服务框架 HSF 使用与配置

一江 更新时间：2011-7-20

目录

HSF 常用 OPS 和开发工具介绍

HSF 常用链接入口：http://hsf.taobao.net，包括 HSF 服务查询 OPS，文档中心，下载中心等链接。

HSF 的使用和配置

下载和安装 HSF

第一步：从 HSF 主页访问 HSF 软件下载中心，下载 JBoss 4.2.2 和 HSF 1.4.8.4 压缩包。如下图所示：

图 2-1. HSF 下载中心

第二步： 解压 jboss-4.2.2.GA.zip 包到任意目录，如 D:\。这时 JBoss 应该位于 D:\jboss-4.2.2.GA 目录；解压 taobao-hsf.tgz 到%JBOSS_HOME%\server\default\deploy 目录。至此，JBoss 和 HSF 安装完成。JBoss 服务器目录结构如图 2-2 所示。

本文的后续章节中将，

- 使用%JBOSS_HOME%指代 JBOSS 解压目录 D:\jboss-4.2.2.GA
- 使用%DEPLOY_DIR%指代%JBOSS_HOME%\server\default\deploy 目录

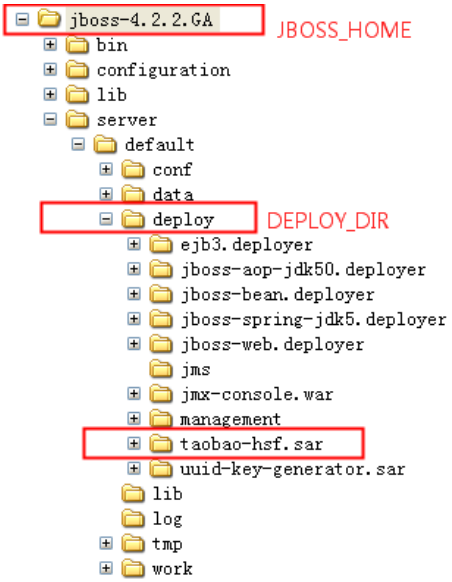

图 2-2. JBoss 和 HSF 安装后目录结构

第三步：执行%JBOSS_HOME%\bin\run.bat 启动 JBoss，这时访问 http://localhost/将能够看到 JBoss 服务器默认首页，如图 2-3 所示。

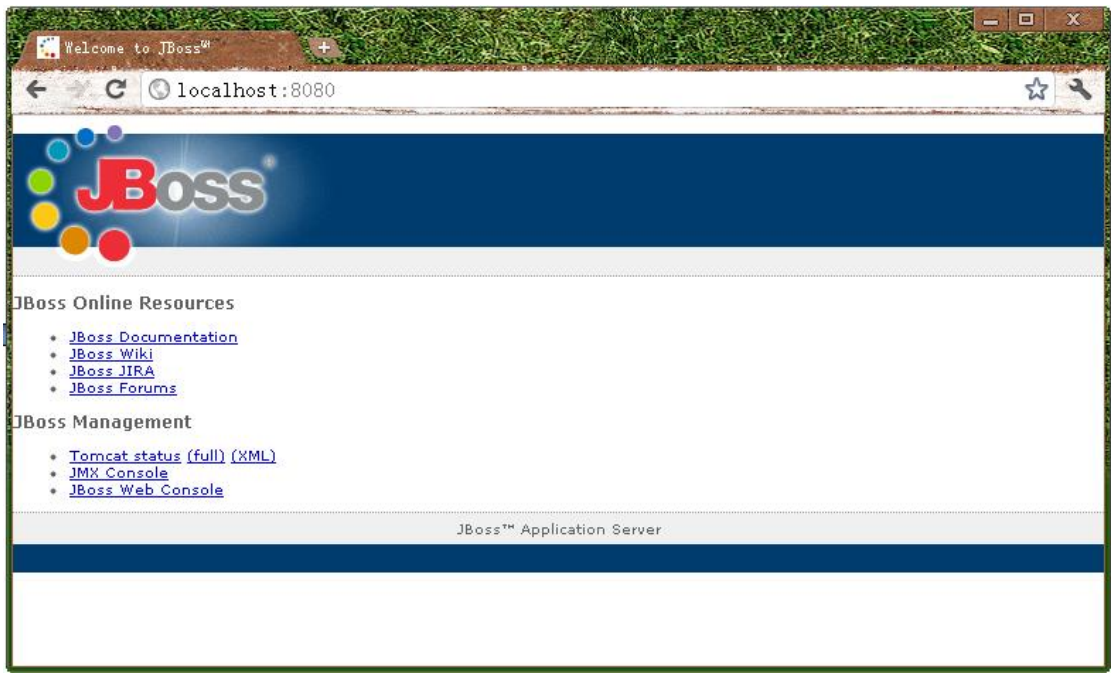

图 2-3. JBoss 服务器默认首页

如果启动时，遇到端口被占用异常：java.net.BindException: Address already in use: JVM_Bind:80，如图 2-4 所示。

```
java.net.BindException: Address already in use: JVM_Bind:80
        at org.apache.tomcat.util.net.JIoEndpoint.init(JIoEndpoint.java:500)
        at org.apache.tomcat.util.net.JIoEndpoint.start(JIoEndpoint.java:514)
        at org.apache.coyote.http11.Http11Protocol.start(Http11Protocol.java:2
        at org.apache.catalina.connector.Connector.start(Connector.java:1146)
        at org.jboss.web.tomcat.service.JBossWeb.startConnectors(JBossWeb.java
        at org.jboss.web.tomcat.service.JBossWeb.handleNotification(JBossWeb.j
        at sun.reflect.GeneratedMethodAccessor4.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcce
        at java.lang.reflect.Method.invoke(Method.java:597)
        at org.jboss.mx.notification.NotificationListenerProxy.invoke(Notificat
        at $Proxy44.handleNotification(Unknown Source)
        at org.jboss.mx.util.JBossNotificationBroadcasterSupport.handleNotifica
java:127)
        at org.jboss.mx.util.JBossNotificationBroadcasterSupport.sendNotificat
va:108)
        at org.jboss.system.server.ServerImpl.sendNotification(ServerImpl.java
        at org.jboss.system.server.ServerImpl.doStart(ServerImpl.java:497)
        at org.jboss.system.server.ServerImpl.start(ServerImpl.java:362)
        at org.jboss.Main.boot(Main.java:200)
        at org.jboss.Main$1.run(Main.java:508)
        at java.lang.Thread.run(Thread.java:619)
15:31:45,098 WARN  [JBossWeb] Failed to startConnectors
LifecycleException:  service.getName(): "jboss.web";  Protocol handler start fa
eady in use: JVM_Bind:80
        at org.apache.catalina.connector.Connector.start(Connector.java:1153)
        at org.jboss.web.tomcat.service.JBossWeb.startConnectors(JBossWeb.java
```

图 2-4. JBoss 启动，端口占用异常

这说明 JBoss Web 默认使用的 80 端口被其他程序占用，这时你可以停掉占用 80 端口的应用或修改 JBoss 端口配置。

修改 JBoss 端口的方法，打开%DEPLOY_DIR%\jboss-web.deployer\server.xml 文件，打到图示的位置，修改端口即可。这里我使用的是 8080 端口。

```
13    <Service name="jboss.web">
14
15    <!-- A "Connector" represents an endpoint by which requests are received
16         and responses are returned. Documentation at :
17         Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
18         Java AJP  Connector: /docs/config/ajp.html
19         APR (HTTP/AJP) Connector: /docs/apr.html
20         Define a non-SSL HTTP/1.1 Connector on port 8080
21    -->                    在这里修改jboss web连接端口
22    <Connector port="8080" address="${jboss.bind.address}"
23         maxThreads="250" maxHttpHeaderSize="8192"
24         emptySessionPath="true" protocol="HTTP/1.1"
25         enableLookups="false" redirectPort="8443" acceptCount="100"
26         connectionTimeout="20000" disableUploadTimeout="true" />
```

图 2-5. 修改 JBoss Web 连接端口

到这里，Jboss 和 HSF 的安装就完成了。部署着 HSF 服务的 war 包类型的应用，可以直接部署到 Jboss 的%DEPLOY_DIR%。

服务开发与部署

本节开发和部署一个简单的 Hello World 服务，使用的开发工具是 eclipse，安装了 m2clipse 插件。

第一步：使用 maven 命令或 eclipse 向导创建一个简单的 Web 工程 hsf-sample。
使用命令方式下，命令如下：

```
mvn archetype:create -DgroupId=com.taobao.hsf.test -DartifactId=hsf-sample
-DpackageName=com.taobao.hsf.test -DarchetypeArtifactId=maven-archetype-webapp
```

使用 eclipse 方式下，过程如下：

1. 选择 File->New->Others（快捷键 Ctrl+N），选择 Maven 中的 Maven Project，Next，

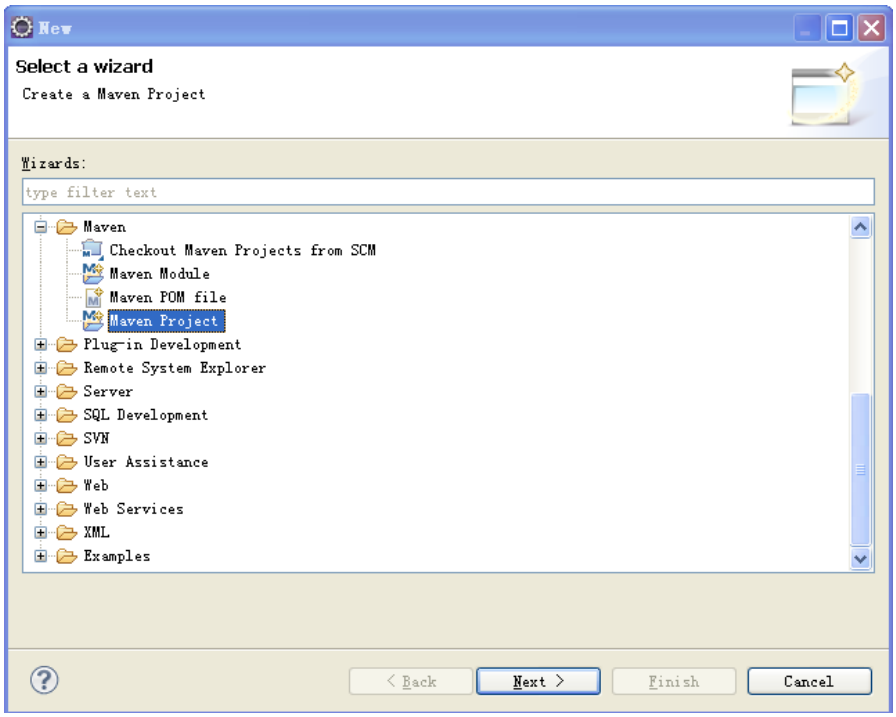

图 2-6. 新建工程

2. 勾选 Create a simple project，Next，

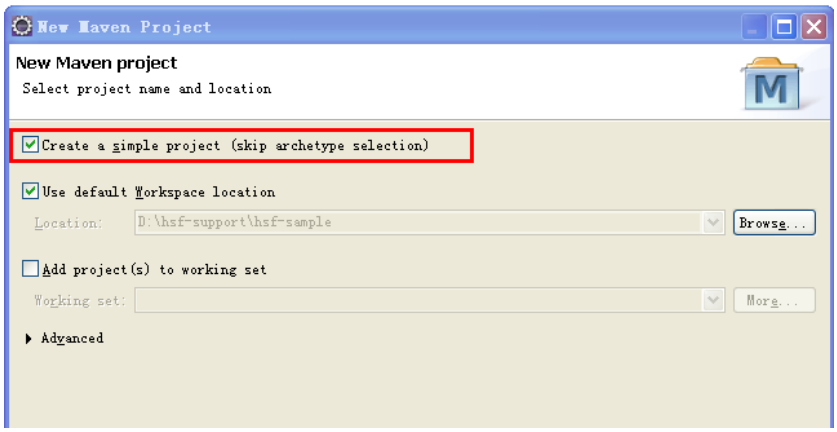

图 2-7. 新建 Maven 工程

3. 进行配置项目属性配置，如下图所示。配置结束后，选择 Finish。

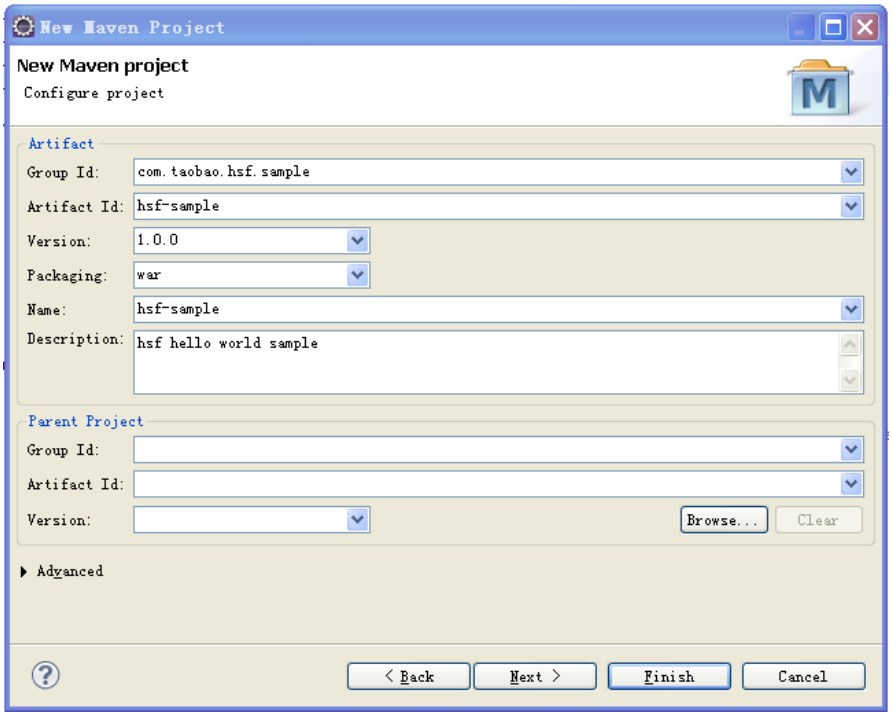

图 2-8. 项目属性配置

4. 创建完成的 eclipse 工程 hsf-sample，如图 2-9 所示。

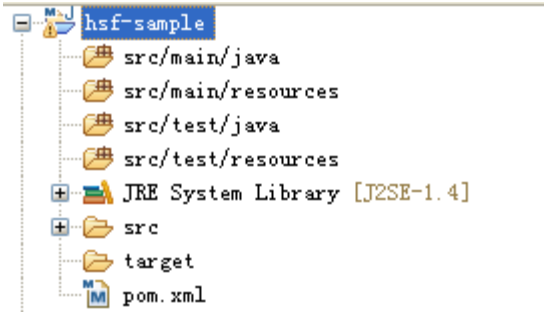

图 2-9. hsf-sample 工程目录结构

提示：

- 使用命令方式创建的工程，同样可以导入到 eclipse 中。有两种方式：
  - File->Import->Existing Maven Projects，然后选择 Maven 工程目录，执行导入
  - 在 Maven 工程目录中执行：mvn eclipse:eclipse，生成 eclipse 配置文件，通过 File->Import->Existing Projects into Workspace,然后选择目录执行导入
- 项目创建成功后，默认的使用的编译器为 JDK 1.4 版本，而实际的项目开发目前以 1.6 为主。下一步中将会通过修改 pom.xml 修改该配置。

第二步：在项目的 pom 配置中加入以下配置信息，用于指定编译和打包使用的 JDK 版本为 1.6，如下所示。

```
<properties>
    <java.version>1.6</java.version>
</properties>
<build>
    <finalName>hsf-sample</finalName>
    <plugins>
        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>${java.version}</source>
                <target>${java.version}</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

修改完成后，右击工程，选择 Maven->Update Project Configuration，eclipse 将会刷新 Maven 配置。

第三步：在工程中添加两个两个基本依赖，添加如下内容到 pom.xml 中：

```
<dependencies>
    <dependency>
        <groupId>org.apache.geronimo.specs</groupId>
        <artifactId>geronimo-servlet_2.5_spec</artifactId>
        <version>1.2</version>
        <type>jar</type>
        <scope>provided</scope>
    </dependency>
```

```
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring</artifactId>
        <version>2.5.6</version>
        <type>jar</type>
        <scope>compile</scope>
    </dependency>
</dependencies>
```

在这里，geronimo-servlet 用于为 Web 工程提供 servlet 支持，spring 用于为 servlet 提供 spring 应用上下文支持。 安装了 m2clipse 插件的 eclipse 将会自动刷新工程依赖，并下载、加入 jar 包。如果没有自动执行，可以使用 mvn eclipse:eclipse 命令或 m2clipse 选项中的 Update Dependencies（见图 2-10）完成该操作。完成这一步后，eclipse 中的工程将会引入图 2-11 所示的依赖包。

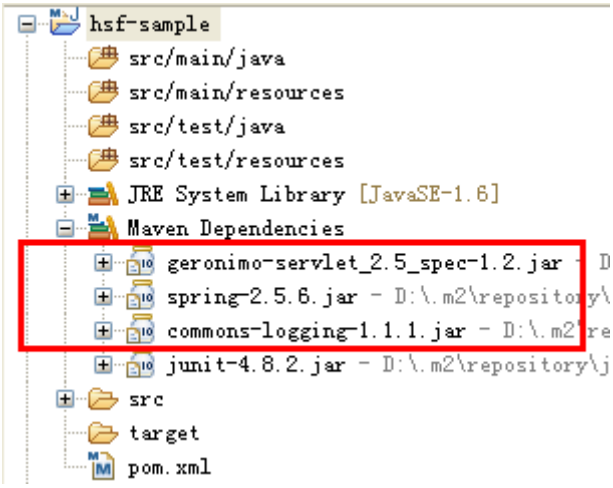

图 2-11. 项目依赖的 jar 包

第三步：在源代码中创建一个简单的 HelloWorld 服务，接口和实现如下：

```
package com.taobao.hsf.sample.service;

public interface HelloWorldService {
    String sayHello(final String name);
}

package com.taobao.hsf.test.service;

public class HelloWorldServiceImpl implements HelloWorldService {

    public String sayHello(final String name) {
        return "Hello, " + name + "!";
    }
}
```

第四步：在 src\main\resources 下添加服务的 Spring 配置：ApplicationContext.xml，内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="helloWorldServiceImpl"
class="com.taobao.hsf.sample.service.HelloWorldServiceImpl" />

    <bean id="helloWorldServiceProvider"
class="com.taobao.hsf.app.spring.util.HSFSpringProviderBean"
        init-method="init">
        <property name="serviceInterface">
            <value>com.taobao.hsf.sample.service.HelloWorldService</value>
        </property>
        <property name="target">
            <ref bean="helloWorldServiceImpl" />
        </property>
        <property name="serviceVersion">
            <value>1.0.0.daily</value>
        </property>
    </bean>
</beans>
```

第五步：定义 Web 工程的配置，创建 src\main\webapp\WEB-INF\web.xml，这里只加入一个 Spring 监听器，用于在应用启动时让 Spring 完成服务初始化。配置如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    version="2.5">
```

```
    <context-param>
        <param-name>contextConfigLocation</param-name>
    <param-value>classpath:ApplicationContext.xml</param-value>
    </context-param>
        <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListene
r</listener-class>
    </listener>
</web-app>
```

第六步：打包部署应用，在创建的项目根目录下执行, mvn clean package 进行打包。打完成功后，target 目录中将会生成名称为 `hsf-sample.war` 的 War 包。将该 War 包拷贝到 JBoss 的%DEPLOY_DIR%中，然后启动 JBoss，这就完成了 HSF 应用的发布。

在发布了一个服务后，如何查看该服务已经被成功发布，别人又如何能知道你这一服务呢？这就要借助于 HSFOPS 平台了，登录 http://hsf.taobao.net/，进入线下查询地址链接，首页就会显示当前机器所订阅和发布的所有服务，如图 2-12 所示。

图 2-12. HSFOPS 服务查询

注意：

- 在这一应用的启动过程中，你可能会看到 log4j 相关的异常，这里缺失日志配置引起的，不会影响服务的发布和使用。
- 这里你可能会问，为什么工程没有依赖任何 HSF 的 jar 包，服务却能够发布成功。原因是 JBoss 服务器中已经部署了 HSF 包，因此，部署在 JBoss 中的所有应用都可以透明地使用 HSF 所提供的功能。

服务查询

所有正常发布出去的 HSF 服务，都可以通过 HSFOPS 平台查询到其发布信息。HSFOPS 平台能够实时从配置中心查询服务的各种信息，包括：服务地址信息、服务的被调用信息，服务地址的实时推送情况等等。下表为 HSFOPS 查询语法的示例。

com.taobao.api.item.ItemService:1.0.0	根据 dataId 精确检索服务

com.taobao.api.*	模糊匹配以 com.taobao.api.开头的服务
item	模糊匹配服务名称中包括 item 的服务
item type:consumer	匹配订阅了*item*的服务订阅者
uic group:HSF	检索 HSF 组内的*uic*服务
uic ip:172.23*	检索 172.23.网段发布的*uic*服务
uic ip:10.* type:consumer	检索 10.网段内订阅了*uic*服务的服务消费者

服务调用

下面介绍如何调用一个 HSF 服务。这里继续使用上一节中创建的 Web 工程 hsf-sample 完成这一实例。

第一步：添加服务调用配置信息，编辑 src\main\resources 中的 ApplicationContext.xml，添加以下内容：

```
<bean id="helloWorldService"
class="com.taobao.hsf.app.spring.util.HSFSpringConsumerBean"
        init-method="init">
    <property name="interfaceName">
    <value>com.taobao.hsf.sample.service.HelloWorldService</value>
    </property>
    <property name="version">
        <value>1.0.0.daily</value>
    </property>
</bean>
```

配置文件中指定了要调用的接口名以及版本，这需要与前一节中的服务提供者的定义相一致。

第二步：编写一个简单的 servlet，内容如下：

```
package com.taobao.hsf.sample.servlet;

import ...

@SuppressWarnings("serial")
public class HelloWorldServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
```

```
        WebApplicationContext context =
WebApplicationContextUtils.getWebApplicationContext(getServletContext(
));
        HelloWorldService helloWorldService = (HelloWorldService)
context.getBean("helloWorldService");

        PrintWriter out = resp.getWriter();
        out.println(helloWorldService.sayHello("yijiang"));
        return;
    }
}
```

第四步：在 web.xml 中添加 servlet，编辑 web.xml，添加以下内容：

```
<servlet>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>com.taobao.hsf.sample.servlet.HelloWorldServlet</serv
let-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>/helloWorld</url-pattern>
</servlet-mapping>
```

第五步：重新编译，打包，并部署到 JBoss 中，使用与前一节相同的命令和部署方式。部署成功后，启动 JBoss，尝试访问：http://localhost:8080/hsf-sample/helloWorld，如果服务器端口经过修改，请将 URL 中的端口替换成修改后的端口。

图 2-13. HelloWorld 实例执行结果

注意：

- 该实例中，服务提供者和服务调用者位于同一个应用中。如果不在同一应用内，服务调用者应用需要依赖相同服务定义接口包。实际项目中，应用的提供和使用往往是在多个项目中的。
- 通过这个例子可以看到，HSF 主要是使用 HSFSpringProviderBean 和 HSFSpringConsumerBean 两个 Spring Bean 工厂类完成了对服务的发布和调用。
- 这里的实例中的编程规范不一定符合线上项目开发规范，不作为开发参考。

HSFUNIT 测试包的使用

HSFUNIT 是目前推荐使用的 HSF 服务测试包，它提供了直接在 JUNIT 或类 JUNIT 测试场景中调用 HSF 服务。下面介绍一下如何使用 HSFUNIT 测试包，HSF 服务以刚才我们发布的服务为例。

第一步：在项目的/src/test/resources 目录添加测试资源：ApplicationContext-Test.xml，该文件中配置了一个 HSFSpringConsumerBean，和上一节中的配置文件内容基本一致，惟一的不同是增加了 target 属性，显式地指定了要调用的服务目标地址。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="helloWorldService"
class="com.taobao.hsf.app.spring.util.HSFSpringConsumerBean"
        init-method="init">
        <property name="interfaceName">

    <value>com.taobao.hsf.sample.service.HelloWorldService</value>
        </property>
        <property name="version">
            <value>1.0.0.daily</value>
        </property>
        <property name="target">
            <value>10.232.36.84:22200</value>
        </property>
    </bean>
</beans>
```

第二步：编写测试用例，测试用例中依赖了两个包 Junit 和 spring-test 包，可以添加以下依赖到 pom 配置中，

```
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.15</version>
            <type>jar</type>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.8.2</version>
            <type>jar</type>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>3.0.5.RELEASE</version>
            <type>jar</type>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.taobao.hsf</groupId>
            <artifactId>hsfunit</artifactId>
            <version>1.0.2</version>
            <type>jar</type>
            <scope>test</scope>
        </dependency>
```

第三步：编写和执行测试用例。测试用例代码如下所示：

```java
package com.taobao.hsf.sample.service;

import ...

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations =
{ "classpath:ApplicationContext-Test.xml" })
public class HelloWorldServiceTestCase {

    static {
        HSFEasyStarter.start("D:\\hsf-support\\hsf", "1.4.8.4");
    }

    @Autowired
    protected HelloWorldService helloWorldService;

    @Test
    public void testHelloWorld() {
        String result = helloWorldService.sayHello("yijiang");
        Assert.assertEquals("Hello, yijiang!", result);
    }
}
```

HSF Jetty 插件的使用

为了方便开发人员使用 HSF，提高开发效率，HSF 提供了支持 Jetty 服务器的 eclipse 插件。下面介绍如何安装和在开发中使用 Jetty 插件。这里采用的 eclipse 版本为 3.5.2。

第一部分：安装 Jetty 插件。在 eclipse 中选择 Help->Install New Software...，

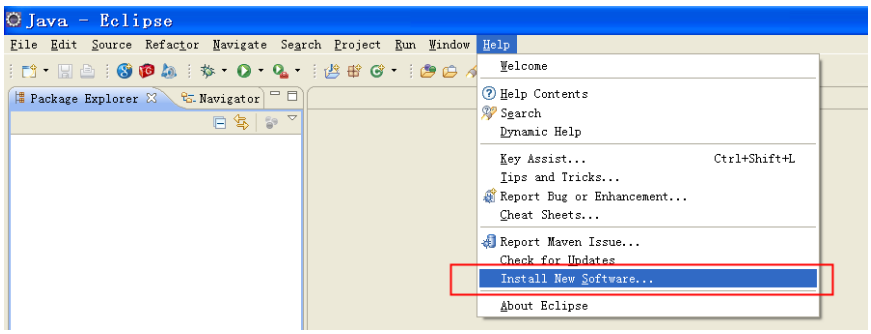

图 3-1. Install New Software...

在弹出的 Install 对话框中，输入 http://hsfops.taobao.net/hsf/hsf_jetty/update_site，然后

选择 Add。在弹出的 Add Site 对话框中直接点击 OK。如图 3-1，3-2 所示。

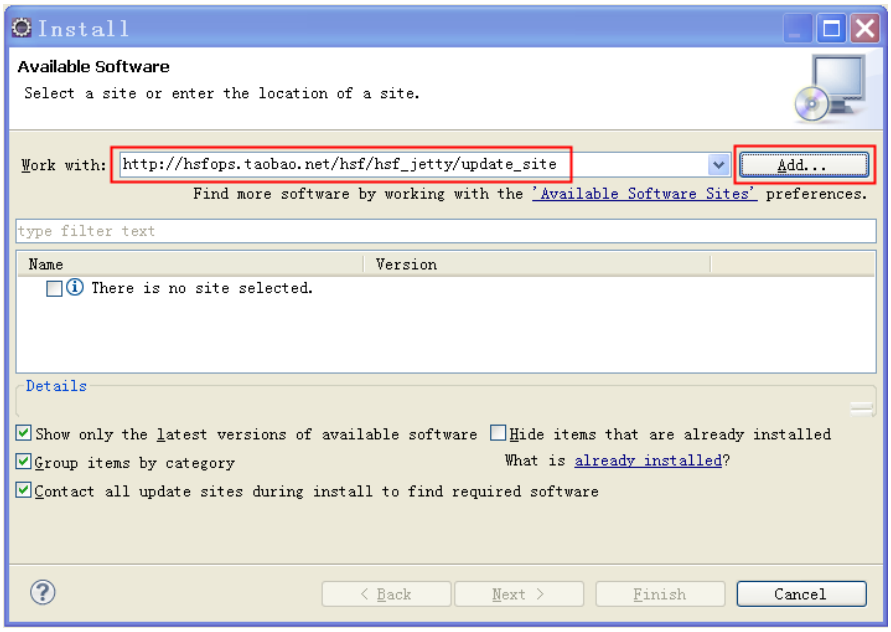

图 3-1. Install 对话框

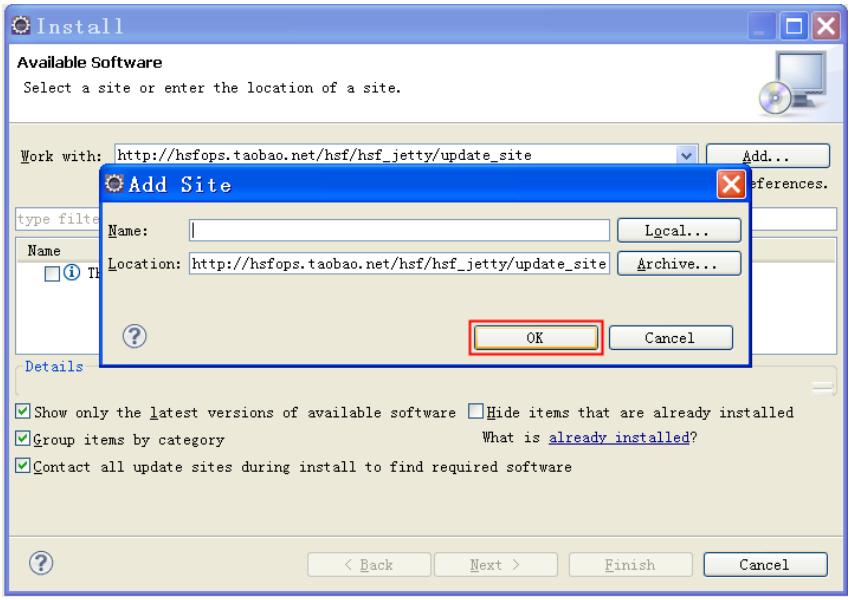

图 3-2. Add Site 对话框

Eclipse 将搜索并显示出要安装的插件，选中插件，Next，如图 3-3 所示。

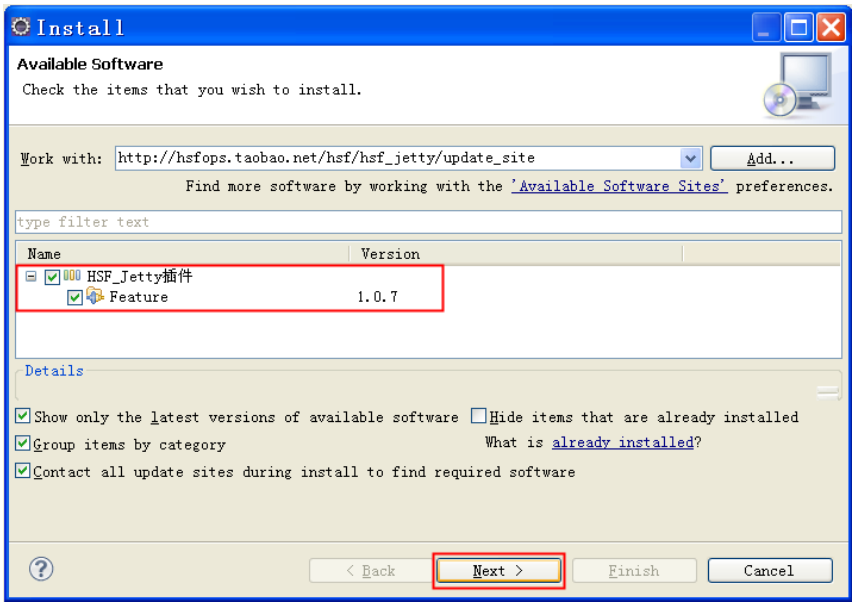

图 3-3. Install 对话框

然后将出现安装描述信息对话框，选中 Feature，Next，如图 3-4 所示。

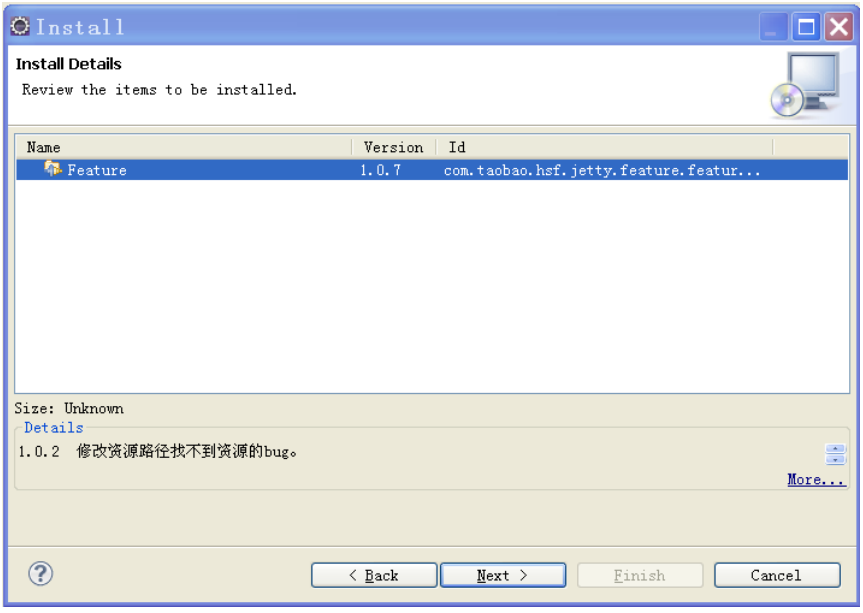

图 3-4. Install Details 对话框

接受授权，Next。如图 3-5 所示。

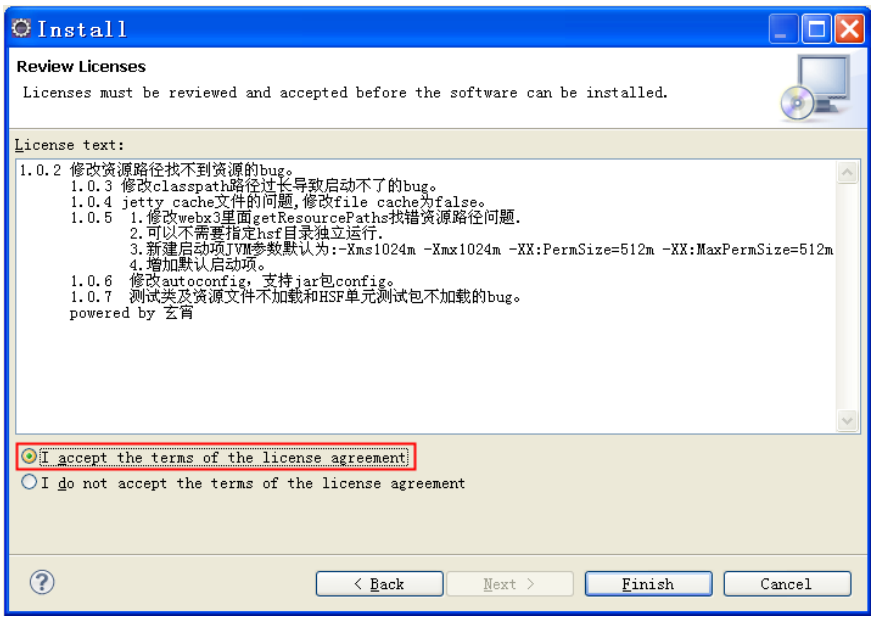

图 3-5. Review License 对话框

接下来，eclispe 将下载并执行安装，中间会有一次弹出安全警告对话框，选择 Ok 确定继续安装。如图 3-6 所示。

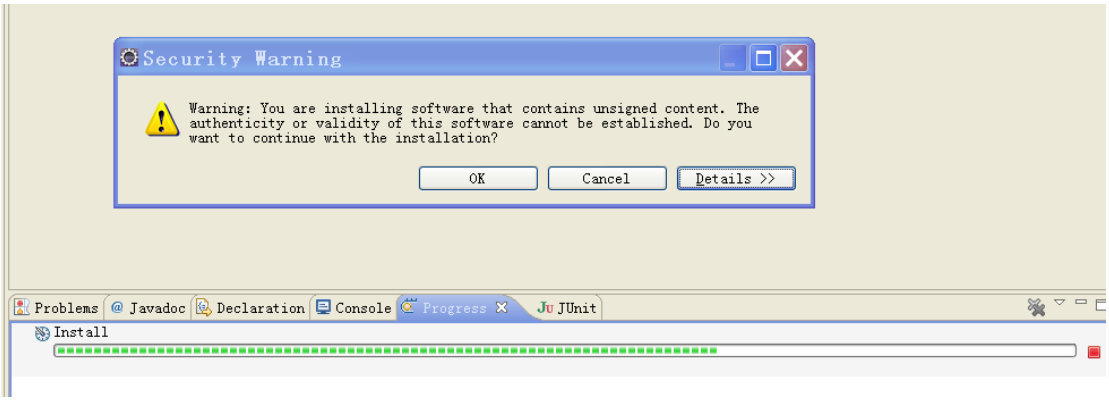

图 3-6. 安全警告对话框

安装完成后 eclipse 会提示重新启动以启用该插件。重启后，Jetty 插件就安装完成了。

第二部分：使用 Jetty 插件。

在 eclipse 中选择 Run->Run Configuration...，如图 3-7 所示。

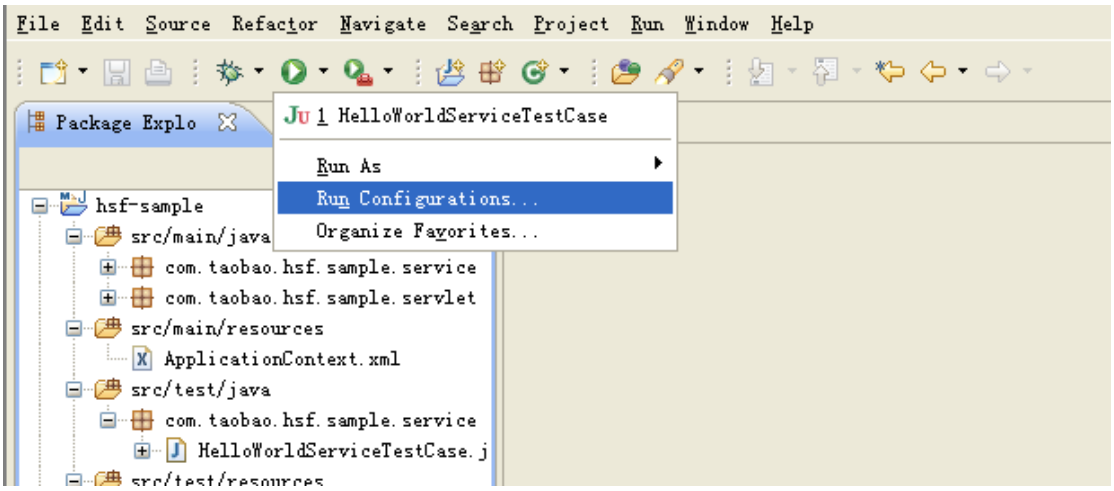

图 3-7. Run Configuration…

按照图 3-8 所示逐一配置各个参数，就完成了配置过程，点击 Run，就可以启动应用了。

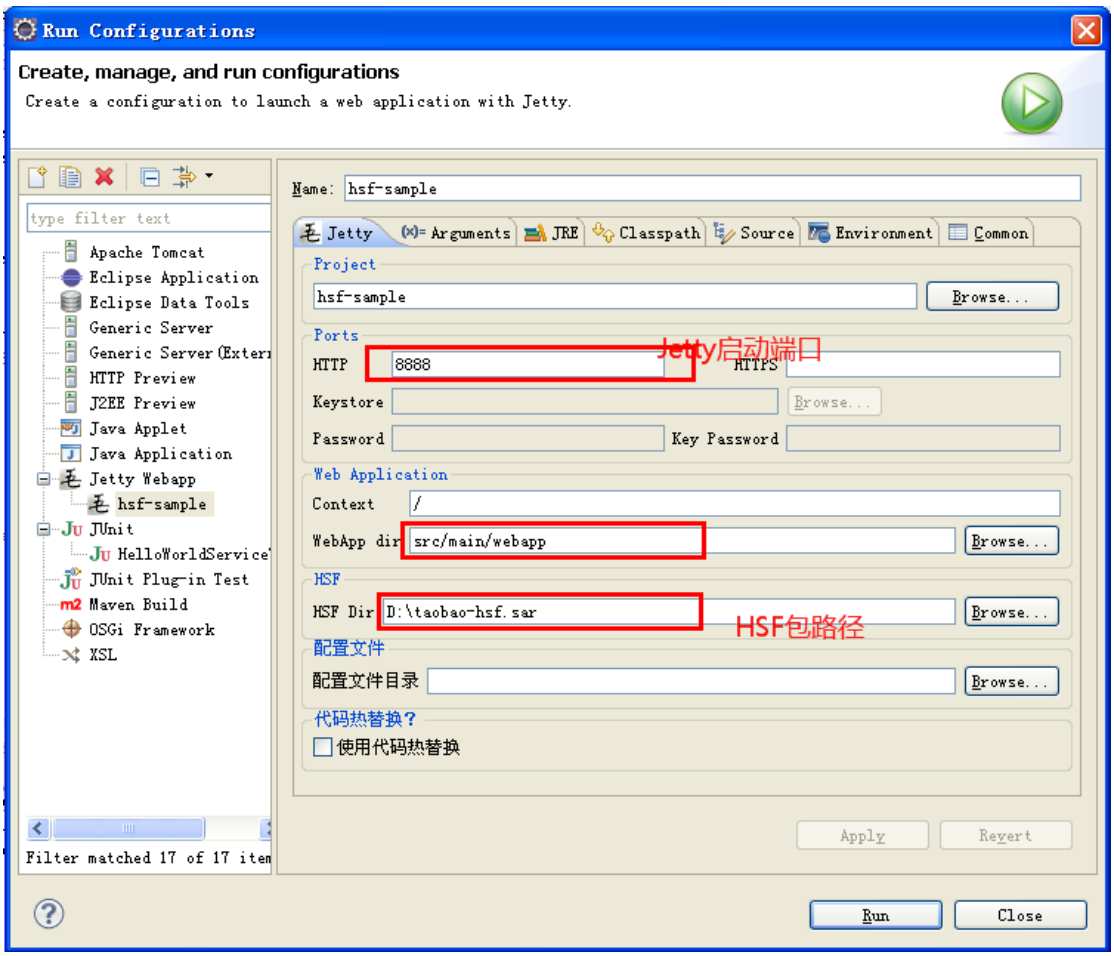

图 3-8. 配置 Jetty 插件运行参数

注意：

启动时如果遇到错误：

```
Error occurred during initialization of VM
Could not reserve enough space for object heap
```

可以在配置对话框的Arguments中将启动参数适当调小。如图3-9所示。

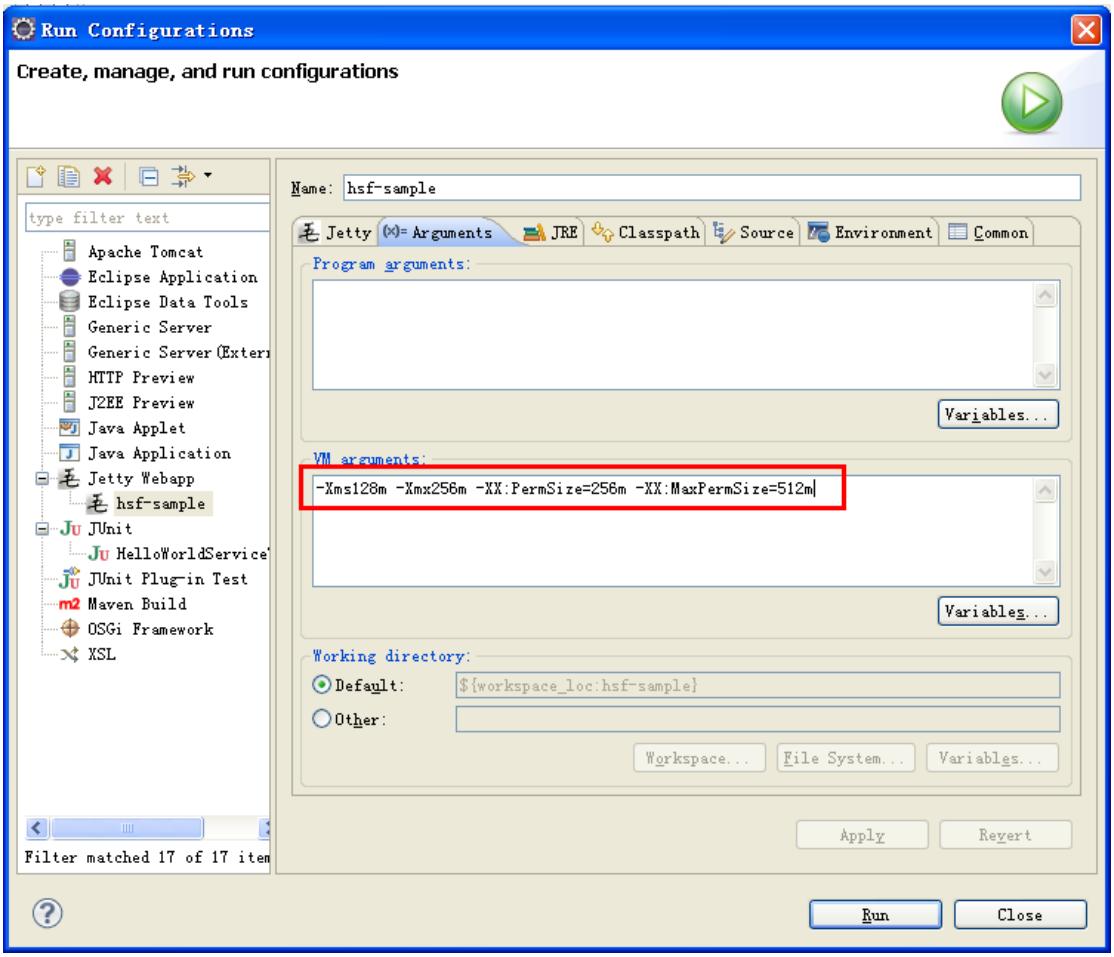

图 3-9. 配置 Jetty 插件启动参数

成功启动 Jetty 插件后，尝试访问 http://localhost:8888/helloWorld ，应该可以看到与图 2-11 一样的返回结果。

Jetty 插件还提供了数据源配置功能，这里不详细介绍，有兴趣的可以参考 HSF Jetty 插件使用指南。

路由规则和限流规则的介绍

HSF 路由规则支持接口路由,方法路由，参数路由,并且在路由规则的情况下支持权重规则，HSF 采用 Groovy 脚本作为路由规则设置内容。详细介绍见 confluence：http://confluence.taobao.ali.com:8080/pages/viewpage.action?pageId=100270328

HSF 限流规则提供了对某应用或服务的访问流量进行控制的功能。详细介绍见 confluence：http://confluence.taobao.ali.com:8080/pages/viewpage.action?pageId=192188098