0) {
            //minimum one poller thread
```

2.1.1



```
                bindState = BindState.BOUND_ON_START;
            }
            startInternal();
        }

        protected final void startAcceptorThreads() {
            int count = getAcceptorThreadCount();
            acceptors = new Acceptor[count];

            for (int i = 0; i < count; i++) {
                acceptors[i] = createAcceptor();
                String threadName = getName() + "-Acceptor-" + i;
                acceptors[i].setThreadName(threadName);
                Thread t = new Thread(acceptors[i], threadName);
                t.setPriority(getAcceptorThreadPriority());
                t.setDaemon(getDaemon());
                t.start();
            }
        }
```

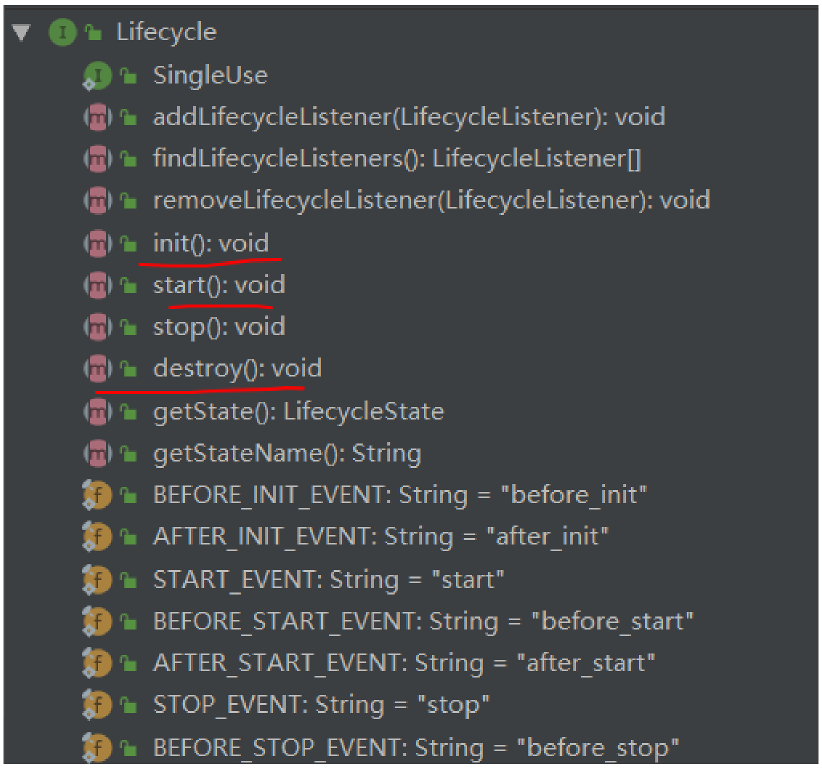
2.1.2

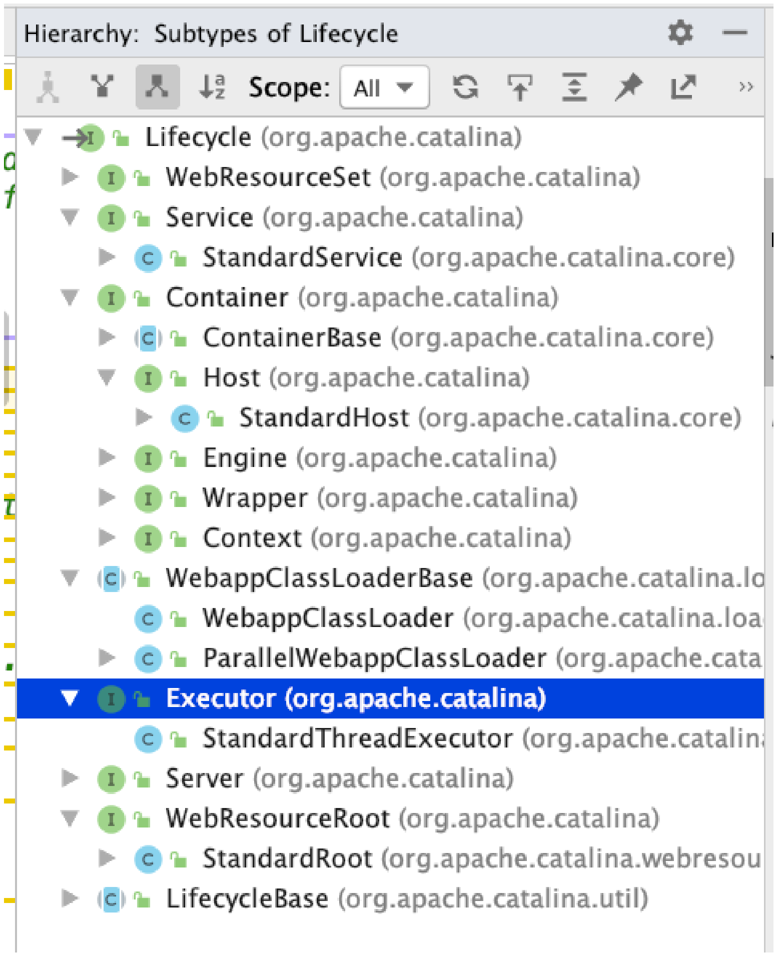

NioEndpoint.java

```java
                state = AcceptorState.RUNNING;

                try {
                    //if we have reached max connections, wait
                    countUpOrAwaitConnection();

                    SocketChannel socket = null;
                    try {
                        // Accept the next incoming connection from the server
                        // socket
                        socket = serverSock.accept();
                    } catch (IOException ioe) {
                        // We didn't get a socket
                        countDownConnection();
                        if (running) {
                            // Introduce delay if necessary
                            errorDelay = handleExceptionWithDelay(errorDelay);
                            // re-throw
                            throw ioe;
                        } else {
                            break;
                        }
                    }
                    // Successful accept, reset the error delay
                    errorDelay = 0;

                    // Configure the socket
                    if (running && !paused) {
                        // setSocketOptions() will hand the socket off to
```

2.1.3

Tomcat中的各容器器组件都会涉及创建、销毁等，因此设计了生命周期接口Lifecycle进行统一规范，各容器组件实现该接口。


```
Lifecycle
    SingleUse
    addLifecycleListener(LifecycleListener): void
    findLifecycleListeners(): LifecycleListener[]
    removeLifecycleListener(LifecycleListener): void
    init(): void
    start(): void
    stop(): void
    destroy(): void
    getState(): LifecycleState
    getStateName(): String
    BEFORE_INIT_EVENT: String = "before_init"
    AFTER_INIT_EVENT: String = "after_init"
    START_EVENT: String = "start"
    BEFORE_START_EVENT: String = "before_start"
    AFTER_START_EVENT: String = "after_start"
    STOP_EVENT: String = "stop"
    BEFORE_STOP_EVENT: String = "before_stop"
```

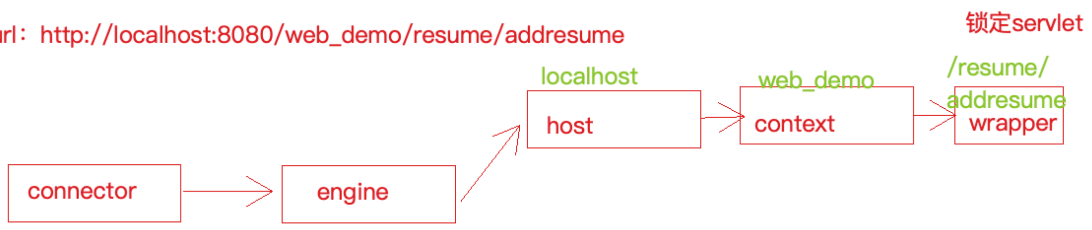


```
Hierarchy:  Subtypes of Lifecycle
Scope: All
▼ Lifecycle (org.apache.catalina)
  ▶ WebResourceSet (org.apache.catalina)
  ▼ Service (org.apache.catalina)
    ▶ StandardService (org.apache.catalina.core)
  ▼ Container (org.apache.catalina)
    ▶ ContainerBase (org.apache.catalina.core)
    ▼ Host (org.apache.catalina)
      ▶ StandardHost (org.apache.catalina.core)
    ▶ Engine (org.apache.catalina)
    ▶ Wrapper (org.apache.catalina)
    ▶ Context (org.apache.catalina)
  ▼ WebappClassLoaderBase (org.apache.catalina.lo
      WebappClassLoader (org.apache.catalina.loa
    ▶ ParallelWebappClassLoader (org.apache.cata
  ▼ Executor (org.apache.catalina)
      StandardThreadExecutor (org.apache.catalin
  ▶ Server (org.apache.catalina)
  ▼ WebResourceRoot (org.apache.catalina)
    ▶ StandardRoot (org.apache.catalina.webresou
  ▶ LifecycleBase (org.apache.catalina.util)
```

Tomcat请求处理流程:

tomcat请求处理流程: 当一个servlet请求到来的时候，tomcat是通过怎样的机制定位到servlet并且执行的

url：http://localhost:8080/web_demo/resume/addresume

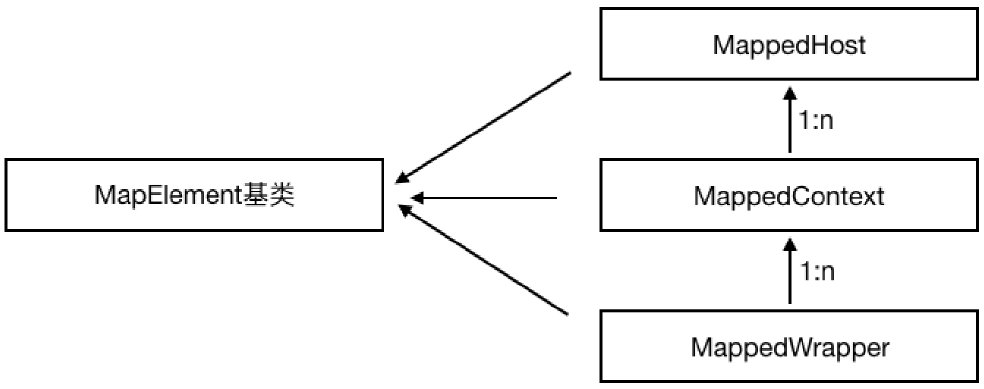


Mapper（映射的意思，这里不是集合）组件完成url和Host、Context、Wrapper等容器的映射

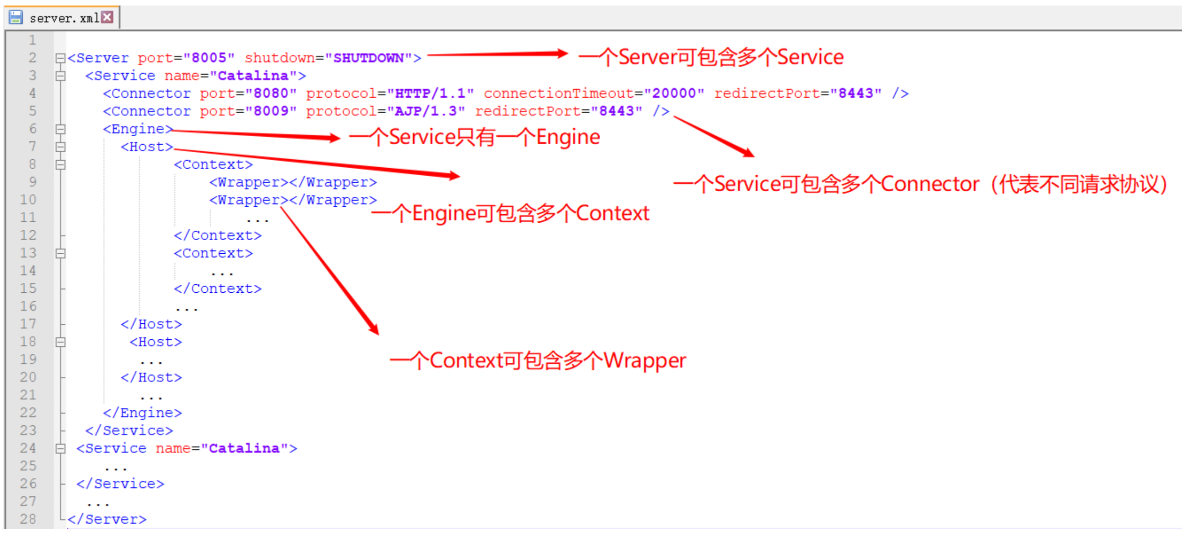
Mapper组件体系结构

web应用案例——>部署到tomcat软件中（不是源代码工程）
最终，希望的是把web应用案例部署到tomcat源代码工程中

Mapper体系结构：



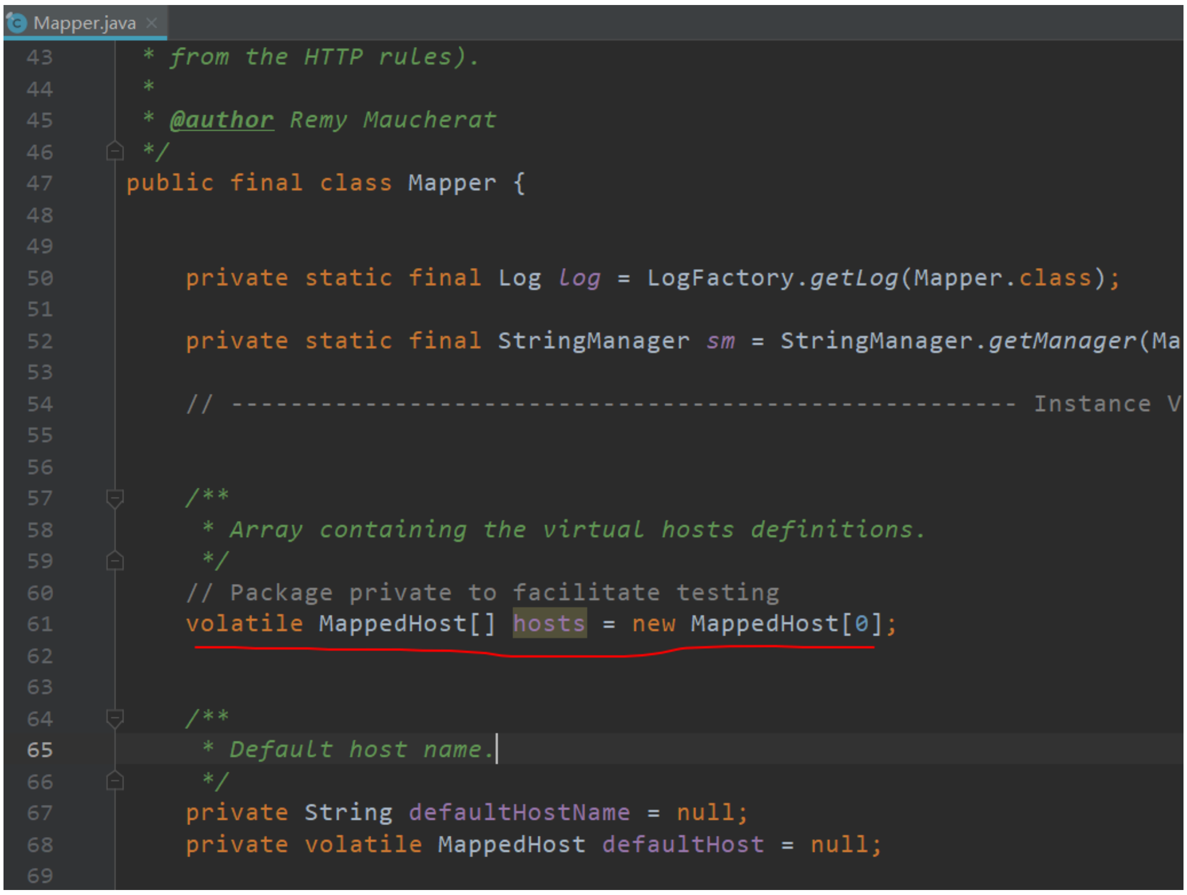
Server.xml层级关系如图2.2.1



```
<Server port="8005" shutdown="SHUTDOWN">
  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
    <Engine>
      <Host>
            <Context>
                <Wrapper></Wrapper>
                <Wrapper></Wrapper>
                    ...
            </Context>
            <Context>
                ...
            </Context>
            ...
      </Host>
       <Host>
          ...
      </Host>
          ...
    </Engine>
  </Service>
 <Service name="Catalina">
    ...
 </Service>
  ...
</Server>
```

server.xml
一个Server可包含多个Service
一个Service只有一个Engine
一个Service可包含多个Connector（代表不同请求协议）
一个Engine可包含多个Context
一个Context可包含多个Wrapper

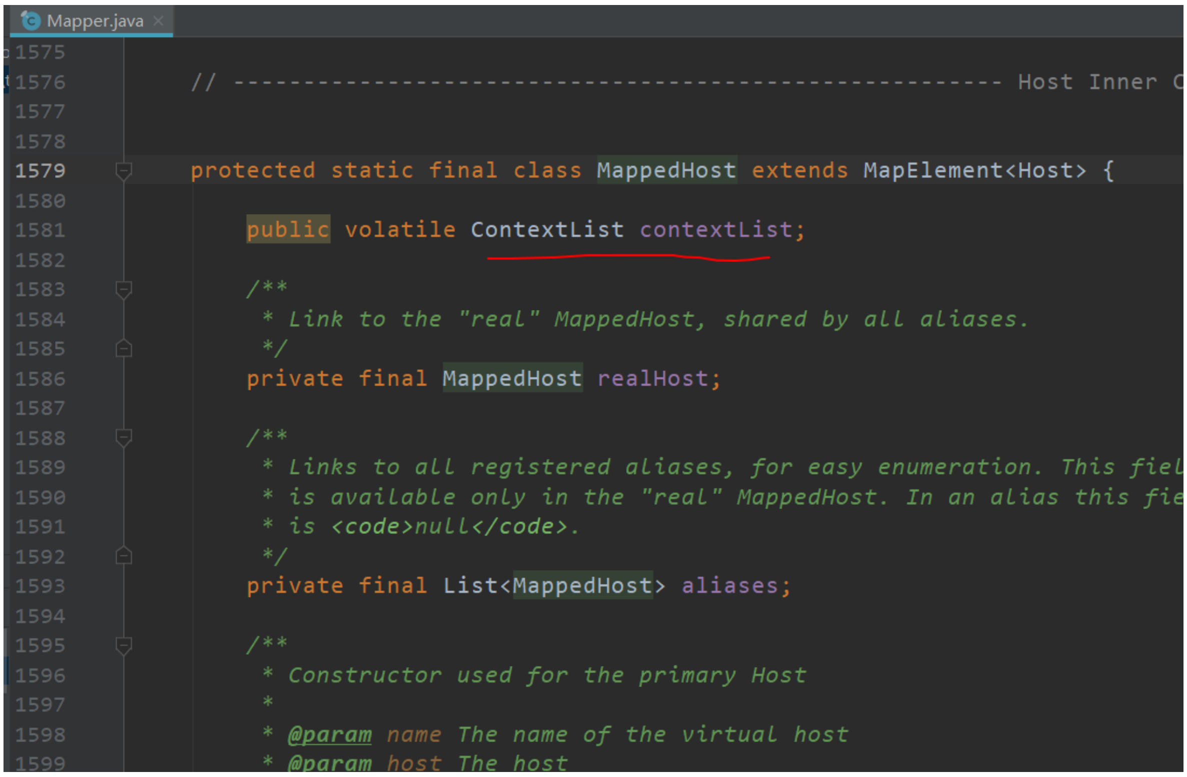
Mapper完成映射：都继承MapElement静态抽象类，MappedHost和MappedContext实现一对多的关系（另外一个类似）代码如图2.2.1、2.2.2、2.2.3。



Mapper.java

```java
 * from the HTTP rules).
 *
 * @author Remy Maucherat
 */
public final class Mapper {


    private static final Log log = LogFactory.getLog(Mapper.class);

    private static final StringManager sm = StringManager.getManager(Ma

    // ---------------------------------------------------- Instance V


    /**
     * Array containing the virtual hosts definitions.
     */
    // Package private to facilitate testing
    volatile MappedHost[] hosts = new MappedHost[0];


    /**
     * Default host name.
     */
    private String defaultHostName = null;
    private volatile MappedHost defaultHost = null;

```

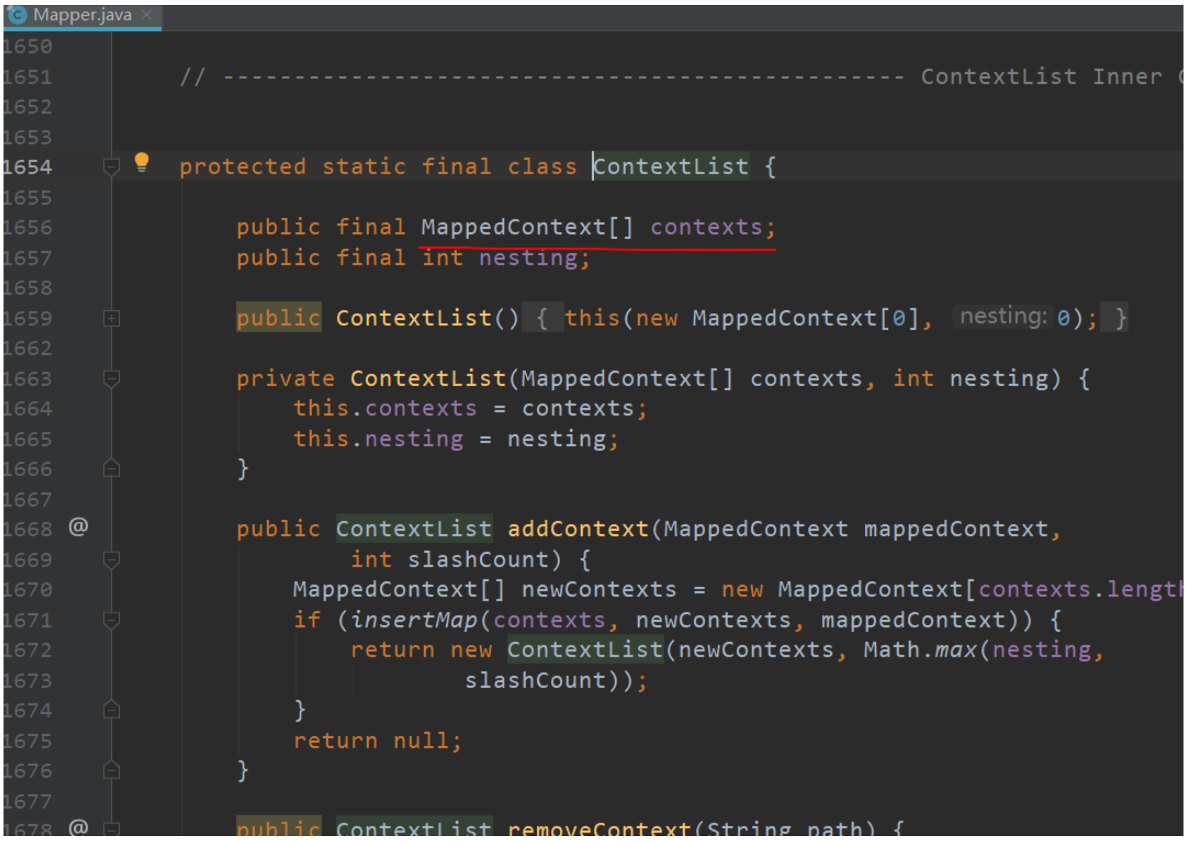
2.2.1



Mapper.java

```java

    // ---------------------------------------------------- Host Inner C

    protected static final class MappedHost extends MapElement<Host> {

        public volatile ContextList contextList;

        /**
         * Link to the "real" MappedHost, shared by all aliases.
         */
        private final MappedHost realHost;

        /**
         * Links to all registered aliases, for easy enumeration. This fiel
         * is available only in the "real" MappedHost. In an alias this fie
         * is <code>null</code>.
         */
        private final List<MappedHost> aliases;

        /**
         * Constructor used for the primary Host
         *
         * @param name The name of the virtual host
         * @param host The host
```

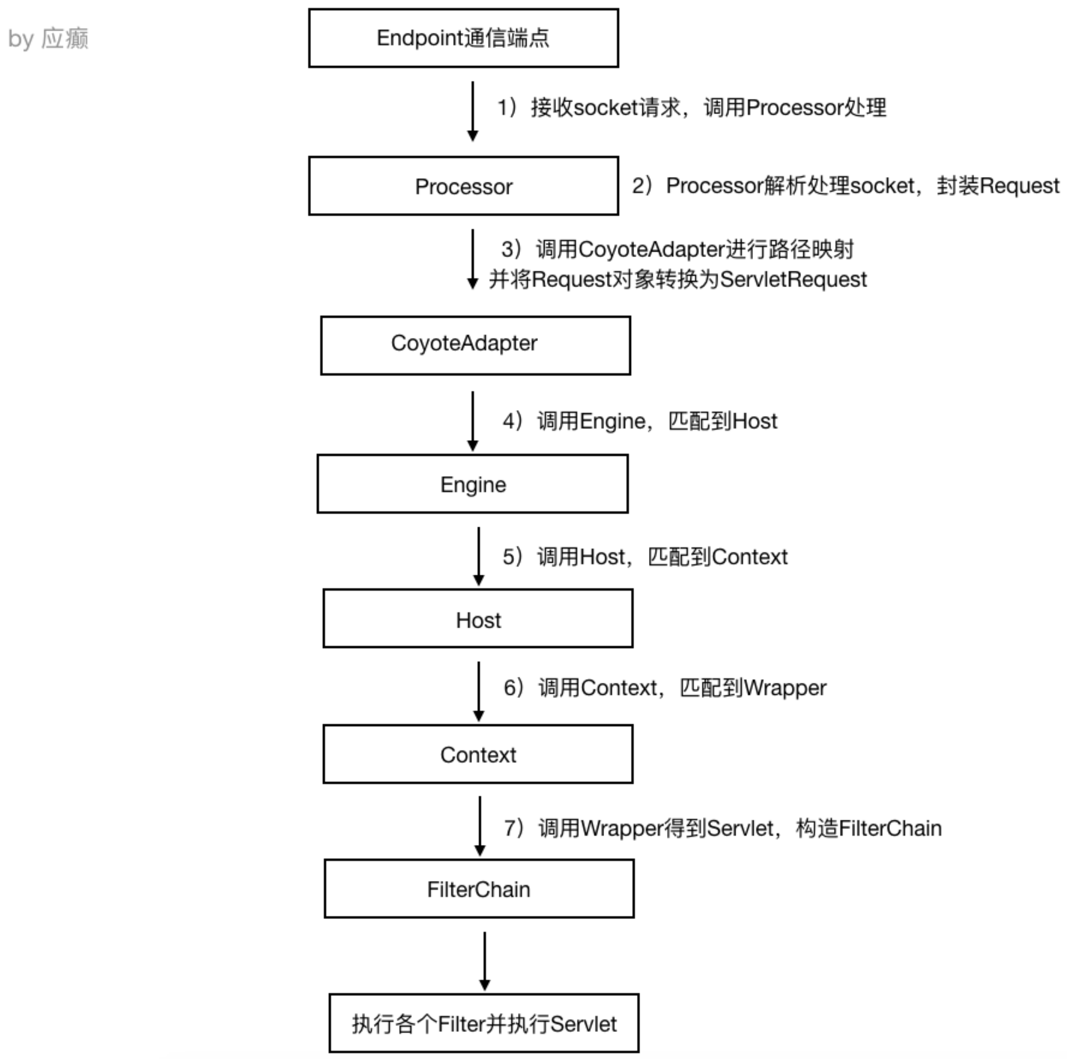
2.2.2



Mapper.java

```java
    // ------------------------------------------------ ContextList Inner C

    protected static final class ContextList {

        public final MappedContext[] contexts;
        public final int nesting;

        public ContextList() { this(new MappedContext[0], nesting: 0); }

        private ContextList(MappedContext[] contexts, int nesting) {
            this.contexts = contexts;
            this.nesting = nesting;
        }

        public ContextList addContext(MappedContext mappedContext,
                int slashCount) {
            MappedContext[] newContexts = new MappedContext[contexts.length
            if (insertMap(contexts, newContexts, mappedContext)) {
                return new ContextList(newContexts, Math.max(nesting,
                        slashCount));
            }
            return null;
        }

        public ContextList removeContext(String path) {
```

请求处理流程示意图：



by 应癫
Endpoint通信端点
1）接收socket请求，调用Processor处理
Processor
2）Processor解析处理socket，封装Request
3）调用CoyoteAdapter进行路径映射
并将Request对象转换为ServletRequest
CoyoteAdapter
4）调用Engine，匹配到Host
Engine
5）调用Host，匹配到Context
Host
6）调用Context，匹配到Wrapper
Context
7）调用Wrapper得到Servlet，构造FilterChain
FilterChain
执行各个Filter并执行Servlet