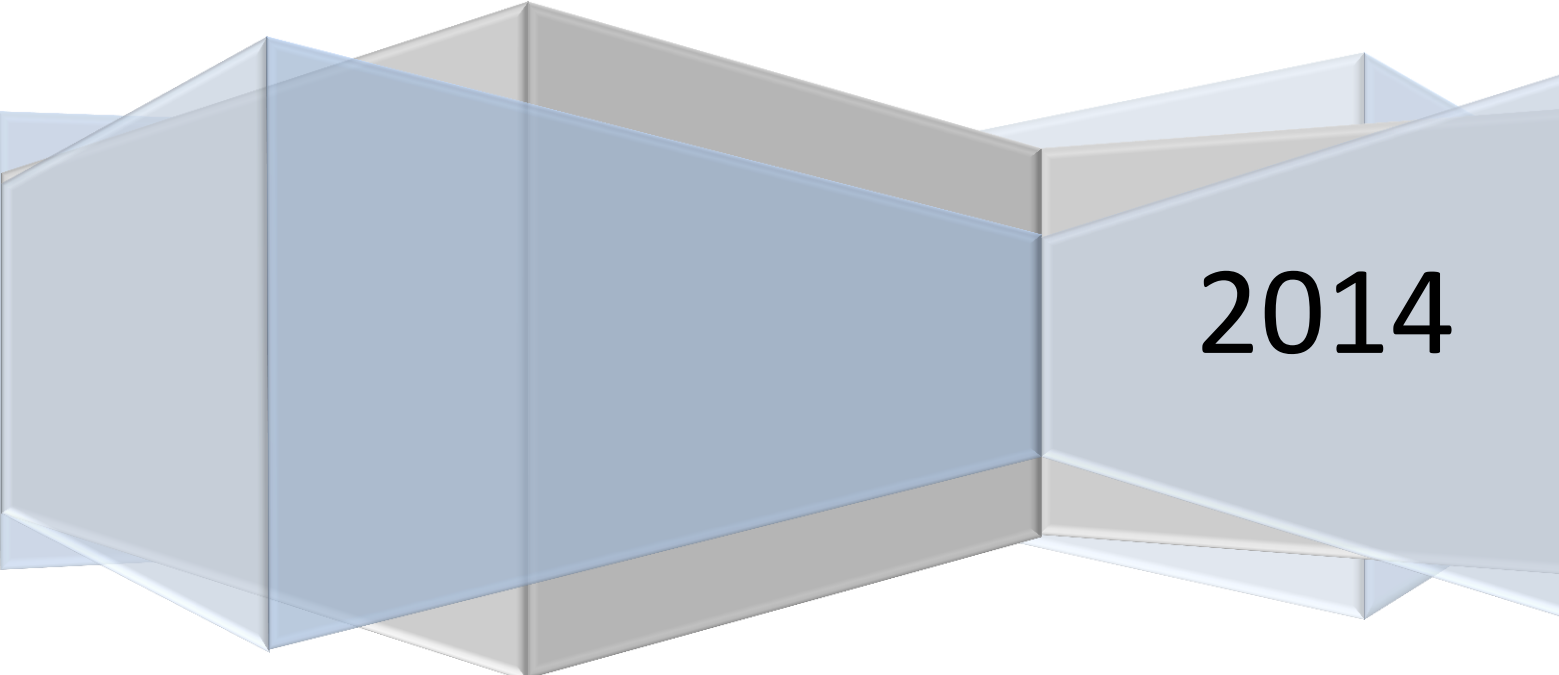


中间件&稳定性平台-服务框架

HSF 新人手册 for 2.X

嵐衫



2014

HSF 2.X 介绍

ConfigServer

Diamond

Pandora

HSF 的初始 Demo

HSF 的下载和安装

Tomcat

HSF sar 包下载

发布 HSF 服务

消费 HSF 服务

HSF 的其他启动方式

Light-Api

HSF-standalone

HSF Bean 配置详解

HSFSpringProviderBean

HSFSpringConsumerBean

HSF 规则使用介绍

路由规则

接口路由

方法路由

参数路由

如何配置路由规则

归组规则

同机房优先规则

权重规则

HSF 异步调用

Future 调用

Callback 调用

HSF 2.X 新特性

Pandora 控制台

泛化调用

优雅上下线

更多的 HSF 2.X 功能

HSF 2.X 介绍

HSF 作为公司的基础中间件组件，旨在为集团应用提供一个分布式的服务框架，HSF 从分布式应用层面以及统一的发布/调用方式层面为大家提供支持，从而可以很容易的开发分布式的应用以及提供或复用公用功能模块，而不用考虑分布式领域中的各种细节技术，例如进程通讯、性能损耗、调用的透明化、同步/异步调用方式的实现等等问题。

HSF 的百科入口 <http://gitlab.alibaba-inc.com/middleware/hsf2-0/wikis/home>

服务治理页面 <http://ops.jm.taobao.net/hsfops>。

下载中心 hsf.taobao.net/hsfversion/

虽然 HSF 使用起来非常方便，但通过与使用 HSF 的同学的接触，发现大家对一些概念并不清楚，在这里简单叙述一下。

ConfigServer

HSF 是一个 RPC 框架，远程调用对端的地址就是由 ConfigServer 来推送的，这样用户只需要配置自己的服务端或者消费端，不需要对自己的地址进行管理。

ConfigServer 不光负责向消费端推送对应服务的服务 IP 列表，还可以区分整个阿里应用的地址环境。众所周知阿里的开发环境分为日常、性能、预发、小淘宝、线上等等，而这些环境的概念，就是靠 ConfigServer 来区分的，例如一个服务端在性能环境注册了一个 HSF 服务，它会连上性能环境的 ConfigServer，这样 ConfigServer 只会把这个服务端的 IP 推送给同样在性能环境的消费端。

更多 ConfigServer 的信息请看

<http://gitlab.alibaba-inc.com/middleware/configserver/wikis/home>。

Diamond

Diamond 也参与环境的区分，用于存放 HSF 的各种规则，是持久化的配置中心。

Pandora

Pandora 是 HSF 生存的容器，对于 HSF2.X 来说，HSF 只是作为 hsf.jar.plugin 这个插件，存活在 pandora 中，由 pandora 来管理整个 HSF 的生命周期和二方包的隔离，以后就没有 HSF 版本这一说，只有 pandora 的版本，其中 HSF 的版本由 pandora 来指定。

HSF 的初始 Demo

HSF 的下载和安装

Tomcat

HSF 2.X 推荐使用 Ali-tomcat, 去下载中心 hsf.taobao.net/hsfversion/ 下载最新版本的 ali-tomcat, 如下图:

其他相关软件下载		
软件名称	版本	版本介绍
JBoss	4.2.2	线上通用的jboss版本, 跟官方版本不一样, 修改了classl
Jetty	7.3.1	线上jboss的应用, 直接使用该版本迁移, 所有的部署结构
阿里官方 ali-tomcat	7.0.52.2	tomcat7, jdk7 下支持 JSR 标准 WebSocket 1.0
阿里官方 ali-tomcat	7.0.54.1	阿里应用统一升级到此版本, tomcat7 最新版本, jdk7 下支
Hotswap Patch	0.2	无
HSF-Standalone	1.0.0	无

下载解压后, 目录结构如下:

bin	2014/7/4 14:47	文件夹	
conf	2014/7/4 14:49	文件夹	
deploy	2014/8/15 23:28	文件夹	
lib	2014/7/7 13:50	文件夹	
logs	2014/7/2 15:35	文件夹	
temp	2014/7/2 15:35	文件夹	
tools	2014/7/2 15:35	文件夹	
work	2014/7/4 14:49	文件夹	
LICENSE	2014/7/2 15:35	文件	56 KB
NOTICE	2014/7/2 15:35	文件	2 KB

用户的 war 包和 HSF 的 sar 包均放在 deploy 目录下。

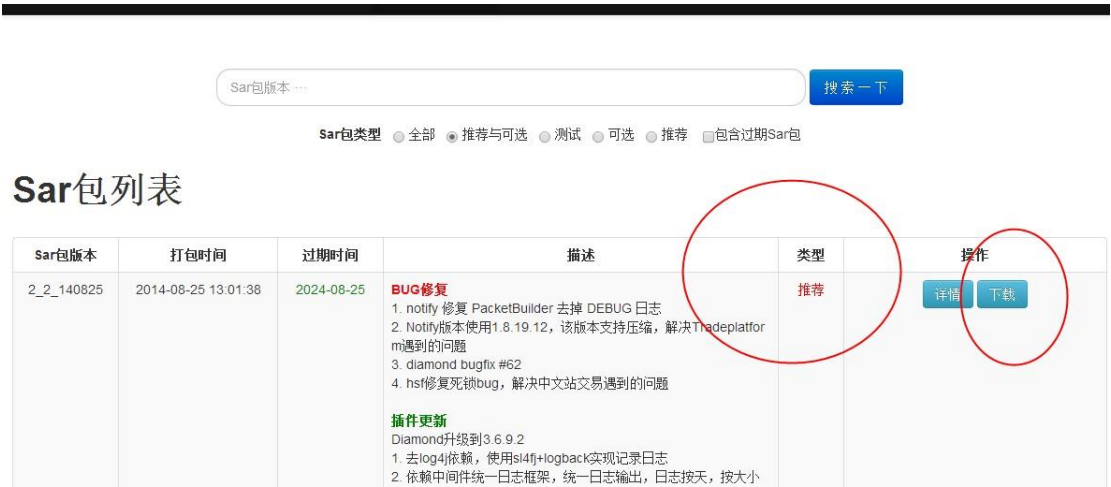
Ali-tomcat 要求 deploy 目录下一定要有 hsf 的 sar 包, 切记!

HSF sar 包下载

HSF 的 sar 包在 2.X 中其实是 pandora 的 sar 包，为了兼容老的 HSF 概念，所以命名还是 taobao-hsf.sar，需要去 pandora 运维页面下载推荐版本：
<http://ops.jm.taobao.org:9999/pandora-web/index.html>



进入 sar 包列表后，选择推荐版本。



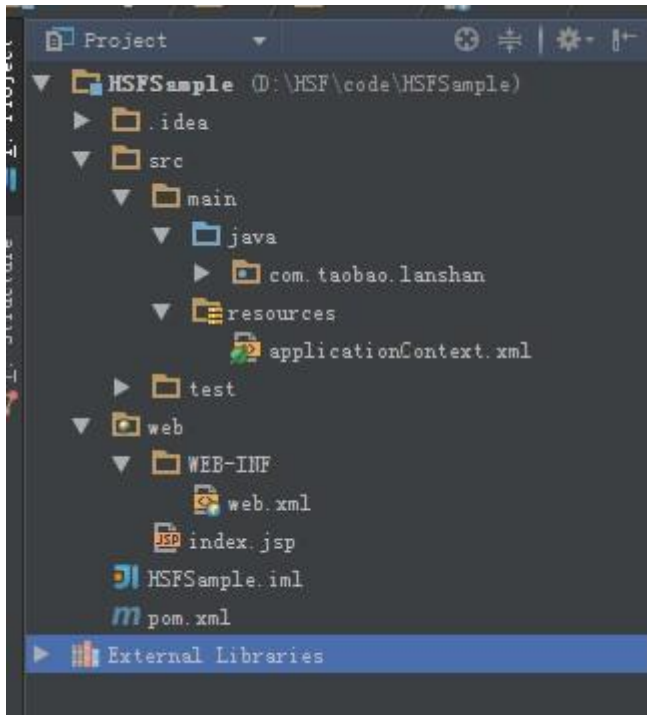
下载完 sar 包之后，放入 tomcat 的 deploy 目录。

到这，HSF 的环境就准备好了。

发布 HSF 服务

由于现在 IDEA 和 Eclipse 使用人数都非常多，这里就不详细介绍 Web 工程的创建过程了，具体步骤可以自行谷歌。

以 IDEA 为例，建立好的 web 工程如下：



想发布 HSF 服务，分为以下几步：

第一：在 **POM.xml** 中配置以下信息，用于指定编译和打包使用的 JDK 版本：

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

第二： 在工程中添加 Spring 和 Servlet 依赖：

```
<dependencies>
    <dependency>
        <groupId>org.apache.geronimo.specs</groupId>
        <artifactId>geronimo-servlet_3.0_spec</artifactId>
        <version>1.0</version>
        <type>jar</type>
        <scope>compile</scope>
    </dependency>

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

第三： 创建一个 HelloWorld 服务，作为服务端，要提供接口和实现：

接口：

```
package com.taobao.lanshan.service;

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

实现类：

```
package com.taobao.lanshan.service;

public class HelloWorldServiceImpl implements HelloWorldService{
    @Override
    public String sayHello(String name) {
        return "hello : " + name;
    }
}
```

第四： 为 HelloWorld 服务编写 Spring 配置文件。

```
<bean
id="HelloWorldServiceImpl"
class="com.taobao.lanshan.service.HelloWorldServiceImpl"/>

<bean
id="HelloWorldService"
class="com.taobao.hsf.app.spring.util.HSFSpringProviderBean" init-method="init">
    <property name="serviceInterface">
        <value>com.taobao.lanshan.service.HelloWorldService</value>
    </property>
    <property name="target">
        <ref bean="HelloWorldServiceImpl"/>
    </property>
    <property name="serviceVersion">
        <value>1.0.0.daily</value>
    </property>
    <property name="serviceGroup">
        <value>HSF</value><!-- 组别一致的服务才可以互相调用 -->
    </property>
    <property name="serviceName"><!-- 仅仅便于管理 -->
        <value>HelloWorld</value>
    </property>
</bean>
```

第五： 在 web.xml 中添加一个 Spring 监听器，让容器启动的时候 Spring 完成服务初始化。

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener
    -class>
</listener>
```


上述步骤完成之后，打成 war 包放在 tomcat 下，启动 tomcat，即可完成服务的发布。查看服务是否发布成功可以去服务治理页面 <http://ops.jm.taobao.net/hsfops> 查询，在服务提供者里看看有没有自己的 IP。

发布者 java:消费者 跨语言消费者				
机器	超时	序列化方式	组别	全部展开
10.68.175.221:12200	3000		[HSF]	展开URL

消费 HSF 服务

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 工程完成这一实例。

第一： 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="HelloWorldConsumer"
class="com.taobao.hsf.app.spring.util.HSFSpringConsumerBean" init-method="init">
    <property name="interfaceName">
        <value>com.taobao.lanshan.service.HelloWorldService</value>
    </property>
    <property name="version">
        <value>1.0.0.daily</value>
    </property>
</bean>
```

配置文件中指定了该服务需要调用的接口名以及版本，这需要与上一节中的服务提供定义相一致。

第二： 编写一个简单的 servlet， 用于访问上一节中的 HSF 服务。

```

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("HelloWorldConsumer");
        PrintWriter out = resp.getWriter();
        out.println(helloWorldService.sayHello("lanshan"));
        return;
    }
}

```

第四：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.lanshan.servlet.HelloWorldServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>

```

第五：重新打包，启动 tomcat，访问对应的 url，我这里是
<http://localhost:8080/HSFSample/HelloWorld>，可以看到执行的结果：



The screenshot shows a web browser window with the address bar containing 'localhost:8080/HSFSample/HelloWorld'. Below the address bar, the text 'hello : lanshan' is displayed on the page.

说明服务调用成功。

HSF 原理上不需要依赖任何 jar 包，如果 IDE 在配置 Spring 文件时报错，可以 provided 级依赖 hsf.app.spring，版本不限。

HSF 的其他启动方式

Light-Api

LightApi 的思想是把 HSF 的服务看做一种资源，由应用自由发挥。

1.在 pom 中加入如下依赖

```
<dependency>
    <groupId>com.taobao.hsf</groupId>
    <artifactId>LightApi</artifactId>
    <version>1.0.2</version>
</dependency>
```

2.定义一个静态变量 ServiceFactory，并初始化需要发布、消费的服务。

调用了 service()和 version()后即可发布、消费服务，其余的方法为可选，其余方法见类的方法说明。

```
public class ServicesContainer{
    public static ServiceFactory factory = ServiceFactory.g
etInstance();//必须为 static，初始化 HSF 资源
    static{
        initServices();
    }

    public static void initServices() {
        factory .provider("hello")//参数是一个标识，初始化后，下
次只需调用 provider("hello")即可拿出对应服务
        .service("com.taobao.lanshan.service.HelloWo
rldService")//服务名
        .version("1.0.0.daily")//版本号
        .group("light")//组别
        // .writeMode("unit",0) //设置单元化服务的 writeMo
de,非 unit 服务第二个参数随意
        .impl(new HelloWorldServiceImpl())//对应的服务
实现
        .publish();//发布服务，至少要调用 service()和 ver
sion()才可以发布服务

        factory .consumer("hello")//参数是一个标识，初始化后，下
次只需调用 consume("hello")即可直接拿出对应服务
        .service("com.taobao.lanshan.service.HelloWo
rldService")//服务名
```

```

        .version("1.0.0.daily") //版本号
        .group("light") //组别
        .subscribe(); //消费服务并获得服务的接口, 至少要调用
service() 和 version() 才可以消费服务
    }
    public static void main(String[] args){
        HelloWorldService helloWorldService = (HelloWorldService)factory.consumer("hello").subscribe(); //用 ID 取出对应服务, subscribe() 方法返回对应的接口
        ServiceUtil.waitForServiceReady(helloWorldService);
        System.out.println(helloWorldService.sayHello("lanshan1"));
    }
}

```

- ServiceFactory 是单例的, 定义为 static 变量是为了优先校验是否使用了 Web 容器。
- ServiceFactory 会缓存所有 provider 和 consumer, 建议在一个静态块里直接初始化所有服务。
- provider("hello")和 consumer("hello")里的参数, 是一个标识。可以在 main 方法中看出, 要使用接口时, 直接通过这个标识拿到 consumer 调用 subscribe()方法。
- service()和 version()为必调方法, 不调用这两个方法无法发布、消费服务, 其余的可选方法参见类的方法说明。

在 Web 中使用

和 Main 方法中使用没有区别, 可以自己封装一个 ServicesContainer, 在初始化的时候去发布、消费服务。

可以再封装一个方法来获取服务。

```

public Object getConsumeService(String desc){
    //这个方法调用之前要保证被初始化过。
    return factory.consumer(desc).subscribe();
}

```

HSF-standalone

hsf-standalone 是针对一些不使用 Web 容器的应用, 推出的产品, 它可以一行代码启动 HSF, 而且不需要存活在 Web 容器中。。

首先在 pom 依赖中加入 hsf-standalone 的依赖, 目前推荐版本是 2.0.7 (最近稳定版本可以参考 gitlab 上的 HSF-wiki)。

```

<dependency>
  <groupId>com.taobao.hsf</groupId>
  <artifactId>hsf-standalone</artifactId>
  <version>2.0.7</version>
</dependency>

```

然后和之前配置 ConsumerBean 一样，在 Spring 配置文件中加入 Consumer 的配置。

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
    <bean id="uicReadService"
        class="com.taobao.hsf.app.spring.util.HSFSpringConsumerBean"
        init-method="init">
        <property name="interfaceName"
            value="com.taobao.uic.common.service.userinfo.UicReadService"/>
        <property name="version" value="1.0.0.daily"/>
    </bean>
</beans>
```

后面就可以直接在 Main 中启动 HSF 了，这里启动 HSF，靠的就是 hsf-standalone 中的 HSEasyStarter.start("release_path","version")来启动 HSF，**注意，这一行启动 HSF 的代码一定要在所有代码之前，包括对日志的操作**。具体的启动例子如下：

```
public static void main(String[] args){
    // Sar 包自动下载、解压缩到 Temp 目录下的 2.1.0.7/taobao-hsf.sar
    // 比如/home/admin/appname/ 会下载到
    /home/admin/appname/2.1.0.7/taobao-hsf.sar/
    // 如果下载不可用，可以手动放置 sar 包。
    HSFEasyStarter.start("d:/tmp/", "2.1.0.7");

    String springResourcePath = "spring-hsf-uic-consumer.xml";
    ClassPathXmlApplicationContext ctx = new
    ClassPathXmlApplicationContext(springResourcePath);
    UicReadService uicReadService = (UicReadService)
    ctx.getBean("uicReadService");

    // 下面这句是等该服务的地址，这句非必须的，如果不加这句，调得太快，可能会出现找不到地址
    的异常。（简单的可以 sleep 一会，3s 差不多够了）
    ServiceUtil.waitForServiceReady(uicReadService);

    BaseUserDO user = uicReadService.getBaseUserByUserId(10000L,
    "detail").getModule();
    System.out.println("user[id:10000L] nick:" + user.getNick());
}
```

HSF Bean 配置详解

HSFSpringProviderBean

HSFSpringProviderBean 是提供服务发布功能的 Spring Bean，它有一系列属性可以配置，用于控制服务的各种配置信息。

红色配置为必选配置，蓝色配置为可选配置。

serviceInterface: 定义了对外提供服务的接口，**必须配置**（与版本号一起作为服务名）。

```
<property name="serviceInterface">
    <value>com.taobao.hsf.lanshan.HelloWorldService/>
</property>
```

target: 为对应服务的具体实现，也是一个 Spring Bean，**必须配置**。

```
<property name="target">
    <ref bean="HelloWorldServiceImpl"/>
</property>
```

serviceVersion: 为服务的版本号，可以利用版本号来区分服务（与接口名一起作为服务名），**必须配置**。

```
<property name="serviceVersion">
    <value>1.0.0.daily</value>
</property>
```

serviceGroup: 为服务的组别，**注意只有同一个组别的服务才可以互相调用**。

```
<property name="serviceGroup">
    <value>HSF</value>
</property>
```

serviceName: 用于方便管理的服务名称，并非服务的 dataId，可选配置，推荐使用，默认为 null。

```
<property name="serviceName">
    <value>HelloWorldService</value>
</property>
```

ServiceDesc: 用于方便管理的服务描述信息，可选配置，默认值为 null。

```
<property name="serviceDesc">
    <value>HelloWorldService provided by HSF</value>
</property>
```

clientTimeout: 对这个服务的所有方法进行统一的超时时间设置，但如果客户端配置了超时时间，则会优先使用客户端超时时间。

```
<property name="clientTimeout">
    <value>3000</value>
</property>
```

methodSpecial: 针对某一个或者多个方法，做单独的超时时间处理，优先级高于 clientTimeout，但低于客户端的 methodSpecial。

```
<property name="methodSpecials">
    <list>
        <bean class="com.taobao.hsf.model.metadata.MethodSpecial">
            <property name="methodName" value="sum" />
            <property name="clientTimeout" value="2000" />
        </bean>
    </list>
</property>
```

serializeType: 序列化的类型，默认为 Hessian，Hessian 的效率比 Java 序列化要高，但使用的时候有很多小问题，可以在这里看看：

http://gitlab.alibaba-inc.com/middleware/hsf2-0/wikis/hessin_problem

碰到疑难的序列化错误，可以考虑使用 Java。

```
<property name="serializeType">
    <value>hessian</value>
</property>
```

HSFSpringConsumerBean

HSFSpringConsumerBean 是消费者用于消费服务的 Spring Bean，它有一系列属性可以配置，用于控制服务的各种配置信息。

红色配置为必选配置，蓝色配置为可选配置。

interfaceName: 定义了对外提供服务的接口，**必须配置**（与版本号一起作为服务名）。

```
<property name="serviceInterface">
    <value>com.taobao.hsf.lanshan.HelloWorldService/>
</property>
```

version: 为服务的版本号，可以利用版本号来区分服务（与接口名一起作为服务名），**必须配置**。

```
<property name="version">
    <value>1.0.0.daily</value>
</property>
```


group: 为服务的组别，注意只有同一个组别的服务才可以互相调用。

```
<property name="group">
    <value>HSF</value>
</property>
```

methodSpecial: 针对某一个或者多个方法，做单独的超时时间处理，优先级高于 clientTimeout，同时高于服务端的 methodSpecial。

```
<property name="methodSpecials">
    <list>
        <bean class="com.taobao.hsf.model.metadata.MethodSpecial">
            <property name="methodName" value="sum" />
            <property name="clientTimeout" value="2000" />
        </bean>
    </list>
</property>
```

clientTimeout: 客户端统一设置接口中所有方法的超时时间(单位 ms)，超时设置优先级由高到低是：客户端 MethodSpecial，客户端接口级别，服务端 MethodSpecial，服务端接口级别。

```
<property name="clientTimeout">
    <value>3000</value>
</property>
```

target: 指定一个 IP 来进行调用，主要用于单元测试环境和 hsf.run.mode=0 的开发环境中，在运行环境下，此属性将无效，而是采用配置中心推送回来的目标服务地址信息。

```
<property name="target">
    <value>10.1.6.57:12200?_TIMEOUT=1000</value>
</property>
```

invokeContextThreadLocal: 设置传递上下文的方式为 ThreadLocal 对象方式，即在发起调用时通过一个 ThreadLocal 对象来设置上下文对象，接口方式和 ThreadLocal 对象方式选其一即可

```
<property name="invokeContextThreadLocal" ref="invokeContextThreadLocal"
/>
```

HSF 规则使用介绍

路由规则

HSF 路由规则支持接口路由，方法路由，参数路由，采用 Groovy 脚本作为路由规则设置内容。

路由规则是通过 Diamond Server 进行推送的，用户想配置路由规则需要去 diamond-ops，选择对应的环境进行配置。

对应的 dataId 为： 服务名（带版本号）.RULES，详细配置说明见后续章节。

配置了路由规则后，HSF 会根据用户配置的路由规则，将客户端的流量引向对应的机器，路由规则分为三级：

接口路由

接口路由针对服务粒度，配置了接口路由后，调用此服务（接口）的客户端，只会访问所配置的 IP。

如果想对服务 A 做接口级别路由，其中提供 A 服务的机器有：192.168.1.2、192.168.1.3，希望调服务 A 的消费者，全部路由到 192.168.1.3，向配置中心推送如下规则：

```
Groovy_v200907@package hqm.test.groovy
public class RoutingRule{

    Map<String, List<String>> routingRuleMap(){
        return [
            "G1":["192.168.1.3:*"]
        ]
    }

    String interfaceRoutingRule(){
        return "G1";
    }

}
```

方法路由

方法路由是针对方法粒度，针对不同的方法，路由到不同的机器去。比如用户需要读写分离，现有 2 台机器提供同一个服务的读和写方法，用户想 1 台机器用做读，1 台用做写服务 向配置中心推送如下规则：

```

Groovy_v200907@package hqm.test.groovy
public class RoutingRule{

    Map<String, List<String>> routingRuleMap(){
        return [
            "read_method_address_filter_Key":["192.168.1.3:*"],
            "write_method_address_filter_Key":["192.168.1.2:*"]
        ]
    }

    String methodRoutingRule(String methodName, String[] paramTypeStrs){
        if(methodName.matches("get.*"))
            return "read_method_address_filter_Key";
        else if (methodName.matches("put.*"))
            return "write_method_address_filter_Key";
        else
            return null;
    }

}

```

参数路由

参数路由针对参数粒度,配置后 HSF 会根据对应的参数规则,讲客户端导向对应的机器。比如用户想对服务 A 做参数级别路由控制,A 服务的地址列表为: 192.168.1.2, 192.168.1.3。如果方法中某个参数大于一定规则时(比如参数传入整数 大于 10, 到某几台机器, 小于 10 到另几台机器), 路由到不同组机器上。 向配置中心推送如下规则: (对参数路由中方法 argsRoutingRule, 返回值最好不为 null,如果为 null, 参数路由会被系统忽略!)

```

Groovy_v200907@package hqm.test.groovy
public class RoutingRule{

    Map<String, List<String>> routingRuleMap(){
        return [
            "BSeller_address_filter_Key":["192.168.1.2"],
            "ASeller_address_filter_Key":["192.168.1.3"]
        ]
    }

}

```

```

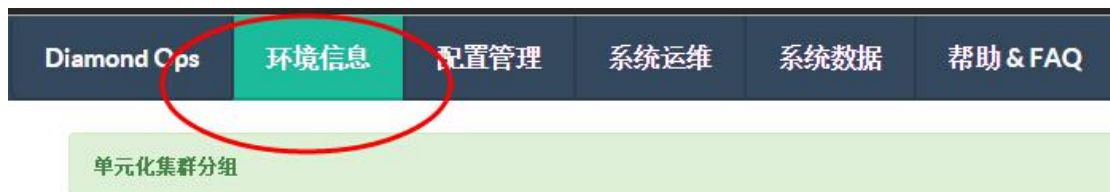
Object argsRoutingRule(String methodName, String[] paramTypeStrs){
    if ( methodName.startsWith("routeByBSeller") ) {
        return {
            Object[] args->
                if(args[0]>250)
                    return "BSeller_address_filter_Key";
                else
                    return "ASeller_address_filter_Key";
        }
    } else if (methodName.equals("aSpecialMethod") ) {
        return {
            Object[] args->
                if(args.length > 1)
                    return "specialMethod_address_filter_Key1";
                else
                    return null;
        }
    }

    return null;
}
}

```

如何配置路由规则

先去 diamond-ops<http://ops.jm.taobao.org:9999/diamond-ops/>，在环境信息那里找到对应的环境：



然后点击右侧的“新增配置”

Well done! unit:线上中心,新配置项发布 (仅允许发布新的配置, 发布已存在的配置会系统会提示失败)

dataId	com.taobao.lanshan.service.HelloWorld:1.0.0.daily.RULES 服务名:RULES
group	HSF 需要与服务所在分组一致
unit	线上中心
content	<pre>Groovy_v200907@package hqm.test.groovy public class RoutingRule{ Map<String, List<String>> routingRuleMap(){ return ["G1":["10.68.137.221:"]] } String interfaceRoutingRule(){ return "G1"; } }</pre> 路由规则内容

线上中心

搜索配置

新增配置

批量发布

配置历史

多环境管理

多单元发布

点我设置

点击发布, 即可发布路由规则。

想要清空路由规则, 可以直接删除对应的 `dataId`, 也可以直接发布一个保留到@符号的规则, 即: `Groovy_v200907@`。

归组规则

HSF GroupingRule 是用于对发布了同一 HSF 服务的所有机器进行统一归组的规则。GroupingRule 作用于 HSF 服务提供端, 能够在 HSF 服务的发布期生效, 使当前机器发布的该 HSF 服务自动归入特定的分组内。这样, 不同分组中的 HSF 服务实例就组成了以组为单位的集群, 针对某些特定的机器提供服务。

发布归组规则与路由规则方法一致:

1. Grouping Rule 使用 Diamond 进行推送, 该规则对一个应用是惟一的, 即一个应用上只能配置一份 Grouping Rule。
2. 发布到 Diamond 时, group 为: HSF, dataId 为: 应用名称.GROUPINGRULE, 如果应用名称为 hfsample, 则 Grouping Rule 的 dataId 为: hfsample.GROUPINGRULE。

需要注意的是:

- 1) HSF 应用中, 要求启动脚本中配置-Dproject.name 参数来指定当前应用的名称。
- 2) 规则中不要使用中文

```
groupingRule@
<rules>
  <rule>
    <services>
      <service>com.taobao.hsf.sample.service.HelloWorldService:1.0.0.lanshan
    </service>
    </services>
    <ips>
      <ip>10.232.36.79</ip>
      <ip>10.232.36.*</ip>
    </ips>
    <group>HSF</group>
  </rule>
  <rule>
    <services>
      <service>com.taobao.hsf.sample.service.HelloWorldService:1.0.0.lanshan
    </service>
    </services>
    <ips>
      <ip>10.232.68.13?</ip>
    </ips>
    <group>NOHSF</group>
  </rule>
</rules>
```

这样配置之后，符合 10.232.68.13? 正则式的机器会被归入 NOHSF 分组，这样只有分组同样是 NOHSF 的客户端，才能调用到它们。

同机房优先规则

HSF 机房流量控制规则用于对跨机房间的 HSF 调用流量进行规划控制，能够保证 HSF 服务消费者在请求 HSF 服务时，优先选择与服务消费者同机房的服务提供者。

发布方式是和路由规则发布在一个 diamond 的 dataId 下。

在发布数据页面，以 com.taobao.hsf.sample.service.HelloWorldService:1.0.0 服务为例

DataId: com.taobao.hsf.sample.service.HelloWorldService:1.0.0.RULES

Group: HSF(需要与服务分组相同)

发布规则内容如下：

```

flowControl@
<flowControl>
  <localPreferredSwitch>on</localPreferredSwitch>
  <threshold>0.2</threshold>
  <exclusions></exclusions>
</flowControl>

```

规则属性

1. localPreferredSwitch: on|off
2. threshold: float 值
3. exclusions: 如果期望该规则只对一部分机器生效，可以使用这一属性配置需要排除的 IP，比如：172.24.*，将表示该规则不会应用于所有 172.24 打头的 IP

生效阈值的计算方法：服务可用比例=本机房可用机器数量/所有服务机器数量 当服务可用比例 \geq threshold 时，启用本地机房优先策略 当服务可用比例 $<$ threshold 时，本地机房优先策略关闭，服务仍然采用随机调用的方式

权重规则

权重规则适用如下场景：用户不想对流量进行硬性导向，但倾向于将流量多往某几台机器上引导，这时候就可以将这几台机器的权重规则配置大一点。

配置权重规则有两种方式，第一是配置在路由规则之后，举例说明 服务 A 有 2 台机器提供服务，客户端调用该服务时，选到 192.168.1.3 的概率为 1/4，选到 192.168.1.2 的概率为 3/4。

```

Groovy_v200907@package hqm.test.groovy
public class RoutingRule{

    Map<String, List<String>> routingRuleMap(){
        return [
            "interface_level_address_filter_Key":["192.168.1.3::1","192.168.1.2::3"]
        ]
    }

    String interfaceRoutingRule(){
        return "interface_level_address_filter_Key";
    }
}

```

还有一种配置方式，是单独配置权重规则，同样也是在 diamond-ops 上配置

dataId: 服务名（带版本号）.WEIGHTINGRULE

group: 与服务所在组别一致。

Content 的例子：

```
weightingRule@
<rules>
  <rule>
    <ips>
      <ip>10.232.36.79</ip>
    </ips>
    <weight>2</weight>
  </rule>
</rules>
```

权重规则详解：

com.taobao.hsf.sample.service.HelloWorldService:1.0.0 对应的服务提供者有 两台， 那么 10.232.36.79 这台被调用的概率就有 $2 \times 1 / (2 \times 1 + 1)$ ，也就是三分之二的概率被调用；流量从二分之一提升到三分之二。

HSF 异步调用

对于客户端来说，并不是所有的 HSF 服务都是需要同步等待服务端返回结果的，对于这些服务，HSF 提供异步调用的形式，让客户端不必同步阻塞在 HSF 操作上。

异步调用在发起调用时，HSF service 的调用结果都是返回值的默认值，如返回类型是 int，则会返回 0，返回类型是 Object，则会返回 null。而真正的结果，是在 HSFResponseFuture 或者回调函数中获得的。

Future 调用

HSF 的 Future 调用类似于 JDK 的 future，通过 HSFResponseFuture.getResponse(int timeout) 拿到调用结果。

首先要在 HSFSpringConsumerBean 中配置 asyncallMethods 参数，如下：

```
<beans>
...
<bean name="helloService"
      class="com.taobao.hsf.app.spring.util.HSFSpringConsumerBean"
      init-method="init">
  <property name="interfaceName"
    value="com.taobao.hsf.sample.service.HelloService" />
  <property name="version" value="1.0.0.daily" />
  <property name="asyncallMethods">
    <list>
      <!--future 的含义为通过 Future 的方式去获取请求执行的结果，例如先调用下远程的接口，接着继续做别的事情，然后再通过 Future 来获取结果-->
      <value>name:sayHello;type:future</value>
    </list>
  </property>
</bean>
...
</beans>
```

调用的方法如下：

```
// 发起调用
String helloString = helloService.sayHello("lanshan"); //这里返回的 helloString 其实是 null
//do something else
//获取结果
helloString = HSFResponseFuture.getResponse(-1); //这里才是真正的返回结果
System.out.println(helloString);
```

Callback 调用

Callback 顾名思义，HSF 提供一种回调机制，当这个 HSF 服务消费完毕拿到结果后，会回调用户的接口，需要用户实现 `HSFResponseCallback` 接口。

首先用户需要依赖 `hsf.app.spring` 这个 jar 包，**scope 记得写成 provided**。

```
<dependency>
  <groupId>com.taobao.hsf</groupId>
  <artifactId>hsf.app.spring</artifactId>
  <version>2.0.1.7</version>
  <scope>provided</scope>
</dependency>
```

在配置 `ConsumerBean` 的时候，和 `Future` 调用一样，要写明异步方法。

```
<beans>
  <bean name="helloService"
class="com.taobao.hsf.app.spring.util.HSFSpringConsumerBean"
  init-method="init">
    <property name="interfaceName"
value="com.taobao.lanshan.service.HelloWorldService" />
    <property name="version" value="1.0.0.daily" />
    <property name="asynccallMethods">
      <list>
        <value>name:sayHello;type:callback;listener:com.taobao.lan
shan.service.MyCallBackListener</value>
      </list>
    </property>
  </bean>
</beans>
```

与普通的服务调用不同的是，需要用户自己实现 `HSFResponseCallback` 接口，真正执行的逻辑是在 `onAppResponse()` 方法里。

```

public class MyCallBackListener implements HSFResponseCallback {
    @Override
    public void onAppException(Throwable t) {
        //收到服务端异常
    }

    @Override
    public void onAppResponse(Object appResponse) {
        System.out.println("callback : " + appResponse);
    }

    @Override
    public void onHSFException(HSFException hsfEx) {
        //收到 HSF 层异常
    }
}

```

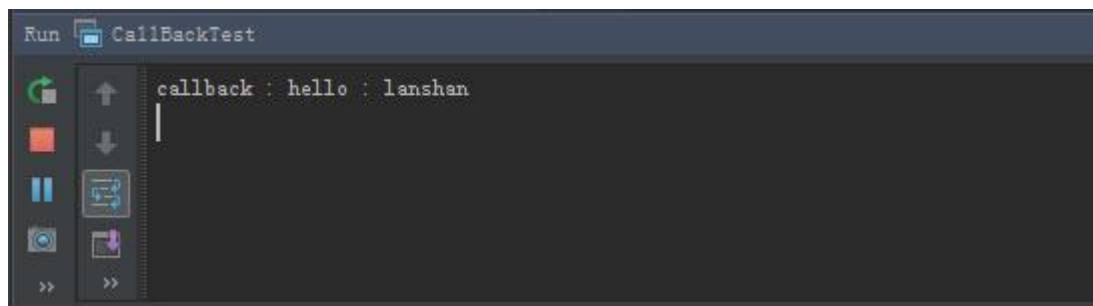
调用代码如下：

```

public class CallBackTest {
    public static void main(String[] args) throws InterruptedException {
        HSFEasyStarter.start("d:/tmp", "2.1.0.7");
        ClassPathXmlApplicationContext ctx = new
        ClassPathXmlApplicationContext("consumer.xml");
        HelloWorldService testParamService =
        (HelloWorldService)ctx.getBean("ConsumerTestService");
        Thread.sleep(9000);
        testParamService.sayHello("lanshan");
    }
}

```

到这里你应该能猜到执行的结果是什么了，如下：



使用 **Callback** 方式调用 **HSF** 服务，有以下几点需要注意：

- 1) 由于只用方法名字来标识方法，所以并不区分重载的方法。同名的方法都会被设置为同样的调用方式。
- 2) 回调函数是由 **io** 线程来调用的，不要在拿到结果后做费时的操作。
- 3) 不能在 **onReponse** 里边再发起 **hsf** 调用，目前这种做法可能导致 **io** 线程挂起，无法恢复。

HSF 2.X 新特性

Pandora 控制台

与 HSF 1.X 不同的是，HSF 2.X 的类加载和隔离体系，不再是 OSGI 的 Bundle 模式，而是托管与 Pandora，对于 HSF 来说，Pandora 是一个管理 HSF 生命周期的容器。

可以通过执行 telnet localhost 12201，进入 Pandora 的控制台，如下图所示：

```
[huangsheng.hs@appcenter010176118195.n.et2 /home/huangsheng.hs]
$telnet localhost 12201
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^['.

welcome to Pandora Console.
Login Success:   Wed Sep 17 11:11:03 CST 2014 from /127.0.0.1:22791

Pandora Host:    10.176.118.195
Pandora Version: 2.0.5.2(2014-04-29 16:55:00)
QOS Version:     2.0.5.2

ll
Total Modules:11
DeployTime      V      P      S      Name
Thu Sep 11 13:32:23 CST 2014 1.1.0-SNAPSHOT 20    1    spas-sdk-client
Thu Sep 11 13:32:23 CST 2014 1.0.1      50    1    monitor
Thu Sep 11 13:32:23 CST 2014 1.3.1      100   1    eagleeye-core
Thu Sep 11 13:32:23 CST 2014 3.6.9      150   1    diamond-client
Thu Sep 11 13:32:23 CST 2014 1.1.0-SNAPSHOT 160   1    spas-sdk-service
Thu Sep 11 13:32:23 CST 2014 1.6.6.2    300   1    config-client
Thu Sep 11 13:32:23 CST 2014 1.0.11     550   1    unitrouter
Thu Sep 11 13:32:23 CST 2014 1.8.19.14  600   1    notify-tr-client-plugin
Thu Sep 11 13:32:23 CST 2014 1.8.19.8   999   1    hsf-notify-client
Thu Sep 11 13:32:23 CST 2014 2.1.0.7    1000  1    hsf
Thu Sep 11 13:32:23 CST 2014 2.0.2      2000  1    pandora.qos.service

pandora>cd hsf
```

可以看到，进入控制台后，执行“ll”操作，即可看到当前 Pandora 容器中的所有插件，HSF 就是其中的一个插件，我们暂时不关心别的插件，执行“cd hsf”进入 hsf 插件命令行。

在 hsf 命令行下，执行“ll”操作，即可看到作为 Provider/Consumer 的服务列表，通过这里，可以观察 HSF 服务是否发布/消费成功。

```
pandora>cd hsf
hsf
hsf>ll
As Provider side:
com.yunos.lifefservice.hsf.client.Card:1.0.0.daily
com.yunos.lifefservice.hsf.client.Service:1.0.0.daily
com.yunos.lifefservice.hsf.client.V2Service:1.0.0.daily
com.yunos.lifefservice.hsf.client.V2Card:1.0.0.daily
com.yunos.lifefservice.hsf.client.V2Tip:1.0.0.daily
com.taobao.mtop.common.GetResponseDesc:1.0.0.lifefservice
com.yunos.lifefservice.hsf.client.Account:1.0.0.daily
com.yunos.lifefservice.hsf.client.Tip:1.0.0.daily
com.yunos.lifefservice.hsf.client.V2Account:1.0.0.daily

As Subscriber side:
com.taobao.uic.common.service.userinfo.UicReadService:1.0.0.daily
com.taobao.uic.common.service.userinfo.UicPaymentAccountService:1.0.0.daily
com.laiwang.pp.service.PubMessageReadService:1.0.0.daily
com.taobao.film.api.RegionAPI:1.0.0.daily
com.taobao.lottery.client.hsf.OrderService:1.0.0.daily
```

其实还有很多 Pandora 命令，比如可以直接查看某个消费的服务的地址，命令为 `getAddressInfo serviceName`，如下图：

```
hsf>getAddressInfo com.yunos.lifefservice.hsf.client.card:1.0.0.daily
As Subscriber side:
hsf>getAddressInfo com.taobao.film.api.ShowAPI:1.0.0.daily
As Subscriber side:com.taobao.film.api.ShowAPI:1.0.0.daily:taobao-film:getAddressInfo all:10.101.104.15:
LETIMEOUT=10&p=4&SERIALIZETYPE=1
10.125.0.249:12200?v=2.0&_TIMEOUT=5000&_IDLETIMEOUT=10&p=4&SERIALIZETYPE=1

local:10.101.104.152:12200?v=2.0&_TIMEOUT=5000&_IDLETIMEOUT=10&p=4&SERIALIZETYPE=1
10.125.0.249:12200?v=2.0&_TIMEOUT=5000&_IDLETIMEOUT=10&p=4&SERIALIZETYPE=1

available:10.101.104.152:12200?v=2.0&_TIMEOUT=5000&_IDLETIMEOUT=10&p=4&SERIALIZETYPE=1
10.125.0.249:12200?v=2.0&_TIMEOUT=5000&_IDLETIMEOUT=10&p=4&SERIALIZETYPE=1

invalid:
hsf>
```

更多的 pandora 命令，可以参见这里：

<http://gitlab.alibaba-inc.com/middleware/hsf2-0/wikis/pandoracommond>

泛化调用

HSF 2.X 支持一种特殊的调用方式——泛化调用。何谓泛化调用呢？就是客户端不需要依赖服务端的二方包，也就是本地不需要持有对应的接口，即可进行调用。对于一些平台化的产品，减轻自身重量是非常有帮助的。

要使用泛化调用，要在配置 ConsumerBean 的时候加上泛化调用的标识：

```
<property name="generic">
    <value>true</value>
</property>
```

这里用一个简单的调用例子来说明泛化调用：

```
GenericService svc = (GenericService) consumerBean.getObject();
System.out.println(svc.$invoke("testRetPrimitive", new String[] { "long" },
    new Object[] { 11 }));
```

可以看到泛化调用有 3 个参数：

- 1) 第一个参数是方法名。
- 2) 第二个参数是参数类型。
- 3) 第三个参数是参数的值。

方法名和参数类型用字符串是很直接的，但是参数的值会复杂些，诸如 Number，String，Date 这些会简单点，直接依赖二方的 Pojo bean 如何表达？一般情况下有两种选择来描述，一是使用结构化更清晰的 Json，二是 key-value 的 Map。我们选择的是简化了接近 Json 的方式来描述参数值。

调用方在不确定格式的情况下写个单元测试，测试依赖二方包，使用 HSF 提供的工具类 `com.taobao.hsf.util.PojoUtils`，使用 `generalize` 方法来生产一个 `pojobean` 的字符串描述格式。

下面给出一个泛化调用的例子，涵盖了大部分参数类型：

```
System.out.println(svc.$invoke("testRetPrimitive", new String[]
{ "long" }, new Object[] { 11 }));
// 传入数组类型，返回数组类型
System.out.println(svc.$invoke("testRetArray", new String[]
{ "[Ljava.lang.String;" },
new Object[] { new String[] { "haha", "hehe", "hoho" } }));
// 传入 map，返回 map
Map<String, String> map = new HashMap<String, String>();
map.put("hello", "world");
System.out.println(svc.$invoke("testRetMap", new String[]
{ "java.util.Map" }, new Object[] { map }));
// 传入 jdk 内置类型对象，返回内置类型对象
System.out.println(svc.$invoke("testRetObject", new String[]
{ Integer.class.getName() },
new Object[] { 7 }));
// 传入业务自定义 pojo，返回业务自定义 pojo
ParamClass param = new ParamClass();
param.setI(11);
param.setName("Miles");
Object comp = PojoUtils.generalize(param);
System.out.println(svc.$invoke("testRetPojo",
new String[] { "com.alibaba.pfmiles.testhsf.pojo.ParamClass" }, new
Object[] { comp }));
```

感谢 miles123456 提供完整的测试程序，这里有他的 demo，大家可以参考，下载链接：

[http://hsf.taobao.net/hsfversion/downloadfile/testhsf.t
ar.gz](http://hsf.taobao.net/hsfversion/downloadfile/testhsf.tar.gz)

优雅上下线

问题的起源:

- 一直以来，核心服务比如 Tradeplatform 重启（分为关闭和启动两个阶段）的时候，关闭阶段会导致周边系统大量调用失败；以至于交易有明显的波动。这些调用失败的请求大多数是因为服务端还在处理而没有返回 response 到客户端；另外有一部分在网络发送途中或者客户端准备发送中。希望通过服务上下线，能够优雅的解决上述场景下的问题。
- 与服务下线对应的一个问题是服务上线。当重启过程中，启动阶段，服务还没有完全初始化成功，就有请求进来了，这时候的处理结果就是失败。安全快速上线就是要解决上述问题。

所以 HSF 针对这种情况，加入了优雅上下线的功能。

优雅上线:

- 1) 在启动参数中加入 `-Dhsf.publish.delayed=true` 来让服务默认不注册到注册中心。
- 2) 需要 PE 配合，在启动 app server(tomcat)前不启动 web server (apache/nginx)，等应用初始化完成后，执行 `curl localhost:12201/hsf/online?k=hsf` 使服务注册到注册中心。
- 3) 通过基于 HTTP 协议发送 `curl localhost:12201/hsf/status` 命令来检测服务是否 successfully 初始化，成功之后再启动 web server (apache/nginx)。

这样，就可以保证流量进来的时候，应用已经具备提供服务的能力。

优雅下线:

- 1) PE 需要配合，在关闭 server 时候，先基于 HTTP 协议发送 `curl localhost:12201/hsf/offline?k=hsf` 命令来下线服务
- 2) 等待 30 秒后，再执行 `shutdown jvm` 的操作。
- 3) 服务下线在 `shutdownhook` 中加入注销服务后等待指定可配置的时间；默认是 10000 毫秒，可以通过 `-Dhsf.shuthook.wait` 来配置。

这样，就可以保证可以处理完所有请求后才关闭应用。

更多的 HSF 2.X 功能

这里只是介绍了一些基本的、常用的 HSF 功能，HSF 2.X 还提供很多别的功能，比如安全校验、白名单、限流、动态设置超时时间、调用上下文等等等等。

如果了解更多的 HSF 功能和用法，请看 gitlab 上 HSF 的 wiki。

<http://gitlab.alibaba-inc.com/middleware/hsf2-0/wikis/home>

岚衫

2014.9.17