

qhs 的用户手册

(qhs 是快速 http 服务器的简称。)

1、qhs 的技术特点

qhs(即快速 http 服务器)是一款参考 rfc7540、rfc7541、rfc6455、rfc8441 而开发设计的 http 服务器，具有以下技术特点：

- (1)、支持单线程模型和多线程模型。
- (2)、可分析来自浏览器的表单数据(Form Data)。
- (3)、支持开发 HTTP 服务模块(Service Module)用于生成动态页面。
- (4)、支持开发 URL 重写器用于重写 URL。
- (5)、支持服务器推送(Server Push)。
- (6)、支持开发全局对象、线程特定对象、连接特定对象等等。
- (7)、跟据具体的 URL 路径，对 URL 路径追加缺省页面文件名、对 http 答复增加定制的头域。
- (8)、支持 WebSocketsOverHTTP/2。

2、qhs 的运行环境

内存：至少 100M 字节。

硬件环境：本软件在 AMD-X86-64 个人计算机开发，可在 AMD-X86-64 计算机运行。

操作系统：本软件可在 UbuntuKylin16、UbuntuKylin19、Ubuntu18 桌面、CentOS6、CentOS7 等 Linux 操作系统上运行。qhs 是一款纯 Java 程序，理论上，qhs 可运行在安装有 JDK11 的一切 Linux 操作系统。

支撑软件：JDK11。

3、示例的运行环境

本手册的所有示例(包括演示文件、脚本命令)需要的运行环境：

- (1)、建议安装 UbuntuKylin16-X86-64 操作系统。用户可在 <http://www.ubuntukylin.com> 下载 UbuntuKylin16。UbuntuKylin19-X86-64 操作系统也是可接受的。
- (2)、在操作系统创建有 http 账户，而且 http 账户的主文件夹(即\$HOME 目录)为/home/http 目录，以 http 账户登录操作系统桌面。
- (3)、OpenJDK11 安装在/usr/java/jdk-11 目录。OpenJDK11 用于运行 ApacheNetBeansIDE13、运行示例。用户可在 <http://jdk.java.net> 下载 OpenJDK11。
- (4)、qhs 安装在/home/http/qhs3.3 目录。
- (5)、Chrome91 以上的浏览器或者 Firefox89 以上的浏览器。在浏览器的“证书机构”栏或“证书颁发机构”栏或“授权中心”栏导入/home/http/qhs3.3/sslFiles/ca.crt 证书文件。ca.crt 证书文件的创建可参考第 [6.1.1](#) 节。
- (6)、安装 ApacheNetBeansIDE13。用户可在 <http://netbeans.apache.org> 下载 ApacheNetBeansIDE13。

4、安装 qhs

qhs 的安装包为 qhs3.3-inst.jar 文件。用户需要在终端窗口输入、运行命令来启动安装程序。

安装程序的运行参数有：

- ①、--InstPath=安装目录。
- ②、--JavaProgPath=可执行程序 java 的绝对路径。

如：在终端窗口运行以下命令来安装 qhs：

```
/usr/java/jdk-11/bin/java -jar qhs3.3-inst.jar \  
--InstPath=/home/http/qhs3.3 \  
--JavaProgPath=/usr/java/jdk-11/bin/java;
```

上一命令把 qhs 安装在/home/http/qhs3.3 目录。

5、启动 qhs、终止 qhs 的运行

在终端运行 qhs 安装目录内部的 startup.sh 脚本。如：qhs 的安装目录为/home/http/qhs3.3，运行以下脚本启动 qhs：

```
/home/http/qhs3.3/startup.sh &
```

若是安装 qhs 后立即运行 startup.sh 脚本而且按照第 3 节对浏览器导入了 ca.crt，可在浏览器的地址栏输入 https://127.0.0.1:10000 尝试对 qhs 发送请求，如果浏览器显示“你好”页面，说明成功启动了 qhs。在 UbuntuKylin16、CentOS6-GNOME 等桌面，如果 base.xml 配置文件内部的 SystemTray 值设置为 true 并且成功启动 qhs，桌面的系统驻留区就显示一个名为“qhs”的图标。

可在终端运行 qhs 安装目录内部的 shutdown.sh 脚本来终止 qhs 的运行。运行 shutdown.sh 脚本时，可选的设置明文密码参数。如：qhs 的安装目录为/home/http/qhs3.3，运行以下脚本：

```
/home/http/qhs3.3/shutdown.sh 123abc;
```

123abc 是安装 qhs 以后的初始密码，用户可以编辑/home/http/qhs3.3/config/shutdown.xml 配置文件来更改密码，shutdown.xml 内部的注释说明了更改密码的方法。

6、配置

qhs 的所有配置都以 XML 文档进行组织、表达，表达配置的 XML 文档都存放在 qhs 安装目录内部的 config 子目录(即配置目录)。如果 qhs 的安装目录为/home/http/qhs3.3，那么配置目录为/home/http/qhs3.3/config。

在启动 qhs 的过程中，qhs 参考配置目录内部的文件进行初始化并创建大量可重复使用的对象。可重复使用的对象包括 HTTP 服务模块、WS 服务模块、http 请求分析器、用于分析请求 URI 与重写 URL 的 URIParser 对象等等。

用户可运行 Linux 的 gedit、pluma 等文本编辑器编辑 XML 文档。

6.1、基本配置

基本配置存放在/home/http/qhs3.3/config/base.xml 文件，base.xml 文件内部包含有比较多的说明注释。

6.1.1、创建密钥库文件、签证书

浏览器可与 qhs 进行 TLSv1.3 加密通信。

创建密钥库文件以及签证书的操作是复杂的，如果用户是新用户而且想了解 qhs 的其它开发，用户可以跳过本节。为了后续章节的例示以及浏览器能够成功连接到 qhs，用户必须按照第 3 节的说明对浏览器导入 ca.crt 证书文件。

在启动 qhs 以前，用户需要编辑 base.xml 配置文件，对其内部的 SSLServerSocket 元素进行适当的更改，设置 SSLServerSocket>KeyStore 元素值为密钥库(keystore)文件的路径。qhs 对密钥库文件有两个要求：

- (1)、一个密钥库文件只存放有一个密钥对(key pair)以及与密钥对相关联的证书。
一个密钥对由一个公钥(public key)以及一个私钥(private key)组成。
- (2)、对于一个密钥库文件，用于加密密钥库文件的密码(password)与用于加密私钥的密码必须完全相同。即用于一个密钥库文件的所有密码必须完全相同。

作为演示，在终端输入的以下命令创建的密钥库只适用于绑定到 127.0.0.1 地址的服务器套接字，要理解下面命令的作用，用户需对 JDK 的 keytool 程序有了解：

```

cd ~;
export JDK_HOME="/usr/java/jdk-11";
export OUTPUT_DIR="/home/http/sslFiles";
export PASSWORD="123456";
export DAY_COUNT="7000";
rm -Rvf $OUTPUT_DIR;
mkdir -vp $OUTPUT_DIR;
$JDK_HOME/bin/keytool -genkeypair -storepass $PASSWORD -keystore $PASSWORD \
    -keyalg RSA -keysize 1024 -keystore $OUTPUT_DIR/ca.ks -alias ca -ext bc:c -dname "CN=ca" \
    -validity $DAY_COUNT;

$JDK_HOME/bin/keytool -exportcert -storepass $PASSWORD -rfc \
    -keystore $OUTPUT_DIR/ca.ks -alias ca -file $OUTPUT_DIR/ca.crt;

$JDK_HOME/bin/keytool -genkeypair -storepass $PASSWORD -keystore $PASSWORD \
    -keyalg RSA -keysize 1024 -keystore $OUTPUT_DIR/httpServer.ks -alias httpServer \
    -dname "CN=127.0.0.1" -validity $DAY_COUNT;

$JDK_HOME/bin/keytool -certreq -storepass $PASSWORD -keystore $OUTPUT_DIR/httpServer.ks \
    -alias httpServer -file $OUTPUT_DIR/httpServer.csr;

$JDK_HOME/bin/keytool -gencert -storepass $PASSWORD -rfc -keystore $OUTPUT_DIR/ca.ks \
    -ext san=ip:127.0.0.1 \
    -alias ca -infile $OUTPUT_DIR/httpServer.csr -outfile $OUTPUT_DIR/httpServer.crt \
    -validity $DAY_COUNT;

$JDK_HOME/bin/keytool -importcert -storepass $PASSWORD -keystore $OUTPUT_DIR/httpServer.ks \
    -file $OUTPUT_DIR/ca.crt -noprompt;

$JDK_HOME/bin/keytool -importcert -storepass $PASSWORD -keystore $OUTPUT_DIR/httpServer.ks \
    -alias httpServer -file $OUTPUT_DIR/httpServer.crt;

```

运行上述命令后对 base.xml 配置文件进行更改。在 base.xml 配置文件，设置 SSLServerSocket>KeyStore 元素值为密钥库文件的路径，设置 SSLServerSocket>Password 元素值为解密密钥库文件的密码：

```

.....
<SSLServerSocket>
  <Created>true</Created>
  .....
  <KeyStore>/home/http/sslFiles/httpServer.ks</KeyStore>
  <Password>123456</Password>
  .....
</SSLServerSocket>
.....

```

在浏览器管理证书时，应在“授权机构”栏或“证书颁发机构”栏导入上面命令创建的/home/http/sslFiles/ca.crt 文件。

如果 base.xml 配置文件内部的 SSLServerSocket>AuthClient 元素值等于 true，就必须创建个人数字证书。在终端输入以下命令创建符合 PKCS#12 格式的个人证书文件/home/http/sslFiles/person.p12，person.p12 作为个人数字证书导入到浏览器：

```

openssl req -newkey rsa:1024 -keyout $OUTPUT_DIR/person.key -out $OUTPUT_DIR/person.csr;

$JDK_HOME/bin/keytool -gencert -storepass $PASSWORD -rfc -keystore $OUTPUT_DIR/ca.ks \
    -alias ca -infile $OUTPUT_DIR/person.csr -outfile $OUTPUT_DIR/person.crt -validity $DAY_COUNT;

openssl pkcs12 -in $OUTPUT_DIR/person.crt -inkey $OUTPUT_DIR/person.key -export \
    -out $OUTPUT_DIR/person.p12;

```

在浏览器管理证书时，应在“您的证书”栏导入上面命令创建的/home/http/sslFiles/person.p12 文件。

Chrome 浏览器要求服务器证书包含有 SAN 值，上面命令签发的服务器证书的 SAN 值等于 127.0.0.1。签发 SAN 值等于多个 IPv4 地址的服务器证书的演示见/home/http/qhs3.3/docs/create-certs-1.txt 文件。签发 SAN 值等于多个域名的服务器证书的演示见/home/http/qhs3.3/docs/create-certs-2.txt 文件。

6.2、调试输出器的配置

调试输出器的配置存放在/home/http/qhs3.3/config/debugOutputer.xml 文件，debugOutputer.xml 文件内部包含有说明注释。

6.3、错误答复创建器的配置

错误答复创建器的配置存放在/home/http/qhs3.3/config/errorResponseCreator.xml 文件，errorResponseCreator.xml 文件内部包含有说明注释。

6.4、文件检索器的配置

文件检索器的配置存放在/home/http/qhs3.3/config/fileRetriever.xml 文件，fileRetriever.xml 文件内部包含有说明注释。

6.5、HTTP 服务模块的配置

HTTP 服务模块的配置存放在/home/http/qhs3.3/config/httpServiceModules.xml 文件，httpServiceModules.xml 文件内部包含有说明注释。

6.6、媒体类型的配置

媒体类型配置存放在/home/http/qhs3.3/config/mediaTypes.xml 文件，mediaTypes.xml 文件内部包含有说明注释。

6.7、缺省页面文件名、定制头域、推送 URL 等配置

缺省页面文件名、定制头域、推送 URL 等配置存放在/home/http/qhs3.3/config/urlPaths.xml 文件。定制头域是用户自主设计的、不是 qhs 自动生成的头域。URL 路径是存在于 URL 内部的 path 部件。如：

`http://127.0.0.1/abc/def/ghi.html?jkl=mno&pqr=stu`
└──────────┬──────────┘
 └─▶ URL 路径：/abc/def/ghi.html

在 qhs 运行的过程中，如果 qhs 未创建 URL 重写器，qhs 就从 http 请求内部的请求 URI(Request-URI)中截取 URL 路径或者从推送 URL 中截取 URL 路径。如果 qhs 创建了 URL 重写器，qhs 就从经过重写以后的 URL 中截取 URL 路径。如果 qhs 发现截取的 URL 路径的最右端不存在文件名，qhs 就尝试在 URL 路径的最右端追加缺省页面文件名，并且把最终形成的 URL 路径用于构建资源文件路径、构建 HTTP 服务模块 ID 等。

在 qhs 启动的过程中，qhs 引用 urlPaths.xml 文件上的除 Config 元素(Config 元素等于根目录)、Header 元素、Push 元素等以外的其它元素并且创建节点树(node tree)，节点树保留了原有的元素之间的父子关系，节点树内部的节点的名称等同于元素的名称(即元素的标记)，而且所有节点可选的设置缺省页面文件名(由 DefaultPage 属性说明)、一个以上的定制答复头域(由一个以上的 Header 元素说明)、一个以上的推送 URL(由一个以上的 Push 元素说明)等。Header、Push 等元素是保留元素。

在 urlPaths.xml，如果一个非保留元素具有 DefaultPage 属性，并且 DefaultPage 属性值说明的缺省页面文件名是

合规的，就可以认定这个元素设置有缺省页面文件名。Config 元素可以设置有 DefaultPage 属性。如果 Config 元素未设置有 DefaultPage 属性，则默认 Config 元素设置的缺省页面文件名为 index.html。

在 urlPaths.xml，如果一个非保留元素包含有 Header 元素，并且 Header 元素具有 Name(用于说明头域名)、Value(用于说明头域值)等属性以及 Name、Value 等属性值是合规的，就可以认定这元素(即 Header 元素的直接父元素)设置有定制头域。一个 Header 元素只能说明一个定制头域。一个父元素可以包含有多个 Header 元素，即一个父元素可以设置有多个定制头域。不要使用 Header 元素说明 date、server、last-modified、etag 等头域，这些头域由 qhs 自动生成。

可以使用 Header 元素说明 expires 头域。如：

```
.....
<Header Name="expires" Value="2020/12/3 4:5:6" />
.....
```

上面的 Value 属性值等于"2020/12/3 4:5:6"，这属性值是配置值而不是标准的 expires 头域值，qhs 分析配置值并且生成标准的 expires 头域值，用于 expires 头域的配置值格式：年/月/日 时:分:秒

可以对 Header 元素增加 WriteType 属性。如：

```
.....
<Header Name="hn1" Value="hv1" WriteType="false" />
<Header Name="hn2" Value="hv2" WriteType="true" />
.....
```

WriteType 属性值等于"true"表示把头域写入到答复动态表，否则，不把头域写入答复动态表。如果 Header 元素缺少 WriteType 属性，就把头域写入答复动态表，效果相当于对 Header 元素增加了 WriteType 属性并且 WriteType 属性值等于"true"。

在 urlPaths.xml，Push 元素用于说明推送数据。Push 元素的 URL 属性用于说明推送 URL。Push 元素的 PartOfLinkHeader 属性用于说明 link 头域的部分域值。如果父元素包含有 Push 元素，则说明父元素所指向的静态资源文件可以发起推送操作。urlPaths.xml 的内容如：

```
<Config>
  <html-pages>
    <page1.html>
      <Header Name="hn1" Value="hv1" WriteType="false" />
    </page1.html>
  </html-pages>
  <pics>
    <pic1.png>
      <Header Name="hn2" Value="hv2" WriteType="true" />
    </pic1.png>
  </pics>
  <abc>
    <def.html>
      <Push URL="/html-pages/page1.html" />
      <Push URL="/html-pages/page2.html" />
      <Push URL="/pics/pic1.png" PartOfLinkHeader=""; rel=preload; as=image" />
      <Push URL="/get-time.id" />
    </def.html>
  </abc>
</Config>
```

从 Config 元素遍历到 def.html 元素的路径形成一个 URL 路径：/abc/def.html，而且 def.html 元素(即 Push 元素的父元素)包含有 4 个 Push 元素，在 qhs 发送文档根目录内部的/abc/def.html 文件以前，qhs 自动推送/html-pages/page1.html、/html-pages/page2.html、/pics/pic1.png、/get-time.id 等 4 个 URL。在推送这 4 个 URL 时，qhs 为这 4 个 URL 分别生成推送答复(pushes response)，并且把推送答复发送给浏览器等客户端。为/html-pages/page1.html 生成推送答复时，对推送答复增加"hn1: hv1"头域，但是不把"hn1: hv1"头域写入答复动态表。为/pics/pic1.png 生成推送答复时，对推送答复增加"hn2: hv2"头域，并且把"hn2: hv2"头域写入答复动态表。qhs 在发送文档根目录内部的/abc/def.ht

ml 文件的 http 答复上, 增加了以下多个 link 头域:

```
link: </html-pages/page1.html>
link: </html-pages/page2.html>
link: </pics/pic1.png>; rel=preload; as=image
link: </get-time.id>
```

为了检索设置在节点上的数据, qhs 使用 URL 路径在节点树上定位节点。定位节点时, qhs 按照观察 URL 路径的从左到右的顺序, 使用 URL 路径左侧的路径段与低深度的节点的名称进行比较, 使用 URL 路径右侧的路径段与高深度的节点的名称进行比较。如果路径段与节点的名称相同, 就继续使用下一路径段与子节点(即更高深度的节点)的名称进行比较。有 3 种情况终止定位节点操作:

- (1)、路径段与节点的名称不相同。
- (2)、没有更多的路径段与节点的名称进行比较。
- (3)、没有更多的节点名称与路径段进行比较。

在默认的情况下, 如果子元素未设置有缺省页面文件名、定制头域等数据, 子元素就自动从父元素继承缺省页面文件名、定制头域等数据。如果用户要求子元素从父元素只继承部分数据, 可以对子元素增加 Inherited 属性。Inherited 属性值等于"headers"表示子元素从父元素继承定制头域。Inherited 属性值等于"defaultPage"表示子元素从父元素继承缺省页面文件名。Inherited 属性值等于"defaultPage, headers"表示子元素从父元素同时继承缺省页面文件名以及定制头域等数据, 这属于默认的情况, 这等同于一个元素没有增设 Inherited 属性的情况。Inherited 属性值等于空串表示子元素不从父元素继承任何数据。需要提醒的是使用 Push 元素说明的推送 URL 是不可继承的。如:

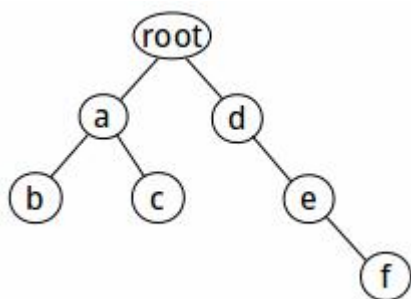
假设 urlPaths.xml 文件的内容:

```
<Config DefaultPage="first-page.html">
  <Header Name="n11" Value="v11" />
  <Header Name="n12" Value="v12" />

  <a DefaultPage="plain.txt">
    <Header Name="n2" Value="v2" />

    <b DefaultPage="pic.png" Inherited="defaultPage" />
    <c>
      <Header Name="n3" Value="v3" />
    </c>
  </a>
  <d>
    <e Inherited="headers">
      <Header Name="n4" Value="v4" />
      <f DefaultPage="about.html" />
    </e>
  </d>
</Config>
```

上面的 urlPaths.xml 文件内容可使 qhs 创建以下节点树:



在节点树上, 其中:

- (1)、root 节点(对应 Config 元素)等于根目录, root 节点设置有"first-page.html"缺省页面文件名、"n11: v11"定制头域、"n12: v12"定制头域。

- (2)、a 节点设置有"plain.txt"缺省页面文件名、"n2:v2"定制头域。
- (3)、b 节点设置有"pic.png"缺省页面文件名，但不从 a 节点继承定制头域。
- (4)、c 节点设置有"n3:v3"定制头域。
- (5)、d 节点没有任何设置。
- (6)、e 节点设置有"n4:v4"定制头域，但不从 d 节点继承缺省页面文件名。
- (7)、f 节点设置有"about.html"缺省页面文件名。

下面“=>”左侧的 URL 路径的最右端不存在文件名，经过 qhs 追加缺省页面文件名后等于下面“=>”右侧的 URL 路径：

```

/                =>  /first-page.html
/x/y/z/          =>  /x/y/z/first-page.html
/a/              =>  /a/plain.txt
/a/x/y/z/        =>  /a/x/y/z/plain.txt
/a/b/            =>  /a/b/pic.png
/a/b/x/y/z/      =>  /a/b/x/y/z/pic.png
/a/c/            =>  /a/c/plain.txt
/a/c/x/y/z/      =>  /a/c/x/y/z/plain.txt
/d/              =>  /d/first-page.html
/d/x/y/z/        =>  /d/x/y/z/first-page.html
/d/e/            =>  /d/e/index.html，e 节点不继承缺省页面文件名，追加了默认的"index.html"缺省页面文件名。
/d/e/x/y/z/      =>  /d/e/x/y/z/index.html，index.html 继承自 e 节点。
/d/e/f/          =>  /d/e/f/about.html
/d/e/f/x/y/z/    =>  /d/e/f/x/y/z/about.html

```

对 http 答复追加定制头域，只适用于发送静态资源文件的情况。下面“=>”左侧的 URL 路径(*表示 URL 路径是经过追加缺省页面文件名后而形成的)用于生成文件路径，并且假设相应的静态资源文件存在于文档根目录。下面“=>”右侧的头域是对 http 答复追加的定制头域：

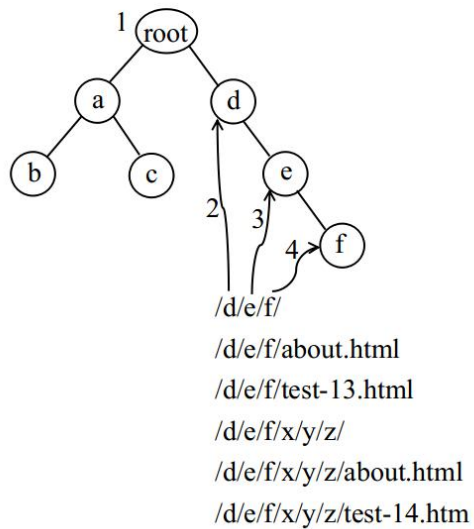
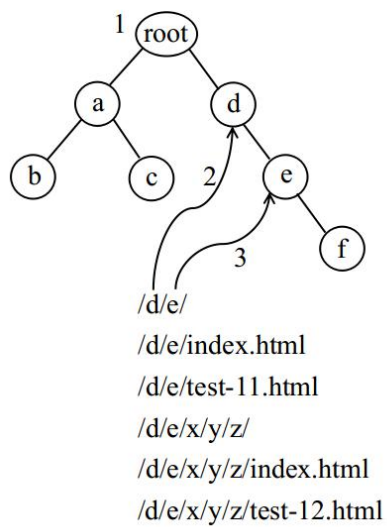
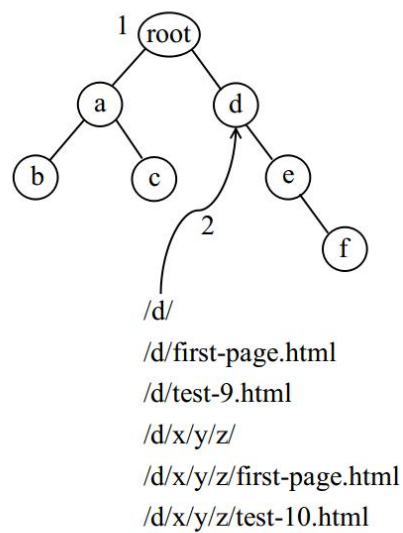
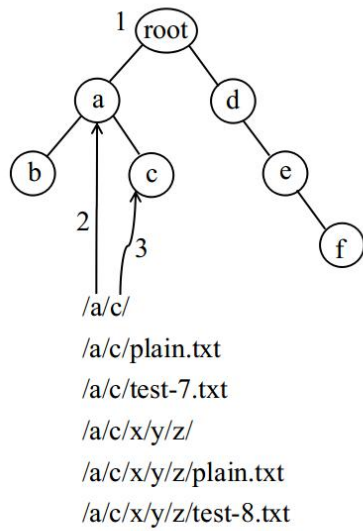
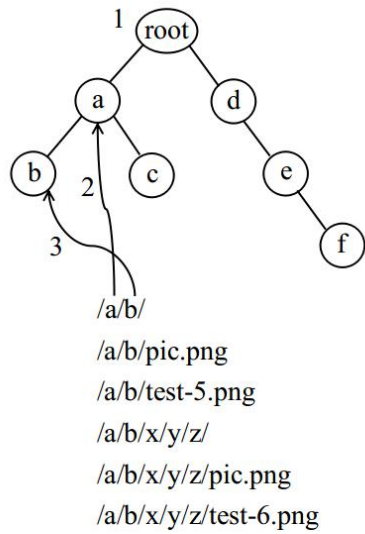
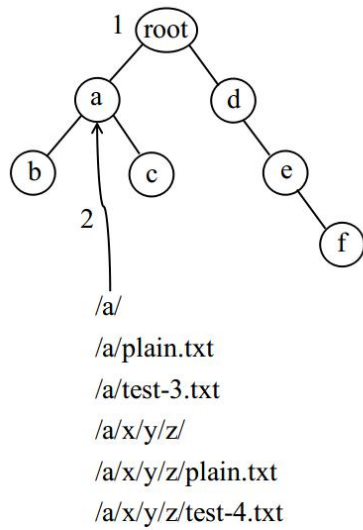
```

* /first-page.html    =>  "n11:v11", "n12:v12"
* /test-1.html        =>  "n11:v11", "n12:v12"
* /x/y/z/first-page.html =>  "n11:v11", "n12:v12"
* /x/y/z/test-2.html  =>  "n11:v11", "n12:v12"
* /a/plain.txt        =>  "n2:v2"
* /a/test-3.txt       =>  "n2:v2"
* /a/x/y/z/plain.txt  =>  "n2:v2"
* /a/x/y/z/test-4.txt =>  "n2:v2"
* /a/b/pic.png        =>  无
* /a/b/test-5.png     =>  无
* /a/b/x/y/z/pic.png  =>  无
* /a/b/x/y/z/test-6.png =>  无
* /a/c/plain.txt      =>  "n3:v3"
* /a/c/test-7.txt     =>  "n3:v3"
* /a/c/x/y/z/plain.txt =>  "n3:v3"
* /a/c/x/y/z/test-8.txt =>  "n3:v3"
* /d/first-page.html  =>  "n11:v11", "n12:v12"
* /d/test-9.html      =>  "n11:v11", "n12:v12"
* /d/x/y/z/first-page.html =>  "n11:v11", "n12:v12"
* /d/x/y/z/test-10.html =>  "n11:v11", "n12:v12"
* /d/e/index.html     =>  "n4:v4"
* /d/e/test-11.html   =>  "n4:v4"
* /d/e/x/y/z/index.html =>  "n4:v4"
* /d/e/x/y/z/test-12.html =>  "n4:v4"
* /d/e/f/about.html   =>  "n4:v4"
* /d/e/f/test-13.html =>  "n4:v4"
* /d/e/f/x/y/z/about.html =>  "n4:v4"
* /d/e/f/x/y/z/test-14.html =>  "n4:v4"

```

下图演示了 qhs 使用上面“=>”左侧的 URL 路径在节点树上定位节点的过程，图中的数字表示定位操作的顺序，

所有定位操作必须从 root 节点开始:



6.8、URL 重写器的配置

URL 重写器的配置存放在/home/http/qhs3.3/config/urlRewriter.xml 文件，urlRewriter.xml 文件内部包含有说明注释。

6.9、WS 服务模块的配置

WS 服务模块的配置存放在/home/http/qhs3.3/config/wsServiceModule.xml 文件，wsServiceModule.xml 文件内部包含有说明注释。

7、开发 HTTP 服务模块

一个 HTTP 服务模块是一个 HTTP 服务模块类的实例。HTTP 服务模块用于构建动态的 http 答复以及推送答复 (pushed response)。遵循下面定义的 Java 类称为 HTTP 服务模块类：

- (1)、定义有一个公开的、无参数的构造器。
- (2)、实现 qhs.interfaces.HTTPServiceModule 接口。

httpServiceModules.xml 配置文件用于说明 HTTP 服务模块类。在 qhs 启动的过程中，qhs 参考 httpServiceModules.xml 配置文件为每个 http 处理线程创建一个 HTTP 服务模块池(一个 HTTP 服务模块池包含有一个以上的 HTTP 服务模块，因此，一个 HTTP 服务模块是一个线程特定对象)。

7.1、构建发送 http 答复

当浏览器等客户端对 HTTP 服务模块发出 http 请求时，qhs 就通过调用 HTTP 服务模块的 invoke()实例方法来构建 http 答复并且把 http 答复发送给客户端。

在 invoke()方法的内部应放置有构建发送 http 答复的代码。一般情况下，invoke()方法的工作有：

- (1)、分析来自客户端的 http 请求。来自客户端的 http 请求缓存在 qhs.interfaces.HTTPRequest 类型对象。
在 HTTP 服务模块的 invoke()方法内部调用 tc.getRequest()返回一个 qhs.interfaces.HTTPRequest 类型对象。
- (2)、构建发送 http 答复。在 HTTP 服务模块的 invoke()方法内部调用 tc.getResponse()返回一个 qhs.interfaces.HTTPResponse 类型对象，再调用返回的 qhs.interfaces.HTTPResponse 类型对象构建发送 http 答复。

构建发送 http 答复的演示文件见/home/http/qhs3.3/demo/httpServiceModule/ServerTime1.java，ServerTime1.java 定义的 ServerTime1 类用于创建 ID 为“server-time-1”的 HTTP 服务模块。

以下步骤演示了从编译源文件到启动 qhs 的过程，示例的运行环境见第 3 节：

- (1)、复制以下命令到终端并且执行命令：

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.3";
export MODULE_DIR="/home/http/qhsModules-01";
rm -Rvf $MODULE_DIR;
mkdir -vp $MODULE_DIR/classes;

cp -v $QHS_HOME/demo/httpServiceModule/*.java $MODULE_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
    -classpath $QHS_HOME/bin/qhs.jar $MODULE_DIR/*.java;

$JDK_HOME/bin/jar cvf $MODULE_DIR/httpServiceModule.jar \
    -C $MODULE_DIR/classes httpServiceModule;
```

(2)、对/home/http/qhs3.3/config/base.xml 文件内部的一个 SSLServerSocket 元素设置为：

```
.....
<SSLServerSocket>
  <Created>true</Created>
  <Address>127.0.0.1</Address>
  <Port>10000</Port>
  <KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>
  <Password>123456</Password>
</SSLServerSocket>
.....
```

(3)、对/home/http/qhs3.3/config/httpServiceModules.xml 文件的内容设置为：

```
<Config>
  <Modules JarFile="/home/http/qhsModules-01/httpServiceModule.jar" Enabled="true" >
    <Class Name="httpServiceModule.ServerTime1" Enabled="true" />
  </Modules>
</Config>
```

(4)、运行/home/http/qhs3.3/startup.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行，并重新启动 qhs)。用户可以打开 log 文件(log 文件由 base.xml 配置文件的 LogFile 元素值说明)查看 qhs 检测到的 HTTP 服务模块类。

(5)、运行浏览器，在浏览器的地址栏输入“https://127.0.0.1:10000/server-time-1.id”对 qhs 发出 http 请求(即对“server-time-1”HTTP 服务模块发出 http 请求)，则 qhs 把包含有服务器时间的 http 答复发送给浏览器。上述的 URL 中，“https://127.0.0.1:10000/”与“.id”之间的“server-time-1”就是服务模块 ID。如果浏览器对其它 HTTP 服务模块发出 http 请求，只需把“server-time-1”改为相应的服务模块 ID 就可以了。

7.2、构建发送推送答复

对于开发用户，构建发送推送答复(pushes response)与构建发送 http 答复的方法是很相似的，区别在：

- (1)、构建发送推送答复的代码放在 push()方法，而构建发送 http 答复的代码放在 invoke()方法。
- (2)、构建发送推送答复的 qhs.interfaces.HTTPResponse 类型对象是 tc.getPushedResponse()的返回值，而构建发送 http 答复的 qhs.interfaces.HTTPResponse 类型对象是 tc.getResponse()的返回值。

构建发送推送答复的演示文件见/home/http/qhs3.3/demo/serverPush-2/httpServiceModule/ServerTime2.java, ServerTime2.java 定义的 ServerTime2 类用于创建“server-time-2”HTTP 服务模块。ServerTime2.java 与上面的 ServerTime1.java 同样用于构建表达时间的答复，区别在，ServerTime2.push()方法用于构建表达时间的推送答复，而 ServerTime1.invoke()方法用于构建表达时间的 http 答复。

下面的命令演示了编译源文件的过程，示例的运行环境见第 3 节，复制以下命令到终端并且执行命令，命令编译生成的 serverTime2.jar 包的使用在第 8 节：

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.3";
export MODULE_DIR="/home/http/qhsModules-02";
rm -Rvf $MODULE_DIR;
mkdir -vp $MODULE_DIR/classes;

cp -v $QHS_HOME/demo/serverTime-2/httpServiceModule/*.java $MODULE_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
  -classpath $QHS_HOME/bin/qhs.jar $MODULE_DIR/*.java;

$JDK_HOME/bin/jar cvf $MODULE_DIR/serverTime2.jar -C $MODULE_DIR/classes httpServiceModule;
```

8、服务器推送

qhs 支持服务器推送(Server Push)。用户在阅读本节以前需要确定所用的浏览器是否支持服务器推送。因为有的浏览器的旧版本支持服务器推送(如: chrome91), 而新版本却不支持服务器推送(如: chrome111)。

8.1、通过静态资源文件发起服务器推送

通过静态资源文件发起服务器推送即 qhs 在文档根目录内部检索文件、发送文件时发起的服务器推送。这要求用户必须在 urlPaths.xml 配置文件中定义有指向文档根目录内部的文件的 URL 路径并且为被指向的文件设置一个以上的推送 URL。本手册的第 6.7 节对 urlPaths.xml 配置文件的格式作了说明。

以/home/http/qhs3.3/demo/serverPush-1/config/urlPaths.xml 配置文件为例, urlPaths.xml 的内容:

```
<Config>
  <pics>
    <Header Name="cache-control" Value="no-store, no-cache" />
  </pics>

  <server-push.html>
    <Header Name="cache-control" Value="no-store, no-cache" />
    <Push URL="/pics/11.png" PartOfLinkHeader=""; rel=preload; as=image" />
    <Push URL="/pics/12.png" PartOfLinkHeader=""; rel=preload; as=image" />
    .....
    <Push URL="/pics/58.png" PartOfLinkHeader=""; rel=preload; as=image" />
    <Push URL="/pics/59.png" PartOfLinkHeader=""; rel=preload; as=image" />

    <Push URL="/server-time-2.id" />
  </server-push.html>
</Config>
```

在上面的配置中, 从 Config 元素遍历到 server-push.html 元素的路径形成一个 URL 路径: /server-push.html, /server-push.html 指向文档根目录内部的/server-push.html 文件。server-push.html 元素包含有多个 Push 子元素, 每一个 Push 元素说明一个推送 URL。当 qhs 使用/server-push.html 在文档根目录内部检索、发送/server-push.html 文件以前, 推送/pics/11.png 至/server-time-2.id 等多个 URL。/pics/11.png 至/pics/59.png 等多个 URL 分别指向文档根目录内部的/pics/11.png 至/pics/59.png 等文件, 这些文件用于构建推送答复并且发送给浏览器等客户端。当 qhs 使用这些文件构建推送答复时, 对推送答复增加"cache-control:no-store, no-cache"头域。/server-time-2.id 用于调用 server-time-2 服务模块构建推送答复并且把推送答复发送给客户端, 即当 qhs 推送/server-time-2.id 时, qhs 就调用 ServerTime2.push()方法构建推送答复并且把推送答复发送给客户端。

以下步骤演示了从编译源文件到启动 qhs 的过程, 示例的运行环境见第 3 节:

- (1)、参考 7.2 节编译生成 serverTime2.jar 包。
- (2)、对/home/http/qhs3.3/config/base.xml 文件内部的 DocumentRoot 元素值以及一个 SSLServerSocket 元素设置为:

```
.....
<DocumentRoot>/home/http/qhs3.3/demo/serverPush-1/documentRoot</DocumentRoot>
.....
<SSLServerSocket>
  <Created>true</Created>
  <Address>127.0.0.1</Address>
  <Port>10000</Port>
  <KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>
  <Password>123456</Password>
</SSLServerSocket>
.....
```

(3)、复制/home/http/qhs3.3/demo/serverPush-1/config/urlPaths.xml 配置文件到/home/http/qhs3.3/config 目录:

```
cp -v /home/http/qhs3.3/demo/serverPush-1/config/urlPaths.xml /home/http/qhs3.3/config;
```

(4)、对/home/http/qhs3.3/config/httpServiceModules.xml 配置文件的内容设置为:

```
<Config>
  <Modules JarFile="/home/http/qhsModules-02/serverTime2.jar" >
    <Class Name="httpServiceModule.ServerTime2" />
  </Modules>
</Config>
```

(5)、运行/home/http/qhs3.3/startup.sh 来启动 qhs(如果 qhs 正在运行,就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行,并重新启动 qhs)。用户可以打开 log 文件(log 文件由 base.xml 配置文件的 LogFile 元素值说明)查看 qhs 检测到的 HTTP 服务模块类。

(6)、运行浏览器,在浏览器的地址栏输入“https://127.0.0.1:10000/server-push.html”对 qhs 发出 http 请求。当浏览器页面显示一幅由 45 个小图像组成的风景图以及页面刷新时间,说明 qhs 成功执行了服务器推送。在浏览器查看 server-push.html 的源代码,可以看到这样的结构:

```
.....

<table border="0" frame="void" cellspacing="1" cellpadding="0" >
  <tr>
    <td></td>
    <td></td>
    .....
    <td></td>
    <td></td>
  </tr>
  <tr><td colspan="4"><iframe width="600" height="80" src="/server-time-2.id" /></td></tr>
  .....

```

在 server-push.html 的源代码中,有多个 src 属性分别引用了/pics/11.png 至/pics/59.png、/server-time-2.id 等多个 URL,每一个 URL 代表一个从 qhs 接收到的推送答复。当浏览器页面显示所有小图像、页面刷新时间等数据时,首先引用的是推送答复,而不是再对 qhs 发出 http 请求后接收到的 http 答复。上述的多个 URL 对应了/home/http/qhs3.3/demo/serverPush-1/config/urlPaths.xml 配置文件内部的为 server-push.html 设置的多个推送 URL。在/home/http/qhs3.3/demo/serverPush-1/documentRoot 文档根目录内部存在 server-push.html、non-server-push.html 等 2 个文件,这 2 个文件的内容是相同的。使用浏览器访问 https://127.0.0.1:10000/server-push.html 地址时能够发起服务器推送,而访问 https://127.0.0.1:10000/non-server-push.html 地址时却不能发起服务器推送,原因是在/home/http/qhs3.3/demo/serverPush-1/config/urlPaths.xml 没有定义指向 non-server-push.html 的 URL 路径。如果用户熟悉浏览器的开发者工具,用户可以在浏览器不断地刷新访问 https://127.0.0.1:10000/server-push.html、https://127.0.0.1:10000/non-server-push.html 等 2 个地址来比较 ServerPush 与 NonServerPush 的速度。

8.2、通过 HTTP 服务模块发起服务器推送

通过 HTTP 服务模块发起服务器推送即在调用 HTTP 服务模块构建发送 http 答复以前调用 ThreadContext.push() 方法推送 URL。通过 HTTP 服务模块发起服务器推送的方法具有自由度和灵活性,发起服务器推送时需要遵循以下编程步骤:

- (1)、调用 ThreadContext.push() 推送一个以上的 URL。
- (2)、对 http 答复增加 link 头域,每一个 link 头域引用一个推送 URL。
- (3)、在 http 答复的载荷体(payload body)增加引用推送 URL 的语句。

以/home/http/qhs3.3/demo/serverPush-2/httpServiceModule/ServerPush.java 文件为例，ServerPush.java 文件结构：

```
.....
public void invoke(qhs.interfaces.ThreadContext tc, qhs.interfaces.Connection conn) throws Throwable {
    .....
    for (String url : urls) {
        tc.push(url.getBytes());
    }
    .....
    for (int i = 0; i < urls.length - 1; i++) {
        response.addHeader(qhs.sys.ResponseHeaderCreator.create(qhs.sys.StaticName.link,
            "<" + urls[i] + ">; rel=preload; as=image"));
    }
    response.addHeader(qhs.sys.ResponseHeaderCreator.create(qhs.sys.StaticName.link,
        "<" + urls[urls.length - 1] + ">"));
    .....
    for (int col = 1; col <= maxCol; col++) {
        response.tryToSendPartOfPayloadBody(getUTF8Bytes("                <td><img "
            + "src=\"" + urls[index++] + "\"在 html 内容引用推送 URL。*/ + "\" /></td>\n"));
    }
    .....
    response.tryToSendPartOfPayloadBody(getUTF8Bytes("                "
        + "<tr>"
        + "<td colspan=\"4\"><iframe width=\"600\" height=\"80\" "
        + "src=\"" + urls[index] + "\"在 html 内容引用推送 URL。*/ + "\" /></td>"
        + "</tr>\n"));
    .....
}
.....
```

上面的 java 源码，调用了 tc.push() 推送 URL，调用了 response.addHeader() 对 http 答复增加 link 头域，调用 response.tryToSendPartOfPayloadBody() 发送的载荷体(属于 html 数据)内部的 src 属性引用了推送 URL。查看/home/http/qhs3.3/demo/serverPush-2/httpServiceModule/ServerPush.java 文件的全部内容，当调用 tc.push() 推送/pics/11.png 至/pics/59.png 等 URL 时，qhs 在文档根目录检索/pics/11.png 至/pics/59.png 等文件，并且使用这些文件构建推送答复，把推送答复发送给客户端。当调用 tc.push() 推送/server-time-2.id 时，qhs 就调用 ServerTime2.push() 方法构建推送答复并且把推送答复发送给客户端。

以下步骤演示了从编译源文件到启动 qhs 的过程，示例的运行环境见第 3 节：

(1)、参考 7.2 节编译生成 serverTime2.jar 包。

(2)、对/home/http/qhs3.3/config/base.xml 文件内部的 DocumentRoot 元素值以及一个 SSLServerSocket 元素设置为：

```
.....
<DocumentRoot>/home/http/qhs3.3/demo/serverPush-2/documentRoot</DocumentRoot>
.....
<SSLServerSocket>
    <Created>true</Created>
    <Address>127.0.0.1</Address>
    <Port>10000</Port>
    <KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>
    <Password>123456</Password>
</SSLServerSocket>
.....
```

(3)、对/home/http/qhs3.3/config/urlPaths.xml 文件的内容设置为：

```
<Config>
    <pics>
```


9、开发 WebSocket 功能

9.1、开发 WS 服务模块

qhs 支持 WebSocketsOverHTTP/2。在 WebSocketsOverHTTP/2，所有与 WebSocket 功能有联系的数据被编码为 HTTP/2 流，这样的 HTTP/2 流称为 WebSocket 流(即 WebSocketStream)，简称为 WS 流。

在传输 WS 流以前，必须首先建立 http 连接或者存在一个可用的 http 连接。以浏览器为例，浏览器调用 JavaScript 的 WebSocket()构造器以前必须(通过某些方法)首先建立浏览器与 qhs 之间的 http 连接，让 WebSocket()构造器查找可用的 http 连接。要是 WebSocket()构造器的 url 参数值上的地址与 qhs 的地址相同，浏览器就起用已建立的 http 连接并且在 http 连接上传输 WS 流。

一般情况下，用户通过开发 WS 服务模块，并在 WS 服务模块内部执行 WebSocket 读写操作。一个 WS 服务模块是一个 WS 服务模块类的实例。遵循下面定义的 Java 类称为 WS 服务模块类：

- (1)、定义有一个公开的、无参数的构造器。
- (2)、实现 qhs.interfaces.WSServiceModule 接口。

wsServiceModule.xml 配置文件用于说明 WS 服务模块类。在 qhs 启动的过程中，qhs 参考 wsServiceModule.xml 配置文件为每个 http 处理线程创建一个 WS 服务模块。因此，一个 WS 服务模块是一个线程特定对象。

实现了 WS 服务模块类的演示文件见/home/http/qhs3.3/demo/websocket/wsServiceModule/WSServiceModule.java。WSServiceModule.java 用于查询 qhs 的运行环境并且把查询结果编码成为 WebSocket 帧以后把 WebSocket 帧传输给浏览器。

以下步骤演示了从编译源文件到启动 qhs 的过程，示例的运行环境见第 3 节：

- (1)、复制以下命令到终端并且执行命令：

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.3";
export MODULE_DIR="/home/http/qhsModules-04";
rm -Rvf $MODULE_DIR;
mkdir -vp $MODULE_DIR/classes;

cp -v $QHS_HOME/demo/websocket-1/wsServiceModule/*.java $MODULE_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
    -classpath $QHS_HOME/bin/qhs.jar $MODULE_DIR/*.java;

$JDK_HOME/bin/jar cvf $MODULE_DIR/wsServiceModule.jar -C $MODULE_DIR/classes wsServiceModule;
```

- (2)、对/home/http/qhs3.3/config/base.xml 文件内部的 DocumentRoot 以及一个 SSLServerSocket 元素设置为：

```
.....
<DocumentRoot>/home/http/qhs3.3/demo/websocket-1/pages</DocumentRoot>
.....
<SSLServerSocket>
    <Created>true</Created>
    <Address>127.0.0.1</Address>
    <Port>10000</Port>
    <KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>
    <Password>123456</Password>
</SSLServerSocket>
```


.....
(3)、对/home/http/qhs3.3/config/wsServiceModule.xml 文件的内容设置为：

```
<Config>
  <Created>true</Created>
  <JarFile>/home/http/qhsModules-04/wsServiceModule.jar</JarFile>
  <Class>wsServiceModule.WSServiceModule</Class>
</Config>
```

(4)、运行/home/http/qhs3.3/startup.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行，并重新启动 qhs)。用户可以打开 log 文件(log 文件由 base.xml 配置文件的 LogFile 元素值说明)查看 qhs 检测到的 WS 服务模块类。

(5)、运行浏览器，在浏览器的地址栏输入“https://127.0.0.1:10000/query-env.html”对 qhs 发出 http 请求从而建立浏览器与 qhs 之间的 http 连接。在浏览器显示的页面，单击“尝试建立 WS 流”按钮建立 WS 流，然后选择命令并且单击“执行”按钮，浏览器显示了从 qhs 返回的命令执行结果。查看 query-env.html 的源码，可以看到 WebSocket()构造器的 url 参数值为“wss://127.0.0.1:10000”，url 参数值上的地址与用户在地址栏输入的地址相同，先前浏览器使用这个相同的地址已建立了浏览器与 qhs 之间的 http 连接，浏览器起用这个先前已经建立的、已存在的 http 连接并且在 http 连接上传输 WS 流。

9.2、编写跨线程的 WebSocket 代码

经由 qhs 创建的 qhs.interfaces.Connection 接口实例以及 qhs.interfaces.WebSocketStream 接口实例是可以跨线程的、可以缓存为全局数据或静态数据，这样：

- (1)、可以对 Connection 接口对象按某种标准(如：客户地址)进行分组管理。
- (2)、用户代码可以在任何代码位置、任意时刻调用 Connection.close()关闭 http 连接。
- (3)、用户代码可以在任何代码位置、任意时刻调用 WebSocketStream.writeText()或 WebSocketStream.writeBinary()主动发送 WebSocket 数据给客户端，有利于减少客户端对 qhs 的轮询请求。用户代码可以调用 Connection.getWebSocketStreams()返回所有在 http 连接上传输的 WS 流。

qhs 为所有 http 连接(属于 qhs.interfaces.Connection 接口类型)分别配置有一个线程锁(即一个 http 连接配置有一个专有的线程锁)，这些被配置的多个线程锁是不相等的，用户代码不能直接引用这些线程锁。为了在多线程环境下保护 http 连接，WebSocketStream 接口实例调用 http 连接的线程锁执行锁定操作(即锁定 http 连接)：在任何代码位置调用定义在 WebSocketStream 接口上的方法时，WebSocketStream 接口实例调用 WS 流所处于的 http 连接的线程锁用于锁定 http 连接，http 连接等于 WebSocketStream.getConnection()的返回值。作为调用 WebSocketStream.invoke()方法的参数值的 WebSocketCode 接口实例，如果用户在 WebSocketCode.invoke()方法的实现方法内部放置有不同类型的线程锁的锁定操作(如：synchronized 方法的调用、synchronized 语句块的执行、调用 ReentrantLock.lock()方法等等)，就必然出现线程锁嵌套。即：先调用 WS 流所处于的 http 连接的线程锁执行锁定操作，再调用 WebSocketCode.invoke()方法的实现方法内部的用户放置的线程锁执行锁定操作。

在多线程环境下，为了同步 WS 流的状态，建议调用 WebSocketStream.invoke()方法来同步 WS 流的状态，否则，使用其它算法会导致 WS 流的状态不完全同步。

/home/http/qhs3.3/demo/websocket-2/modules 目录包含有跨线程的 WebSocket 示例代码，这些代码主要用于发送/proc/meminfo 文件的部分内容给浏览器：

- (1)、WebSocketStreamList 类：缓存所有 WebSocketStream 接口实例。WebSocketStreamList 类是一个同步类，所有类方法说明为 synchronized。WebSocketStreamList 类没有调用定义在 Connection、WebSocketStream 等接口上的方法，避免了线程锁嵌套及死锁。
- (2)、SendingThread 类：用于创建发送线程，发送线程每隔 300 毫秒从/proc/meminfo 文件读取 6 个变量值，并且把这些变量值通过 WS 流发送给浏览器。SendingThread.run()通过调用 WebSocketStream.invoke()(如：“stream.invoke(wsc);”语句)来同步 WS 的状态并且把变量值发送给浏览器。
- (3)、StreamState 类：记录 WS 流的状态。
- (4)、WSServiceModule 类：实现 WS 服务模块。WSServiceModule 类主要用于分析浏览器发送给 qhs 的 WebSocket 数据、分析出命令并且变改 WS 流的状态。WSServiceModule.invoke()通过调用 WebSocketStream.invoke

()(如：“readableStream.invoke(wsc);”语句)来同步 WS 的状态。

- (5)、上面的发送线程与运行在 http 处理线程的 WSServiceModule 类实例同时引用有 WebSocketStream 接口实例，被引用的 WebSocketStream 接口实例是跨线程的。

以下步骤演示了从编译 WebSocket 示例代码到启动 qhs 的过程，示例的运行环境见第 3 节：

- (1)、复制以下命令到终端并且执行命令：

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.3";
export MODULE_DIR="/home/http/qhsModules-05";
rm -Rvf $MODULE_DIR;
mkdir -vp $MODULE_DIR/classes;

cp -Rvf $QHS_HOME/demo/websocket-2/modules/sendingthread $MODULE_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
    -classpath $QHS_HOME/bin/qhs.jar $MODULE_DIR/sendingthread/*.java;

$JDK_HOME/bin/jar cvf $MODULE_DIR/sendingThread.jar -C $MODULE_DIR/classes sendingthread;

cp -Rvf $QHS_HOME/demo/websocket-2/modules/wsServiceModule $MODULE_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
    -classpath $QHS_HOME/bin/qhs.jar:$MODULE_DIR/sendingThread.jar \
    $MODULE_DIR/wsServiceModule/*.java;

$JDK_HOME/bin/jar cvf $MODULE_DIR/wsServiceModule.jar \
    -C $MODULE_DIR/classes wsServiceModule;
```

- (2)、对/home/http/qhs3.3/config/base.xml 文件内部的 DocumentRoot 以及一个 SSLServerSocket 元素设置为：

```
.....
<DocumentRoot>/home/http/qhs3.3/demo/websocket-2/pages</DocumentRoot>
.....
<SSLServerSocket>
  <Created>true</Created>
  <Address>127.0.0.1</Address>
  <Port>10000</Port>
  <KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>
  <Password>123456</Password>
</SSLServerSocket>
.....
```

- (3)、对/home/http/qhs3.3/config/wsServiceModule.xml 文件的内容设置为：

```
<Config>
  <Created>true</Created>
  <JarFile>/home/http/qhsModules-05/wsServiceModule.jar</JarFile>
  <Class>wsServiceModule.WSServiceModule</Class>
</Config>
```

- (4)、对/home/http/qhs3.3/startup.sh 的内容设置为：

```
#!/bin/bash
exec "/usr/java/jdk-11/bin/java" -classpath \
"/home/http/qhs3.3/bin/qhs.jar:\
/home/http/qhsModules-05/sendingThread.jar:\
/home/http/qhs3.3/bin/gson.jar" \
qhs.Start --ServerRoot="/home/http/qhs3.3" --Config="/home/http/qhs3.3/config";
```

- (5)、运行/home/http/qhs3.3/startup.sh 来启动 qhs(如果 qhs 正在运行, 就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行, 并重新启动 qhs)。用户可以打开 log 文件(log 文件由 base.xml 配置文件的 LogFile 元素值说明)查看 qhs 检测到的 WS 服务模块类。
- (6)、运行浏览器, 在浏览器的地址栏输入“https://127.0.0.1:10000/mem-info.html”对 qhs 发出 http 请求从而建立浏览器与 qhs 之间的 http 连接。在浏览器显示的页面, 单击“尝试建立 WS 流”按钮建立 WS 流以后, qhs 主动不断地发送/proc/meminfo 文件的部分内容给浏览器。

10、开发文件检索器

文件检索器用于检索资源文件。qhs 内建有系统默认的文件检索器, 这文件检索器只在文档根目录(通过 base.xml 配置文件的 DocumentRoot 元素说明)的内部检索文件。

购买付费的 qhs 实例, 用户可以开发定制的文件检索器用于替换系统默认的文件检索器。免费 qhs 实例的用户无需读本节。

通过实现 qhs.interfaces.FileRetriever、qhs.interfaces.File 等接口, 用户可以开发定制的文件检索器并且把各种来源的数据(如: 操作系统目录内部的文件、字节数组、来自 Redis 集群的键值、来自数据库的记录、来自消息队列的数据等等)作为虚拟文件。qhs 把虚拟文件设置成为 http 答复或者推送答复(push response)的载荷体(payload-body)并且传送给浏览器。qhs 对文档根目录内部的资源文件的所有操作同样适用于虚拟文件, 如:

- (1)、使用指向虚拟文件的 URL 路径在 urlPaths 系统对象定位节点并且在 http 答复、推送答复上设置定制头域。
- (2)、使用指向虚拟文件的 URL 路径右侧的文件扩展名, 对 http 答复、推送答复设置 content-type 头域。
- (3)、当虚拟文件作为 http 答复或者推送答复的载荷体时, qhs 在 http 答复或者推送答复增加一个 etag 头域。在下次对相同的虚拟文件发出 http 请求时, 浏览器在 http 请求的内部增加 if-none-match 头域并且把 etag 头域的值设置为 if-none-match 头域的值。etag 头域与 if-none-match 头域等 2 个头域的使用, 为有效地减少网络数据传输量提供了一种方法。要是 http 请求内部的 if-none-match 头域值被成功匹配, 那么 qhs 发送状态码为 304 的 http 答复给浏览器, 而不是重新发送虚拟文件作为载荷体的 http 答复给浏览器。一个状态码为 304 的 http 答复的明文长度小于 40 个字节。要是虚拟文件的长度大于 70 个字节, 那么减少了 30 以上个字节的网络数据传输量。

一个文件检索器是一个遵循下面定义的 Java 类的实例:

- (1)、定义有一个公开的、无参数的构造器。
- (2)、实现 qhs.interfaces.FileRetriever 接口。

fileRetriever.xml 配置文件用于说明创建文件检索器的类。在 qhs 启动的过程中, qhs 参考 fileRetriever.xml 配置文件为每个 http 处理线程创建一个文件检索器, 因此, 一个文件检索器是一个线程特定对象。

演示文件/home/http/qhs3.3/demo/fileRetriever/TempFileRetriever.java 定义的 TempFileRetriever 类用于创建文件检索器。演示文件/home/http/qhs3.3/demo/fileRetriever/TempFile.java 定义的 TempFile 类用于创建虚拟文件的外层。

以下步骤演示了从编译源文件到启动 qhs 的过程, 示例的运行环境见第 3 节:

- (1)、复制以下命令到终端并且执行命令:

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.3";
export MODULE_DIR="/home/http/qhsModules-06";
rm -Rvf $MODULE_DIR;
mkdir -vp $MODULE_DIR/classes;

cp -v $QHS_HOME/demo/fileRetriever/*.java $MODULE_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
```

```
-classpath $QHS_HOME/bin/qhs.jar $MODULE_DIR/*.java;
```

```
$JDK_HOME/bin/jar cvf $MODULE_DIR/fileRetriever.jar -C $MODULE_DIR/classes fileretriever;
```

- (2)、对/home/http/qhs3.3/config/base.xml 文件内部的 DocumentRoot 以及一个 SSLServerSocket 元素设置为：

```
.....
<DocumentRoot>/dev/shm</DocumentRoot>
.....
<SSLServerSocket>
  <Created>true</Created>
  <Address>127.0.0.1</Address>
  <Port>10000</Port>
  <KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>
  <Password>123456</Password>
</SSLServerSocket>
.....
```

- (3)、对/home/http/qhs3.3/config/fileRetriever.xml 文件的内容设置为：

```
<Config>
  <Created>true</Created>
  <JarFile>/home/http/qhsModules-06/fileRetriever.jar</JarFile>
  <Class>fileretriever.TempFileRetriever</Class>
</Config>
```

- (4)、运行/home/http/qhs3.3/startup.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行，并重新启动 qhs)。用户可以打开 log 文件(log 文件由 base.xml 配置文件的 LogFile 元素值说明)查看 qhs 检测到的用于创建文件检索器的类。

- (5)、运行浏览器，在浏览器的地址栏输入“https://127.0.0.1:10000/index.html”对 qhs 发出 http 请求，浏览器显示媒体类型为 text/html 的虚拟文件。

11、开发套接字通道读写器

套接字通道读写器(即 SocketChannelReadWrite，简称为 SocketChannelRW)用于对套接字通道进行读写操作。

购买付费的 qhs 实例用于开发套接字通道读写器，免费 qhs 实例的用户无需读本节。

开发套接字通道读写器的优点：

- (1)、有利于开发用户专有的通信加密协议。qhs 与浏览器的通信加密协议是 TLSv1.3。TLS 协议是复杂而强大的加密协议。然而，TLS 并不是处处适用。比如，服务器与客户端之间的 TLS 握手会消耗大量计算资源，导致应用变慢，不利于用户体验。又比如，TLS 不是为 http 专门定制，不能获知数据的重要程度以及数据的类型，对所有数据(包括重要的数据、不重要的数据、不同类型的数据)施用相同的加密算法及加密流程，缺少灵活性。通过开发套接字通道读写器，可开发用户专有的通信加密协议，对重要的数据(如：用户密码、交易金额等)施用高强度的加密算法，对不重要的数据(如：html 数据)施用低强度的加密算法，有利于平衡网络数据的安全性与用户体验。开发用户专有的通信加密协议通常适用于 qhs 与手机 APP 之间的直接通信。
- (2)、有利于定制 qhs 与反向代理之间的明文数据通信格式。这里的反向代理既可以通过软件实现的也可以通过硬件实现的，必须具有与浏览器进行加密通信的能力，必须具有与后端服务器进行明文通信的能力。反向代理把来自浏览器的密文数据解密得出明文数据，并且把明文数据转发给后端服务器。若是 qhs 作为后端服务器而且 qhs 与反向代理进行明文数据通信，就有需要通过开发套接字通道读写器来定制明文数据通信格式。

一个套接字通道读写器是一个遵循下面定义的 Java 类的实例：

- (1)、定义有一个公开的、无参数的构造器。

(2)、实现 qhs.interfaces.SocketChannelRW 接口。

socketChannelRW.xml 配置文件用于说明创建套接字通道读写器的类。在 qhs 运行的过程中, qhs 允许 TLS 加密通信(通过 base.xml 配置文件的 SSLServerSocket 元素进行说明)与非 TLS(non-TLS)加密通信(通过 base.xml 配置文件的 ServerSocket 元素进行说明)等两者同时并存。要是存在非 TLS 加密通信而且 socketChannelRW.xml 配置文件说明了创建套接字通道读写器的类, 那么, 对于一切进行非 TLS 加密通信的 http 连接, 所有对套接字通道的读写操作交给套接字通道读写器。一个套接字通道读写器是一个连接特定对象, qhs 为每个进行非 TLS 加密通信的 http 连接创建一个套接字通道读写器。

演示文件/home/http/qhs3.3/demo/socketChannelRW/SocketChannelRW_base.java 定义的类用于创建套接字读写器, 这个类的实例对套接字通道只执行最基本的读写操作。演示文件/home/http/qhs3.3/demo/socketChannelRW/SocketChannelRW_cipher.java 定义的类用于创建套接字读写器:

- (1)、在 SocketChannelRW_cipher.read(), 示例代码首先调用 sc.read() 读取足够长度的密文数据(密文数据来自客户端), 然后调用 AESCipher.decrypt() 解密密文数据得出明文数据并且把明文数据缓存在 buffer 参数。缓存在 buffer 参数上的明文数据提供给 qhs 进行分析。如果明文数据不是合格的 HTTP/2 数据, qhs 就关闭 http 连接。
- (2)、在 SocketChannelRW_cipher.write(), 示例代码首先从 buffer 参数获取 HTTP/2 明文数据(HTTP/2 明文数据经由 qhs 生成), 接着调用 AESCipher.encrypt() 加密 HTTP/2 明文数据得出密文数据并且编码合并密文数据的长度, 然后调用 sc.write() 把密文数据发送给客户端。

以下步骤演示了从编译源文件到启动 qhs 的过程, 示例的运行环境见第 3 节:

- (1)、复制以下命令到终端并且执行命令:

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.3";
export MODULE_DIR="/home/http/qhsModules-07";
rm -Rvf $MODULE_DIR;
mkdir -vp $MODULE_DIR/classes;

cp -v $QHS_HOME/demo/socketChannelRW/*.java $MODULE_DIR;
$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
    -classpath $QHS_HOME/bin/qhs.jar $MODULE_DIR/*.java;

$JDK_HOME/bin/jar cvf $MODULE_DIR/socketChannelRW.jar \
    -C $MODULE_DIR/classes socketChannelRW;
```

- (2)、对/home/http/qhs3.3/config/base.xml 文件内部的一个 ServerSocket 元素设置为:

```
.....
<ServerSocket>
    <Created>true</Created>
    <Address>127.0.0.1</Address>
    <Port>8899</Port>
</ServerSocket>
.....
```

- (3)、对/home/http/qhs3.3/config/socketChannelRW.xml 文件的内容设置为:

```
<Config>
    <Enabled>true</Enabled>
    <JarFile>/home/http/qhsModules-07/socketChannelRW.jar</JarFile>
    <Class>socketChannelRW.SocketChannelRW_cipher</Class>
</Config>
```

- (4)、运行/home/http/qhs3.3/startup.sh 来启动 qhs(如果 qhs 正在运行, 就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行, 并重新启动 qhs)。用户可以打开 log 文件(log 文件由 base.xml 配置文件的 LogFile 元素值说明)查看 qhs 检测到的用于创建套接字通道读写器的类。

- (5)、运行用户定制的客户端，连接到 127.0.0.1:8899 地址。客户端可以调用 AESCipher.encrypt()加密数据，可以调用 AESCipher.decrypt()解密数据。

12、开发 URL 重写器

URL 重写器(URLRewriter)既用于重写 URL 又用于确定重写后的 URL 的用途。

一个 URL 重写器是一个遵循下面定义的 Java 类的实例：

- (1)、定义有一个公开的、无参数的构造器。
- (2)、实现 qhs.interfaces.URLRewriter 接口。

urlRewriter.xml 配置文件用于说明创建 URL 重写器的类。在 qhs 启动的过程中，qhs 参考 urlRewriter.xml 配置文件为每个 http 处理线程创建一个 URL 重写器，因此，一个 URL 重写器是一个线程特定对象。

演示文件/home/http/qhs3.3/demo/urlRewriter/URLRewriter.java 定义了一个简单的、用于创建 URL 重写器的 URLRewriter 类，开发人员可以对这 URLRewriter 类增加代码。

以下步骤演示了从编译源文件到启动 qhs 的过程，示例的运行环境见第 3 节：

- (1)、复制以下命令到终端并且执行命令：

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.3";
export MODULE_DIR="/home/http/qhsModules-08";
rm -Rvf $MODULE_DIR;
mkdir -vp $MODULE_DIR/classes;

cp -v $QHS_HOME/demo/urlRewriter/*.java $MODULE_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
    -classpath $QHS_HOME/bin/qhs.jar $MODULE_DIR/*.java;

$JDK_HOME/bin/jar cvf $MODULE_DIR/urlRewriter.jar \
    -C $MODULE_DIR/classes urlRewriter;
```

- (2)、对/home/http/qhs3.3/config/base.xml 文件内部的一个 SSLServerSocket 元素设置为：

```
.....
<SSLServerSocket>
  <Created>true</Created>
  <Address>127.0.0.1</Address>
  <Port>10000</Port>
  <KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>
  <Password>123456</Password>
</SSLServerSocket>
.....
```

- (3)、对/home/http/qhs3.3/config/urlRewriter.xml 文件的内容设置为：

```
<Config>
  <Created>true</Created>
  <JarFile>/home/http/qhsModules-08/urlRewriter.jar</JarFile>
  <Class>urlRewriter.URLRewriter</Class>
</Config>
```

- (3)、运行/home/http/qhs3.3/startup.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行，并重新启动 qhs)。用户可以打开 log 文件(log 文件由 base.xml 配置文件的 LogFile 元素说明)查看 qhs 检测到的、用于创建 URL 重写器的 urlRewriter.URLRewriter 类。

- (4)、运行浏览器，在浏览器的地址栏输入“https://127.0.0.1:10000/asd.html?pr”对 qhs 发出 http 请求，qhs 生成并且发送表达永久重定向的 http 答复给浏览器。在浏览器的地址栏输入“https://127.0.0.1:10000/fgh.html?tr”对 qhs 发出 http 请求，qhs 生成并且发送表达临时重定向的 http 答复给浏览器。如果用户熟悉浏览器的开发者工具，用户可以在浏览器的开发者工具查看到表达永久重定向(状态码为 301)的 http 答复以及表达临时重定向(状态码为 307)的 http 答复。

13、开发调试输出器

调试输出器(DebugOutputer)用于输出、显示调试信息。如果 qhs 创建了调试输出器，在 qhs 的运行过程中，qhs 就尝试把各种调试信息输出到调试输出器。一般情况下，调试输出器只与 NetBeansIDE 等开发工具结合使用，开发人员可以通过观察调试输出器显示的信息来了解 qhs 的运作过程，并为开发 HTTP 服务模块、URL 重写器、WS 服务模块等 qhs 模块排除错误。开发人员也可以通过观察调试输出器显示的信息来了解浏览器与 qhs 的互动，从而对自编代码进行优化以及优化 qhs 的配置。

一个调试输出器是一个遵循下面定义的 Java 类的实例：

- (1)、定义有一个公开的、无参数的构造器。
- (2)、实现 qhs.interfaces.DebugOutputer 接口。

debugOutputer.xml 配置文件用于说明创建调试输出器的类。在 qhs 启动的过程中，qhs 参考 debugOutputer.xml 配置文件尝试创建一个全局的调试输出器，这个调试输出器被所有 http 处理线程引用，因此，开发调试输出器时必须要考虑线程安全。

演示文件/home/http/qhs3.3/demo/debugOutputer/DebugOutputer.java 定义了一个简单的、用于创建调试输出器的 DebugOutputer 类，开发人员可以对这 DebugOutputer 类进行更改、增加代码。演示文件/home/http/qhs3.3/demo/debugOutputerGUI/DebugOutputer.java 文件也定义了一个用于创建调试输出器的 DebugOutputer 类。

以下步骤演示了从编译源文件到启动 qhs 的过程，示例的运行环境见第 3 节：

- (1)、复制以下命令到终端并且执行命令：

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.3";
export MODULE_DIR="/home/http/qhsModules-09";
rm -Rvf $MODULE_DIR;
mkdir -vp $MODULE_DIR/classes;

cp -v $QHS_HOME/demo/debugOutputerGUI/*.java $MODULE_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
    -classpath $QHS_HOME/bin/qhs.jar $MODULE_DIR/*.java;

$JDK_HOME/bin/jar cvf $MODULE_DIR/debugOutputer.jar \
    -C $MODULE_DIR/classes debugOutputerGUI;
```

- (2)、对/home/http/qhs3.3/config/base.xml 文件内部的一个 SSLServerSocket 元素更改为：

```
.....
<SSLServerSocket>
  <Created>true</Created>
  <Address>127.0.0.1</Address>
  <Port>10000</Port>
  <KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>
  <Password>123456</Password>
```

```
</SSLServerSocket>
```

```
.....
```

- (3)、对/home/http/qhs3.3/config/debugOutputer.xml 文件的内容设置为：

```
<Config>
  <Created>true</Created>
  <JarFile>/home/http/qhsModules-09/debugOutputer.jar</JarFile>
  <Class>debugOutputerGUI.DebugOutputer</Class>
</Config>
```

- (4)、运行/home/http/qhs3.3/startup.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行，并重新启动 qhs)。用户可以打开 log 文件(log 文件由 base.xml 配置文件的 LogFile 元素值说明)查看 qhs 检测到的、用于创建调试输出器的 debugOutputerGUI.DebugOutputer 类。如果成功启动 qhs，qhs 就显示调试输出器窗口，用户可以在浏览器的地址栏输入任意 URL(如：https://127.0.0.1:10000)对 qhs 发送 http 请求，并在调试输出窗口查看调试信息。

14、开发错误答复创建器

错误答复创建器(ErrorResponseCreator)主要用于构建表达错误原因(如：未找到资源文件、服务无效等等)的 http 答复或推送答复。

一个错误答复创建器是一个遵循下面定义的 Java 类的实例：

- (1)、定义有一个公开的、无参数的构造器。
- (2)、实现 qhs.interfaces.ErrorResponseCreator 接口。

errorResponseCreator.xml 配置文件用于说明创建错误答复创建器的类。在 qhs 启动的过程中，qhs 参考 errorResponseCreator.xml 配置文件为每个 http 处理线程创建一个错误答复创建器，因此，一个错误答复创建器是一个线程特定对象。

演示文件/home/http/qhs3.3/demo/errorResponseCreator/ErrorResponseCreator.java 定义了一个用于创建错误答复创建器的 ErrorResponseCreator 类，开发人员可以对这 ErrorResponseCreator 类进行更改、增加代码。

以下步骤演示了从编译源文件到启动 qhs 的过程，示例的运行环境见第 3 节：

- (1)、复制以下命令到终端并且执行命令：

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.3";
export MODULE_DIR="/home/http/qhsModules-10";
rm -Rvf $MODULE_DIR;
mkdir -vp $MODULE_DIR/classes;

cp -v $QHS_HOME/demo/errorResponseCreator/*.java $MODULE_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
  -classpath $QHS_HOME/bin/qhs.jar $MODULE_DIR/*.java;

$JDK_HOME/bin/jar cvf $MODULE_DIR/errorResponseCreator.jar \
  -C $MODULE_DIR/classes ErrorResponseCreator;
```

- (2)、对/home/http/qhs3.3/config/base.xml 文件内部的一个 SSLServerSocket 元素更改为：

```
.....
<SSLServerSocket>
  <Created>true</Created>
  <Address>127.0.0.1</Address>
  <Port>10000</Port>
```



```

<KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>
<Password>123456</Password>
</SSLServerSocket>
.....

```

(3)、对/home/http/qhs3.3/config/errorResponseCreator.xml 文件的内容设置为：

```

<Config>
  <Created>true</Created>
  <JarFile>/home/http/qhsModules-10/errorResponseCreator.jar</JarFile>
  <Class>errorResponseCreator.ErrorResponseCreator</Class>
</Config>

```

(4)、运行/home/http/qhs3.3/startup.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行，并重新启动 qhs)。用户可以打开 log 文件(log 文件由 base.xml 配置文件的 LogFile 元素值说明)查看 qhs 检测到的、用于创建错误答复创建器的 ErrorResponseCreator.ErrorResponseCreator 类。成功启动 qhs 后，在浏览器的地址栏输入不能定位资源的地址，对 qhs 发出 http 请求并且在浏览器查看表达错误原因的页面。如：

- ①、https://127.0.0.1:10000/unknown-page-abcdefg.html
- ②、https://127.0.0.1:10000/unknown-service-module-uvwxyz.id
- ③、https://127.0.0.1:10000/abc%xx.txt

15、分析表单数据

表单数据(FormData)存在于浏览器等客户端发送给 qhs 的 http 请求中。表单数据属于 application/x-www-form-urlencoded 媒体类型或者 multipart/form-data 媒体类型。这两种媒体类型有一个相同的功能：使用适当的数据格式来组织控件名称、控件值。在用户代码中，通常要检测、选取合适的控件名称并且缓存与控件名称相对应的控件值。所谓控件名称(ControlName)即在浏览器显示的 html 页面上的图形界面部件的名称。控件名称由 html 页面数据的 input、button、select、textarea 等元素的 name 属性进行说明。控件值(ControlValue)有：

- (1)、显示在图形界面部件上的数据，包括图形界面部件显示的初始值(InitialValue)、浏览器用户对图形界面部件输入的数据。
- (2)、上传的文件。

如：

```

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
  </head>
  <body>
    <form method="get"
      enctype="application/x-www-form-urlencoded"
      action="list-names-and-values-u.id">
      <label>书名:</label>
      <input type="text" name="book-title" value="The Java Tutorials">
      <label>种类:</label>
      <select name="book-type">
        <option selected="true" >编程语言</option>
        <option>Web 技术</option>
        <option>大数据</option>
      </select>
      <input type="submit" value="查找">
    </form>
  </body>
</html>

```

上面的 html 页面数据中，input 元素的 name 属性说明了控件名称“book-title”，select 元素的 name 属性说明了控

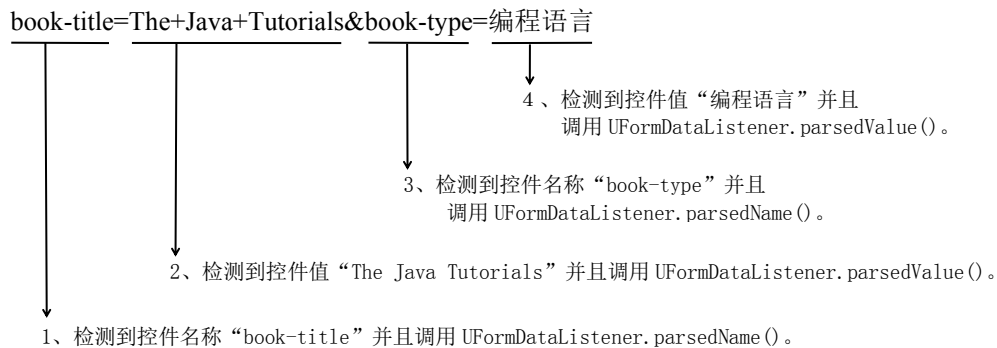
件名称“book-type”，名称为“book-title”的控件的初始值为“The Java Tutorials”，名称为“book-type”的控件的初始值为“编程语言”。

14.1、分析属于 application/x-www-form-urlencoded 媒体类型的表单数据

application/x-www-form-urlencoded 媒体类型使用以下数据格式来组织控件名称、控件值并形成表单数据：

控件名称 1=控件值 1[&控件名称 2=控件值 2][&控件名称 3=控件值 3].....[&控件名称 n=控件值 n]

qhs 内置有 application/x-www-form-urlencoded 媒体类型数据分析器，数据分析器按从左到右的顺序分析上述格式数据，并在分析的过程中，调用定义在监听器(属于 qhs.interfaces.UFormDataListener 接口类型的对象)上的各方法。在分析的过程中，如果数据分析器检测到控件名称，就调用监听器的 parsedName()方法，如果数据分析器检测到控件值，就调用监听器的 parsedValue()方法。如：



在浏览器生成 http 请求的过程中，浏览器跟据不同的请求方法把属于 application/x-www-form-urlencoded 媒体类型的表单数据放置在不同位置。如果请求方法是 GET，则浏览器把表单数据放置在 URL 的 query 部件位置上，并且使表单数据成为 URL 的 query 部件。如果请求方法是 POST，则浏览器把表单数据放置在 http 请求的载荷体(payload body)位置上，并且使表单数据成为 http 请求的载荷体。如果表单数据成为 URL 的 query 部件，在 HTTP 服务模块类的 invoke(qhs.interfaces.ThreadContext tc, ...)方法的内部，用户代码可以调用 tc.getRequest().getRequestURI().parseQuery(UFormDataListener listener, ...)方法来分析表单数据。如果表单数据成为 http 请求的载荷体，在 HTTP 服务模块类的 invoke(qhs.interfaces.ThreadContext tc, ...)方法的内部，用户代码可以调用 tc.getRequest().parsePayloadBody(UFormDataListener listener, ...)方法来分析表单数据。上述的 parseQuery()、parsePayloadBody()等两个方法具有创建 application/x-www-form-urlencoded 媒体类型数据分析器并且调用数据分析器分析表单数据的功能。在数据分析器分析表单数据的过程中，数据分析器把 qhs.interfaces.UFormDataListener 接口类型对象作为监听器，并调用定义在监听器上的各方法。

/home/http/qhs3.3/demo/uFormData/HTTPServiceModule_1.java、/home/http/qhs3.3/demo/uFormData/HTTPServiceModule_2.java 等两个演示文件用于分析属于 application/x-www-form-urlencoded 媒体类型的表单数据。其中，HTTPServiceModule_1.invoke()用于生成包含有 form 元素的 http 答复载荷体(这载荷体属于 text/html 媒体类型数据，即一般 html 页面数据)，HTTPServiceModule_2.invoke()用于分析表单数据并且把所有存在于表单数据内部的控件名称、控件值用于生成 http 答复。

以下步骤演示了从编译源文件到启动 qhs 的过程，示例的运行环境见第 3 节：

(1)、复制以下命令到终端并且执行命令：

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.3";
export MODULE_DIR="/home/http/qhsModules-11";
rm -Rvf $MODULE_DIR;
mkdir -vp $MODULE_DIR/classes;

cp -v $QHS_HOME/demo/uFormData/*.java $MODULE_DIR;
```

```
$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
    -classpath $QHS_HOME/bin/qhs.jar $MODULE_DIR/*.java;

$JDK_HOME/bin/jar cvf $MODULE_DIR/parsingUFormData.jar \
    -C $MODULE_DIR/classes uFormData;
```

(2)、对/home/http/qhs3.3/config/base.xml 文件内部的一个 SSLServerSocket 元素设置为：

```
.....
<SSLServerSocket>
  <Created>true</Created>
  <Address>127.0.0.1</Address>
  <Port>10000</Port>
  <KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>
  <Password>123456</Password>
</SSLServerSocket>
.....
```

(3)、对/home/http/qhs3.3/config/httpServiceModules.xml 文件的内容设置为：

```
<Config>
  <Modules JarFile="/home/http/qhsModules-11/parsingUFormData.jar" Enabled="true" >
    <Class Name="uFormData.HTTPServiceModule_1" Enabled="true" />
    <Class Name="uFormData.HTTPServiceModule_2" Enabled="true" />
  </Modules>
</Config>
```

(4)、运行/home/http/qhs3.3/startup.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行，并重新启动 qhs)。用户可以打开 log 文件(log 文件由 base.xml 配置文件的 LogFile 元素值说明)查看 qhs 检测到的 HTTP 服务模块类。

(5)、运行浏览器：

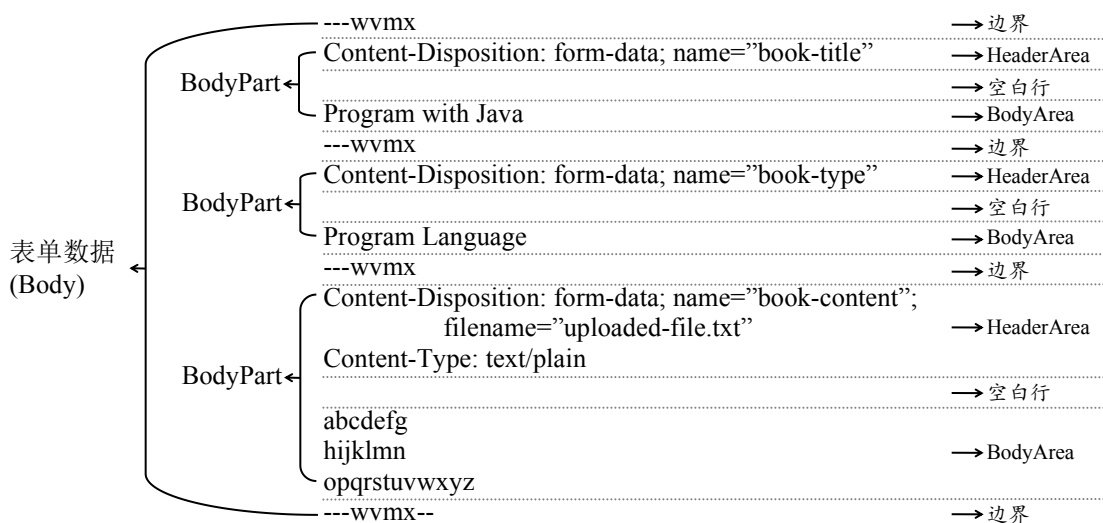
- ①、在浏览器的地址栏输入“https://127.0.0.1:10000/get-form-page-u.id”对 qhs 发出 http 请求，进入第②步骤。
- ②、在浏览器新显示的页面上点击 GET 或 POST，进入第③步骤。
- ③、在浏览器新显示的页面上输入书名、选择种类后，单击“查找”按钮，进入第④步骤。
- ④、在浏览器新显示的页面上显示有控件名称、控件值。

14.2、分析属于 multipart/form-data 媒体类型的表单数据

本节内容是作者查阅《HTML 4.01 Specification》的第 17 节、rfc2046 的第 5.1 节后并且结合实验而编写的。

属于 multipart/form-data 媒体类型的表单数据只存在于请求方法为 POST 的 http 请求内部，而且表单数据是 http 请求的载荷体。虽然 HTTP/2(见 rfc7540)要求 http 请求内部的所有头域名称必须由小写字母构成，但是，Firefox58、Chrome81 等浏览器生成的 multipart/form-data 媒体类型表单数据仍旧包含有由大小写字母构成的头域名称。在 HTTP 服务模块类的 invoke(qhs.interfaces.ThreadContext tc, ...)方法的内部，用户代码可以调用 tc.getRequest().parsePayloadBody(qhs.interfaces.MFormDataListener listener, ...)方法来分析表单数据。parsePayloadBody(qhs.interfaces.MFormDataListener listener, ...)方法具有创建 multipart/form-data 媒体类型数据分析器并且调用数据分析器分析表单数据的功能。在数据分析器分析表单数据的过程中，数据分析器把 listener 参数值用作为监听器，并调用定义在监听器上的各方法。

一个属于 multipart/form-data 媒体类型的表单数据就是一个 Body。一个 Body 由一个以上的 BodyPart 构成。多个 BodyPart 之间由边界(boundary)进行分隔。一个 BodyPart 由一个 HeaderArea、一个空白行(BlankLine)、一个 BodyArea 等组成。每一个控件名称作为 name 参数的值存在于 HeaderArea 内部的 Content-Disposition 头域。每一个控件值等于一个 BodyArea。下图说明了表单数据的结构，中间部分是一个完整的表单数据，左右侧部分是注释：



上图中，第 1 个 Content-Disposition 头域的 name 参数说明了控件名称“book-title”，第 2 个 Content-Disposition 头域的 name 参数说明了控件名称“book-type”，第 3 个 Content-Disposition 头域的 name 参数说明了控件名称“book-content”，第 1 个 BodyArea 说明了控件值“Program with Java”(即名称为“book-title”的控件的值)，第 2 个 BodyArea 说明了控件值“Program Language”(即名称为“book-type”的控件的值)，第 3 个 BodyArea 说明了控件值：

```
abcdefg
hijklmn
opqrstuvwxyz
```

即名称为“book-content”的控件的值，第 3 个 Content-Disposition 头域的 filename 参数说明了第 3 个 BodyArea 是文件内容等等。

qhs 内置有 multipart/form-data 媒体类型数据分析器，数据分析器按从左到右、从上至下的顺序分析表单数据，并在分析的过程中，调用定义在监听器(属于 qhs.interfaces.MFormDataListener 接口类型)上的各方法。在分析的过程中，如果数据分析器检测到 HeaderArea，就调用监听器的 parsedHeaders()方法，如果数据分析器检测到 BodyArea 或者正在读取 BodyArea，就调用监听器的 readingBodyArea()方法，如果数据分析器完成读取整个 BodyArea，就调用监听器的 readBodyArea()方法。

/home/http/qhs3.3/demo/mFormData/HTTPServiceModule_1.java、/home/http/qhs3.3/demo/mFormData/HTTPServiceModule_2.java 等两个演示文件用于分析属于 multipart/form-data 媒体类型的表单数据。其中，HTTPServiceModule_1.invoke()用于生成包含有 form 元素的 http 答复载荷体(这载荷体属于 text/html 媒体类型数据，即一般 html 页面数据)，HTTPServiceModule_2.invoke()用于分析表单数据并且把所有存在于表单数据内部的控件名称、控件值用于生成 http 答复。

以下步骤演示了从编译源文件到启动 qhs 的过程，示例的运行环境见第 3 节：

(1)、复制以下命令到终端并且执行命令：

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.3";
export MODULE_DIR="/home/http/qhsModules-12";
rm -Rvf $MODULE_DIR;
mkdir -vp $MODULE_DIR/classes;

cp -v $QHS_HOME/demo/mFormData/*.java $MODULE_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $MODULE_DIR/classes \
```

```
-classpath $QHS_HOME/bin/qhs.jar $MODULE_DIR/*.java;
```

```
$JDK_HOME/bin/jar cvf $MODULE_DIR/parsingMFormData.jar \  
-C $MODULE_DIR/classes mFormData;
```

(2)、对/home/http/qhs3.3/config/base.xml 文件内部的一个 SSLServerSocket 元素、HttpRequest 元素设置为：

```
.....  
<SSLServerSocket>  
  <Created>true</Created>  
  <Address>127.0.0.1</Address>  
  <Port>10000</Port>  
  <KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>  
  <Password>123456</Password>  
</SSLServerSocket>  
.....  
<HttpRequest>  
  <HeaderBlockBufferSize>20k</HeaderBlockBufferSize>  
  <PayloadBodyBufferSize>17m</PayloadBodyBufferSize>  
</HttpRequest>  
.....
```

(3)、对/home/http/qhs3.3/config/httpServiceModules.xml 文件的内容设置为：

```
<Config>  
  <Modules JarFile="/home/http/qhsModules-12/parsingMFormData.jar" Enabled="true" >  
    <Class Name="mFormData.HTTPServiceModule_1" Enabled="true" />  
    <Class Name="mFormData.HTTPServiceModule_2" Enabled="true" />  
  </Modules>  
</Config>
```

(4)、运行/home/http/qhs3.3/startup.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行，并重新启动 qhs)。用户可以打开 log 文件(log 文件由 base.xml 配置文件的 LogFile 元素值说明)查看 qhs 检测到的 HTTP 服务模块类。

(5)、运行浏览器：

- ①、在浏览器的地址栏输入“https://127.0.0.1:10000/get-form-page-m.id”对 qhs 发出请求，进入第②步骤。
- ②、在浏览器新显示的页面上输入书名、选择种类、选择文件(被选择的多个文件的合计尺寸最大值等于 base.xml 配置文件内部的 HttpRequest>PayloadBodyBufferSize 元素值)后，单击“上传”按钮，进入第③步骤。
- ③、在浏览器新显示的页面上显示有控件名称、控件值。

16、在 NetBeansIDE 调试

开发人员可以在 NetBeansIDE 开发、调试所有 qhs 模块。本节内容使用 NetBeansIDE13(Linux 版)的截图说明最基本的项目设置、项目调试过程。在 NetBeansIDE 调试时，可用两种方法：本机调试、远程调试。

16.1、本机调试

本机调试要求 NetBeansIDE 与 qhs 运行在同一计算机主机上。

使用本调试方法时，必须对所有 NetBeansIDE 项目执行以下设置：

- (1)、导入 gson.jar 包、qhs.jar 包。
- (2)、设置运行项目的主类为 qhs.Start，qhs.Start 类是 qhs 运行的起始，处于 qhs.jar 包。
- (3)、设置--ServerRoot 运行参数，参数的格式为“--ServerRoot=qhs 的安装目录”。

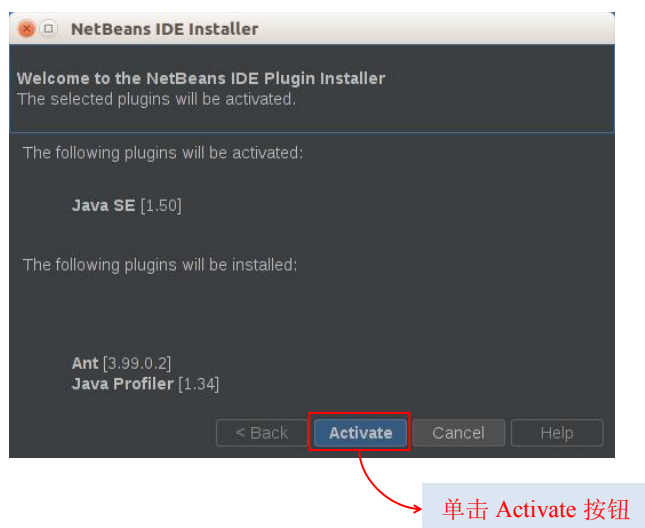
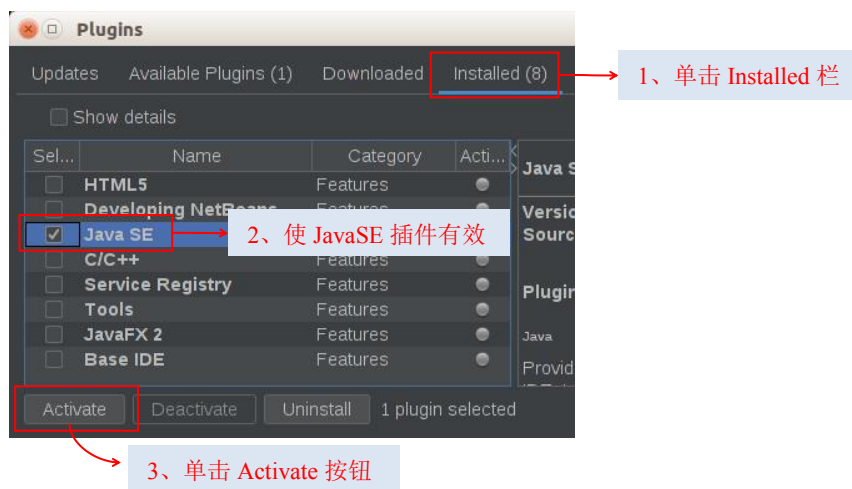
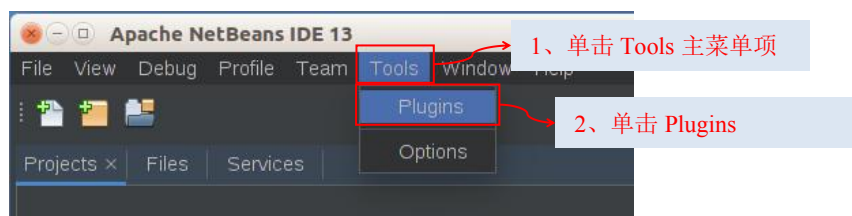
(4)、设置--Config 运行参数，参数的格式为“--Config=配置目录”。

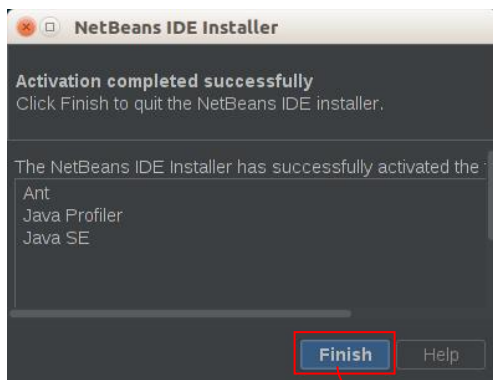
(5)、适当编辑配置目录内部的配置文件。

下面的步骤、截图演示了项目的创建、设置、调试等等，示例的运行环境见第 3 节：

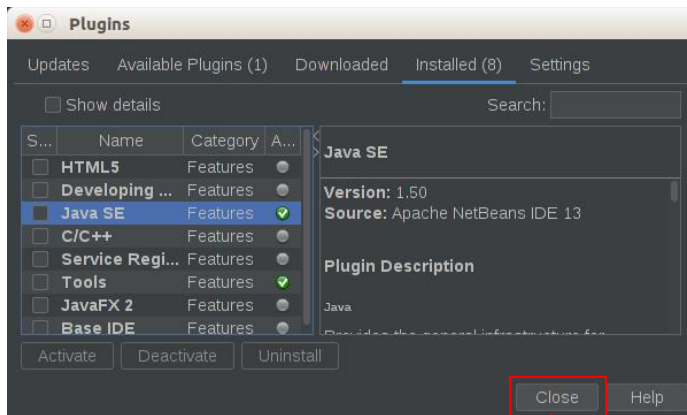
(1)、启动 NetBeansIDE。

(2)、激活"JavaSE"插件。本步骤是可选的，如果已激活"JavaSE"插件，就跳过本步骤。





单击 Finish 按钮

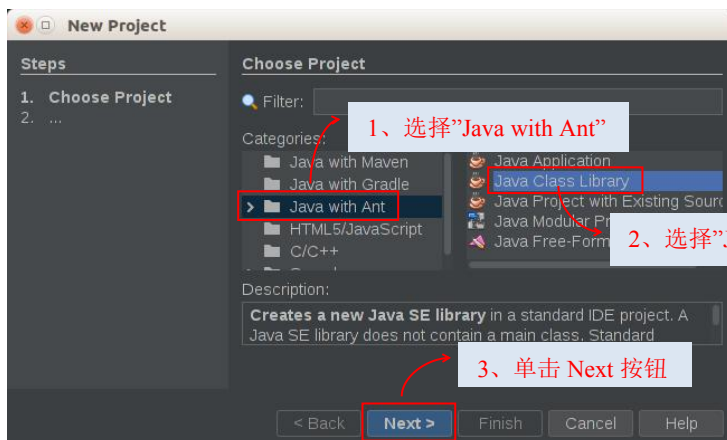


单击 Close 按钮

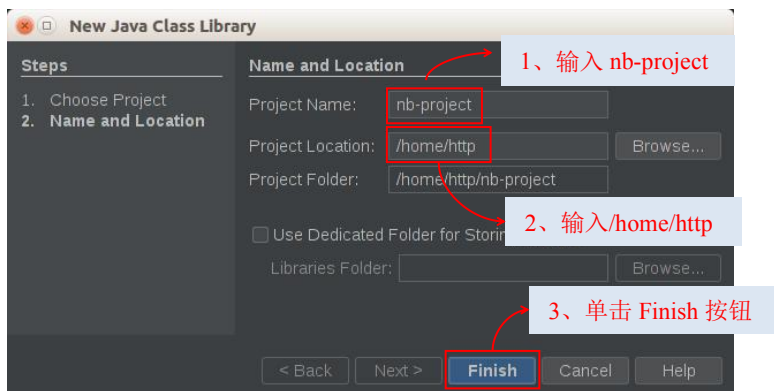
(3)、在 NetBeansIDE 创建 nb-project 项目。



(4)、



(5)、



(6)、



(7)、对 nb-project 项目导入 gson.jar 包、qhs.jar 包。



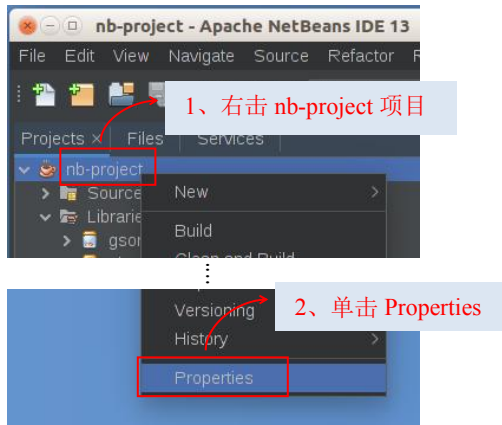
(8)、



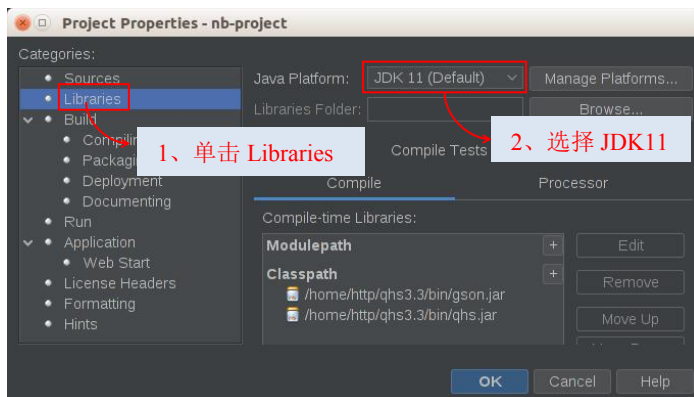
(9)、



(10)、

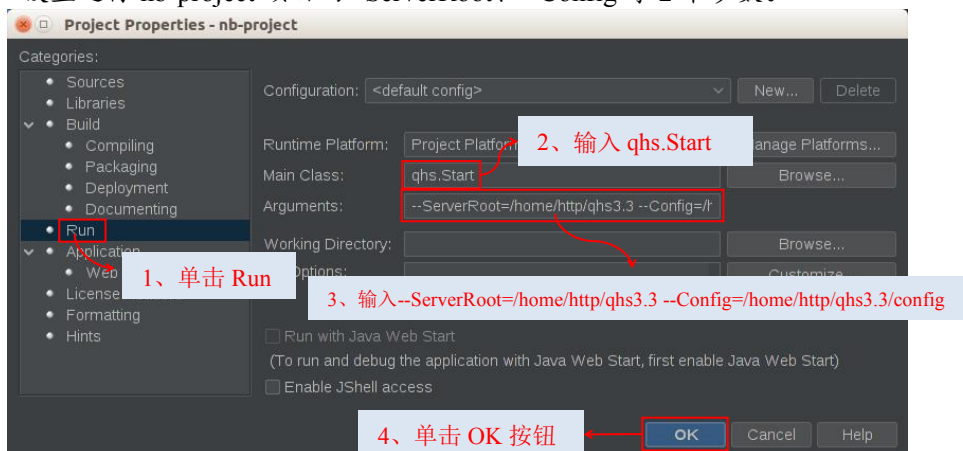


(11)、



(12)、设置运行 nb-project 项目的主类为 qhs.Start。

设置运行 nb-project 项目的--ServerRoot、--Config 等 2 个参数。



(13)、在终端执行以下命令，复制演示文件到 nb-project 项目：

```
export DST=/home/http/nb-project/src/httpServiceModule;  
rm -Rvf $DST;  
mkdir -p $DST;  
cp -v /home/http/qhs3.3/demo/httpServiceModule/*.java $DST;
```

(14)、对/home/http/qhs3.3/config/base.xml 配置文件内部的一个 SSLServerSocket 元素设置为：

```
.....  
<SSLServerSocket>  
  <Created>true</Created>  
  <Address>127.0.0.1</Address>  
  <Port>10000</Port>  
  <KeyStore>/home/http/qhs3.3/sslFiles/httpServer.ks</KeyStore>  
  <Password>123456</Password>  
</SSLServerSocket>  
.....
```

(15)、对/home/http/qhs3.3/config/httpServiceModules.xml 配置文件的内容设置为：

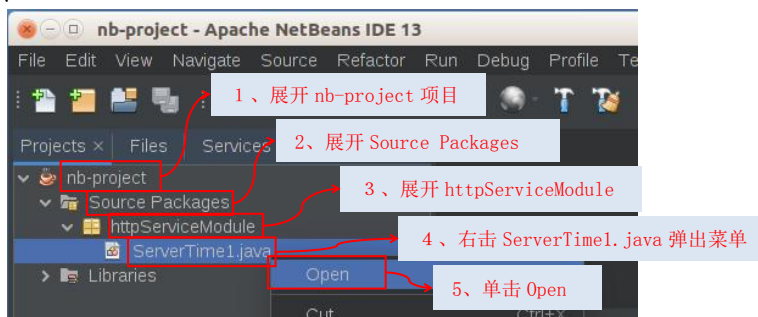
```
<Config>  
  <Modules JarFile="/home/http/nb-project/dist/nb-project.jar" Enabled="true" >  
    <Class Name="httpServiceModule.ServerTime1" Enabled="true" />  
  </Modules>  
</Config>
```

(16)、编译 nb-project 项目。

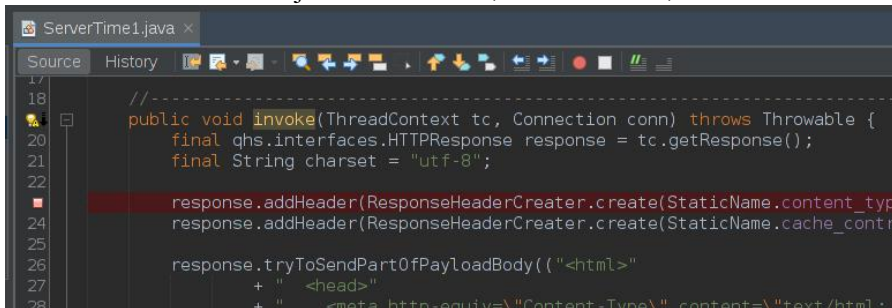


执行上面的所有步骤以后，可以立即调试 nb-project 项目。调试 nb-project 项目时，应首先在源码上设置一个或多个断点，然后在 Debug 主菜单项上选择调试方法。如：

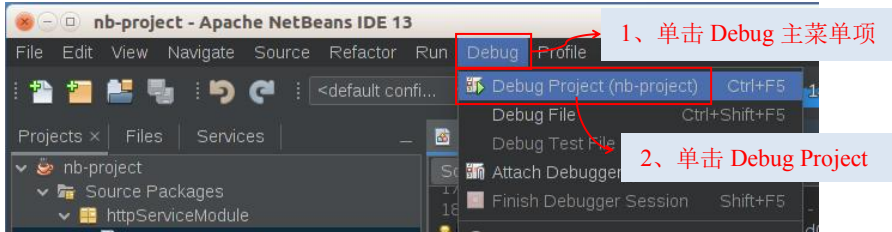
(17)、



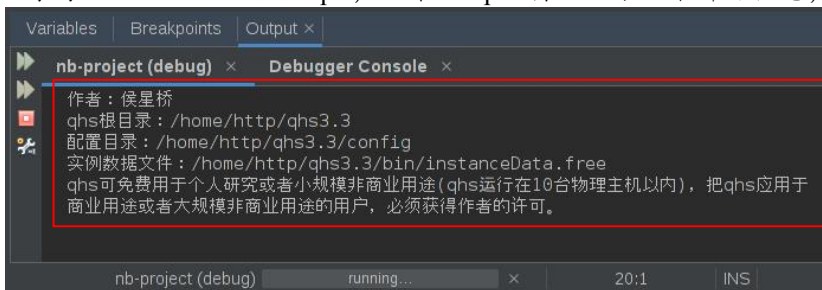
(18)、在名称为 ServerTime1.java 的窗口左侧，单击行号 23，设置断点的源码行的背景变为红色。



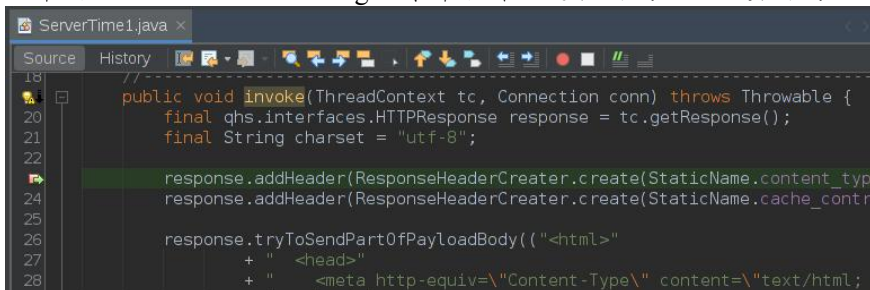
(19)、



(20)、等待 NetBeansIDE 启动 qhs，如果 Output 窗口显示以下许可信息，说明成功启动 qhs。



(21)、启动浏览器，在浏览器的地址栏输入“https://127.0.0.1:10000/server-time-1.id”，使浏览器对 qhs 发送 http 请求。如果设置了断点的源码行的背景从红色变为绿色，说明 qhs 已接收到 http 请求。此时，开发人员可以单击 NetBeansIDE 的 Debug 主菜单项并且选取调试方法继续调试。



16.2、远程调试

使用本调试方法时，必须：

- (1)、对 /home/http/qhs3.3/startup.sh 脚本增加 JVM 运行选项-agentlib。
- (2)、使 NetBeansIDE 连接到调试器。

下面的步骤、截图演示了项目的设置、调试等等，示例的运行环境见第 3 节：

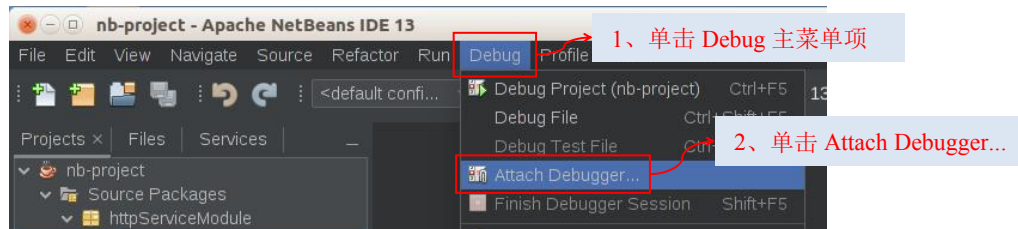
- (1)、经过 16.1 节的所有步骤创建了 nb-project 项目。
- (2)、对 /home/http/qhs3.3/startup.sh 脚本改为：

```
#!/bin/bash
exec "/usr/java/jdk-11/bin/java" \
-agentlib.jdwp=transport=dt_socket,server=y,address=localhost:20000,suspend=n \
-classpath "/home/http/qhs3.3/bin/qhs.jar:/home/http/qhs3.3/bin/gson.jar" \
qhs.Start \
--ServerRoot="/home/http/qhs3.3" \
--Config="/home/http/qhs3.3/config" \
;
```

- (3)、在终端运行/home/http/qhs3.3/startup.sh 脚本启动 qhs。(如果 qhs 正在运行，就运行/home/http/qhs3.3/shutdown.sh 来终止 qhs 的运行，并重新启动 qhs)。如果终端显示以下许可信息说明成功启动 qhs：

```
http@http:~$ /home/http/qhs3.3/startup.sh
Listening for transport dt_socket at address: 20000
产品名称: qhs(快速http服务器)
版本: 3.3
作者: 侯星桥
qhs根目录: /home/http/qhs3.3
配置目录: /home/http/qhs3.3/config
实例数据文件: /home/http/qhs3.3/bin/instanceData.free
qhs可免费用于个人研究或者小规模非商业用途(qhs运行在10台物理主机以内)，把qhs应用于商业用途或者大规模非商业用途的用户，必须获得作者的许可。
```

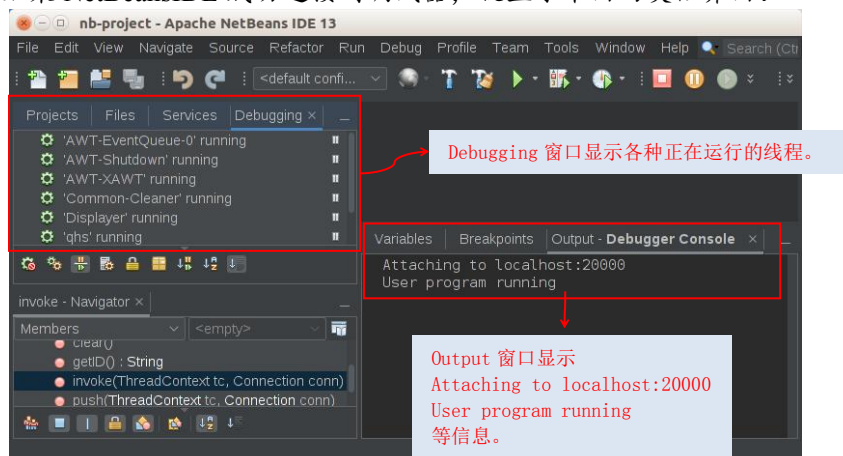
- (4)、在 NetBeansIDE 尝试连接调试器。



- (5)、



- (6)、如果 NetBeansIDE 成功连接到调试器，就显示下面的类似界面：



- (7)、参考 16.1 节的第(17)、(18)等 2 个步骤，对源码行设置断点。

- (8)、参考 16.1 节的第(21)步骤，启动浏览器，在浏览器的地址栏输入”https://127.0.0.1:10000/server-time-1.id”，使浏览器对 qhs 发送 http 请求。如果设置了断点的源码行的背景从红色变为绿色，说明 qhs 已接收到 http 请求。此时，开发人员可以单击 NetBeansIDE 的 Debug 主菜单项并且选取调试方法继续调试。