Tomcat 优化

主要内容

Tomcat 工作原理
Tomcat 应用
Tomcat 优化配置
Tomcat 优化案例
Tomcat 优化总结

学习目标

知识点	要求
Tomcat 工作原理	掌握
Tomcat 应用	掌握
Tomcat 优化配置	掌握
Tomcat 优化案例	掌握
Tomcat 优化总结	掌握

一、 Tomcat 工作原理

1 服务器分类

1.1 JavaEE 更名

JavaEE，很多人都知道它是社区驱动的企业软件标准。JavaEE 是利用 Java Community Process 开发的，每个版本都集成了符合业界需求的新特性，提供了一个丰富的企业软件平台，提高了应用可移植性，提高了开发人员的工作效率。

2018-03-05 日，据国外媒体报道，开源组织 Eclipse 基金会宣布将 JavaEE(Enterprise Edition)被更名为 JakartaEE(雅加达)。这是 Oracle 将 Java 移交给开源组织 Eclipse 后实现对 Java 品牌控制的最新举措。

1.2 应用服务器(JavaEE Application Server)

应用服务器是 Java EE 规范的具体实现，可以执行/驱动基于 JavaEE 平台开发的 web 项目。绝大部分的应用服务器都是付费产品。

常见的应用服务：

Weblogic（BEA）

WebLogic 是 BEA 公司的产品。WebLogic 支持企业级的、多层次的和完全分布式的 Web 应用，并且服务器的配置简单、界面友好。对于那些正在寻求能够提供 JavaEE 平台所拥有的一切应用服务器的用户来说，WebLogic 是一个十分理想的选择。

Webshpere（IBM）

WebSphere 是 IBM 公司的产品，WebSphere Application Server 是基于 JavaEE 的应用环境，可以运行多种操作系统平台，用于建立、部署和管理 JavaEE 应用程序。

JBoss（RedHad）

JBoss 是一个种遵从 JavaEE 规范的、开放源代码的、纯 Java 的 EJB 服务器，对于 JavaEE 有很好的支持。

Geronimo（Apache）

Geronimo 是 Apache 软件基金会的开放源码 J2EE 服务器，是一个符合 J2EE 标准的应用服务器。

1.3 Web 容器(Web Server)

只实现了 JavaEE 平台下部分技术标准，如 Servlet，Jsp，JNDI，JavaMail。Web 容器是开源免费的。

Tomcat（Apache 开源免费）

Tomcat 是 Apache 软件基金会的一款完全开源免费的 Servlet 容器，它是全世界最著名的基于 Java 语言的轻量级应用服务器，它支持 HTML、JS 等静态资源的处理，因此又可以作为轻量级 Web 容器使用。

Jetty（Jetty 开源免费）

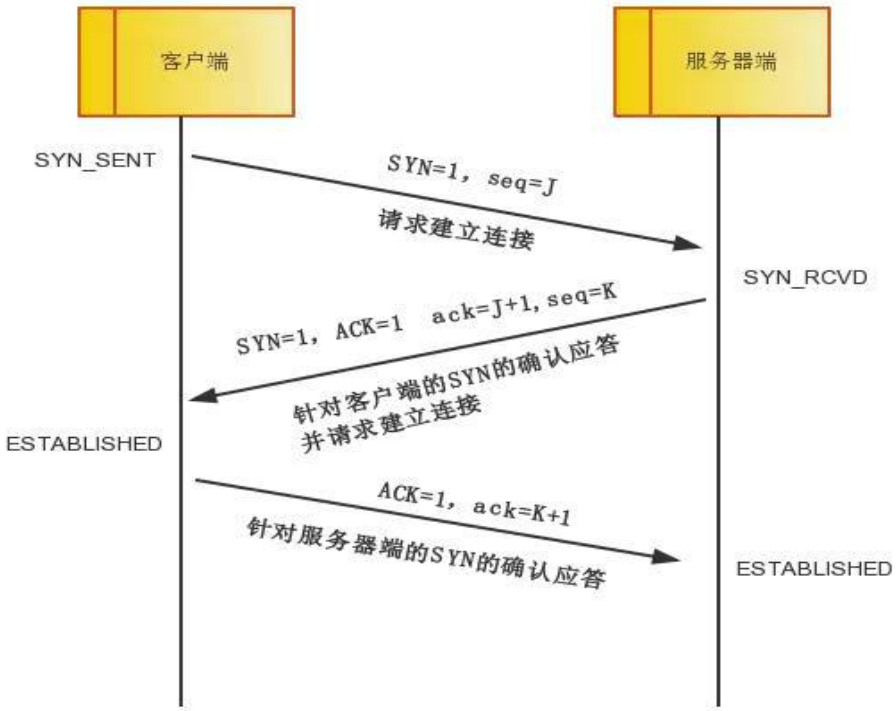
Jetty 是挂在 eclipse 基金会下，基于 Java 语言编写的实现 Servlet 规范的 Web 容器。

2 TCP 协议

TCP 是一个可靠的传输协议，在创建连接时会经历三次握手，在断开连接时会经历四次挥手。

2.1建立连接的三次握手

所谓三次握手是指建立一个 TCP 连接时需要客户端和服务器端总共发送三个包以确认连接的建立。



A 和 B 打电话：

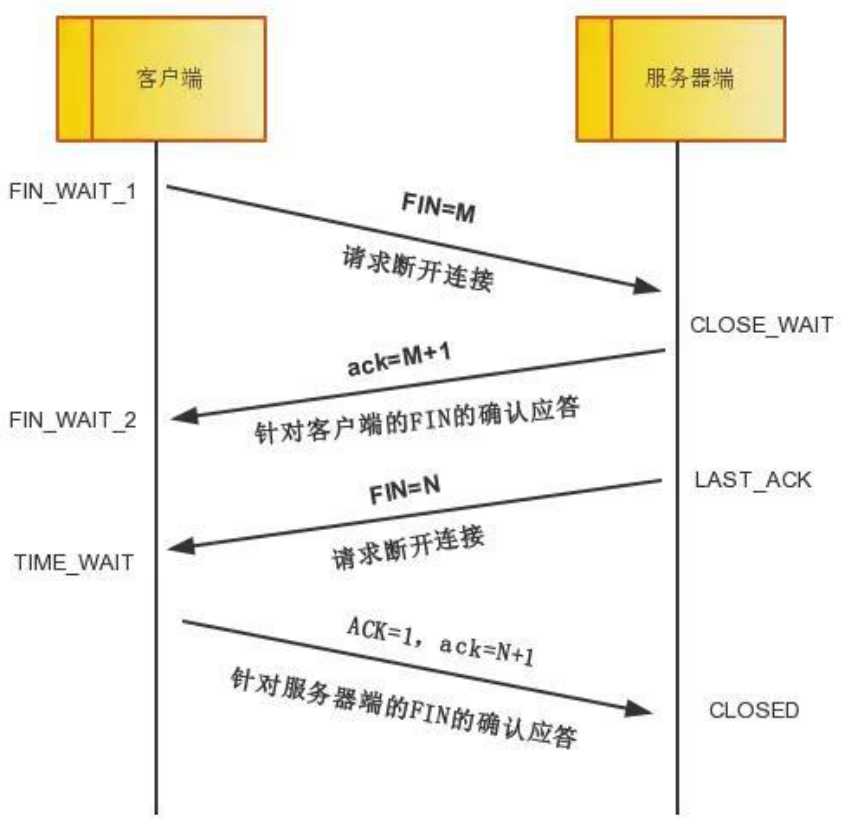
A 对 B 说：你好，我是 A,你能听到我说话吗？

B 对 A 说：嗯，我能听到你说话。

A 对 B 说：好，那我们开始聊天吧。

2.2 断开连接的四次挥手

四次挥手即终止 TCP 连接，就是指断开一个 TCP 连接时，需要客户端和服务端总共发送 4 个包以确认连接的断开。



A 和 B 打电话：

A 对 B 说：我说完了，我要挂电话了

B 对 A 说：等一下，我还没说完

B 继续对 A 说：我说完了，你可以挂电话了

A 对 B 说：好，我挂电话了

3 Http 协议

3.1 Http 协议简介

超文本传输协议（英文：HyperText Transfer Protocol，缩写：HTTP）是一种用于分布式、协作式和超媒体信息系统的应用层协议。HTTP 是万维网的数据通信的基础。

HTTP 是一个客户端终端（用户）和服务器端（网站）请求和应答的标准（TCP）。通过

使用网页浏览器、网络爬虫或者其它的工具，客户端发起一个 HTTP 请求到服务器上指定端口（默认端口为 80）。我们称这个客户端为用户代理程序（user agent）。应答的服务器上存储着一些资源，比如 HTML 文件和图像。我们称这个应答服务器为源服务器（origin server）。

通常，由 HTTP 客户端发起一个请求，创建一个到服务器指定端口（默认是 80 端口）的 TCP 连接。HTTP 服务器则在那个端口监听客户端的请求。一旦收到请求，服务器会向客户端返回一个状态，比如"HTTP/1.1 200 OK"，以及返回的内容，如请求的文件、错误消息、或者其它信息。

3.2 Http 协议特点

- 单向性
- 无状态

3.3 Http 工作原理

HTTP 请求/响应的步骤：

1) 客户端连接到 Web 服务器

一个 HTTP 客户端，通常是浏览器，与 Web 服务器的 HTTP 端口（默认为 80）建立一个 TCP 套接字连接。例如，http://www.baidu.com。

2) 发送 HTTP 请求

通过 TCP 套接字，客户端向 Web 服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据 4 部分组成。

3) 服务器接受请求并返回 HTTP 响应

Web 服务器解析请求，定位请求资源。服务器将资源复本写到 TCP 套接字，由客户端读取。一个响应由状态行、响应头部、空行和响应数据 4 部分组成。

4) 释放连接 TCP 连接

若 connection 模式为 close，则服务器主动关闭 TCP 连接，客户端被动关闭连接，释放 TCP 连接;若 connection 模式为 keepalive，则该连接会保持一段时间，在该时间内可以继续接收请求。

5) 客户端浏览器解析 HTML 内容

客户端浏览器首先解析状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的 HTML 文档和文档的字符集。客户端浏览器读取响应数据 HTML，根据 HTML 的语法对其进行格式化，并在浏览器窗口中显示。

例如：在浏览器地址栏键入 URL，按下回车之后会经历以下流程：

浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址;

解析出 IP 地址后，根据该 IP 地址和默认端口 80，和服务器建立 TCP 连接;

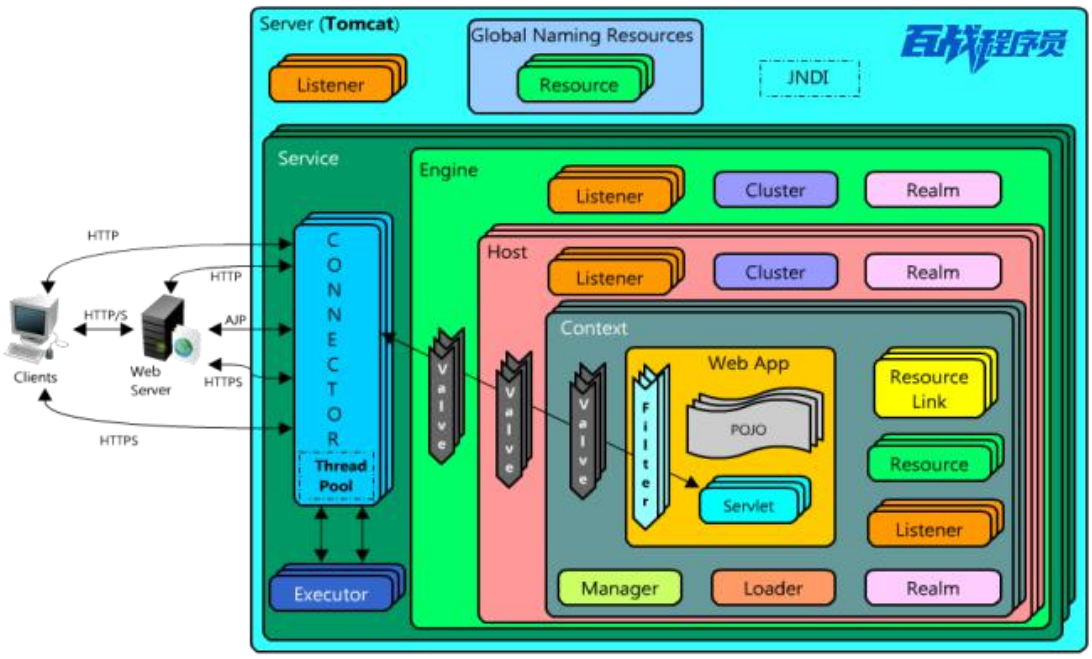
浏览器发出读取文件(URL 中域名后面部分对应的文件)的 HTTP 请求，该请求报文作为 TCP 三次握手的第三个报文的数据发送给服务器;

服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器;

释放 TCP 连接;

浏览器将该 html 文本并显示内容;

4 Tomcat 架构介绍



在架构图中包含了 Tomcat 内部的主要组件，每个组件之间的层次包含关系很清楚。Tomcat 大体上可以看成由 Connector 和 Container 组件组成。Connector 组件负责在服务器端处理客户端连接（客户端连接、接收客户端消息报文、消息报文的解析等），Container 负责对客户端的请求进行逻辑处理，并把结果返回给客户端。

对应 server.xml

```
<Server>
  <Listener/>
  <GlobalNamingResources>
    <Resource/>
  </GlobalNamingResources>
  <Service>
    <Executor/>
    <Connector/>
    <Engine>
      <Cluster/>
      <Realm/>
        <Host>
            <Context/>
        </Host>
```

```
        </Engine>
      </Service>
    </Server>
```

5 Tomcat 组件介绍

5.1 Server 组件

Server 是最顶级的组件，代表 Tomcat 的运行实例，一个 JVM 中只会有一个 Server。

5.2 Listener 组件

在 Tomcat 生命周期中完成不同的工作。

5.3 GlobalNamingResources 组件

集成 JNDI。

5.4 Service 组件

指的是一个服务，主要的功能是把 connector 组件和 engine 组织起来，使得通过 connector 组件与整个容器通讯的应用可以使用 engine 提供的服务。

5.5 Connector 组件

连接器组件，可以配置多个连接器支持多种协议，如 http，APJ 等。

5.6 Engine 组件

服务引擎，这个可以理解为一个真正的服务器，内部提供了多个虚拟主机对外服务。

5.7 Host 组件

虚拟主机，每一个虚拟主机相当于一台服务器，并且内部可以部署多个应用，每个虚拟主机可以绑定一个域名，并指定多个别名。

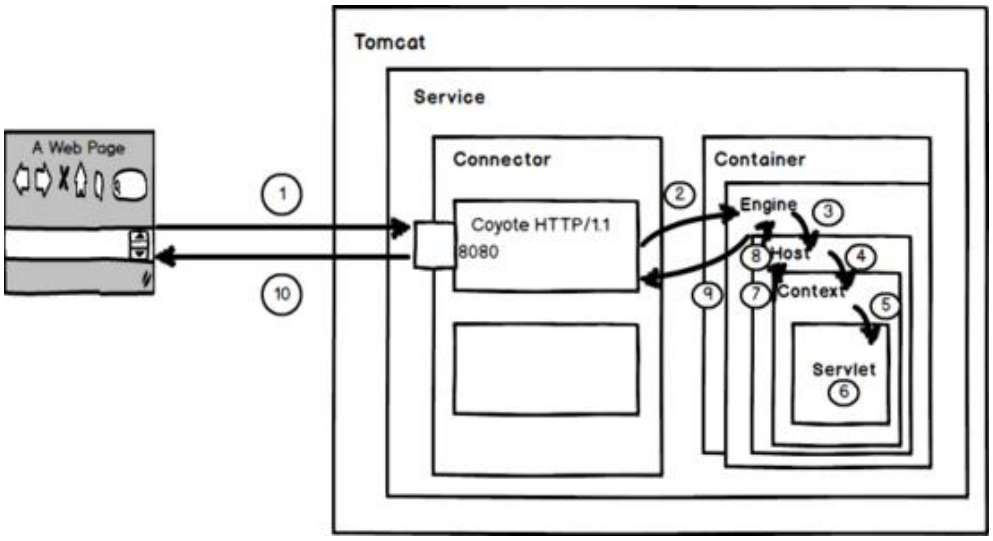
5.8 Context 组件

应用上下文，每一个 webapp 都有一个单独的 context，也可以理解为每一个 context 代表一个 webapp。

5.9 Valva 组件

Access Log Valve 用来创建日志文件，格式与标准的 web server 日志文件相同。可以使用用日志分析工具对日志进行分析，跟踪页面点击次数、用户会话的活动等。

6 Tomcat 处理请求过程



① 用户访问 localhost:8080/test/index.jsp，请求被发送到 Tomcat，被监听 8080 端口并处理 HTTP/1.1 协议的 Connector 获得。

② Connector 把该请求交给它所在的 Service 的 Engine 来处理，并等待 Engine 的回应。

③ Engine 获得请求 localhost/test/index.jsp，匹配所有的虚拟主机 Host。

④ Engine 匹配到名为 localhost 的 Host 虚拟主机来处理/test/index.jsp 请求（即使匹配不到会请求交给默认 Host 处理），Host 会根据/test 匹配它所拥有的所有的 Context。

⑤ 匹配到的 Context 获得请求/index.jsp。

⑥ 构造 HttpServletRequest 对象和 HttpServletResponse 对象，作为参数调用 JspServlet 的 doGet（ ）或 doPost（ ）.执行业务逻辑、数据存储等程序。

⑦ Context 把执行完之后的结果通过 HttpServletResponse 对象将响应内容返回给 Host。

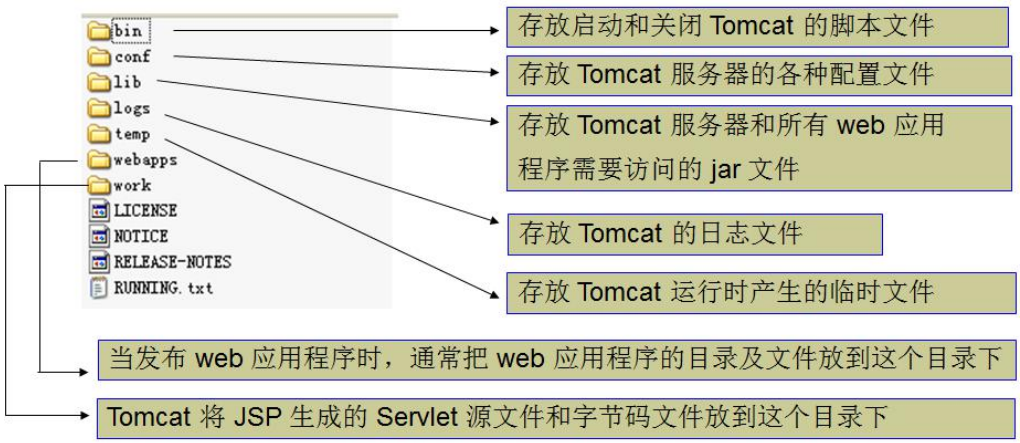
⑧ Host 将响应内容返回给 Engine。

⑨ Engine 将响应内容返回 Connector。

⑩ Connector 将响应内容返回给客户 Browser。

二、 Tomcat 应用

1 Tomcat 目录结构



- bin

  脚本文件目录。

- conf

  存放配置文件，最重要的是 server.xml。

- lib

  存放所有 web 项目都可以访问的公共 jar 包（使用 Common 类加载器加载）。

- logs

  存放日志文件。

- temp

Tomcat 运行时候存放临时文件用的。

- webapps

  web 应用发布目录。

- work

  Tomcat 把各种由 jsp 生成的 servlet 文件放在这个目录下。删除后，启动时会自动创建。

2 Tomcat 配置文件详解

2.1 Tomcat 配置文件介绍

Tomcat 的配置文件由 4 个 xml 组成，分别是 context.xml、web.xml、server.xml、tomcat-users.xml。每个文件都有自己的功能与配置方法。

2.1.1 context.xml

Context.xml 是 Tomcat 公用的环境配置。 Tomcat 服务器会定时去扫描这个文件。一旦发现文件被修改（时间戳改变了），就会自动重新加载这个文件，而不需要重启服务器 。

2.1.2 web.xml

Web 应用程序描述文件，都是关于是 Web 应用程序的配置文件。所有 Web 应用的 web.xml 文件的父文件。

2.1.3 server.xml

是 tomcat 服务器的核心配置文件 ,server.xml 的每一个元素都对应了 tomcat 中的一个组件，通过对 xml 中元素的配置，实现对 tomcat 中的各个组件和端口的配置。

2.1.4 tomcat-users.xml

配置访问 Tomcat 的用户以及角色的配置文件。

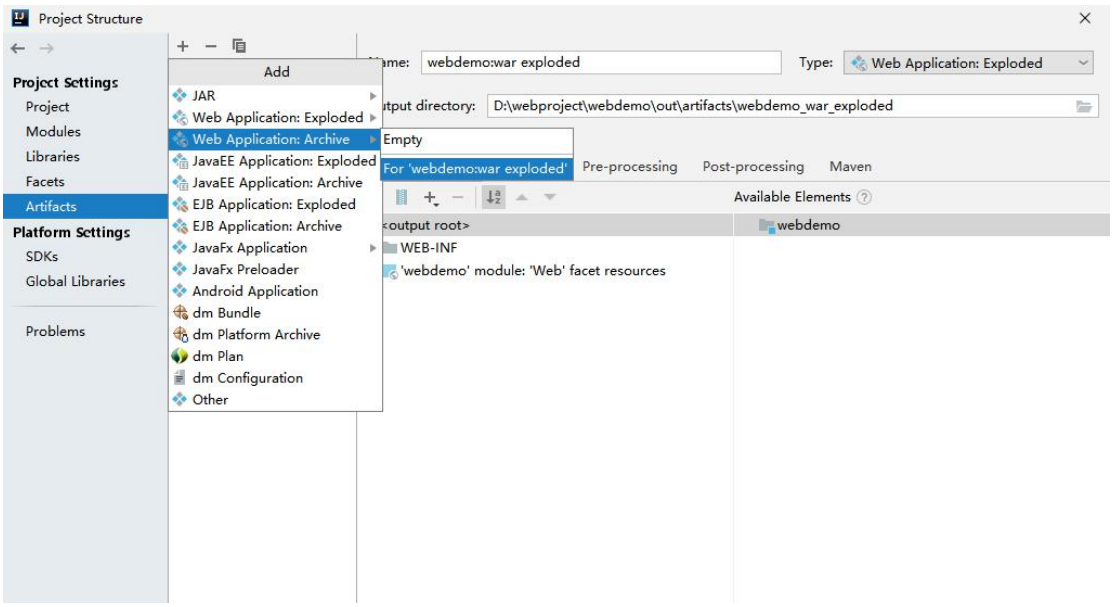
3 Tomcat 项目部署方式

3.1部署方式一

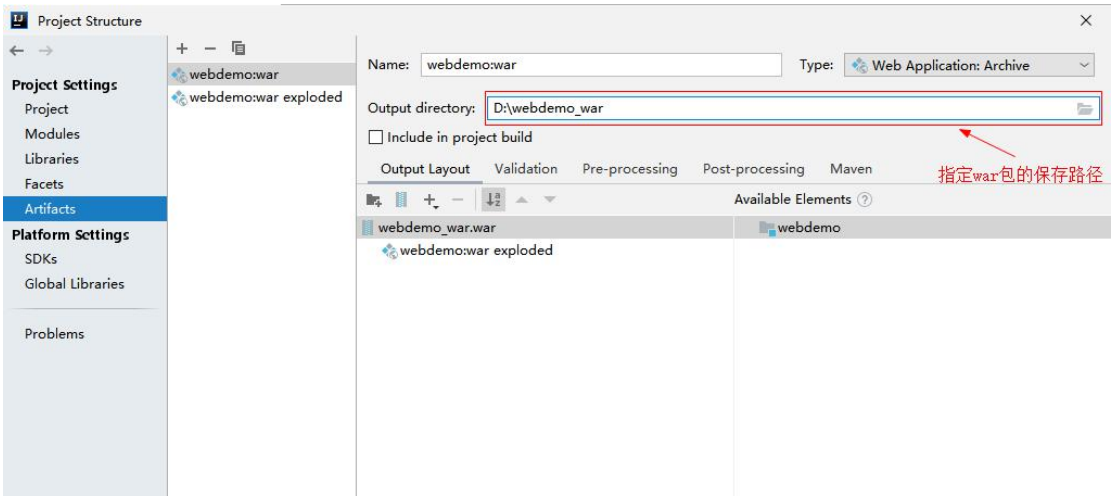
将项目的 war 包直接部署到 webapps 目录中。

导出项目的 war 包

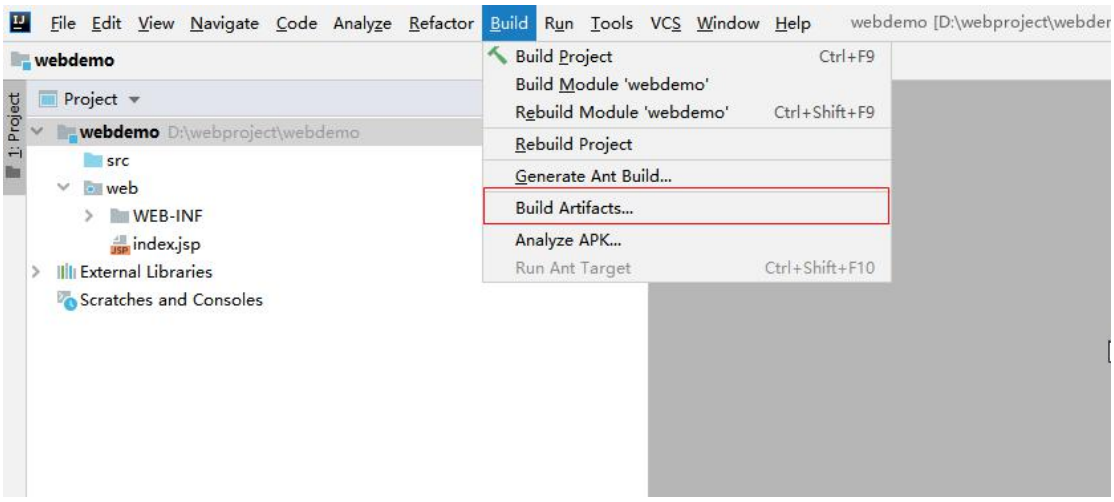
Step 1



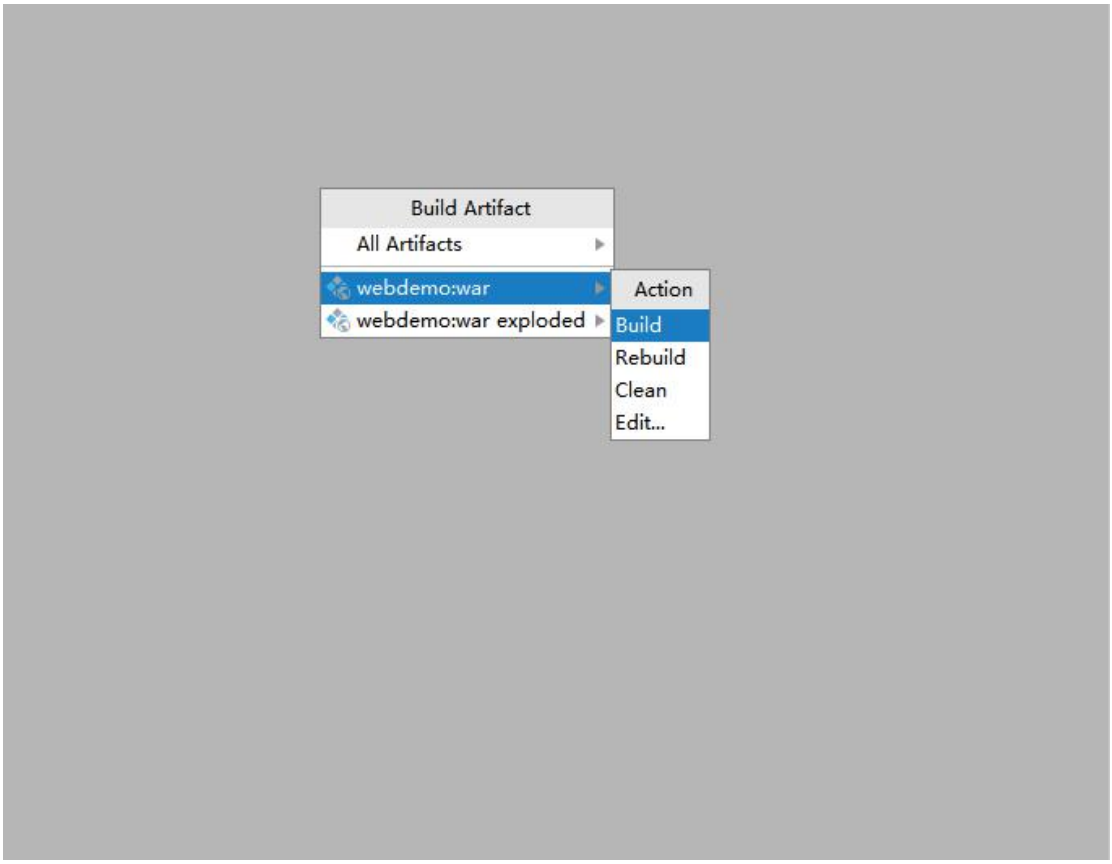
Step2



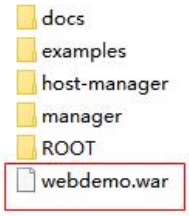
Step3


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
webdemo [D:\webproject\webdemo]
webdemo
Project
1: Project
webdemo D:\webproject\webdemo
src
web
WEB-INF
index.jsp
External Libraries
Scratches and Consoles
Build Project Ctrl+F9
Build Module 'webdemo'
Rebuild Module 'webdemo' Ctrl+Shift+F9
Rebuild Project
Generate Ant Build...
Build Artifacts...
Analyze APK...
Run Ant Target Ctrl+Shift+F10

Step4


Build Artifact
All Artifacts
webdemo:war
webdemo:war exploded
Action
Build
Rebuild
Clean
Edit...

项目部署



3.2部署方式二

通过修改 server.xml 配置文件，将项目部署在指定位置。

修改 conf/server.xml，在<Host> </Host>标签中添加：

<Context path="/webdemo" docBase="D:/webdemo" reloadable="true" />

- path:浏览器访问时的路径名。
- docBase:web 项目的 WebRoot 所在的路径，注意是 WebRoot 的路径，不是项目的路径。其实也就是编译后的项目。
- reloadble:设定项目有改动时，tomcat 是否重新加载该项目。

3.3部署方式三

通过新建 xml 配置文件，将项目部署在指定位置。

在 apache-tomcat\conf\Catalina\localhost 目录中新建一个<项目名.xml>的配置文件。

那个新建的 xml 配置文件中，添加如下配置：

<Context docBase="D:/webdemo" reloadable="true"/>

注意：没有 path 属性。

3.4部署方式总结

- 第一种方法比较普通，但是我们需要将项目以 war 包的形式导出并 copy 到 webapps 目录下，多出了两步操作。
- 第二种方法直接在 server.xml 文件中配置，但是从 Tomcat5.0 版本开始后，server.xml 文件作为 Tomcat 启动的主要配置文件，一旦 Tomcat 启动后，便不会再读取这个文件，因此无法再 Tomcat 服务启动后发布 web 项目。
- 第三种方法是最好的，每个项目分开配置，Tomcat 将以\conf\Catalina\localhost 目录下的 xml 文件的文件名作为 web 应用的上下文路径，而不需要在<Context>的 path 属性中配置访问路径，所以该方式在一个 Tomcat 中部署多个项目时更灵活。

三、 Tomcat 优化配置

1 连接优化

优化连接配置.这里以 Tomcat8 的参数配置为例，需要修改 conf/server.xml 文件。

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

1.1port

Tomcat 作为一个网络 server 端,它需要暴露一个 socket 端口来 accept 客户端的链接,可以通过 port 指定。

1.2protocol

使用的网络协议,表示 Tomcat 使用何种方式来接受和处理 Client 端请求,"HTTP/1.1"是默认值。

1.3 connectionTimeout

当 Client 与 Tomcat 建立连接之后,在"connectionTimeout"时间之内,仍然没有得到 Client 的请求数据，此时连接将会被断开。此值的设定需要考虑到网络稳定型，同时也有性能的考虑。默认值为 20000 毫秒，即 20 秒；

1.4 maxHttpHeaderSize

http 头的最大尺寸，默认为 8192，单位为“字节”。

1.5 URIEncoding

http-get 请求中，使用何种字符集对查询字符串进行编码,默认为"iso-8859-1"。

1.6 acceptorThreadCount

默认为 1，表示用于 accept 新链接的线程个数，如果在多核 CPU 架构下，此值可以设置为 2，官方不建议设定超过 2 个的值。

1.7 enableLookups

设置为 true 是否要调用以 request.getRemoteHost()执行 DNS 查找以返回远程客户端的实际主机名。设置为 false 跳过 DNS 查找并改为以字符串形式返回 IP 地址（从而提高性能）。默认情况下，DNS 查找被禁用。

案例：

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

```
maxHttpHeaderSize="8192"
URIEncoding="UTF-8"
acceptorThreadCount="1"
redirectPort="8443" />
```

2 网络优化

2.1 protocol

Tomcat 目前支持：BIO、NIO、NIO2、APR 四种 IO 模型，Tomcat7 默认为 BIO，Tomcat8 默认为 NIO。对于互联网应用，我们应该在 NIO、NIO2 之间做选择，因为它能够有效的提升性能（主要是并发能力），其中 NIO2 即为 AIO，需要 JDK 1.7+。

org.apache.coyote.http11.Http11NioProtocol 非阻塞 Java NIO 连接器

org.apache.coyote.http11.Http11Nio2Protocol 非阻塞 Java NIO2 连接器-APR

org.apache.coyote.http11.Http11AprProtocol 本地连接器。

2.2 pollerThreadCount

表示用于 polling IO 事件的线程个数，默认为 1。在多核 CPU 架构下，我们可以设置为 2 来提高 polling 的能力，官方不建议设置大于 2 的值，因为锁的竞争会导致性能下降，事实上一个线程也足够快速。

2.3 useSendfile

是否开启 sendfile 特性，默认为 true。对于 web 应用而言，通常 project 中还会包含一定数量的静态资源，比如图片、CSS、js、html 等，sendfile 在一定程度上可以提高性能。

2.4 selectorTimeout

选择器阻塞的时间，如果你进行过 NIO 开发，应该知道此参数的含义，默认值为 1000

毫秒。此值不要太大，因为 selector 线程本身还需要用来清理已关闭的链接等。

```
<Connector port="8080" protocol="HTTP/1.1"
                connectionTimeout="20000"
                maxHttpHeaderSize="8192"
                URIEncoding="UTF-8"
                maxKeepAliveRequests="32"
                acceptorThreadCount="1"

                pollerThreadCount="2"
                selectorTimeout="1000"
                useSendfile="true"
                redirectPort="8443" />
```

3 压缩优化

启用压缩，消耗 CPU，减小网络传输大小。

3.1 compression

是否对 http 响应数据启用 Gzip 压缩,可选值为"off"或者"on";这是一个值得商榷的参数;如果开启压缩,意味着较少的网络传输量,但是将消耗一定的 CPU.如果你的应用有较高的 CPU 性能结余,且响应数据均是一些文本字符串,那么开启压缩,会有较大的收益.(并不是所有的浏览器都能够合理的支持 gzip 压缩，特别是低版本）。

3.2 compressionMinSize

如果压缩设置为“on”，则此属性可用于指定压缩输出之前的最小数据量。如果未指定，则此属性默认为“2048”。

3.3 compressibleMimeType

该值为逗号分隔的 MIME 类型列表，可以对其使用 HTTP 压缩。默认值为 text/html,text/xml,text/plain,text/css,text/javascript,application/javascript,application/json,application/xml。

案例：

```
<Connector port="8080" protocol="HTTP/1.1"
                connectionTimeout="20000"
                maxHttpHeaderSize="8192"
                URIEncoding="UTF-8"
                maxKeepAliveRequests="32"
                acceptorThreadCount="1"
                pollerThreadCount="2"
                selectorTimeout="1000"
                useSendfile="true"
                redirectPort="8443"

                compression="on"
                compressionMinSize="2048"
    compressibleMimeType="text/html,text/xml,text/plain,text/css,text/javascript,a
            pplication/javascript" />
```

4 线程池优化

在 Tomcat 中每一个用户请求都是一个线程，所以可以优化线程池提高性能。

4.1方式一

在连接器中优化线程池。

4.1.1 acceptCount

当 Tomcat 请求处理线程池中的所有线程都处于忙碌状态时，此时新建的链接将会被放入到 pending 队列，acceptCount 即是此队列的容量，如果队列已满，此后所有的建立链接的请求(accept)，都将被拒绝。默认为 100。在高并发/短链接较多的环境中，可以适当增大此值。

4.1.2 maxConnections

Tomcat 允许接收和处理的最大链接数 ,对于 BIO 而言此值默认与 maxThreads 参数一样，

对于 NIO 和 NIO2 而言此值默认为 10000。

4.1.3 minSpareThreads

线程池中，保持活跃的线程的最小数量，默认为 10。

4.1.4 maxThreads

用于接收和处理 client 端请求的最大线程数 ,Tomcat 底层将采取线程池的方式来处理客户端请求,此参数标识这线程池的尺寸。maxThreads 意味着 tomcat 能够并发执行 request 的个数.此值默认为 200。一般情况下,在 production 环境中(根据物理机器配置，或者虚拟机的限制来做参考值)，通常会有微调.较大的值并不能提升 Tomcat 的负载能力，事实上"200"个线程数，已经足够大了.一般为 maxThreads=120。如果执行器与此连接器相关联，则此属性将被忽略。

4.2 maxKeepAliveRequests

Tomcat 需要保持的最大请求数，即处于 keepAlive 状态的请求的个数，建议此值为 maxThreads * 0.5,不得大于 maxThreads，否则将得不到预期的效果。-1 表示不限制，1 表示关闭 keepAlive 机制。默认值为 100。

案例：

```
<Connector port="8080" protocol="HTTP/1.1"
                    connectionTimeout="20000"
                    maxHttpHeaderSize="8192"
                    URIEncoding="UTF-8"
                    acceptorThreadCount="1"
                    pollerThreadCount="2"
                    selectorTimeout="1000"
                    useSendfile="true"
                    redirectPort="8443"
```

```
                acceptCount="1024"
                maxConnections="10000"
                minSpareThreads="12"
                maxThreads="128"
                 maxKeepAliveRequests="64"
            />
```

4.3方式二

在执行器中优化线程池。

4.3.1 namePrefix

为 Tomcat 的线程号追加前缀、默认值：tomcat-exec-。

4.3.2 maxQueueSize

最大的等待队列数，超过则拒绝请求

4.3.3 prestartminSpareThreads

在 Tomcat 初始化的时候就初始化 minSpareThreads 的参数值，如果不等于 true，minSpareThreads 的值就没啥效果了

```
<Executor name="tomcatThreadPool"
        namePrefix="catalina-exec-"
        maxThreads="500"
        minSpareThreads="80"
        maxQueueSize="100"
        prestartminSpareThreads="true" />

<Connector executor="tomcatThreadPool" port="8080" protocol="HTTP/1.1"
                connectionTimeout="20000"
                maxHttpHeaderSize="8192"
                URIEncoding="UTF-8"
                acceptorThreadCount="1"
                pollerThreadCount="2"
                selectorTimeout="1000"
```

```
                useSendfile="true"
                redirectPort="8443"
                acceptCount="1024"
            />
```

四、 Tomcat 优化案例

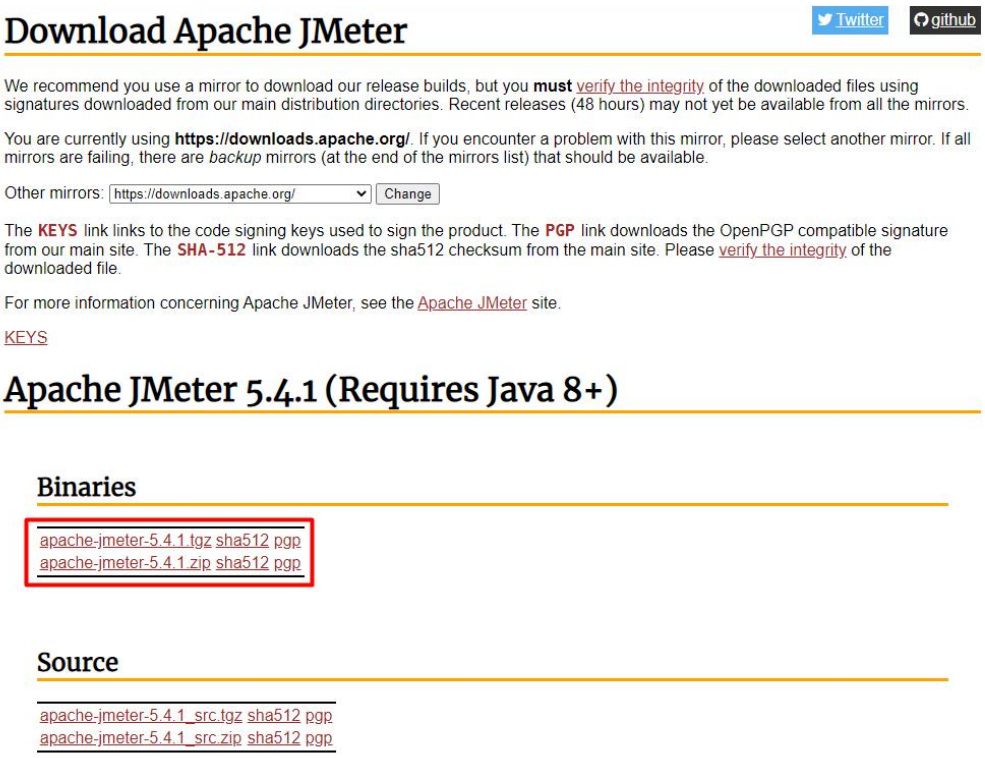
1 测试工具 Jmeter

1.1 Jmeter 简介

Apache JMeter 是 Apache 组织开发的基于 Java 的压力测试工具，是用来负载功能测试和性能测试的纯 Java 开源软件。您可以使用 JMeter 对 web 应用或各种各样服务的性能进行分析和度量。性能测试是指针对重负载、多并发用户流量的 web 应用测试。JMeter 最初被设计用于 Web 应用或 FTP 应用测试，现在扩展到功能测试、数据库服务器测试等。

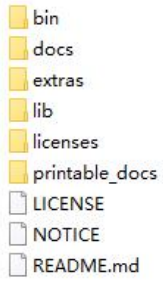
1.2 Jmeter 的下载

Jmeter 下载地址：http://jmeter.apache.org/download_jmeter.cgi

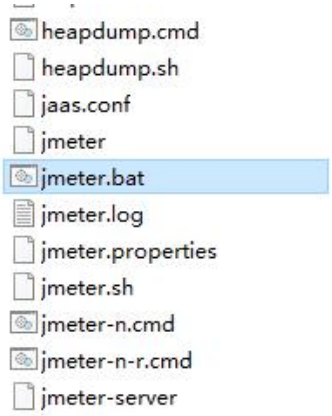


1.3 Jmeter 的使用

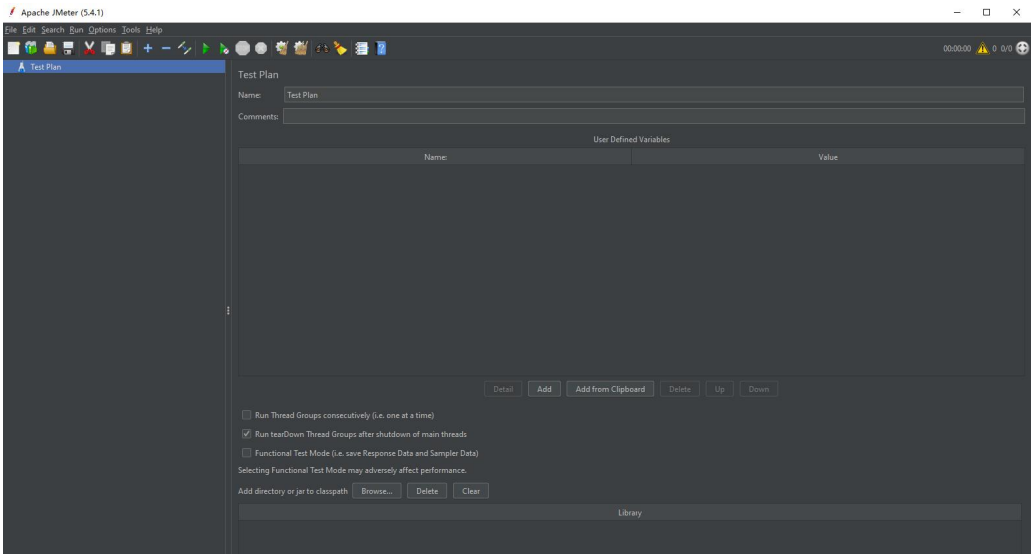
在使用 Jmeter5.4.1 时系统中需要安装 jdk8 及其以上版本的环境。直接将下载好的 zip 压缩包进行解压即可。



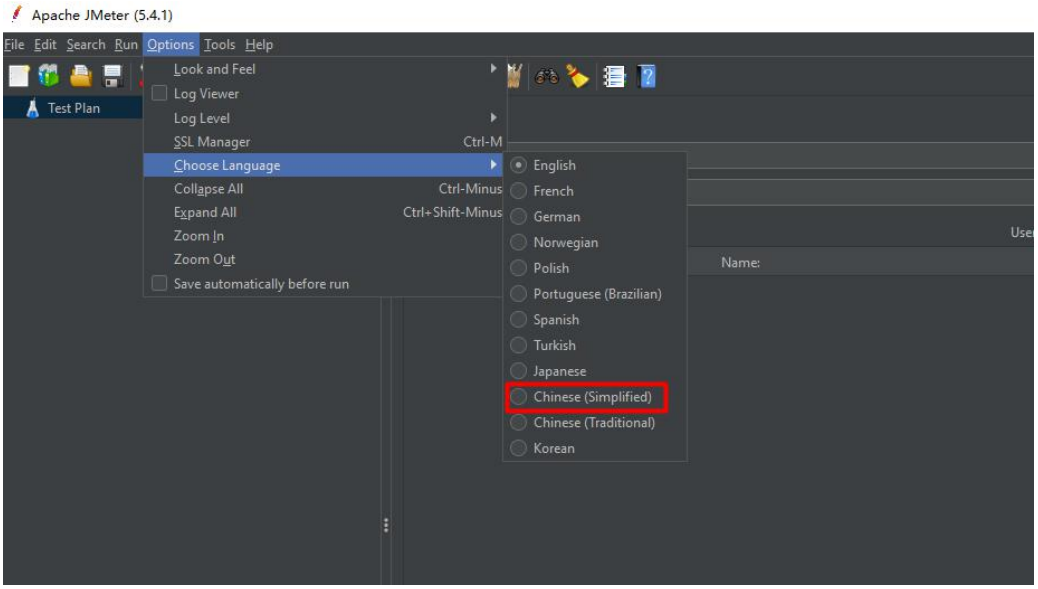
进入 bin 目录，找到 jmeter.bat 文件，双机打开即可启动。

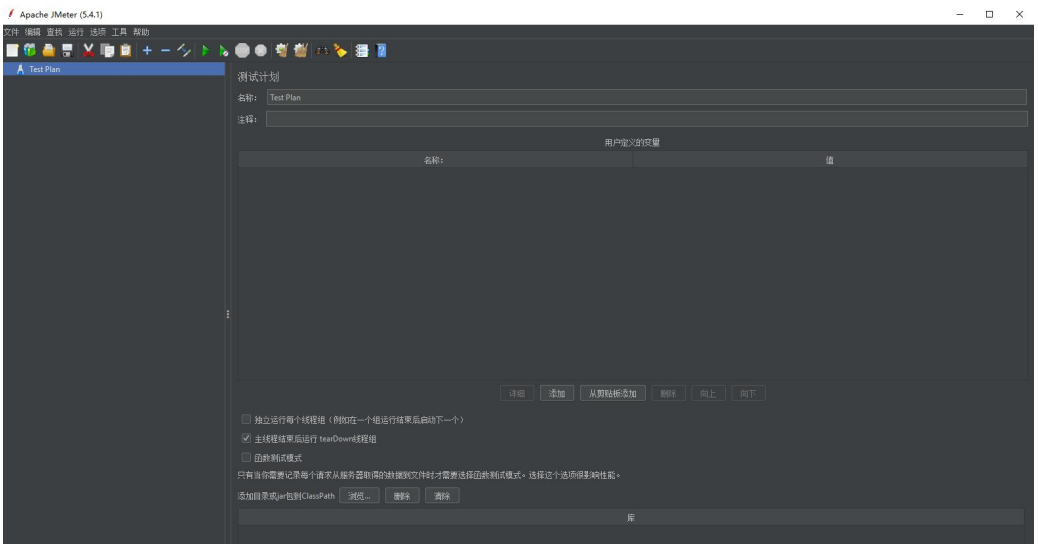


Step1 Jmeter 主界面

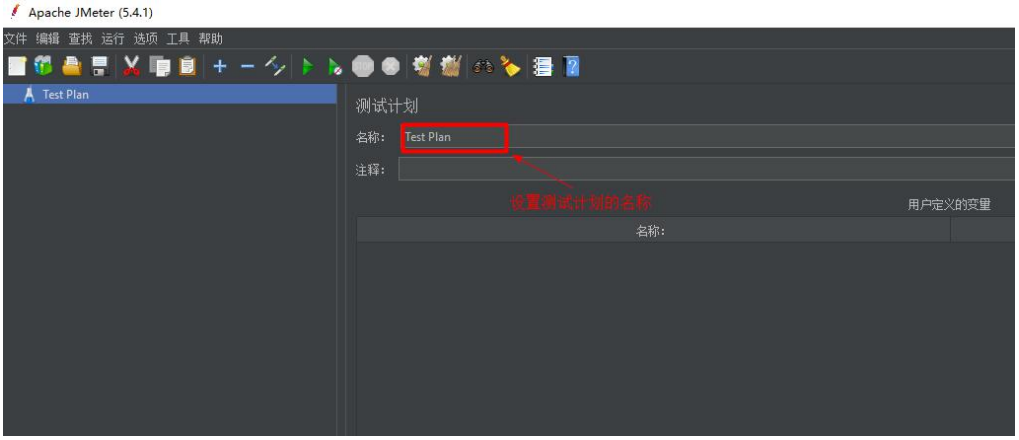


Step2 修改语言

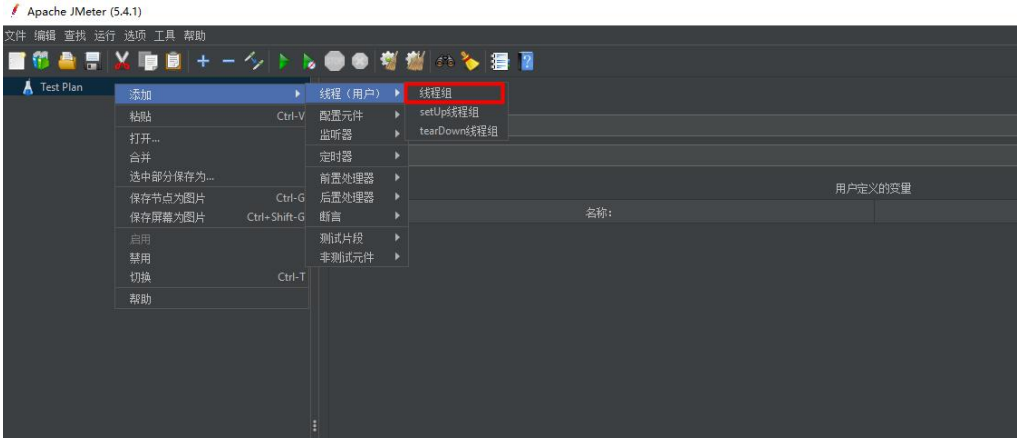


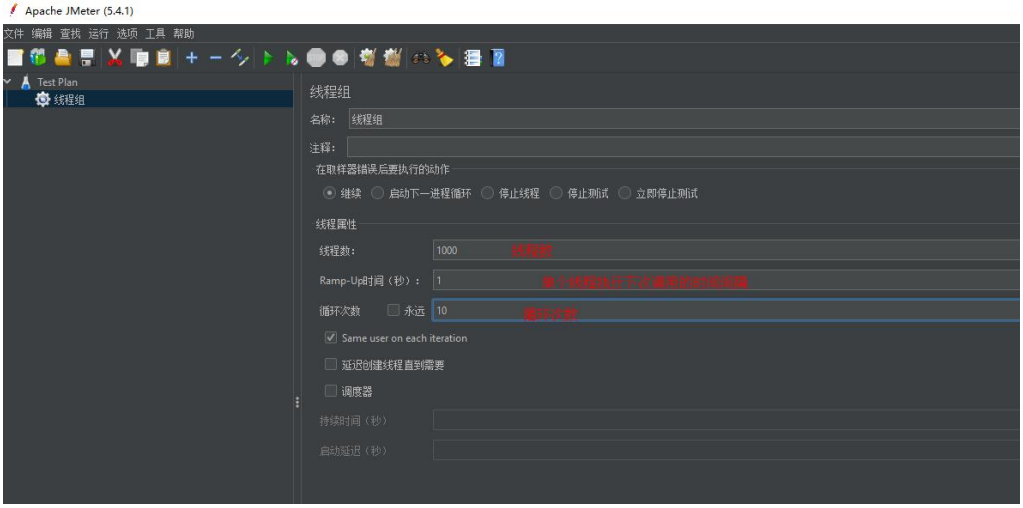


Step3 设置测试计划的名称



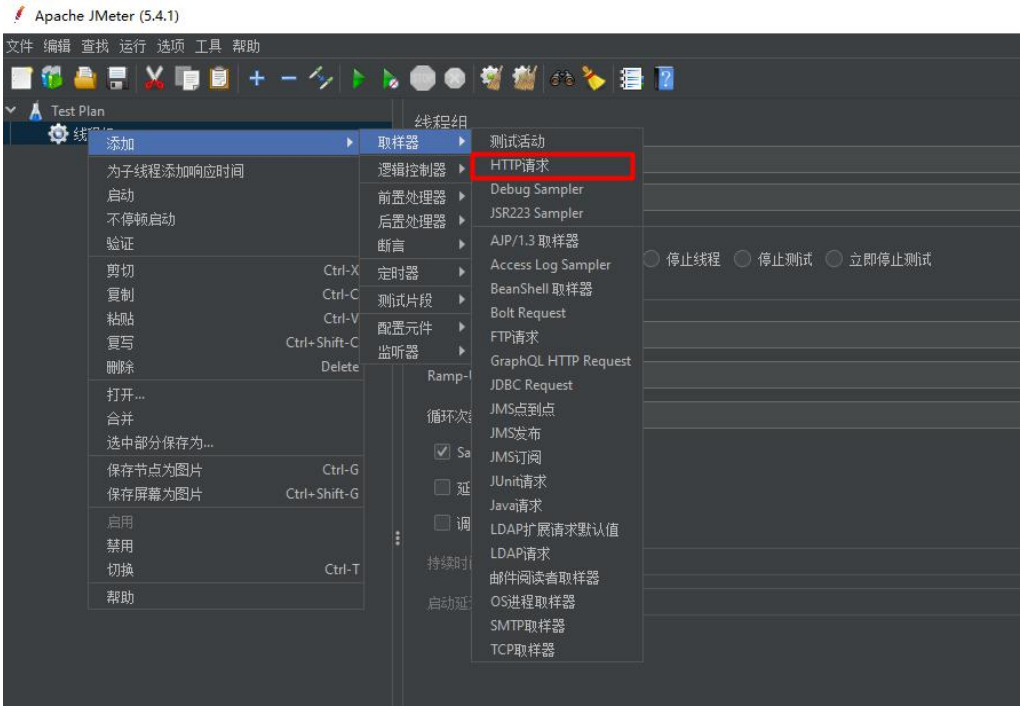
Step4 添加线程组，使用线程模拟用户的并发。



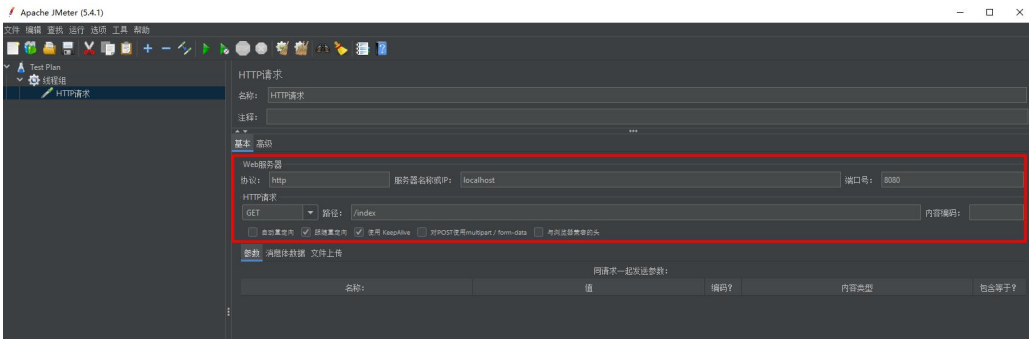


1000 个线程，每个线程循环 10 次，也就是 tomcat 会接收到 10000 个请求。

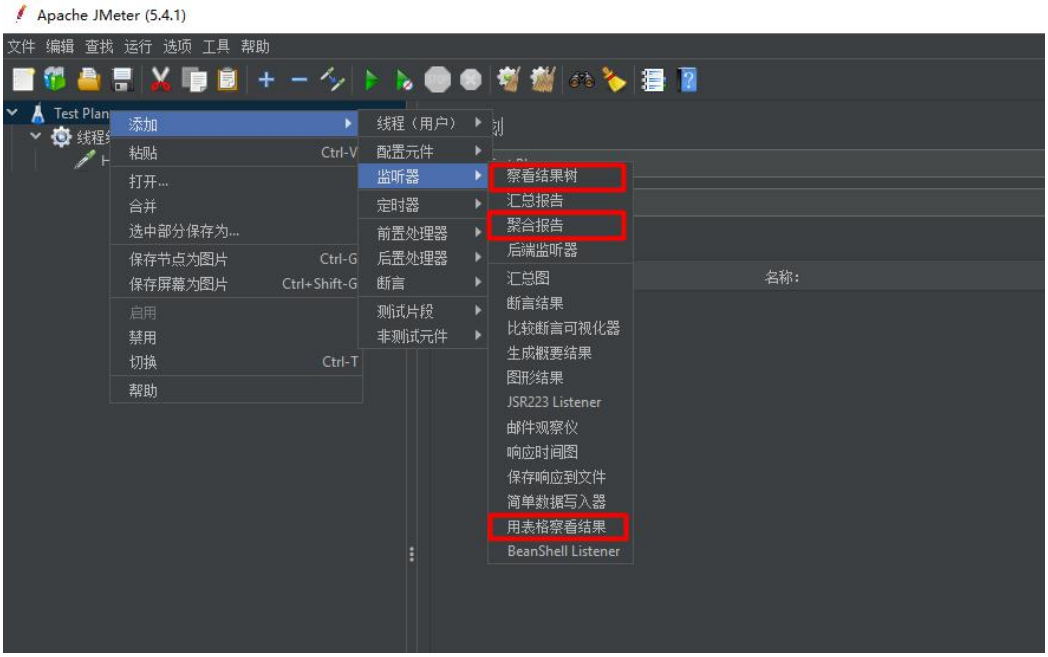
Step5 添加 http 请求



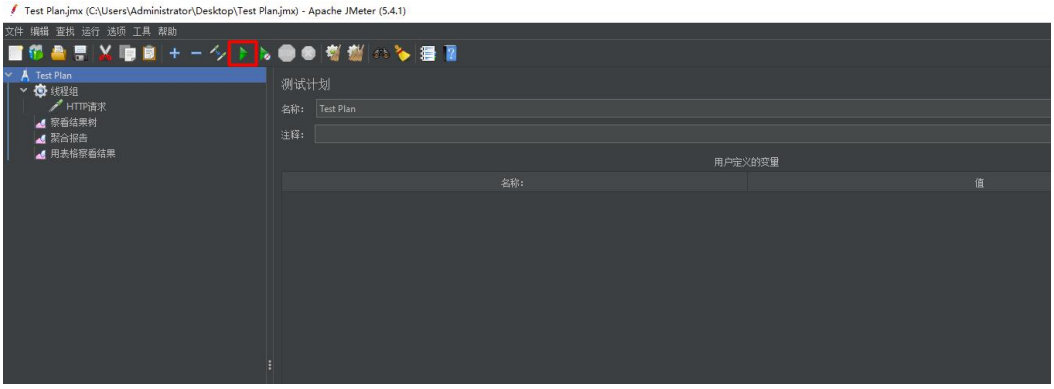
Step6 设置 Http 请求



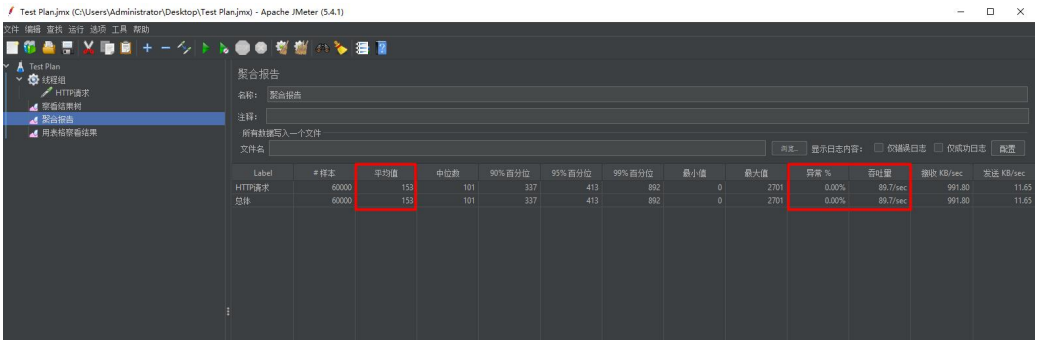
Step6 添加请求监控



Step7 启动与进行接口测试



Step8 查看测试报告



2 搭建测试环境

系统：Linux(CentOS8)

JDK：1.8

Tomcat：apache-tomcat-8.5.65

EditPlus：5.3

3 优化 Tomcat

3.1连接优化

```
<Connector port="8080" protocol="HTTP/1.1"
                connectionTimeout="20000"
                maxHttpHeaderSize="8192"
                URIEncoding="UTF-8"
                acceptorThreadCount="1"
                redirectPort="8443" />
```

3.2线程池优化

3.2.1 设置最大线程数与初始化线程数

```
<Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
        maxThreads="500" minSpareThreads="50" prestartminSpareThreads="true"/>
```

3.2.2 设置最大等待队列数

```
<!--最大等待数为 100-->
<Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
        maxThreads="500"  minSpareThreads="50"  prestartminSpareThreads="true"
maxQueueSize="100"/>
```

3.3网络优化

```
<Connector port="8080" protocol="org.apache.coyote.http11.Http11Nio2Protocol"
                connectionTimeout="20000"
                maxHttpHeaderSize="8192"
                URIEncoding="UTF-8"
                acceptorThreadCount="1"
                redirectPort="8443" />
```

4 优化 JVM

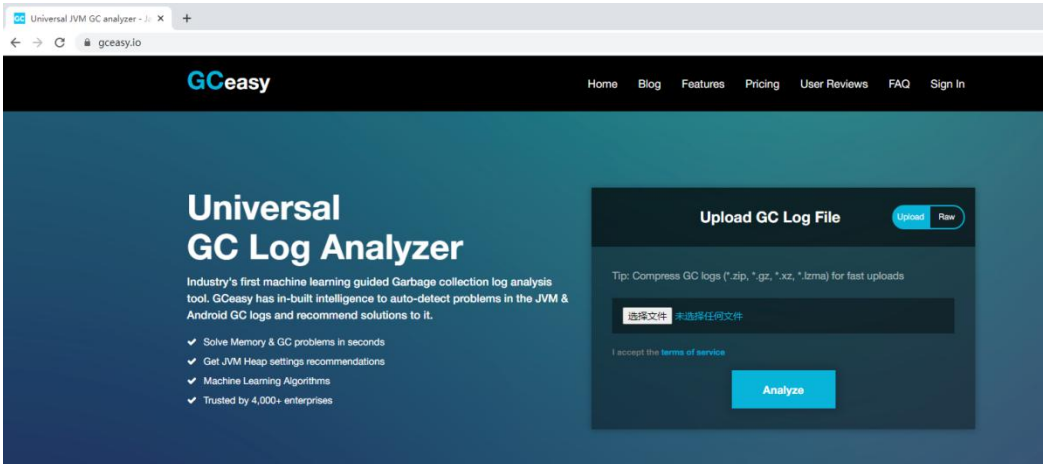
4.1设置并行垃圾回收器

在/bin/catalina.sh 文件第一行添加如下参数，gc 日志输出到/logs/gc.log

年轻代、老年代均使用并行收集器，初始堆内存 64M，最大堆内存 512M

```
JAVA_OPTS="-XX:+UseParallelGC -XX:+UseParallelOldGC -Xms64m -Xmx512m -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC -Xloggc:../logs/gc.log"
```

通过 gceasy.io 网站分析内存情况



4.2调整年轻代大小

将初始堆大小设置为 128m，最大为 1024m，初始年轻代大小 64m，年轻代最大 256m。

```
JAVA_OPTS="-XX:+UseParallelGC -XX:+UseParallelOldGC -Xms128m -Xmx1024m -XX:NewSize=64m -XX:MaxNewSize=256m -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC -Xloggc:../logs/gc.log"
```

4.3设置 G1 垃圾回收器

设置 G1 垃圾收集器，设置了最大停顿时间 100 毫秒，初始堆内存 128m，最大堆内存 1024m。

```
JAVA_OPTS="-XX:+UseG1GC -XX:MaxGCPauseMillis=100 -Xms128m -Xmx1024m -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC -Xloggc:../logs/gc.log"
```

4.4JVM 内存优化参数说明

```
JAVA_OPTS="-Dfile.encoding=UTF-8-server -Xms1024m -Xmx2048m -XX:NewSize=512m -XX:MaxNewSize=1024m -XX:PermSize=256m -XX:MaxPermSize=256m -XX:NewRatio=4 -XX:MaxTenuringThreshold=10 -XX:+DisableExplicitGC"
```

file.encoding 默认文件编码

-Xmx2048m 设置 JVM 最大可用内存为 2048MB

-Xms1024m 设置 JVM 最小内存为 1024m。此值可以设置与-Xmx 相同，以避免每次垃圾回收完成后 JVM 重新分配内存。

-XX:NewSize 设置年轻代大小

-XX:MaxNewSize 设置最大的年轻代大小

-XX:PermSize 设置永久代大小

-XX:MaxPermSize 设置最大永久代大小

-XX:NewRatio=4 设置年轻代（包括 Eden 和两个 Survivor 区）与终身代的比值（除去永

久代）。设置为 4，则年轻代与终身代所占比值为 1：4，年轻代占整个堆栈的 1/5

-XX:MaxTenuringThreshold=0 设置垃圾最大年龄，默认为：15。如果设置为 0 的话，则年轻代对象不经过 Survivor 区，直接进入年老代。对于年老代比较多的应用，可以提高效率。

-XX:+DisableExplicitGC 这个将会忽略手动调用 GC 的代码使得 System.gc()的调用就会变成一个空调用，完全不会触发任何 GC。

五、 Tomcat 优化总结

1 Tomcat 端口

Tomcat 的缺省端口号是 8080。

修改 Tomcat 端口号：

- 找到 Tomcat 目录下的 conf 目录。
- 进入 conf 目录里面找到 server.xml 文件。
- 打开 server.xml 文件。
- 修改<Connector>标签中的 port 属性的值。

```
<Connector port=”8080” protocol=”HTTP/1.1” connectionTimeout=”20000” redirectPort=”8443” maxThreads=”500” minSpareThreads=”20” disableUploadTimeout=”true” enableLookups=”false” URIEncoding=”UTF-8” />
```

2 如何优化 Tomcat

2.1优化连接

```
<Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               maxHttpHeaderSize="8192"
               URIEncoding="UTF-8"
               acceptorThreadCount="1"
```

```
                redirectPort="8443"/>
```

2.2优化线程池

```
    <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
              maxThreads="500"    minSpareThreads="50"    prestartminSpareThreads="true"
maxQueueSize="100"/>
```

2.3优化网络

Tomcat7 protocol="HTTP/1.1"默认 BIO 模型，Tomcat8 protocol="HTTP/1.1"默认 NIO 模型。

org.apache.coyote.http11.Http11NioProtocol 非阻塞 Java NIO 连接器

org.apache.coyote.http11.Http11Nio2Protocol 非阻塞 Java NIO2 连接器-APR

org.apache.coyote.http11.Http11AprProtocol 本地连接器。

```
    <Connector port="8080" protocol="org.apache.coyote.http11.Http11Nio2Protocol"
                connectionTimeout="20000"
                maxHttpHeaderSize="8192"
                URIEncoding="UTF-8"
                acceptorThreadCount="1"
                redirectPort="8443" />
```

2.4优化 JVM

```
    JAVA_OPTS="-Dfile.encoding=UTF-8-server -Xms1024m -Xmx2048m -XX:NewSize=512m
-XX:MaxNewSize=1024m -XX:PermSize=256m -XX:MaxPermSize=256m -XX:NewRatio=4
-XX:MaxTenuringThreshold=10 -XX:+DisableExplicitGC"
```