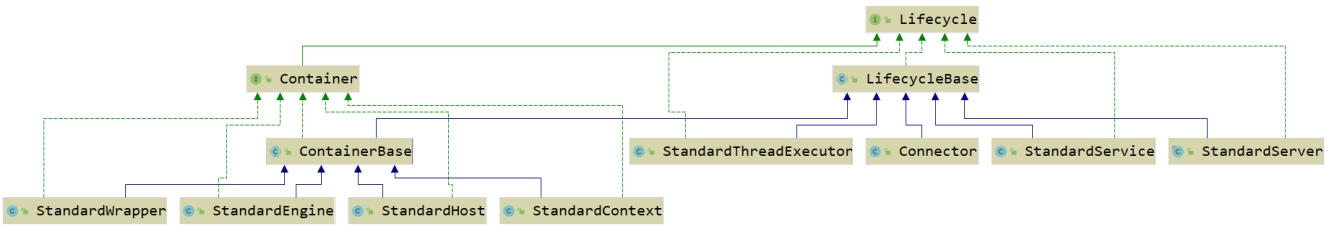
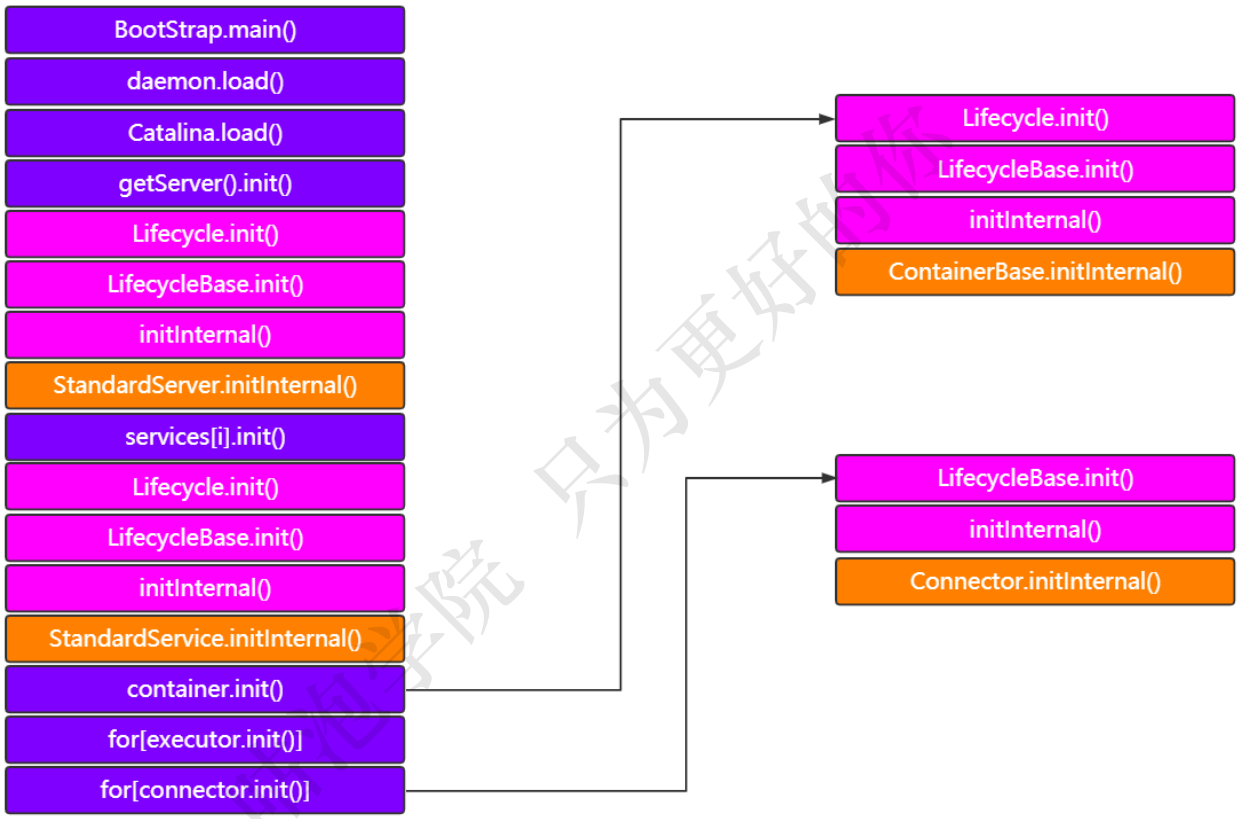
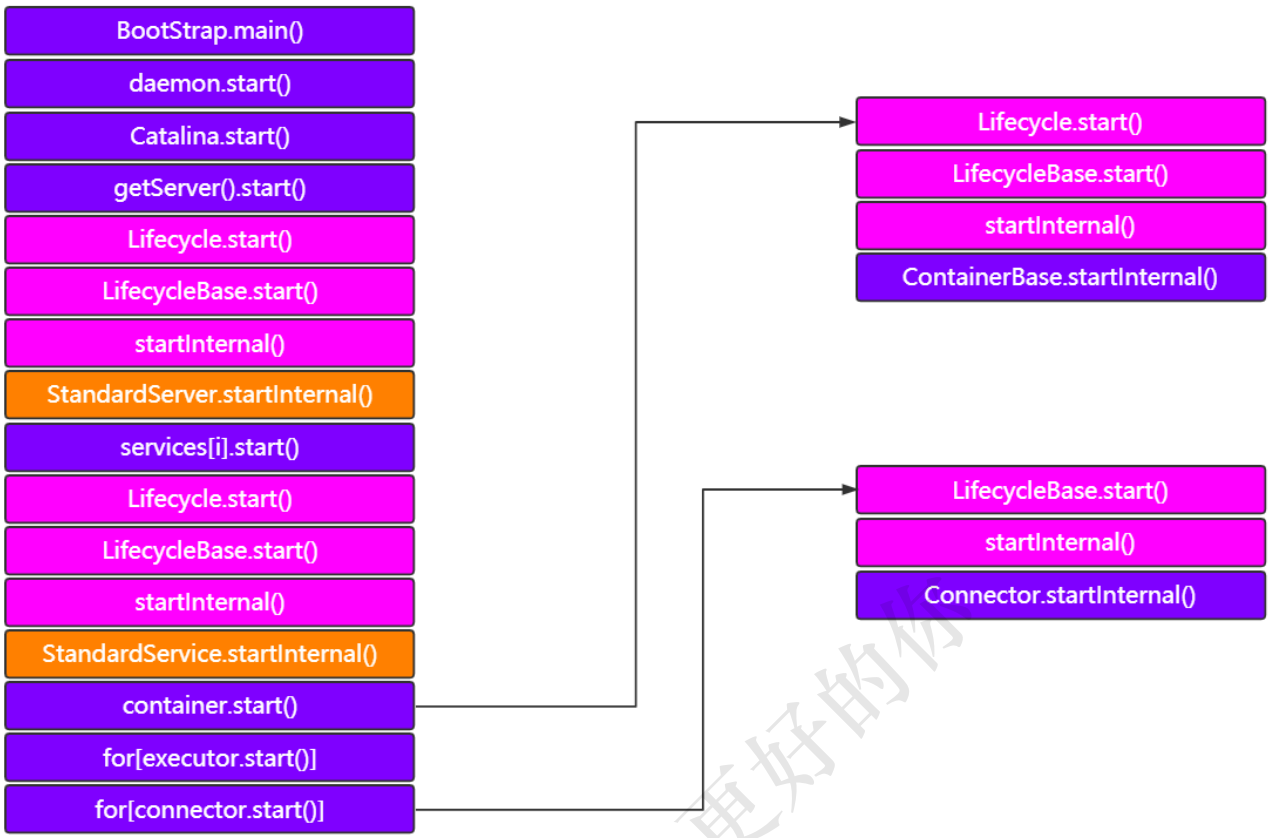
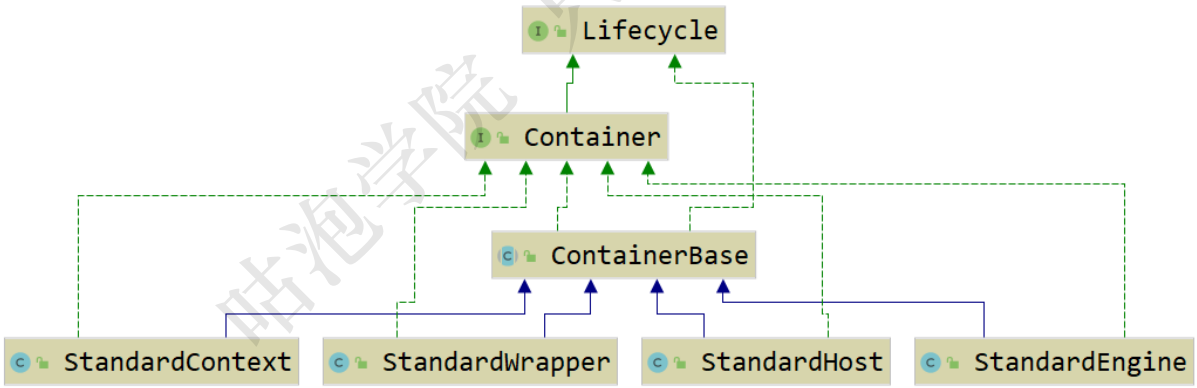
00 上节课内容的延续


Lifecycle
Container
LifecycleBase
ContainerBase
StandardThreadExecutor
Connector
StandardService
StandardServer
StandardWrapper
StandardEngine
StandardHost
StandardContext


BootStrap.main()
daemon.load()
Catalina.load()
getServer().init()
Lifecycle.init()
LifecycleBase.init()
initInternal()
StandardServer.initInternal()
services[i].init()
Lifecycle.init()
LifecycleBase.init()
initInternal()
StandardService.initInternal()
container.init()
for[executor.init()]
for[connector.init()]
Lifecycle.init()
LifecycleBase.init()
initInternal()
ContainerBase.initInternal()
LifecycleBase.init()
initInternal()
Connector.initInternal()



ContainerBase的类关系图



关注到上述图解中的ContainerBase.startInternal()方法

```
// Start our child containers, if any
Container children[] = findChildren();
List<Future<Void>> results = new ArrayList<>();
for (int i = 0; i < children.length; i++) {
   // 这句代码就是会调用ContainerBase下的一个个子容器的call方法
   results.add(startStopExecutor.submit(new StartChild(children[i])));
}
```

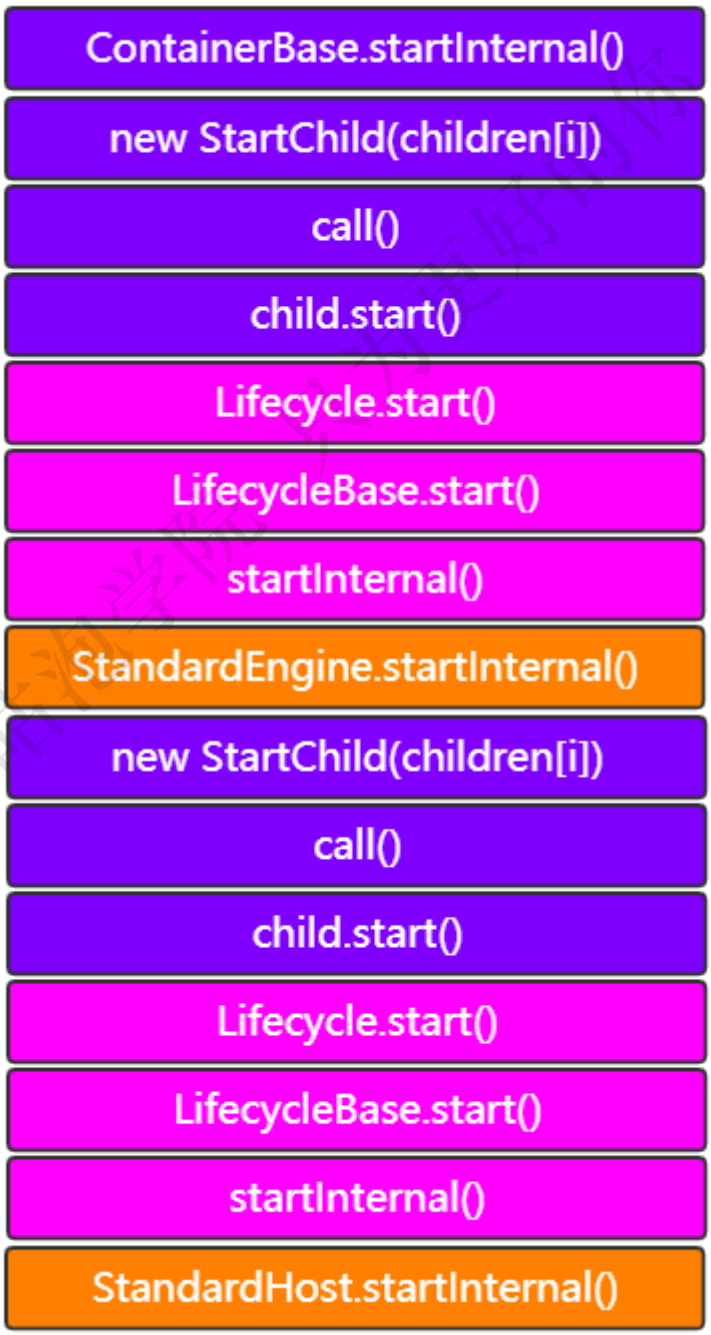
查看new StartChild要执行的call方法

```
    private static class StartChild implements Callable<Void> {
```

```
        private Container child;

        public StartChild(Container child) {
            this.child = child;
        }

        @Override
        public Void call() throws LifecycleException {
            child.start();
            return null;
        }
    }
```



> StandardHost将一个个web项目部署起来


```
// Deploy XML descriptors from configBase
deployDescriptors(configBase, configBase.list());
// Deploy WARs
deployWARs(appBase, filteredAppPaths);
// Deploy expanded folders
deployDirectories(appBase, filteredAppPaths);
```

> StandardContext.startInternal()解析web.xml和添加wrapper

`ContextConfig.webConfig()`的`step9`解析到`servlets`包装成`wrapper`对象

StandardContext.startInternal()->最终会调用`if (!loadOnStartup(findChildren()))`

官网：https://tomcat.apache.org/tomcat-8.0-doc/architecture/startup/serverStartup.pdf

01 性能优化

> Tomcat性能指标：吞吐量、响应时间、错误数、线程池、CPU 、内存等。
>
> 使用`jmeter`进行压测，然后观察相关指标

> **使用命令查看相关指标**

```
01 查看tomcat进程pid
    ps -ef | grep tomcat

02 查看进程的信息
    cat /pro/pid/status

03 查看进程的cpu和内存
    top -p pid
```

> **使用工具查看相关指标**

```
jconsole、jvisualvm、arthas、psi-probe等
```

1.1 优化思路

1.1.1 conf/server.xml核心组件

- Server

官网描述:Server interface which is rarely customized by users.【pass】

- Service

官网描述:The Service element is rarely customized by users.【pass】

- Connector

官网描述:Creating a customized connector is a significant effort.【need】

- Engine

官网描述:The Engine interface may be implemented to supply custom Engines, though this is uncommon.【pass】

- Host

官网描述:Users rarely create custom Hosts because the StandardHost implementation provides significant additional functionality.【pass】

- Context

官网描述:The Context interface may be implemented to create custom Contexts, but this is rarely the case because the StandardContext provides significant additional functionality.【maybe】

Context既然代表的是web应用，是和我们比较接近的，这块我们考虑对其适当的优化

> conclusion:Connector and Context

1.1.2 conf/server.xml非核心组件

官网：https://tomcat.apache.org/tomcat-8.0-doc/config/index.html

- Listener

Listener(即监听器)定义的组件，可以在特定事件发生时执行特定的操作；被监听的事件通常是Tomcat的启动和停止。

```
<Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
<!--监听内存溢出-->
<Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
<Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
<Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
```

- Global Resources

GlobalNamingResources元素定义了全局资源，通过配置可以看出，该配置是通过读取$TOMCAT_HOME/conf/tomcat-users.xml实现的。

The GlobalNamingResources element defines the global JNDI resources for the [Server](https://tomcat.apache.org/tomcat-8.0-doc/config/server.html)

```
<GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
    type="org.apache.catalina.UserDatabase"
    description="User database that can be updated and saved"
    factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
    pathname="conf/tomcat-users.xml" />
</GlobalNamingResources>
```

- Valve

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log" suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
```

- Realm

Realm，可以把它理解成“域”；Realm提供了一种用户密码与web应用的映射关系，从而达到角色安全管理的作用。在本例中，Realm的配置使用name为UserDatabase的资源实现。而该资源在Server元素中使用GlobalNamingResources配置

A Realm element represents a "database" of usernames, passwords, and roles (similar to Unix groups) assigned to those users.

```
<Realm className="org.apache.catalina.realm.LockOutRealm">
    <!-- This Realm uses the UserDatabase configured in the global JNDI
    resources under the key "UserDatabase".  Any edits
    that are performed against this UserDatabase are immediately
    available for use by the Realm.  -->
    <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
    resourceName="UserDatabase"/>
</Realm>
```

1.1.3 conf/web.xml

全局的web.xml文件有些标签用不到的，可以删除掉。

1.1.4 JVM层面

因为Tomcat运行起来本身就是一个Java进程，所以这块可以参照JVM部分的优化思路。

1.2 配置优化

1.2.1 减少web.xml/server.xml中标签

`最终观察tomcat启动日志[时间/内容]，线程开销，内存大小，GC等`

- DefaultServlet

`官网`:User Guide->Default Servlet

The default servlet is the servlet which serves static resources as well as serves the directory listings (if directory listings are enabled).

```
<servlet>
        <servlet-name>default</servlet-name>
        <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>0</param-value>
        </init-param>
        <init-param>
            <param-name>listings</param-name>
            <param-value>false</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>/</url-pattern>
</servlet-mapping>
```

- JspServlet

```
<servlet>
    <servlet-name>jsp</servlet-name>
    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
    <init-param>
        <param-name>fork</param-name>
        <param-value>false</param-value>
    </init-param>
    <init-param>
        <param-name>xpoweredBy</param-name>
        <param-value>false</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
</servlet>
```

```
<servlet-mapping>
    <servlet-name>jsp</servlet-name>
    <url-pattern>*.jsp</url-pattern>
    <url-pattern>*.jspx</url-pattern>
</servlet-mapping>
```

- welcome-list-file

```
<welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

- mime-mapping移除响应的内容

`支持的下载打开类型`

```
<mime-mapping>
    <extension>123</extension>
    <mime-type>application/vnd.lotus-1-2-3</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>3dml</extension>
    <mime-type>text/vnd.in3d.3dml</mime-type>
</mime-mapping>
```

- session-config

`默认jsp页面有session，就是在于这个配置`

```
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```


1.2.2 调整优化server.xml中标签

1.2.2.1 Connector标签

- protocol属性

```
<Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

对于protocol="HTTP/1.1"，查看源码

`构造函数`

```
public Connector(String protocol) {
        setProtocol(protocol);
}
```

`setProtocol(protocol)因为配置文件中传入的是HTTP/1.1`

`并且这里没有使用APR，一会我们会演示APR`

```
else {
    if ("HTTP/1.1".equals(protocol)) {
    setProtocolHandlerClassName
    ("org.apache.coyote.http11.Http11NioProtocol");
    } else if ("AJP/1.3".equals(protocol)) {
    setProtocolHandlerClassName
    ("org.apache.coyote.ajp.AjpNioProtocol");
    } else if (protocol != null) {
    setProtocolHandlerClassName(protocol);
    }
}
```

`发现这里调用的是Http11NioProtocol，也就是说明tomcat8.0.x中默认使用的是NIO`

> 使用同样的方式看tomcat7和tomcat8.5，你会发现tomcat7默认使用的是BIO，tomcat8.5默认使用的是NIO

`针对BIO和NIO的方式进行压测`

（1）BIO

来到tomcat官网Configuration/HTTP/protocol

```
org.apache.coyote.http11.Http11Protocol - blocking Java connector
org.apache.coyote.http11.Http11NioProtocol - non blocking Java NIO connector
org.apache.coyote.http11.Http11Nio2Protocol - non blocking Java NIO2 connector
org.apache.coyote.http11.Http11AprProtocol - the APR/native connector.
```

（2）NIO

tomcat8.0中默认使用的是NIO

（3）APR

调用本地方法库进行IO操作

- executor属性

`最佳线程数公式`:((线程等待时间+线程cpu时间)/线程cpu时间) * cpu数量

```
The Executor represents a thread pool that can be shared between components in Tomcat.
Historically there has been a thread pool per connector created but this allows you to
share a thread pool, between (primarily) connector but also other components when those get
configured to support executors
```

默认的可以查看StandardExecutor类

设置一些属性

`官网:https://tomcat.apache.org/tomcat-8.0-doc/config/http.html`

（1）acceptCount:达到最大连接数之后，等待队列中还能放多少连接，超过即拒绝，配置太大也没有意义

```
The maximum queue length for incoming connection requests when all possible request
processing threads are in use. Any requests received when the queue is full will be
refused. The default value is 100.
```

（2）maxConnections

达到这个值之后，将继续接受连接，但是不处理，能继续接受多少根据acceptCount的值

BIO:maxThreads

NIO/NIO2:10000 ———— AbstractEndpoint.maxConnections

APR:8192

```
The maximum number of connections that the server will accept and process at any given
time. When this number has been reached, the server will accept, but not process, one
further connection. This additional connection be blocked until the number of connections
being processed falls below maxConnections at which point the server will start accepting
and processing new connections again. Note that once the limit has been reached, the
operating system may still accept connections based on the acceptCount setting. The default
value varies by connector type. For BIO the default is the value of maxThreads unless an
Executor is used in which case the default will be the value of maxThreads from the
executor. For NIO and NIO2 the default is 10000. For APR/native, the default is 8192.

Note that for APR/native on Windows, the configured value will be reduced to the highest
multiple of 1024 that is less than or equal to maxConnections. This is done for performance
reasons.
If set to a value of -1, the maxConnections feature is disabled and connections are not
counted.
```

（3）maxThreads:最大工作线程数，也就是用来处理request请求的，默认是200，如果自己配了executor，并且和Connector有关联了，则之前默认的200就会被忽略，取决于CPU的配置。监控中就可以看到所有的工作线程是什么状态，通过监控就能知道开启多少个线程合适

```
The maximum number of request processing threads to be created by this Connector, which
therefore determines the maximum number of simultaneous requests that can be handled. If
not specified, this attribute is set to 200. If an executor is associated with this
connector, this attribute is ignored as the connector will execute tasks using the executor
rather than an internal thread pool. Note that if an executor is configured any value set
for this attribute will be recorded correctly but it will be reported (e.g. via JMX) as -1
to make clear that it is not used.
```

（4）minSpareThreads

最小空闲线程数

```
The minimum number of threads always kept running. This includes both active and idle
threads. If not specified, the default of 10 is used. If an executor is associated with
this connector, this attribute is ignored as the connector will execute tasks using the
executor rather than an internal thread pool. Note that if an executor is configured any
value set for this attribute will be recorded correctly but it will be reported (e.g. via
JMX) as -1 to make clear that it is not used.
```

可以实践一下，Connector配合自定义的线程池

```
 <Connector executor="tomcatThreadPool"
               port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
```

```
<Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
        maxThreads="150" minSpareThreads="4"/>
```

其实这块最好的方式是结合BIO来看，因为BIO是一个request对应一个线程

```
值太低，并发请求多了之后，多余的则进入等待状态。
值太高，启动Tomcat将花费更多的时间。
比如可以改成250。
```

- enableLookups

设置为false

- 删掉AJP的Connector

1.2.2.2 Host标签

autoDeploy :Tomcat运行时，要用一个线程拿出来进行检查，生产环境之下一定要改成false

```
This flag value indicates if Tomcat should check periodically for new or updated web
applications while Tomcat is running. If true, Tomcat periodically checks the appBase and
xmlBase directories and deploys any new web applications or context XML descriptors found.
Updated web applications or context XML descriptors will trigger a reload of the web
application. The flag's value defaults to true. See Automatic Application Deployment for
more information.
```

1.2.2.3 Context标签

reloadable:false

```
reloadable:如果这个属性设为true，tomcat服务器在运行状态下会监视在WEB-INF/classes和WEB-INF/lib目录下
class文件的改动，如果监测到有class文件被更新的，服务器会自动重新加载Web应用。
在开发阶段将reloadable属性设为true，有助于调试servlet和其它的class文件，但这样用加重服务器运行负荷，建议
在Web应用的发存阶段将reloadable设为false。
```

```
Set to true if you want Catalina to monitor classes in /WEB-INF/classes/ and /WEB-INF/lib
for changes, and automatically reload the web application if a change is detected. This
feature is very useful during application development, but it requires significant runtime
overhead and is not recommended for use on deployed production applications. That's why the
default setting for this attribute is false. You can use the Manager web application,
however, to trigger reloads of deployed applications on demand.
```

1.3 启动速度优化

- 删除没用的web应用

因为tomcat启动每次都会部署这些应用

- 关闭WebSocket

websocket-api.jar和tomcat-websocket.jar

- 随机数优化

设置JVM参数：-Djava.security.egd=file:/dev/./urandom

- 多个线程启动Web应用

```
<Host startStopThreads="0">

</Host>
```

1.4 其他方面的优化

- Connector

  配置压缩属性compression="500"，文件大于500bytes才会压缩
- 数据库优化

  减少对数据库访问等待的时间，可以从数据库的层面进行优化，或者加缓存等等各种方案。
- 开启浏览器缓存，nginx静态资源部署

1.5 常见问题排查

1.5.1 CPU使用率过高

> 可能原因

GC频繁或者创建了很多业务线程

> 排查

哪些线程比较消耗CPU，或者多线程上下文频繁切换

> 解决思路

top -H -p pid 查看某个Java进程各个线程使用CPU的情况，找到哪个线程占用CPU比较高

jstack pid 打印出线程信息，定位到上述的线程名称

1.5.2 拒绝连接

- java.net.BindException: Address already in use: JVM_Bind

端口被占用，可以使用netstat -an 查看端口占用情况，关闭对应的进程或者tomcat换端口

- java.net.ConnectException: Connection refused: connect

ping一下服务端的IP，可能服务端机器有问题

- java.net.SocketException: Too many open files

可能在高并发的情况下，创建的Socket过多，文件句柄不够用了，可以关闭无用的句柄，如果都有用，可以增加文件句柄数：ulimit -n 10000