qhs 的用户手册

(qhs 是快速 http 服务器的简称。)

1、qhs3.2 更改、新增的功能

略。

2、qhs 的技术特点

qhs(即快速 http 服务器)是一款参考 rfc7540、rfc7541 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。

3、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。理论上，qhs 可运行在一切安装有 JDK11 的 Linux 操作系统，因为 qhs 是一款纯 Java 程序。
支撑软件：JDK11。

4、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/downloads/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。
(3)、以 http 账户登录操作系统桌面。
(4)、OpenJDK11 安装在/usr/java/jdk-11 目录。OpenJDK11 用于运行 ApacheNetBeansIDE11.3、运行示例。用户可在 http://jdk.java.net 下载 OpenJDK11。
(5) 、qhs 安装在/home/http/qhs3.2 目录。
(6) 、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/qhs3.2/sslFiles/ca.crt 证书文件。ca.crt 证书文件的创建可参考第 8.1.1 节。
(7)、安装 ApacheNetBeansIDE11.3。用户可在 http://netbeans.apache.org/download/下载 ApacheNetBeansIDE11.3。在安装 ApacheNetBeansIDE11.3 以前需要安装 OpenJDK8，OpenJDK8 用于运行 ApacheNetBeansIDE11.3 安装程序，用户可在 http://jdk.java.net 下载 OpenJDK8。

5、安装 qhs

qhs 的安装包为 qhs3.2-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 qhs3.2-inst.jar \
    --InstPath=/home/http/qhs3.2 \
    --JavaProgPath=/usr/java/jdk-11/bin/java;
```

上一命令把 qhs 安装在/home/http/qhs3.2 目录。

6、启动 qhs、终止 qhs 的运行

在终端运行 qhs 安装目录内部的 startup[1-8].sh 脚本。如：qhs 的安装目录为/home/http/qhs3.2，运行以下脚本启动 qhs：

```
/home/http/qhs3.2/startup1.sh &
```

若是安装 qhs 后立即运行 startup1.sh 脚本而且按照第 4 节对浏览器导入了 ca.crt，可在浏览器的地址栏输入 https://127.0.0.1:10010 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[1-8].sh 脚本来终止 qhs 的运行。运行 shutdown[1-8].sh 脚本时，可选的设置有明文密码参数。如：qhs 的安装目录为/home/http/qhs3.2，运行以下脚本：

```
/home/http/qhs3.2/shutdown1.sh 123abc;
```

123abc 是安装 qhs 以后的初始密码，用户可以编辑/home/http/qhs3.2/config[1-8]/shutdown.xml 配置文件来更改密码，shutdown.xml 内部有注释说明更改密码的方法。

7、运行多个 qhs 实例

在一个 Linux 操作系统，允许运行任意个 qhs 实例，每一个 qhs 实例是一个独立的 Linux 进程(process)，每一个 qhs 实例引用一个独立的配置目录。

为了便于用户操作，在安装 qhs 的过程中，安装程序预先创建了 8 个 qhs 实例配置(包括配置目录以及引用配置目录的脚本)：

在终端运行以下命令显示有/home/http/qhs3.2/startup1.sh 至/home/http/qhs3.2/startup8.sh 脚本：

```
ls  /home/http/qhs3.2/startup[1-8].sh;
```

在终端运行以下命令显示有/home/http/qhs3.2/shutdown1.sh 至/home/http/qhs3.2/shutdown8.sh 脚本：

```
ls  /home/http/qhs3.2/shutdown[1-8].sh;
```

在终端运行以下命令显示有/home/http/qhs3.2/config1 至/home/http/qhs3.2/config8 目录：

```
ls  -d  /home/http/qhs3.2/config[1-8];
```

startup1.sh 用于启动第 1 个 qhs 实例，shutdown1.sh 用于终止第 1 个 qhs 实例的运行，startup1.sh 与 shutdown1.sh 同时引用 config1 配置目录。

startup2.sh 用于启动第 2 个 qhs 实例，shutdown2.sh 用于终止第 2 个 qhs 实例的运行，startup2.sh 与 shutdown2.sh 同时引用 config2 配置目录。

startup3.sh 用于启动第 3 个 qhs 实例，shutdown3.sh 用于终止第 3 个 qhs 实例的运行，startup3.sh 与 shutdown3.sh 同时引用 config3 配置目录。

.....

startup8.sh 用于启动第 8 个 qhs 实例，shutdown8.sh 用于终止第 8 个 qhs 实例的运行，startup8.sh 与 shutdown8.sh 同时引用 config8 配置目录。

在启动多个 qhs 实例以前，需要确保各配置目录内部的 base.xml、shutdown.xml 等文件的 ServerSocket>Addres

s、ServerSocket>Port、Port 等元素值说明的 IP 地址与端口的组合必须是不相同的，否则，可能出现因不能创建服务器套接字而不能启动 qhs 实例。

安装 qhs 以后，qhs 的安装目录包含有 free-instance-data.txt 文件，free-instance-data.txt 文件用于运行免费的 qhs 实例。如果用户购买了付费的 qhs 实例数据包，可以解压数据包并且复制 qhs 实例数据文件(如：instance-data.001 文件)到上述配置目录(如：/home/http/qhs3.2/config1 目录)，详细的操作说明见 lcs-manual.pdf 文件。

8、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[1-8]子目录（即配置目录）。如果 qhs 的安装目录为/home/http/qhs3.2，那么配置目录为/home/http/qhs3.2/config[1-8]。

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请求分析器、用于分析请求 URI 与重写 URL 的 URIParser 对象、全局对象、线程特定对象、连接特定对象等等。

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

8.1、基本配置

基本配置存放在/home/http/qhs3.2/config[1-8]/base.xml 文件，base.xml 文件内部包含有比较多的说明注释。

8.1.1、创建密钥库文件、签证书

浏览器可与 qhs 进行 TLSv1.3 加密通信。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口令必须完全相同。

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 -keypass $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 -keypass $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“证书机构”栏或“证书颁发机构”栏或“授权中心”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/qhs3.2/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/qhs3.2/docs/create-certs-2.txt 文件。

8.2、媒体类型配置

媒体类型配置存放在/home/http/qhs3.2/config[1-8]/mediaTypes.xml 文件，mediaTypes.xml 文件内部包含有说明注释。

8.3、服务模块配置

模块配置存放在/home/http/qhs3.2/config[1-8]/modules.xml 文件，modules.xml 文件内部包含有说明注释。

8.4、全局对象、线程特定对象、连接特定对象等配置

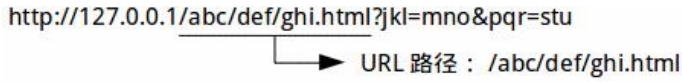
全局对象、线程特定对象、连接特定对象等配置存放在/home/http/qhs3.2/config[1-8]/objects.xml 文件，objects.xml 文件内部包含有说明注释。

8.5、JVM 运行选项的配置

JVM 运行选项的配置存放在/home/http/qhs3.2/config[1-8]/options.xml 文件，options.xml 文件内部包含有说明注释。JVM 运行选项是指调用 java 可执行文件启动 qhs 时所用的 JVM 选项。用户可在终端输入“java -h”或“java -X”命令来显示可用的 JVM 运行选项。

8.6、缺省页面文件名、定制头域、推送 URL 等配置

本配置存放在/home/http/qhs3.2/config[1-8]/urlPaths.xml 文件，urlPaths.xml 文件存放与 URL 路径相关联的缺省页面文件名、定制头域等配置。定制头域即由用户自主设计的、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：



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 就从重写后的 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有缺省页面文件名（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有效的，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 等属性值是有效的，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 属性值说明的是配置值而不是标准的 expires 头域值，qhs 分析配置值并生成标准的 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 元素到 def.html 元素的路径形成一个 URL 路径：/abc/def.html，而且 def.html 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，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(pushed 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。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.html 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。

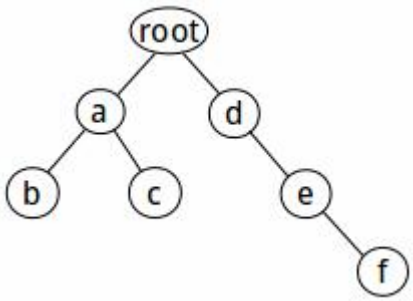
(3)、没有更多的节点名称与路径段进行比较。

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继承缺省页面文件名、定制头域等数据，这属于默认的情况。Inherited 属性值等于空串表示子元素不从父元素继承数据。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：

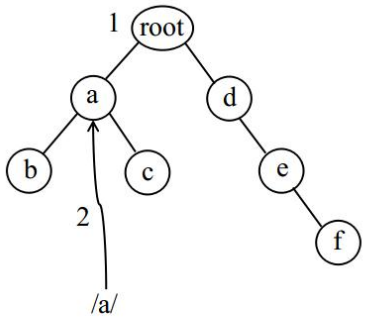
```
/               =>  /first-page.html
/x/y/z/         =>  /x/y/z/first-page.html
/a/             =>  /a/plain.txt
/a/x/y/z/       =>  /a/x/y/z/plain.txt
/a/b/           =>  /a/b/pic.png
/a/b/x/y/z/     =>  /a/b/x/y/z/pic.png
/a/c/           =>  /a/c/plain.txt
/a/c/x/y/z/     =>  /a/c/x/y/z/plain.txt
/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

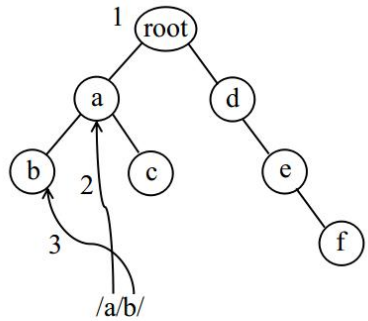
对 http 答复追加定制头域，只适用于发送静态资源文件的情况。下面“=>”左侧的 URL 路径（'*'表示 URL 路径是经过追加缺省页面文件名后而形成的）用于生成文件路径，并且假设相应的静态资源文件存在于文档根目录。下面“=>”右侧的头域是对 http 答复追加的定制头域：

```
*  /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"
```

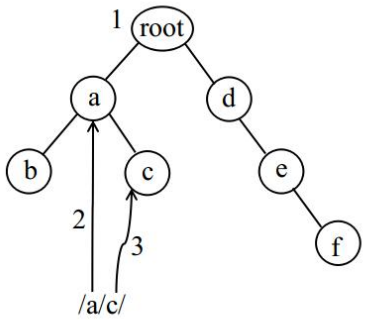
下图演示了 qhs 使用上面“=>”左侧的 URL 路径在节点树上定位节点的过程，图中的数字表示定位操作的顺序，所有定位操作必须从 root 节点开始：



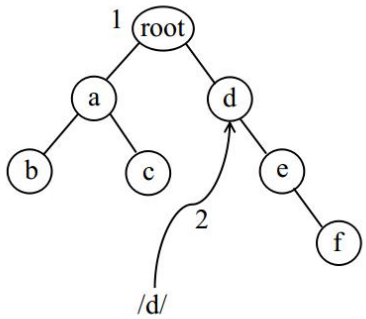
/a/
/a/plain.txt
/a/test-3.txt
/a/x/y/z/
/a/x/y/z/plain.txt
/a/x/y/z/test-4.txt



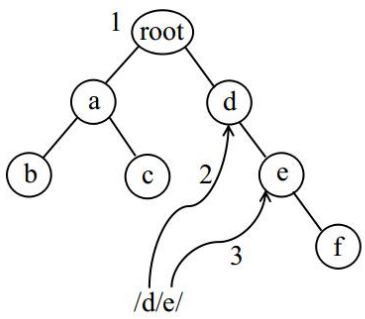
/a/b/
/a/b/pic.png
/a/b/test-5.png
/a/b/x/y/z/
/a/b/x/y/z/pic.png
/a/b/x/y/z/test-6.png



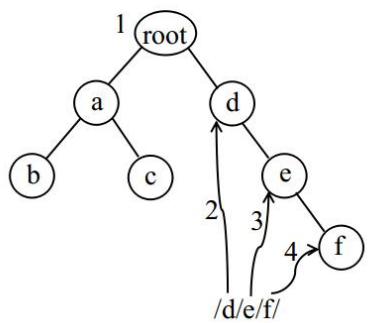
/a/c/
/a/c/plain.txt
/a/c/test-7.txt
/a/c/x/y/z/
/a/c/x/y/z/plain.txt
/a/c/x/y/z/test-8.txt



/d/
/d/first-page.html
/d/test-9.html
/d/x/y/z/
/d/x/y/z/first-page.html
/d/x/y/z/test-10.html



/d/e/
/d/e/index.html
/d/e/test-11.html
/d/e/x/y/z/
/d/e/x/y/z/index.html
/d/e/x/y/z/test-12.html



/d/e/f/
/d/e/f/about.html
/d/e/f/test-13.html
/d/e/f/x/y/z/
/d/e/f/x/y/z/about.html
/d/e/f/x/y/z/test-14.htm

9、开发服务模块

遵循下面定义的 Java 类称为服务模块类。一个服务模块是一个服务模块类的实例。服务模块类的规模可大可小、由开发人员决定。服务模块构建用于显示动态页面的 http 答复、推送答复(pushed response)。

服务模块类的定义：

(1)、public static String getServiceModuleID() {
//.....
}

qhs 调用服务模块类的 getServiceModuleID()类方法取得模块的 ID(即服务模块 ID)并且移除服务模块 ID 上的空格、回车、换行等空白字符。

(2)、一个公开的、无参数的构造器。

(3)、public void invoke(qhs.interfaces.ThreadContext tc) {
//.....
}

qhs 通过调用服务模块的 invoke()实例方法来构建 http 答复。在 invoke()方法的内部应放置有构建 http 答复的代码。每个服务模块类必须定义有本方法。

(4)、public void push(qhs.interfaces.ThreadContext tc) {
//.....
}

qhs 通过调用服务模块的 push()实例方法来构建推送答复。在 push()方法的内部应放置有构建推送答复的代码。每个服务模块类必须定义有本方法。

(5)、public String update(String str) {
//.....
}

update()实例方法用于更新服务模块的内部状态。用户代码通过调用定义在 serviceModulePool 系统对象上的 updateServiceModule()方法来更新服务模块的内部状态，updateServiceModule()方法直接调用此处说明的 update()方法。

(6)、public void clear() {
//.....
}

clear()实例方法用于清除服务模块的内部状态。在 qhs 终止运行的过程中，qhs 调用 clear()方法。

module.xml 配置文件用于说明服务模块类。在 qhs 启动的过程中，qhs 参考 module.xml 配置文件创建多个服务模块并且把创建的服务模块分配给每个 http 处理线程。

9.1、构建发送 http 答复

当浏览器等客户端对服务模块发出 http 请求时，qhs 就通过调用服务模块的 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/qhs3.2/demo/serviceModules/ServiceModule1.java，ServiceModule1.java 定义的 ServiceModule1 类用于创建 ID 为“speed-test”的服务模块（简称为 speed-test 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 4 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/qhs3.2";
export TEST_DIR="/home/http/qhsModules-01";
rm -Rvf $TEST_DIR;
mkdir -vp $TEST_DIR;

cp -v $QHS_HOME/demo/serviceModules/ServiceModule1.java $TEST_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $TEST_DIR \
        -classpath $QHS_HOME/bin/qhs.jar \
        $TEST_DIR/ServiceModule1.java;

$JDK_HOME/bin/jar cvf $TEST_DIR/serviceModules.jar -C $TEST_DIR serviceModules;
```

(2)、对/home/http/qhs3.2/config1/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/qhs3.2/sslFiles/httpServer.ks</KeyStore>
    <Password>123456</Password>
</SSLServerSocket>
......
```

(3)、对/home/http/qhs3.2/config1/modules.xml 文件的内容设置为：

```
<Config>
    <Modules JarFile="/home/http/qhsModules-01/serviceModules.jar" Enabled="true" >
        <Class Name="serviceModules.ServiceModule1" Enabled="true" />
    </Modules>
</Config>
```

(4)、运行/home/http/qhs3.2/startup1.sh 来启动 qhs（如果 qhs 正在运行，就运行/home/http/qhs3.2/shutdown1.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/speed-test.id”对 qhs 发出 http 请求（即对 speed-test 服务模块发出 http 请求），则 qhs 把包含有速度测试结果的 http 答复发送给浏览器。如果在浏览器的地址栏输入“https://127.0.0.1:10000/speed-test.id?reset”对 qhs 发出 http 请求，则 qhs 清除先前的测速数据然后发送 http 答复给浏览器。上述的 URL 中，“https://127.0.0.1:10000/”与“.id”之间的“speed-test”就是服务模块 ID。如果浏览器对其它服务模块发出 http 请求，只需把“speed-test”改为相应的服务模块 ID 就可以了。

其实，上述的 speed-test 服务模块，可用于测试 qhs 调用服务模块的速度，用户可以运行各种性能测试程序对 speed-test 服务模块发出 http 请求。上面的 SSLServerSocket 元素创建的服务器套接字是以 h2 接收客户端的连接。如果用户的性能测试程序支持无 Upgrade 头域的 direct-h2，可以使用 ServerSocket 元素创建服务器套接字。direct-h2 功能只存在于付费的 qhs 实例。如果用户想测试 qhs 的 direct-h2 功能，只有购买 qhs 实例才能获得 direct-h2 功能。qhs 的作者通过自研的性能测试程序测试了 qhs 调用服务模块的速度，以下测试条件、实测数据供用户参考：

(1)、CPU 型号：AMD A4-6300。
(2)、内存：DDR3、4096MB。
(3)、操作系统：UbuntuKylin16。
(4)、JDK：OpenJDK11。
(5)、请求 URL：/speed-test.id。
(6)、qhs 的线程模型：单线程模型。
(7)、服务器套接字：direct-h2。
(8)、http 连接：阻塞型，一个 http 连接最多接收 100 个 http 请求。
(9)、性能测试程序：套接字多路复用、单线程模型，创建 20 个 http 连接，运行时不断随机选择 1 个 http 连接。
(10)、性能测试程序与 qhs 同时运行在同一操作系统。
(11)、性能测试程序的运行时间：10 分钟。
(12)、测速结果：19000requests/second 至 21000requests/second(因电力状况不同测速结果处于本范围以内)。

9.2、构建发送推送答复

对于开发用户，构建发送推送答复(pushed 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/qhs3.2/demo/serviceModules/TimeShower.java，TimeShower.java 定义的 TimeShower 类用于创建 time-shower 服务模块。

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 4 节，复制以下命令到终端并且执行命令，命令编译生成的 timeShower.jar 包的使用在第 10 节：

```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.2";
export TEST_DIR="/home/http/qhsModules-02";
rm -Rvf $TEST_DIR;
mkdir -vp $TEST_DIR;

cp -v $QHS_HOME/demo/serviceModules/TimeShower.java $TEST_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $TEST_DIR \
        -classpath $QHS_HOME/bin/qhs.jar \
        $TEST_DIR/TimeShower.java;

$JDK_HOME/bin/jar cvf $TEST_DIR/timeShower.jar -C $TEST_DIR serviceModules;
```

10、服务器推送

10.1、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(Server Push)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 8.6 节对 urlPaths.xml 配置文件的格式作了说明。

以/home/http/qhs3.2/demo/serverPush/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="/time-shower.id" />
    </server-push.html>
</Config>
```

在上面的配置中，从 Config 元素到 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.htm 文件以前，推送/pics/11.png 至/time-shower.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。/time-shower.id 用于调用 time-shower 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/time-shower.id 时，qhs 就调用 TimeShower.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 4 节:

(1)、参考 9.2 节编译生成 timeShower.jar 包。

(2)、对/home/http/qhs3.2/config1/base.xml 文件内部的 DocumentRoot 元素值以及一个 SSLServerSocket 元素设置为:

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

(3)、复制/home/http/qhs3.2/demo/serverPush/config/urlPaths.xml 配置文件到/home/http/qhs3.2/config1 目录:

```
cp -v /home/http/qhs3.2/demo/serverPush/config/urlPaths.xml /home/http/qhs3.2/config1;
```

(4)、对/home/http/qhs3.2/config1/modules.xml 配置文件的内容设置为:

```
<Config>
    <Modules JarFile="/home/http/qhsModules-02/timeShower.jar" >
        <Class Name="serviceModules.TimeShower" />
    </Modules>
</Config>
```

(5)、运行/home/http/qhs3.2/startup1.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.2/shutdown1.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><img src="/pics/11.png" /></td>
      <td><img src="/pics/12.png" /></td>

.....

      <td><img src="/pics/58.png" /></td>
      <td><img src="/pics/59.png" /></td>
    </tr>
    <tr><td colspan="4"><iframe width="600" height="80" src="/time-shower.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、/time-shower.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/qhs3.2/demo/serverPush/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/qhs3.2/demo/serverPush/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 地址时却不能够发起服务器推送，原因是在/home/http/qhs3.2/demo/serverPush/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调试功能，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。

10.2、通过服务模块发起服务器推送

通过服务模块发起服务器推送即在调用服务模块构建发送 http 答复以前调用 ThreadContext.push()方法推送 URL。通过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/qhs3.2/demo/serviceModules/ServerPush.java 文件为例，ServerPush.java 文件结构：

```
.....
public void invoke(qhs.interfaces.ThreadContext tc) throws Throwable {
    .....
    for (String url : urls) {
        tc.push(url.getBytes());
    }
    .....
    for (int i = 0; i < urls.length - 1; i++) {
        response.addHeader(qhs.sys.ResponseHeaderCreater.create("link",
                "<" + urls[i] + ">; rel=preload; as=image"));
    }
    response.addHeader(qhs.sys.ResponseHeaderCreater.create("link",
            "<" + urls[urls.length - 1] + ">; rel=preload; as=document"));
    .....
        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/qhs3.2/demo/serviceModules/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/time-shower.id 时，qhs 就调用 TimeShower.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 4 节:

(1)、参考 9.2 节编译生成 timeShower.jar 包。

(2)、对/home/http/qhs3.2/config1/base.xml 文件内部的 DocumentRoot 元素值以及一个 SSLServerSocket 元素设置为:

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

(3)、对/home/http/qhs3.2/config1/urlPaths.xml 文件的内容设置为:

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

(4)、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/qhs3.2";
export TEST_DIR="/home/http/qhsModules-03";
rm -Rvf $TEST_DIR;
mkdir -vp $TEST_DIR;
cp -v $QHS_HOME/demo/serviceModules/ServerPush.java $TEST_DIR;
$JDK_HOME/bin/javac -encoding utf-8 -d $TEST_DIR \
    -classpath $QHS_HOME/bin/qhs.jar \
    $TEST_DIR/ServerPush.java;
$JDK_HOME/bin/jar cvf $TEST_DIR/serverPush.jar -C $TEST_DIR serviceModules;
```

(5)、对/home/http/qhs3.2/config1/modules.xml 文件的内容设置为:

```
<Config>
```

```
    <Modules JarFile="/home/http/qhsModules-02/timeShower.jar" >
        <Class Name="serviceModules.TimeShower" />
    </Modules>
    <Modules JarFile="/home/http/qhsModules-03/serverPush.jar" >
        <Class Name="serviceModules.ServerPush" />
    </Modules>
</Config>
```

(6)、运行/home/http/qhs3.2/startup1.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.2/shutdown1.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 检测到的服务模块类。

(7)、运行浏览器，在浏览器的地址栏输入“https://127.0.0.1:10000/server-push.id”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.id 的源代码，可以看到这样的结构：

```
.....

<table border="0" frame="void" cellspacing="1" cellpadding="0" >
    <tr>
      <td><img src="/pics/11.png" /></td>
      <td><img src="/pics/12.png" /></td>

.....

      <td><img src="/pics/58.png" /></td>
      <td><img src="/pics/59.png" /></td>
    </tr>
    <tr><td colspan="4"><iframe width="600" height="80" src="/time-shower.id" /></td></tr>

.....
```

在 server-push.id 的源代码中，有多个 src 属性分别引用了/pics/11.png 至/pics/59.png、/time-shower.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。

11、qhs 对象

qhs 对象分为系统对象、用户开发的对象。

系统对象分为系统全局对象、系统线程特定对象。系统全局对象有 classLoaderPool、urlPaths 等对象，系统线程特定对象有 serviceModulePool、sysThreadSpecificObjectPool、urlRewriter、userThreadSpecificObjectPool 等对象。这些对象的用途及说明见/home/http/qhs3.2/docs/sysobjs 目录。在 qhs 启动的过程中，qhs 创建了一个 classLoaderPool 对象、一个 urlPaths 对象，qhs 为所有 http 处理线程分别创建了一个 serviceModulePool 对象、一个 sysThreadSpecificObjectPool 对象、一个 urlRewriter 对象、一个 userThreadSpecificObjectPool 对象。
系统对象功能只存在于付费的 qhs 实例。如果用户想测试 qhs 的系统对象，只有购买 qhs 实例才能获得系统对象功能。

用户(或开发人员)开发的对象分为用户全局对象、用户线程特定对象、用户连接特定对象。
系统对象、用户全局对象、用户线程特定对象、用户连接特定对象等对象一经创建，就长驻在 qhs，直到 qhs 终止运行，用户代码不能对这些对象进行动态加载、卸载。

11.1、调用系统对象

用户代码通过 qhs.interfaces.SysObjInvoker 接口类型对象来调用系统对象。在服务模块类的 invoke(qhs.interfaces.ThreadContext tc)方法的内部，用户代码可以调 tc.getServerStatus().getSysObjInvoker()来返回一个 SysObjInvoker 接口类型对象。关于 SysObjInvoker 接口类型的帮助文件见/home/http/qhs3.2/docs/api 目录。

/home/http/qhs3.2/demo/serviceModules/Test_urlPaths.java 演示了调用 urlPaths 对象，urlPath 对象是系统全局对象。

/home/http/qhs3.2/demo/serviceModules/Test_serviceModulePool.java 演示了调用 serviceModulePool 对象，serviceModulePool 对象是系统线程特定对象。

通过上述演示文件开发人员可了解如何调用系统对象。

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 4 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/qhs3.2";
export TEST_DIR="/home/http/qhsModules-04";
rm -Rvf $TEST_DIR;
mkdir -vp $TEST_DIR;

cp -v $QHS_HOME/demo/serviceModules/Test_serviceModulePool.java $TEST_DIR;
cp -v $QHS_HOME/demo/serviceModules/Test_urlPaths.java $TEST_DIR;
cp -v $QHS_HOME/demo/temp/ServiceModule.java $TEST_DIR;
$JDK_HOME/bin/javac -encoding utf-8 -d $TEST_DIR \
        -classpath $QHS_HOME/bin/qhs.jar \
        $TEST_DIR/Test_serviceModulePool.java \
        $TEST_DIR/Test_urlPaths.java \
        $TEST_DIR/ServiceModule.java;

$JDK_HOME/bin/jar cvf ~/temp-serivce-module.jar \
        -C $TEST_DIR temp/ServiceModule.class;

$JDK_HOME/bin/jar cvf $TEST_DIR/serviceModules.jar \
        -C $TEST_DIR serviceModules;
```

(2)、对/home/http/qhs3.2/config1/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/qhs3.2/sslFiles/httpServer.ks</KeyStore>
    <Password>123456</Password>
</SSLServerSocket>
.....
```

(3)、对/home/http/qhs3.2/config1/modules.xml 文件的内容设置为：

```
<Config>
    <Modules JarFile="/home/http/qhsModules-04/serviceModules.jar" >
        <Class Name="serviceModules.Test_urlPaths" />
        <Class Name="serviceModules.Test_serviceModulePool" />
    </Modules>
</Config>
```

(4)、运行/home/http/qhs3.2/startup1.sh 来启动 qhs(如果 qhs 正在运行,就运行/home/http/qhs3.2/shutdown1.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 检测到的服务模块类。

(5)、运行浏览器，在浏览器的地址栏输入以下 URL 对 qhs 发出 http 请求:

①、https://127.0.0.1:10000/test-urlPaths.id
②、https://127.0.0.1:10000/test-serviceModulePool.id
③、https://127.0.0.1:10000/test-serviceModulePool.id?load
④、https://127.0.0.1:10000/test-serviceModulePool.id?unload

输入上面的第③或第④个 URL 对 qhs 发出 http 请求后，可输入以下 URL 测试是否加载或卸载了 temp-service-module 服务模块：

https://127.0.0.1:10000/temp-service-module.id

11.2、开发用户全局对象、用户线程特定对象、用户连接特定对象

11.2.1、用户全局对象

用于创建用户全局对象的类称为用户全局对象类。用户全局对象（即用户全局对象类的实例）主要用于存放公用数据。在 qhs 的启动过程中，qhs 使用所有用户全局对象类分别创建一个用户全局对象，即一个用户全局对象类只用于创建一个用户全局对象。

在 qhs 运行的过程中，不同的 http 处理线程使用相同的对象 ID 作为参数调用 qhs.interfaces.ThreadContext.getGlobalObject(char[] id)方法而返回的用户全局对象是相同的。由于一个用户全局对象可同时被多个 http 处理线程引用、调用，因此，用户全局对象必须要线程安全。

一个用户全局对象类的定义：

(1)、一个公开的、返回值类型为 java.lang.String、无参数的 getGlobalObjectID()类方法。
getGlobalObjectID()方法用于返回对象 ID。qhs 使用本方法返回的 ID 来标识、检索用户全局对象。
(2)、一个公开的、无参数的构造器。
(3)、可选地定义有一个公开的、返回值类型为 void、无参数的 clear()实例方法，如：

```
public void clear() {
   //.....
}
```

在 qhs 终止运行的过程中，qhs 调用 clear()方法。

定义用户全局对象类的演示文件见/home/http/qhs3.2/demo/objects/AccessTimes.java，AccessTimes.java 定义了一个用于创建 ID 为“access-times”的用户全局对象（简称为 access-times 全局对象）的类。用于创建 get-access-times 服务模块的/home/http/qhs3.2/demo/serviceModules/ServiceModule2.java 引用、调用 access-times 全局对象。

11.2.2、用户线程特定对象

用于创建用户线程特定对象的类称为用户线程特定对象类。用户线程特定对象（即用户线程特定对象类的实例）主要用于存放对数据库服务器的连接、对内部网络主机的连接、cookie 加/解密器等。在 qhs 启动的过程中，qhs 尝试使用所有用户线程特定对象类创建多组用户线程特定对象，并且为所有 http 处理线程分别分配一组用户线程特定对象。一组用户线程特定对象只专用于一个 http 处理线程。

例如，qhs 在 objects.xml 检测出 test1.AAA、test2.BBB、test3.CCC 等 3 个用户线程特定对象类，而且有 5 个 http 处理线程（即 base.xml 内部的 ThreaCount 元素值为 5），则 qhs 创建 5 组用户线程特定对象，每组用户线程特定对象是 test1.AAA、test2.BBB、test3.CCC 等类的一个实例的集合，qhs 把一组用户线程特定对象(即一个 test1.AAA 类实例、一个 test2.BBB 类实例、一个 test3.CCC 类实例)分配给一个 http 处理线程。

在 qhs 运行的过程中，不同的 http 处理线程使用相同的对象 ID 作为参数调用 qhs.interfaces.ThreadContext.getThreadSpecificObject(char[] id)方法而返回的用户线程特定对象是不相同的。

一个用户线程特定对象类的定义：

(1)、一个公开的、返回值类型为 java.lang.String、无参数的 getThreadSpecificObjectID()类方法。
getThreadSpecificObjectID()方法用于返回对象 ID。qhs 使用本方法返回的 ID 来标识、检索用户线程特定对象。
(2)、一个公开的、无参数的构造器。
(3)、可选地定义有一个公开的、返回值类型为 java.lang.String、参数类型为 java.lang.String 的 update()实例方法，如：

```
public String update(String str) {
   //.....
}
```

update()实例方法用于更新用户线程特定对象的内部状态。用户代码通过调用定义在 userThreadSpecificObjectPool 系统线程特定对象上的 update()方法来跨线程地直接调用此处定义的 update()方法，因此，如果用户线程特定对象类定义了 update()方法，用户代码就必须对线程特定对象类进行同步处理。

(4)、可选地定义有一个公开的、返回值类型为 void、无参数的 clear()实例方法，如：

```
public void clear() {
   //.....
}
```

在 qhs 终止运行的过程中，qhs 调用 clear()方法。

定义了用户线程特定对象类的演示文件见/home/http/qhs3.2/demo/objects/RedisConnection.java，RedisConnection.java 定义了一个用户线程特定对象类，用于创建 ID 为“redis-connection”的用户线程特定对象（简称为 redis-connection 线程特定对象）。用于创建 login 服务模块的/home/http/qhs3.2/demo/serviceModules/ServiceModule3.java 引用、调用 redis-connection 线程特定对象。

11.2.3、用户连接特定对象

用于创建用户连接特定对象的类称为用户连接特定对象类。用户连接特定对象（即用户连接特定对象类的实例）主要用于存放与连接有关联的数据。qhs 启动的过程中，qhs 尝试使用所有用户连接特定对象类创建多组用户连接特定对象，并且为所有连接分别分配一组用户连接特定对象。一组用户连接特定对象只专用于一个连接。

例如，qhs 在 objects.xml 检测出 test4.DDD、test5.EEE、test6.FFF 等 3 个用户连接特定对象类，而且 qhs 允许创建 456 个连接（456 等于 base.xml 内部所有“ServerSocket>ConnectionMaxCount”、“SSLServerSocket>ConnectionMaxCount”元素值的总和），则 qhs 创建 456 组用户连接特定对象，每组用户连接特定对象是 test4.DDD、test5.EEE、test6.FFF 等类的一个实例的集合，qhs 把一组用户连接特定对象(即一个 test4.DDD 类实例、一个 test5.EEE 类实例、一个 test6.FFF 类实例)分配给一个连接。

在 qhs 运行的过程中，不同的 http 处理线程使用相同的对象 ID 作为参数调用 qhs.interfaces.ThreadContext.getConnection().getObject(char[] id)方法而返回的用户连接特定对象是不相同的。

一个用户连接特定对象类的定义：

(1)、一个公开的、返回值类型为 java.lang.String、无参数的 getConnectionSpecificObjectID()类方法。
getConnectionSpecificObjectID()方法用于返回对象 ID。qhs 使用本方法返回的 ID 来标识、检索用户连接特定对象。
(2)、一个公开的、无参数的构造器。
(3)、可选地定义有一个公开的、返回值类型为 void、无参数的 clear()实例方法，如：

```
public void clear() {
   //.....
}
```

clear()方法的主要作用是为清除用户连接特定对象的状态提供机会。clear()方法的调用有利于避免用户连接特定对象的状态遗留给下一个使用用户连接特定对象的连接。如果用户连接特定对象类定义有 clear()

方法，则 qhs 在关闭连接的过程中调用 clear()方法。

定义了用户连接特定对象类的演示文件见/home/http/qhs3.2/demo/objects/QueryRecorder.java，QueryRecorder.java 定义了一个用于创建 ID 为“query-recorder”的用户连接特定对象（简称为 query-recorder 连接特定对象）的类。用于创建 get-queries 服务模块的/home/http/qhs3.2/demo/serviceModules/ServiceModule4.java 引用、调用 query-recorder 连接特定对象。

编译并生成 jar 包时，开发人员必须把用户全局对象类、用户线程特定对象类、用户连接特定对象类与其它 qhs 模块（如：请求监听器、URL 重写器、服务模块等）隔离，不要把用户全局对象类、用户线程特定对象类、用户连接特定对象类与其它 qhs 模块混合编译生成 jar 包，否则有可能出现类名冲突，因为用户全局对象类、用户线程特定对象类、用户连接特定对象类处于最顶层的类加载器，而其它 qhs 模块处于子类加载器(child-classloader)。即用户全局对象类、用户线程特定对象类、用户连接特定对象类与其它 qhs 模块分别处于不同深度的类加载器。最适合的做法是把所有用户全局对象类、用户线程特定对象类、用户连接特定对象类等类编译生成为一个以上的独立 jar 包，在编译其它 qhs 模块时只需导入或引用这些独立的 jar 包即可，示例见第 16.1 节。

在启动 qhs 的过程中，qhs 参考 objects.xml 配置文件说明的用户全局对象类、用户线程特定对象类、用户连接特定对象类用于创建用户全局对象、用户线程特定对象、用户连接特定对象。

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 4 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/qhs3.2";
export TEST_DIR="/home/http/qhsModules-05";
rm -Rvf $TEST_DIR;
mkdir -vp $TEST_DIR;

cp -v $QHS_HOME/demo/objects/AccessTimes.java $TEST_DIR;
cp -v $QHS_HOME/demo/objects/RedisConnection.java $TEST_DIR;
cp -v $QHS_HOME/demo/objects/QueryRecorder.java $TEST_DIR;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule2.java $TEST_DIR;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule3.java $TEST_DIR;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule4.java $TEST_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $TEST_DIR \
        -classpath $QHS_HOME/bin/qhs.jar:$QHS_HOME/demo/jedis-2.1.0.jar \
        $TEST_DIR/AccessTimes.java \
        $TEST_DIR/RedisConnection.java \
        $TEST_DIR/QueryRecorder.java;

$JDK_HOME/bin/jar cvf $TEST_DIR/objects.jar \
        -C $TEST_DIR objects;

$JDK_HOME/bin/javac -encoding utf-8 -d $TEST_DIR \
        -classpath $QHS_HOME/bin/qhs.jar:$TEST_DIR/objects.jar \
        $TEST_DIR/ServiceModule2.java \
        $TEST_DIR/ServiceModule3.java \
        $TEST_DIR/ServiceModule4.java;

$JDK_HOME/bin/jar cvf $TEST_DIR/serviceModules.jar \
        -C $TEST_DIR serviceModules;
```

(2)、对/home/http/qhs3.2/config1/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/qhs3.2/sslFiles/httpServer.ks</KeyStore>
    <Password>123456</Password>
</SSLServerSocket>
.....
```

(3)、对/home/http/qhs3.2/config1/objects.xml 文件的内容设置为：

```
<Config>
    <Objects JarFile="/home/http/qhsModules-05/objects.jar" Enabled="true" >
        <Class Name="objects.AccessTimes" Created="true" />
        <Class Name="objects.RedisConnection" Created="true" />
        <Class Name="objects.QueryRecorder" Created="true" />
    </Objects>
</Config>
```

(4)、对/home/http/qhs3.2/config1/modules.xml 文件的内容设置为：

```
<Config>
    <Modules JarFile="/home/http/qhsModules-05/serviceModules.jar" Enabled="true" >
        <Class Name="serviceModules.ServiceModule2" Enabled="true" />
        <Class Name="serviceModules.ServiceModule3" Enabled="true" />
        <Class Name="serviceModules.ServiceModule4" Enabled="true" />
    </Modules>
</Config>
```

(5)、对/home/http/qhs3.2/config1/options.xml 文件的内容设置为：

```
<Config>
    <!--RedisConnection 类引用 jedis-2.1.0.jar 包内的类。 -->
    <JarFile>/home/http/qhs3.2/demo/jedis-2.1.0.jar</JarFile>
</Config>
```

(6)、运行无参数的 redis-server 命令（即启动 Redis 服务器）：

```
redis-server &
```

(7)、运行 redis-cli 命令（即运行 Redis 客户端）设置 2 个键值：

```
redis-cli set abc def;
redis-cli set temp-user-name temp-password;
```

(8)、运行/home/http/qhs3.2/startup1.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.2/shutdown1.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 检测到的服务模块类、用户全局对象类、用户线程特定对象类、用户连接特定对象类。

(9)、运行浏览器，在浏览器的地址栏输入以下地址对 qhs 发出 http 请求：

①、https://127.0.0.1:10000/get-access-times.id
②、https://127.0.0.1:10000/login.id?abc&def
③、https://127.0.0.1:10000/login.id?Xabc&def
④、https://127.0.0.1:10000/login.id?temp-user-name&temp-password
⑤、https://127.0.0.1:10000/login.id?temp-user-name&Xtemp-password
⑥、https://127.0.0.1:10000/get-queries.id

12、开发 URL 重写器

URL 重写器（URL Rewriter）既用于重写 URL 又用于确定重写后的 URL 的用途。一个 URL 重写器是一个线

程特定对象。用户代码可以调用定义在 sysThreadSpecificObjectPool 系统对象上的 createURLRewriter()、removeURLRewriter()方法来动态创建、移除 URL 重写器，可以调用定义在 urlRewriter 系统对象上的 update()方法来动态更新 URL 重写器的内部状态。在 qhs 启动的过程中，qhs 参考 base.xml 配置文件内部的 URLRewriter 元素为所有 http 处理线程分别创建一个 URL 重写器。

一个用于创建 URL 重写器的类必须定义有以下方法：

(1)、一个公开的、无参数的构造器。

(2)、public qhs.sys.RewriteResult rewrite(qhs.interfaces.ThreadContext tc, qhs.interfaces.URIParser uriParser){
    //.....
}

跟据 tc.isPushing()的返回值可以判断存放在 uriParser 参数的 URL 的来源：

①、tc.isPushing()的返回值等于 false，表示存放在 uriParser 参数的 URL 来自浏览器发送给 qhs 的 http 请求中的请求 URI(Request-URI)。从 rewrite()方法返回以后，存放在 uriParser 参数的 URL 用于生成 http 答复。

②、tc.isPushing()的返回值等于 true，表示存放在 uriParser 参数的 URL 来自 urlPaths.xml 配置文件的 Push 元素说明的推送 URL 或者来自调用 tc.push()的 url 参数。从 rewrite()方法返回以后，存放在 uriParser 参数的 URL 用于生成推送答复。

要重写 URL(即更改存放在 uriParser 参数的 URL)，只需调用定义在 uriParser 参数上的各方法(如：uriParser.setPath()、uriParser.setQuery()等等)，最终重写后的 URL 仍然要缓存在 uriParser 参数。

从 rewrite()方法返回以后，qhs 跟据 rewrite()的返回值，把存放在 uriParser 参数的 URL 用于以下用途之一：

①、构建并且发送表达临时重定向的答复给客户端。

②、构建并且发送表达永久重定向的答复给客户端。

③、构建资源文件路径(可能经过 urlPaths.xml 配置文件追加缺省页面文件名)，并且在文档根目录的内部检索、发送文件给客户端。

④、构建服务模块 ID(可能经过 urlPaths.xml 配置文件追加缺省页面文件名)，并且调用服务模块生成、发送答复给客户端。

如果重写后的 URL 用于构建资源文件路径在文档根目录内部检索文件或者用于构建服务模块 ID，qhs 为了提高 URL 的安全性，限制了构成 URL 路径的字符集。使用已转义(escaped)的 URL 路径作为参数调用 uriParser.setPath()时，已转义的 URL 路径应由以下字符构成：'/'、'a'至'z'、'A'至'Z'、'0'至'9'、'-'、'_'、'.'、空格的已转义形式、解码为汉字的 utf-8 字节组的已转义形式等等，否则，qhs 发送表达错误的答复给客户端。同样，使用未转义(unescaped)的 URL 路径作为参数调用 uriParser.setPath()时，未转义的 URL 路径应由以下字符构成：'/'、'a'至'z'、'A'至'Z'、'0'至'9'、'-'、'_'、'.'、空格、解码为汉字的 utf-8 字节组等等，否则，qhs 发送表达错误的答复给客户端。

(3)、public String update(String str) {
    //.....
}

本方法用于更新 URL 重写器的内部状态。

/home/http/qhs3.2/demo/urlRewriter/URLRewriter.java 文件定义了一个最基本的、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。

base.xml 配置文件的 URLRewriter 元素说明用于创建 URL 重写器的类。

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 4 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/qhs3.2";
export TEST_DIR="/home/http/qhsModules-06";
rm -Rvf $TEST_DIR;
mkdir -vp $TEST_DIR;

cp -v $QHS_HOME/demo/urlRewriter/URLRewriter.java $TEST_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $TEST_DIR \
        -classpath $QHS_HOME/bin/qhs.jar \
        $TEST_DIR/URLRewriter.java;


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

(2)、对/home/http/qhs3.2/config1/base.xml 文件内部的一个 SSLServerSocket 元素、URLRewriter 元素更改为:

```
......
<SSLServerSocket>
    <Created>true</Created>
    <Address>127.0.0.1</Address>
    <Port>10000</Port>
    <KeyStore>/home/http/qhs3.2/sslFiles/httpServer.ks</KeyStore>
    <Password>123456</Password>
</SSLServerSocket>
......
<URLRewriter>
    <Created>true</Created>
    <JarFile>/home/http/qhsModules-06/urlRewriter.jar</JarFile>
    <Class>urlRewriter.URLRewriter</Class>
</URLRewriter>
......
```

(3)、运行/home/http/qhs3.2/startup1.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.2/shutdown1.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 URL 重写器的 urlRewriter.URLRewriter 类。

13、开发调试输出器

调试输出器（Debug Outputer）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，并为开发服务模块、请求监听器、URL 重写器等 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。在 qhs 启动的过程中，qhs 尝试创建一个全局的调试输出器，这调试输出器被所有 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。

一个用于创建调试输出器的类必须定义有以下方法:

(1)、一个公开的、无参数的构造器。

(2)、public void initialize() {

```
 //.....
}
```

initialize()方法为初始化调试输出器提供机会。qhs 在启动的过程中，调用 initialize()方法。

(3)、public void output(String s) {

```
  //.....
}
```

output()方法用于输出调试信息。s 参数值为调试信息串。

(4)、public void outputRequest(long threadID, String json) {

```
  //.....
}
```

outputRequest()方法用于输出 http 请求。threadID 参数值为 http 处理线程 id。json 参数值为表达 http 请求的 json 编码。

(5)、public void outputResponse(long threadID, String json) {

```
  //.....
}
```

outputResponse()方法用于输出 http 答复或推送答复。threadID 参数值为 http 处理线程 id。json 参数值为表达 http 答复或推送答复的 json 编码。

(6)、public void outputRequestDynamicTableState(long threadID, String json) {

```
  //.....
}
```

outputRequestDynamicTableState()方法用于输出请求动态表的状态。threadID 参数值为 http 处理线程 id。json 参数值为表达请求动态表的 json 编码。

(7)、public void outputResponseDynamicTableState(long threadID, String json) {

```
  //.....
}
```

outputResponseDynamicTableState()方法用于输出请求动态表的状态。threadID 参数值为 http 处理线程 id。json 参数值为表达答复动态表的 json 编码。

(8)、public void outputReceivedFrame(long threadID, String json) {

```
  //.....
}
```

outputReceivedFrame()方法用于输出接收到的数据帧。threadID 参数值为 http 处理线程 id。json 参数值为表达数据帧的 json 编码。

(9)、public void outputSendingFrame(long threadID, String json) {

```
  //.....
}
```

outputSendingFrame()方法用于输出正在发送的数据帧。threadID 参数值为 http 处理线程 id。json 参数值为表达数据帧的 json 编码。

(10)、public void finish() {

```
  //.....
  }
```

finish()方法为清除调试输出器状态提供机会。 qhs 在终止运行的过程中，调用 finish()方法。

/home/http/qhs3.2/demo/debugOutputer/DebugOutputer.java 文件定义了一个最基本的、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。/home/http/qhs3.2/demo/debugOutputerGUI/DebugOutputer.java 文件也定义了一个用于创建调试输出器的 DebugOutputer 类。

base.xml 配置文件的 DebugOutputer 元素说明用于创建调试输出器的类。

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 4 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/qhs3.2";
export TEST_DIR="/home/http/qhsModules-07";
rm -Rvf $TEST_DIR;
mkdir -vp $TEST_DIR;

cp -v $QHS_HOME/demo/debugOutputerGUI/DebugOutputer.java $TEST_DIR;
cp -v $QHS_HOME/demo/debugOutputerGUI/DebugDialog.java $TEST_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $TEST_DIR \
        -classpath $QHS_HOME/bin/qhs.jar \
        $TEST_DIR/DebugOutputer.java $TEST_DIR/DebugDialog.java;
$JDK_HOME/bin/jar cvf $TEST_DIR/debugOutputer.jar \
        -C $TEST_DIR debugOutputerGUI;
```

(2)、对/home/http/qhs3.2/config1/base.xml 文件内部的一个 SSLServerSocket 元素、DebugOutputer 元素更改为：

```
......
<SSLServerSocket>
    <Created>true</Created>
    <Address>127.0.0.1</Address>
    <Port>10000</Port>
    <KeyStore>/home/http/qhs3.2/sslFiles/httpServer.ks</KeyStore>
    <Password>123456</Password>
</SSLServerSocket>
......
<DebugOutputer>
    <Created>true</Created>
    <JarFile>/home/http/qhsModules-07/debugOutputer.jar</JarFile>
    <Class>debugOutputerGUI.DebugOutputer</Class>
</DebugOutputer>
......
```

(3)、运行/home/http/qhs3.2/startup1.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.2/shutdown1.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，用户可以在浏览器输入 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（Error Response Creater）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线程特定对象。用户代码可以调用定义在 sysThreadSpecificObjectPool 系统对象上的 createErrorResponseCreater()、removeErrorResponseCreater()方法来动态创建、移除错误答复创建器，可以调用定义在 errorResponseCreater 系统对象上的 update()方法来动态更新错误答复创建器的内部状

态。在 qhs 启动的过程中，qhs 参考 base.xml 配置文件内部的 ErrorResponseCreater 元素为所有 http 处理线程分别创建一个错误答复创建器。

一个用于创建错误答复创建器的类必须定义有以下方法:

(1)、一个公开的、无参数的构造器。
(2)、public void invoke(qhs.interfaces.ThreadContext tc, qhs.interfaces.ErrorReason er) {
//.....
}
(3)、public String update(String str) {
//.....
}
update()实例方法用于更新错误答复创建器的内部状态。

/home/http/qhs3.2/demo/errorResponseCreater/ErrorResponseCreater.java 文件定义了一个用于创建错误答复创建器的 ErrorResponseCreater 类,开发人员可以对这 ErrorResponseCreater 类进行更改、增加代码。
base.xml 配置文件的 ErrorResponseCreater 元素说明用于创建错误答复创建器的类。

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 4 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/qhs3.2";
export TEST_DIR="/home/http/qhsModules-08";
rm -Rvf $TEST_DIR;
mkdir -vp $TEST_DIR;

cp -v $QHS_HOME/demo/errorResponseCreater/ErrorResponseCreater.java $TEST_DIR;
$JDK_HOME/bin/javac -encoding utf-8 -d $TEST_DIR \
        -classpath $QHS_HOME/bin/qhs.jar \
        $TEST_DIR/ErrorResponseCreater.java;
$JDK_HOME/bin/jar cvf $TEST_DIR/errorResponseCreater.jar \
        -C $TEST_DIR errorResponseCreater;
```

(2)、对/home/http/qhs3.2/config1/base.xml 文件内部的一个 SSLServerSocket 元素、ErrorResponseCreater 元素更改为:

```
.....
<SSLServerSocket>
    <Created>true</Created>
    <Address>127.0.0.1</Address>
    <Port>10000</Port>
    <KeyStore>/home/http/qhs3.2/sslFiles/httpServer.ks</KeyStore>
    <Password>123456</Password>
</SSLServerSocket>
.....
<ErrorResponseCreater>
    <Created>true</Created>
    <JarFile>/home/http/qhsModules-08/errorResponseCreater.jar</JarFile>
    <Class>errorResponseCreater.ErrorResponseCreater</Class>
</ErrorResponseCreater>
.....
```

(3)、运行/home/http/qhs3.2/startup1.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.2/shutdown1.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 errorResponseCreater.ErrorResponseCreater 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（Form Data）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（control name）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（control value）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（initial value）、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>
```

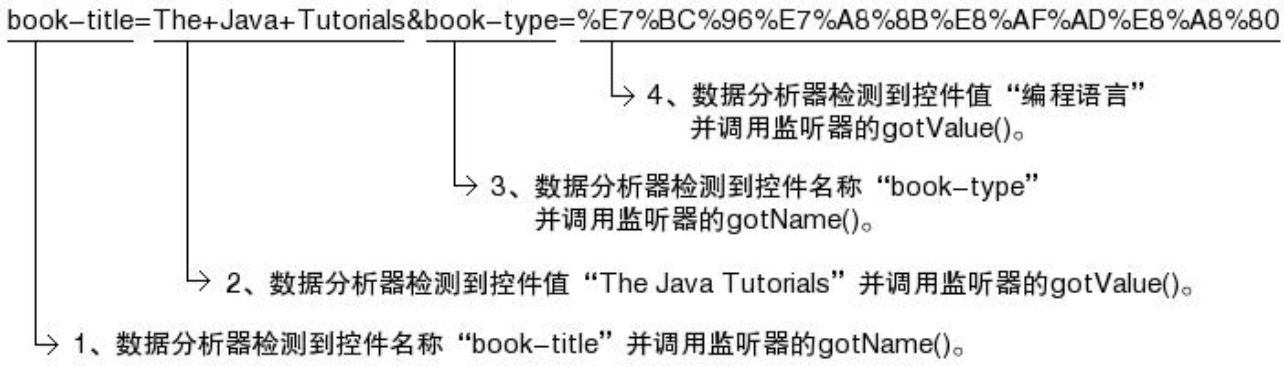
上面的 html 页面数据中，input 元素的 name 属性说明了控件名称“book-title”，select 元素的 name 属性说明了控件名称“book-type”，名称为“book-title”的控件的初始值为“The Java Tutorials”，名称为“book-type”的控件的初始值为“编程语言”。

15.1、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）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 gotName()方法，如果数据分析器检测到控件值，就调用监听器的 gotValue()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, char[])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, byte[], char[])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 listener 参数值用作为监听器，并调用定义在监听器上的各方法。

/home/http/qhs3.2/demo/serviceModules/ServiceModule51.java、 /home/http/qhs3.2/demo/serviceModules/ServiceModule52.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，ServiceModule51.java 实现的 get-form-page-u 服务模块用于生成包含有 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），ServiceModule52.java 实现的 list-names-and-values-u 服务模块用于分析表单数据并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 4 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/qhs3.2";
export TEST_DIR="/home/http/qhsModules-09";
rm -Rvf $TEST_DIR;
mkdir -vp $TEST_DIR;

cp -v $QHS_HOME/demo/serviceModules/ServiceModule51.java $TEST_DIR;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule52.java $TEST_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $TEST_DIR \
        -classpath $QHS_HOME/bin/qhs.jar \
        $TEST_DIR/ServiceModule51.java \
        $TEST_DIR/ServiceModule52.java;

$JDK_HOME/bin/jar cvf $TEST_DIR/serviceModules.jar \
        -C $TEST_DIR serviceModules;
```

(2)、对/home/http/qhs3.2/config1/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/qhs3.2/sslFiles/httpServer.ks</KeyStore>
```

```
    <Password>123456</Password>
</SSLServerSocket>
.....
```

(3)、对/home/http/qhs3.2/config1/modules.xml 文件的内容设置为:

```
<Config>
    <Modules JarFile="/home/http/qhsModules-09/serviceModules.jar" Enabled="true" >
        <Class Name="serviceModules.ServiceModule51" Enabled="true" />
        <Class Name="serviceModules.ServiceModule52" Enabled="true" />
    </Modules>
</Config>
```

(4)、运行/home/http/qhs3.2/startup1.sh 来启动 qhs(如果 qhs 正在运行,就运行/home/http/qhs3.2/shutdown1.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。

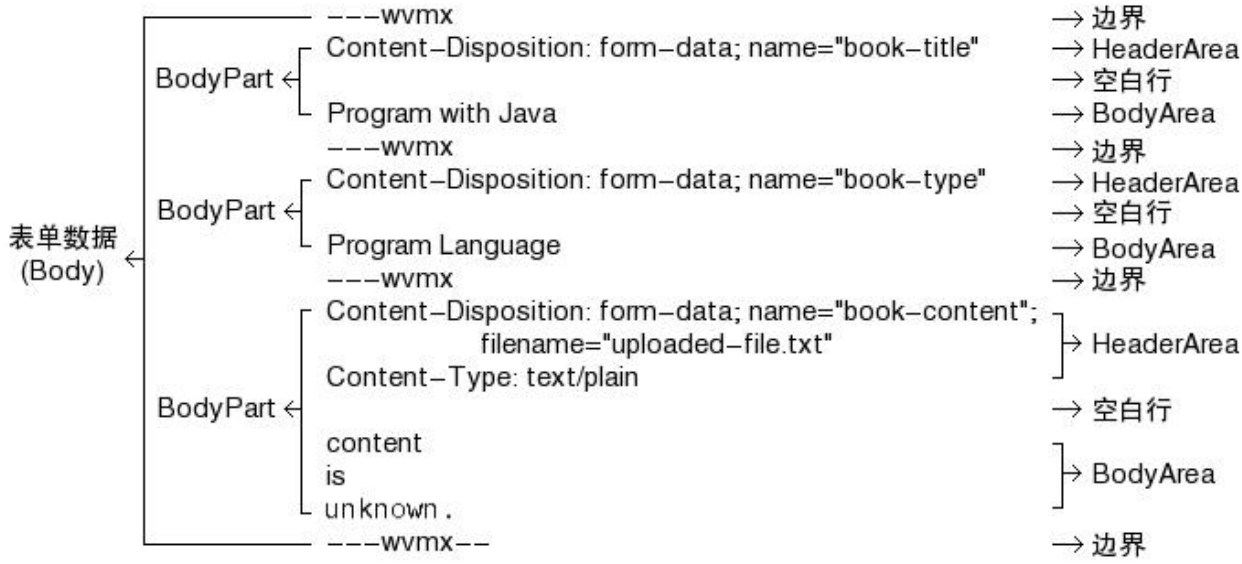
④、在浏览器新显示的页面上显示有控件名称、控件值。

15.2、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(MFormDataListener, byte[])方法来分析表单数据。parsePayloadBody(MFormDataListener, byte[])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（blank line)、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“content is unknown.”（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/qhs3.2/demo/serviceModules/ServiceModule61.java、/home/http/qhs3.2/demo/serviceModules/ServiceModule62.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，ServiceModule61.java 实现的 get-form-page-m 服务模块用于生成包含有 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)，ServiceModule62.java 实现的 list-names-and-values-m 服务模块用于分析表单数据并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 4 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/qhs3.2";
export TEST_DIR="/home/http/qhsModules-10";
rm -Rvf $TEST_DIR;
mkdir -vp $TEST_DIR;

cp -v $QHS_HOME/demo/serviceModules/ServiceModule61.java $TEST_DIR;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule62.java $TEST_DIR;

$JDK_HOME/bin/javac -encoding utf-8 -d $TEST_DIR \
        -classpath $QHS_HOME/bin/qhs.jar \
        $TEST_DIR/ServiceModule61.java \
        $TEST_DIR/ServiceModule62.java;
```

```
$JDK_HOME/bin/jar cvf $TEST_DIR/serviceModules.jar \
        -C $TEST_DIR serviceModules;
```

(2)、对/home/http/qhs3.2/config1/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/qhs3.2/sslFiles/httpServer.ks</KeyStore>
    <Password>123456</Password>
</SSLServerSocket>
.....
<HTTPRequest>
    <HeaderBlockBufferSize>20k</HeaderBlockBufferSize>
    <PayloadBodyBufferSize>7m</PayloadBodyBufferSize>
</HTTPRequest>
.....
```

(3)、对/home/http/qhs3.2/config1/modules.xml 文件的内容设置为:

```
<Config>
    <Modules JarFile="/home/http/qhsModules-10/serviceModules.jar" Enabled="true" >
        <Class Name="serviceModules.ServiceModule61" Enabled="true" />
        <Class Name="serviceModules.ServiceModule62" Enabled="true" />
    </Modules>
</Config>
```

(4)、运行/home/http/qhs3.2/startup1.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.2/shutdown1.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、选择尺寸小于 1MB 的文件后，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 NetBeansIDE11.3(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。

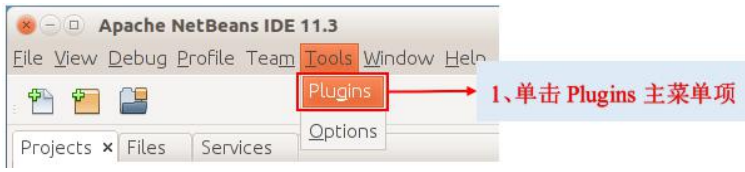
16.1、本机调试

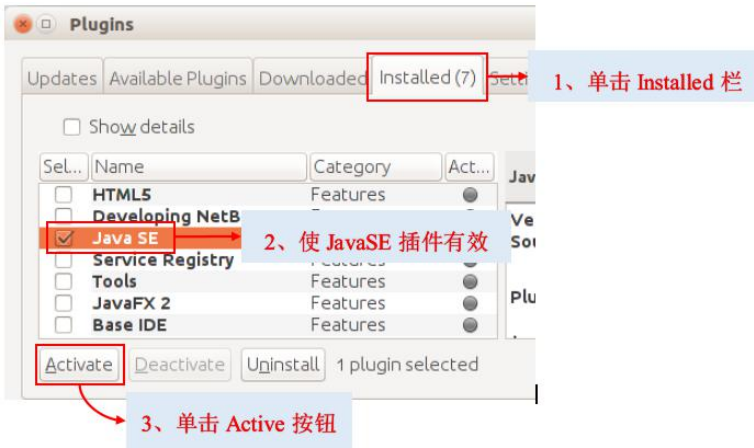
本机调试要求 NetBeansIDE 与 qhs 运行在同一计算机主机上。
使用本调试方法时，必须对所有 NetBeansIDE 项目执行以下设置:

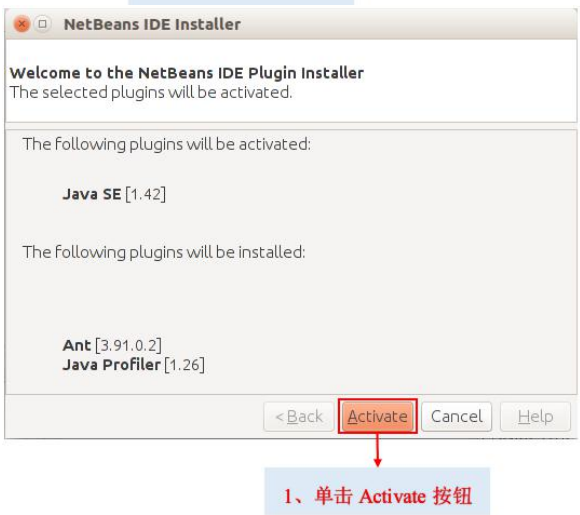
(1)、导入 qhs.jar、gson.jar 包。
(2)、设置运行项目的主类为 qhs.Start，qhs.Start 类是 qhs 运行的起始，处于 qhs.jar 包。
(3)、设置--ServerRoot 运行参数，参数的格式为“--ServerRoot=qhs 的安装目录”。
(4)、设置--Config 运行参数，参数的格式为“--Config=配置目录”。
(5)、导入其它必需的 jar 包（如：包含 JDBC 驱动程序的 jar 包等等）。

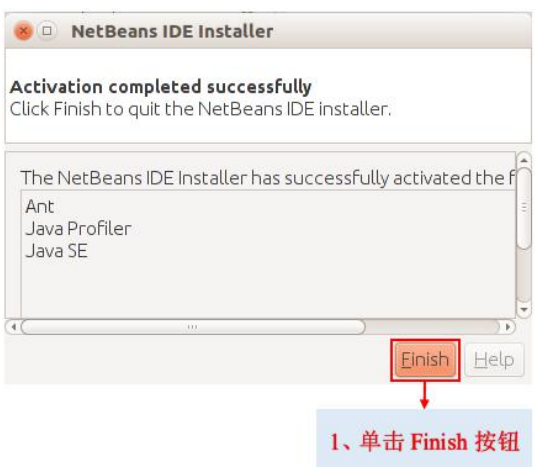
下面的步骤、截图演示了项目的创建、设置、调试等等，示例的运行环境见第 4 节：

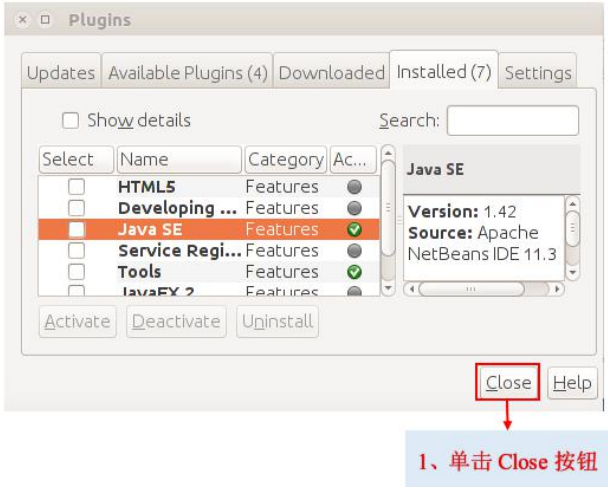
(1)、启动 NetBeansIDE。
(2)、激活"JavaSE"插件。本步骤是可选的，如果已激活"JavaSE"插件，就跳过本步骤。



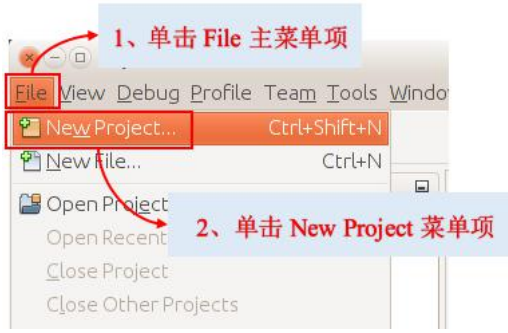




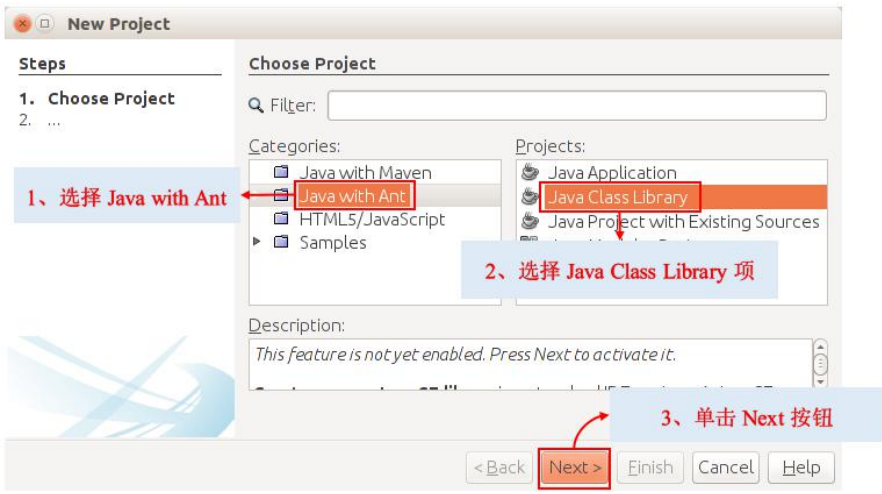




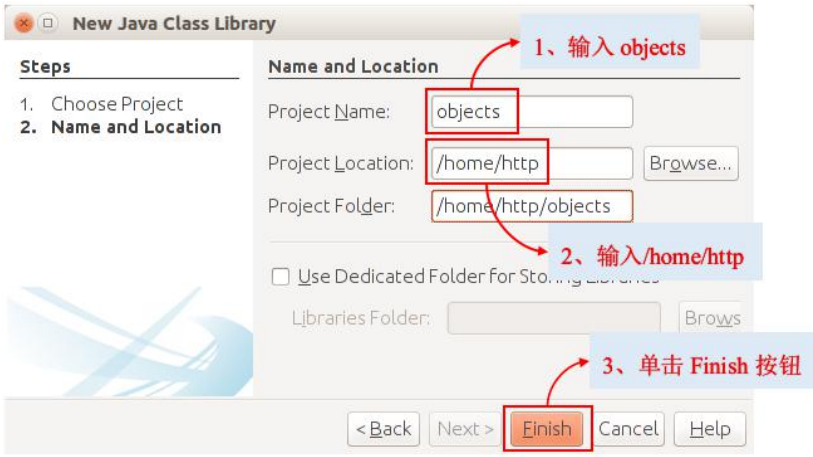
(3)、在 NetBeansIDE 创建 objects 项目。objects 项目集中存放全局对象类、线程特定对象类、连接特定对象类。objects 项目生成的/home/http/objects/dist/objects.jar 被本节创建的其它项目导入。



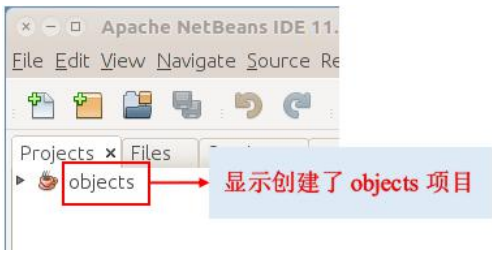
(4)、



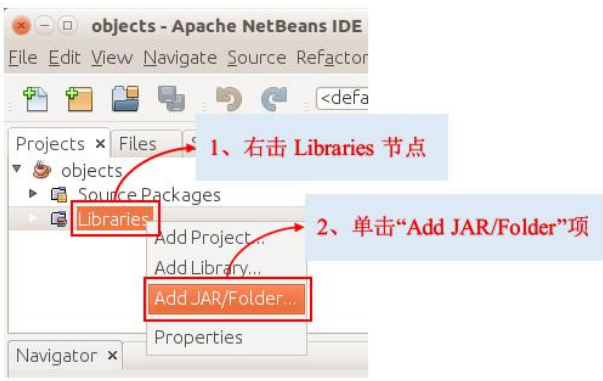
(5)、



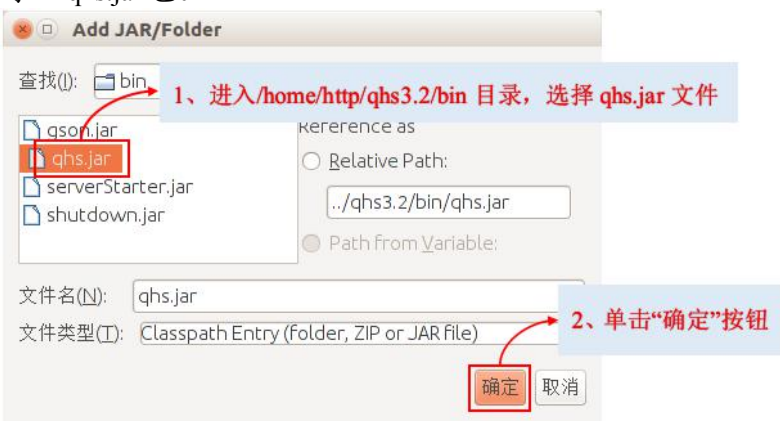
(6)、



(7)、

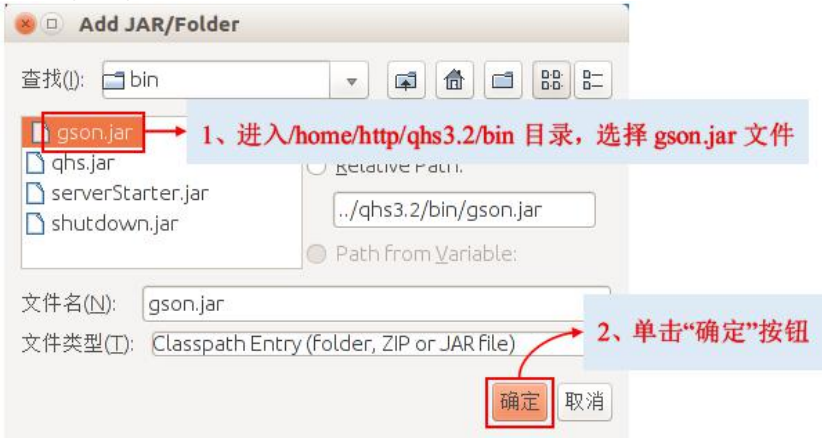


(8)、导入 qhs.jar 包。



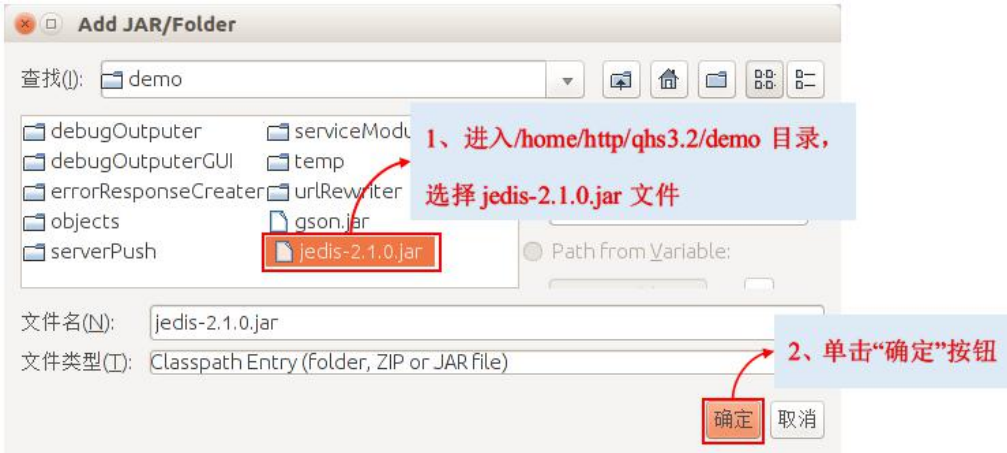
(9)、重复第(7)步骤。

(10)、导入 gson.jar 包。



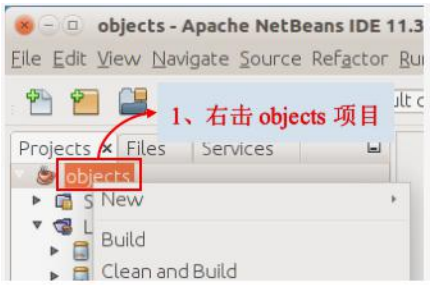
(11)、重复第(7)步骤。

(12)、导入 jedis-2.1.0.jar，objects 项目内部的 RedisConnection.java 引用 jedis-2.1.0.jar 包内部的类。

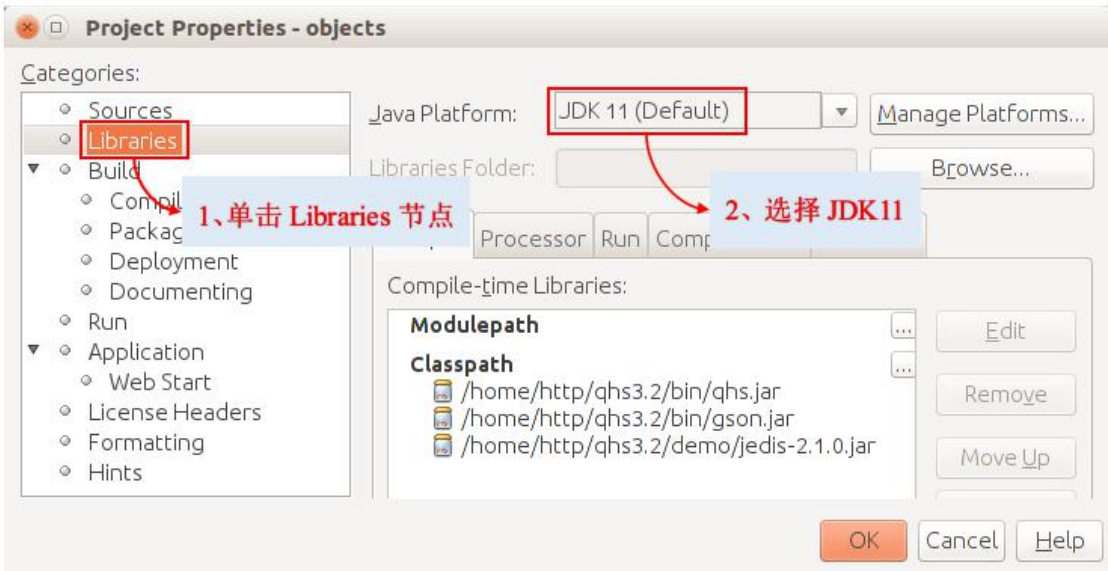


(13)、

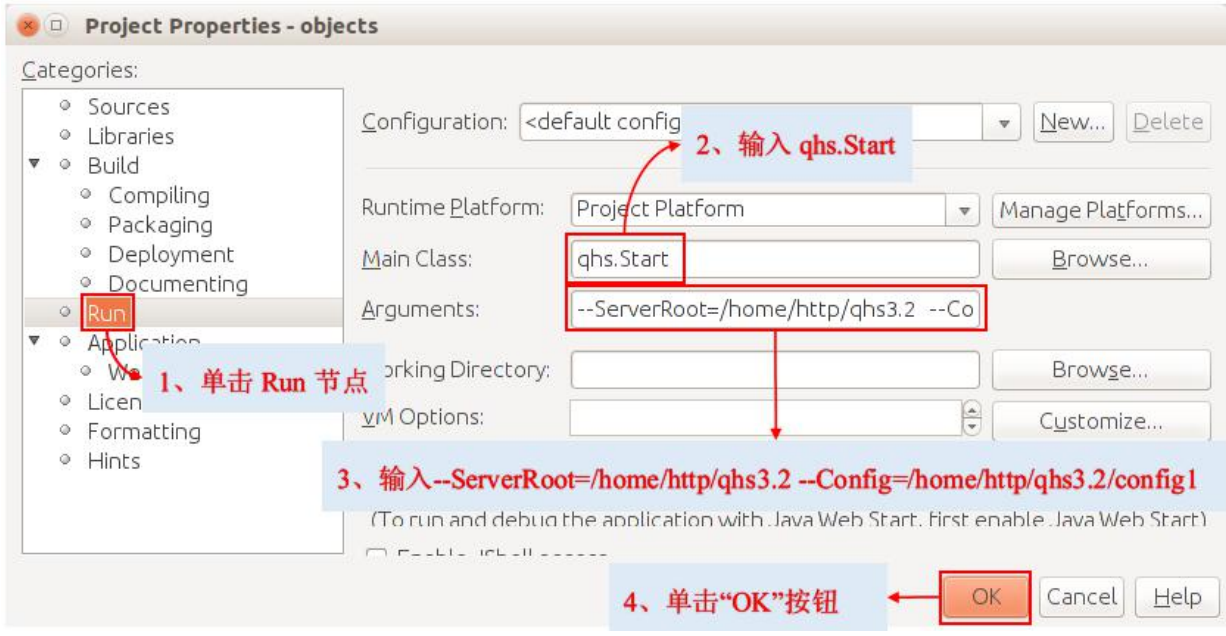
(14)、设置运行项目的主类、参数。



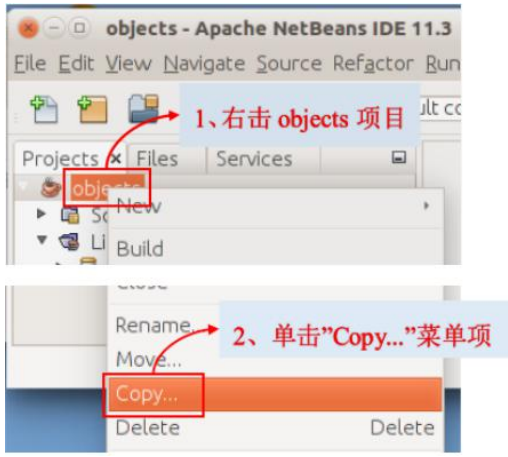
(15)、



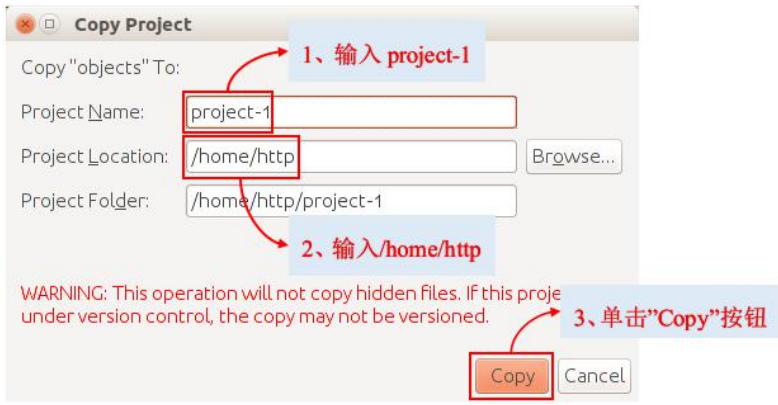
(16)、



(17)、

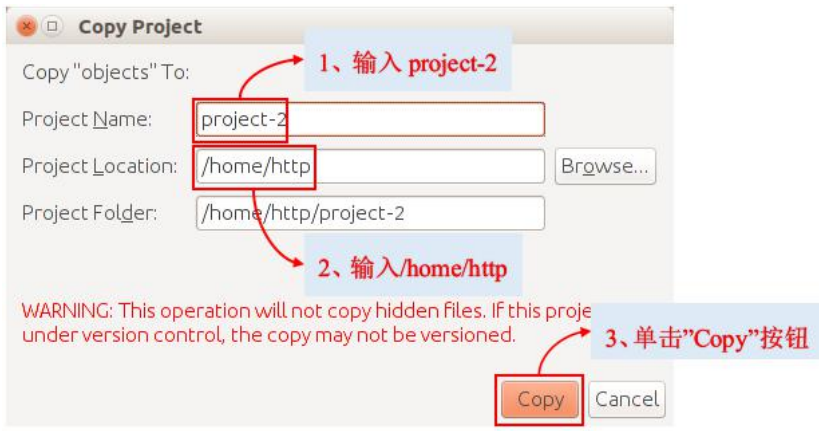


(18)、



(19)、重复第(17)步骤。

(20)、

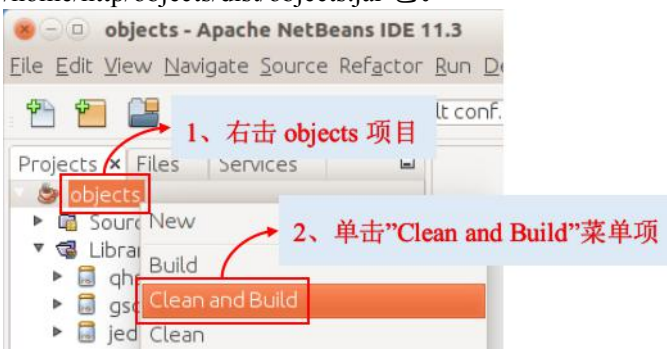


(21)、通过复制来创建了 project-1、project-2 项目，project-1、project-2 项目所包含的各种设置（包括导入的 jar 包、运行主类、运行参数等）与 objects 项目相同。

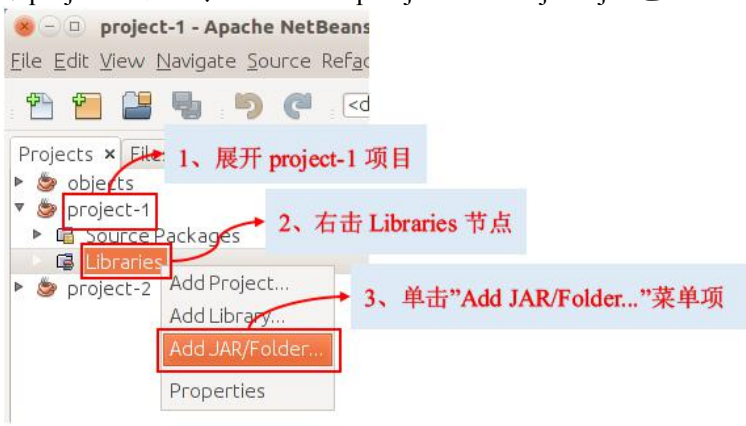
(22)、复制以下命令到终端并且执行命令，命令的结果是复制演示文件到 objects 项目：

```
export QHS_HOME="/home/http/qhs3.2";
export DST_DIR_1="/home/http/objects/src";
rm -Rvf $DST_DIR_1/*;
cp -Rvf $QHS_HOME/demo/objects $DST_DIR_1;
```

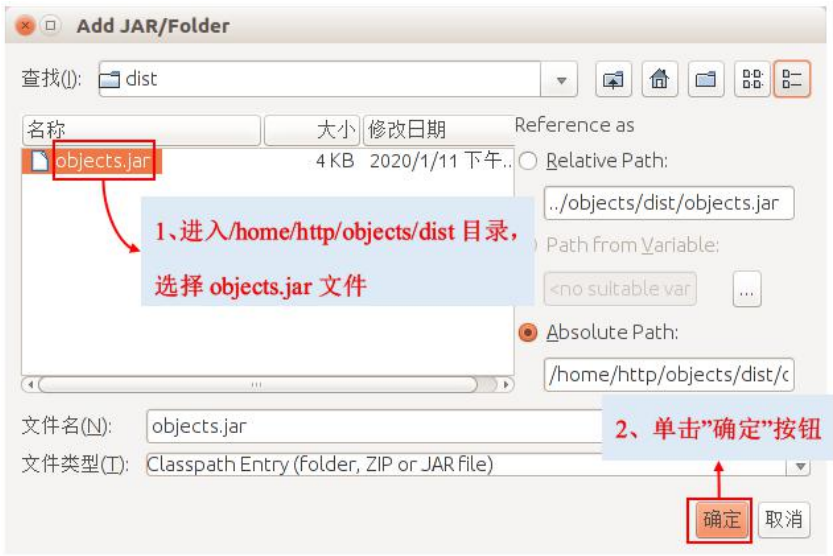
(23)、编译、生成/home/http/objects/dist/objects.jar 包，project-1、project-2 项目要导入 /home/http/objects/dist/objects.jar 包。



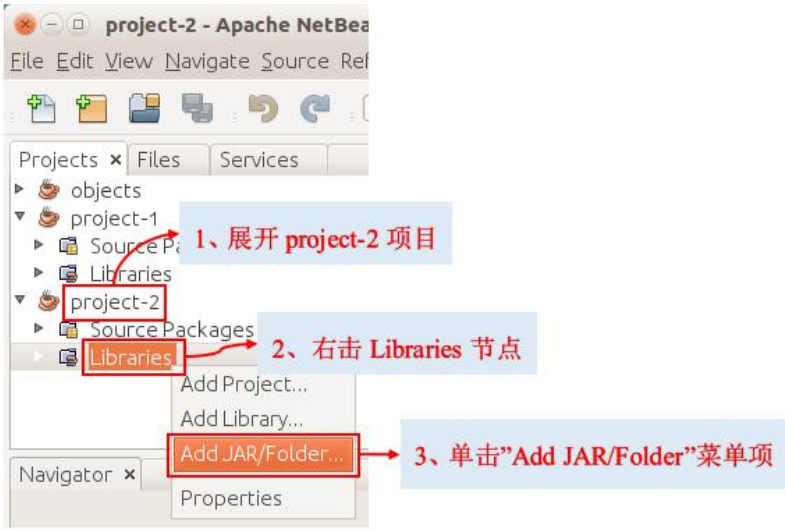
(24)、对 project-1 项目导入/home/http/objects/dist/objects.jar 包。



(25)、



(26)、对 project-2 项目导入/home/http/objects/dist/objects.jar 包。



(27)、重复第(25)步骤。

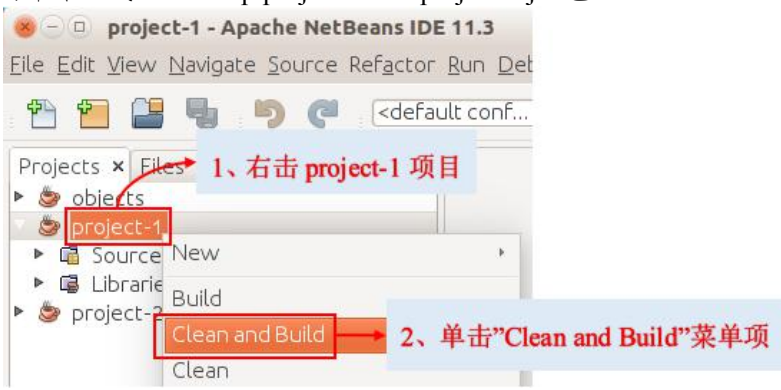
(28)、现时，objects 项目导入了 qhs.jar、gson.jar、jedis-2.1.0.jar 等包，project-1、project-2 项目导入了 qhs.jar、gson.jar、jedis-2.1.0.jar、objects.jar 等包。用户应使用 project-1 或者 project-2 项目启动 qhs 并进入调试状态，因为这两项目导入的 jar 包比 objects 项目更全面（增加导入了 objects.jar 包）。

(29)、复制以下命令到终端并且执行命令，命令的结果是复制演示文件到 project-1、project-2 项目：

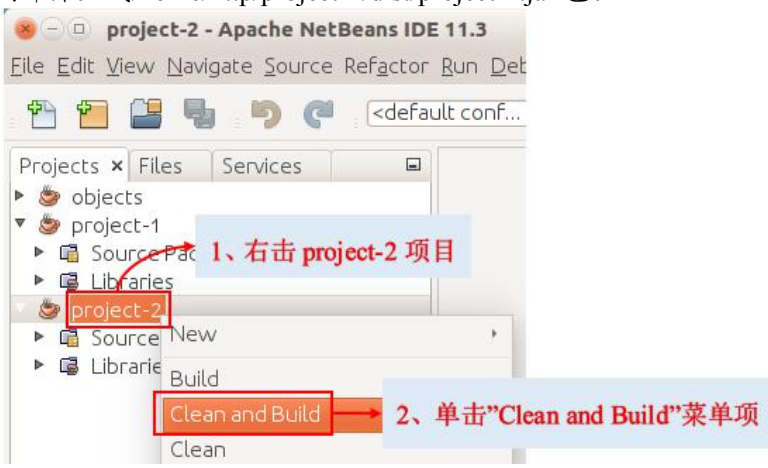
```
cd ~;
export QHS_HOME="/home/http/qhs3.2";
export DST_DIR_2="/home/http/project-1/src";
rm -Rvf $DST_DIR_2/*;
mkdir -vp $DST_DIR_2/serviceModules;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule2.java \
        $DST_DIR_2/serviceModules;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule3.java \
        $DST_DIR_2/serviceModules;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule4.java \
        $DST_DIR_2/serviceModules;
cp -v $QHS_HOME/demo/serviceModules/ServerPush.java \
        $DST_DIR_2/serviceModules;
cp -v $QHS_HOME/demo/serviceModules/TimeShower.java \
        $DST_DIR_2/serviceModules;

export DST_DIR_3="/home/http/project-2/src";
rm -Rvf $DST_DIR_3/*;
mkdir -vp $DST_DIR_3/serviceModules;
cp -Rvf $QHS_HOME/demo/debugOutputerGUI    $DST_DIR_3;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule1.java \
        $DST_DIR_3/serviceModules;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule51.java \
        $DST_DIR_3/serviceModules;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule52.java \
        $DST_DIR_3/serviceModules;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule61.java \
        $DST_DIR_3/serviceModules;
cp -v $QHS_HOME/demo/serviceModules/ServiceModule62.java \
        $DST_DIR_3/serviceModules;
cp -v $QHS_HOME/demo/serviceModules/Test_serviceModulePool.java \
        $DST_DIR_3/serviceModules;
cp -v $QHS_HOME/demo/serviceModules/Test_urlPaths.java \
        $DST_DIR_3/serviceModules;

cp -Rvf $QHS_HOME/demo/errorResponseCreater $DST_DIR_3;
```

cp -Rvf $QHS_HOME/demo/urlRewriter $DST_DIR_3;

(30)、编译并生成/home/http/project-1/dist/project-1.jar 包。



(31)、编译并生成/home/http/project-2/dist/project-2.jar 包。



(32)、编辑/home/http/qhs3.2/config1/base.xml 配置文件，对 DocumentRoot、SSLServerSocket、HTTPRequest、ErrorResponseCreater、URLRewriter、DebugOutputer 等元素设置为：

```
.....
<DocumentRoot>/home/http/qhs3.2/demo/serverPush/documentRoot</DocumentRoot>
.....
<SSLServerSocket>
      <Created>true</Created>
      <Address>127.0.0.1</Address>
      <Port>10000</Port>
      <KeyStore>/home/http/qhs3.2/sslFiles/httpServer.ks</KeyStore>
      <Password>123456</Password>
</SSLServerSocket>
.....
<HTTPRequest>
      <HeaderBlockBufferSize>20k</HeaderBlockBufferSize>
      <PayloadBodyBufferSize>7m</PayloadBodyBufferSize>
</HTTPRequest>
.....
<ErrorResponseCreater>
      <Created>true</Created>
      <JarFile>/home/http/project-2/dist/project-2.jar</JarFile>
      <Class>errorResponseCreater.ErrorResponseCreater</Class>
</ErrorResponseCreater>
.....
<URLRewriter>
    <Created>true</Created>
    <JarFile>/home/http/project-2/dist/project-2.jar</JarFile>
    <Class>urlRewriter.URLRewriter</Class>
```

```
</URLRewriter>
......
<DebugOutputer>
    <Created>true</Created>
    <JarFile>/home/http/project-2/dist/project-2.jar</JarFile>
    <Class>debugOutputerGUI.DebugOutputer</Class>
</DebugOutputer>
......
```

(33)、对/home/http/qhs3.2/config1/modules.xml 配置文件的内容设置为：

```
<Config>
    <Modules JarFile="/home/http/project-1/dist/project-1.jar" Enabled="true" >
        <Class Name="serviceModules.ServiceModule2" Enabled="true" />
        <Class Name="serviceModules.ServiceModule3" Enabled="true" />
        <Class Name="serviceModules.ServiceModule4" Enabled="true" />
        <Class Name="serviceModules.ServerPush" Enabled="true" />
        <Class Name="serviceModules.TimeShower" Enabled="true" />
    </Modules>

    <Modules JarFile="/home/http/project-2/dist/project-2.jar" Enabled="true" >
        <Class Name="serviceModules.ServiceModule1" Enabled="true" />
        <Class Name="serviceModules.ServiceModule51" Enabled="true" />
        <Class Name="serviceModules.ServiceModule52" Enabled="true" />
        <Class Name="serviceModules.ServiceModule61" Enabled="true" />
        <Class Name="serviceModules.ServiceModule62" Enabled="true" />
        <Class Name="serviceModules.Test_serviceModulePool" Enabled="true" />
        <Class Name="serviceModules.Test_urlPaths" Enabled="true" />
    </Modules>
</Config>
```

(34)、对/home/http/qhs3.2/config1/objects.xml 配置文件的内容设置为：

```
<Config>
    <Objects JarFile="/home/http/objects/dist/objects.jar" Enabled="true" >
        <Class Name="objects.AccessTimes" Created="true" />
        <Class Name="objects.RedisConnection" Created="true" />
        <Class Name="objects.QueryRecorder" Created="true" />
    </Objects>
</Config>
```

(35)、在终端执行以下命令复制文件为/home/http/qhs3.2/config1/urlPaths.xml 配置文件：

```
cp -v /home/http/qhs3.2/demo/serverPush/config/urlPaths.xml /home/http/qhs3.2/config1;
```

(36)、对/home/http/qhs3.2/config1/options.xml 配置文件的内容设置为：

```
<Config>
    <!--RedisConnection 类引用 jedis-2.1.0.jar 包内的类。 -->
    <JarFile>/home/http/qhs3.2/demo/jedis-2.1.0.jar</JarFile>
</Config>
```

(37)、复制以下命令到终端并且执行命令，编译、生成 temp-service-module.jar，project-2 项目的 Test_serviceModulePool.java 要加载上述 jar 包：

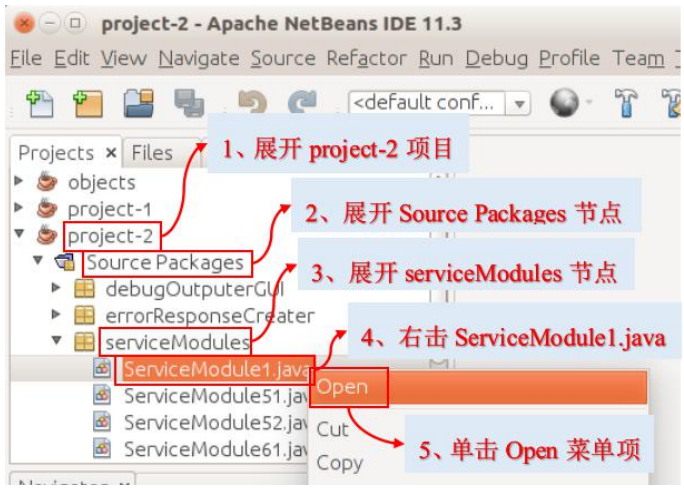
```
cd ~;
export JDK_HOME="/usr/java/jdk-11";
export QHS_HOME="/home/http/qhs3.2";
export DST_DIR_4="/home/http/qhsModules-11";
```

```
rm -Rvf $DST_DIR_4;
mkdir -vp $DST_DIR_4;
cp -v $QHS_HOME/demo/temp/ServiceModule.java $DST_DIR_4;

$JDK_HOME/bin/javac -encoding utf-8 -d $DST_DIR_4 \
        -classpath $QHS_HOME/bin/qhs.jar \
        $DST_DIR_4/ServiceModule.java;

$JDK_HOME/bin/jar cvf ~/temp-serivce-module.jar \
        -C $DST_DIR_4 temp/ServiceModule.class;
```

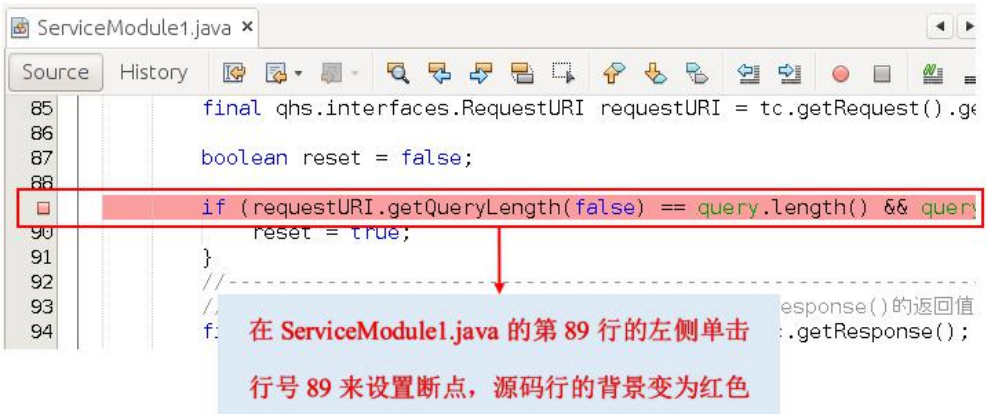
(38)、启动 Redis 服务器并且运行 Redis 客户设置 2 个键值，见 11.2 节。

执行上面的所有步骤后，可以立即调试 objects、project-1、project-2 等项目，这三个项目生成的 jar 包被 qhs 的配置文件所引用或者被 qhs 的类加载器池所加载，因此，可进入调试状态，否则，项目不能进入调试状态。调试上述项目时，应首先在源码上设置一个或多个断点，然后右击项目并单击 Debug 菜单项。如：

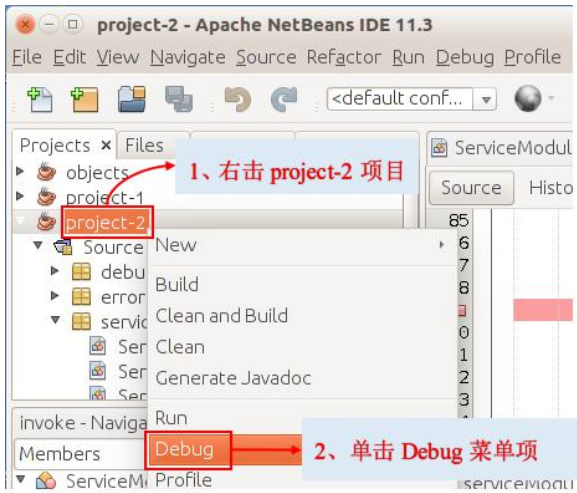
(1)、



(2)、



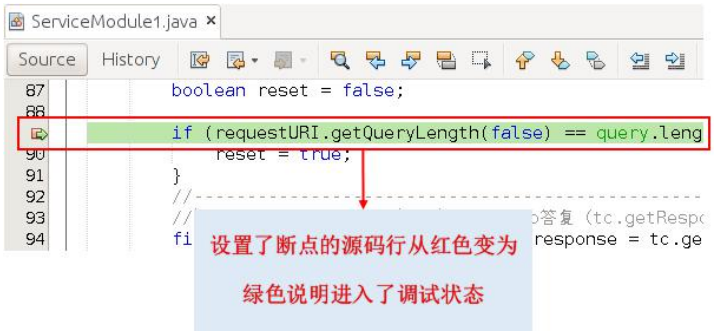
(3)、



(4)、执行第(3)步骤时，NetBeansIDE 尝试启动 qhs。如果 NetBeansIDE 显示出稳定的调试界面，说明 NetBeans IDE 成功启动 qhs，就进入第(5)步骤。如果 NetBeansIDE 不能成功启动 qhs，就弹出显示错误的对话框或者尝试把错误原因写入 log 文件（由/home/http/qhs3.2/config1/base.xml 配置文件的 LogFile 元素值说明）。查看对话框显示的错误、log 文件的内容并纠正错误后，重复第(3)步骤。

(5)、运行浏览器，在浏览器的地址栏输入“https://127.0.0.1:10000/speed-test.id”对 qhs 发送 http 请求。

(6)、



处于调试状态时，开发人员可以在键盘按下 F8、F7、F4 等键对 ServiceModules1.java 继续调试，也可以单击 NetBeansIDE 的 Debug 主菜单项并选取相关调试菜单项。

对于付费的 qhs 实例，处于调试状态时，如果开发人员想立即结束调试，就必须首先按照第 6 节说明的方法来终止 qhs 的运行，然后单击 NetBeansIDE 的“Debug >Continue”菜单项，让 qhs 实例运行到终止。如果用户不按照正确的步骤来结束调试，lcs 与时钟锁认为“qhs 实例仍然正在运行”，影响下一次 qhs 实例的启动。

16.2、远程调试

使用本调试方法时，必须：

(1)、对 NetBeansIDE 项目导入 qhs.jar 包。

(2)、对 NetBeansIDE 项目导入其它必需的 jar 包。

(3)、对 options.xml 配置文件设置-Xdebug、-agentlib 等 JVM 运行选项。

(4)、使 NetBeansIDE 连接到调试器。

另外：

(1)、无需对 NetBeansIDE 项目设置运行主类 qhs.Start、运行参数--ServerRoot、运行参数--Config。

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 4 节：

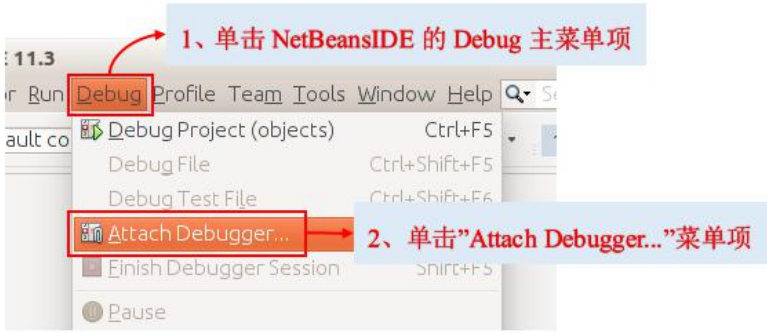
(1)、经过 16.1 节的所有步骤创建了 objects、project-1、project-2 等项目。

(2)、对/home/http/qhs3.2/config1/options.xml 配置文件设置为：

```
<Config>
    <!--RedisConnection 类引用 jedis-2.1.0.jar 包内的类。 -->
    <JarFile>/home/http/qhs3.2/demo/jedis-2.1.0.jar</JarFile>
    <Option>-Xdebug</Option>
    <Option>-agentlib:jdwp=transport=dt_socket,server=y,address=localhost:20000,suspend=n</Option>
</Config>
```

(3)、运行/home/http/qhs3.2/startup1.sh 来启动 qhs(如果 qhs 正在运行，就运行/home/http/qhs3.2/shutdown1.sh 来终止 qhs 的运行，并重新启动 qhs)。

(4)、

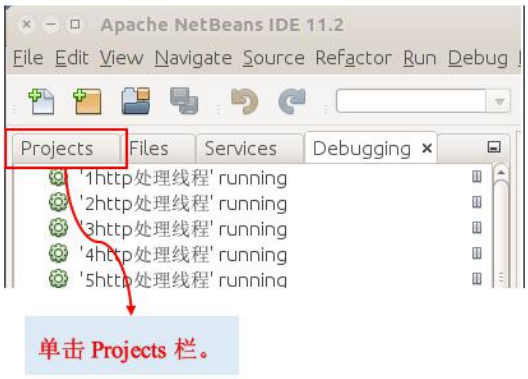


(5)、

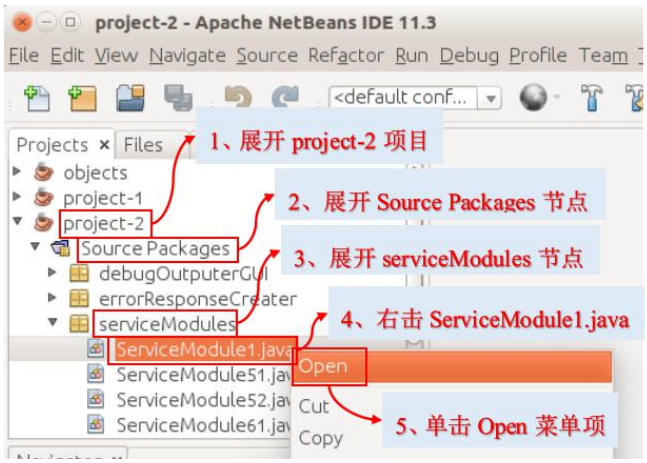
(6)、如果 NetBeansIDE 成功连接到调试器，NetBeansIDE 就显示类似下面情形的界面，否则，就要重新检查、执行上面的(1)至(5)步骤。

执行上面的所有步骤后，可以立即调试 objects、project-1、project-2 等项目，这三个项目生成的 jar 包被 qhs 的配置文件所引用或者被 qhs 的类加载器池所加载，因此，可进入调试状态，否则，项目不能进入调试状态。调试上述项目时，应首先在源码上设置一个或多个断点。如：

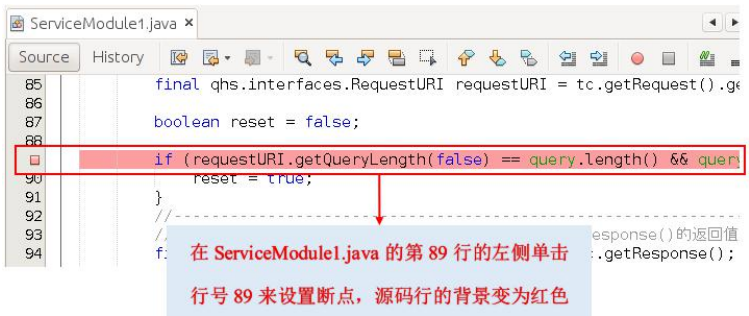
(1)、



(2)、

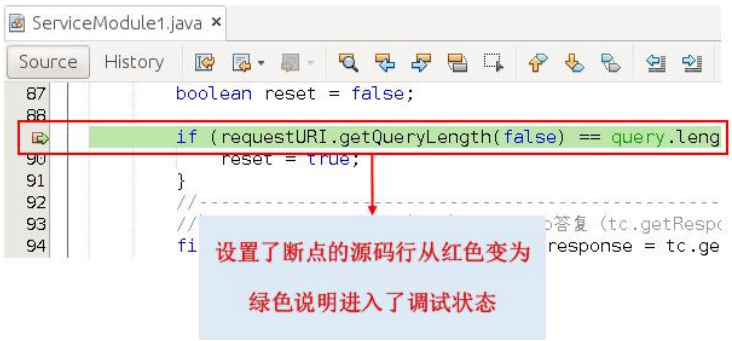


(3)、



(4)、运行浏览器，在浏览器的地址栏输入“https://127.0.0.1:10000/speed-test.id”对 qhs 发送 http 请求。

(5)、



处于调试状态时，开发人员可以在键盘按下 F8、F7、F4 等键对 ServiceModules1.java 继续调试，也可以单击 NetBeansIDE 的 Debug 主菜单项并选取相关调试菜单项。

对于付费的 qhs 实例，处于调试状态时，如果开发人员想立即结束调试，就必须首先按照第 6 节说明的方法来终止 qhs 的运行，然后单击 NetBeansIDE 的“Debug >Continue”菜单项，让 qhs 实例运行到终止。如果用户不按照正确的步骤来结束调试，lcs 与时钟锁认为“qhs 实例仍然正在运行”，影响下一次 qhs 实例的启动。