企业级分布式应用服务 EDAS

开发指南

开发指南

开发工具准备

Ali-Tomcat 为 EDAS 中的服务运行时必须依赖的一个容器，主要集成了服务的发布、订阅、调用链追踪等一系列的核心功能，无论是开发环境还是运行时，均必须将应用程序发布在该容器中。

Ali-Tomcat 的安装步骤如下：

注意：请使用 JDK 1.7及以上版本。

下载 Ali-Tomcat，保存后解压至相应的目录（如：d:\work\tomcat\）。

下载 Pandora 容器，保存后将内容解压至上述保存的 Ali-Tomcat 的 deploy 目录(d:\work\tomcat\deploy)下。

查看 Pandora 容器的目录结构。

Linux 系统中，在相应路径下执行 **tree** -L 2 deploy/ 命令查看目录结构。

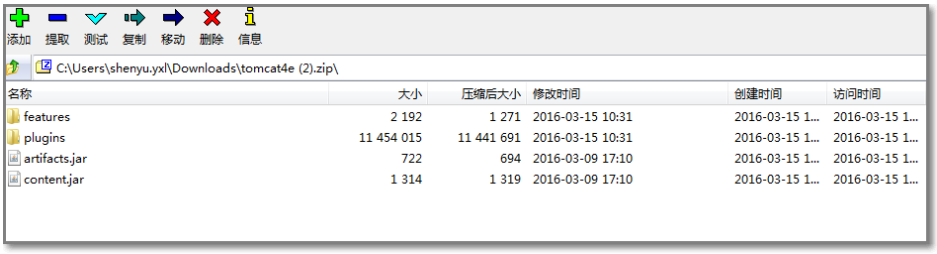
```
d:\work\tomcat >  tree -L 2 deploy/
deploy/
└── taobao-hsf.sar
├── META-INF
├── lib
├── log.properties
├── plugins
├── sharedlib
└── version.properties
```

Windows中，直接进入相应路径进行查看。

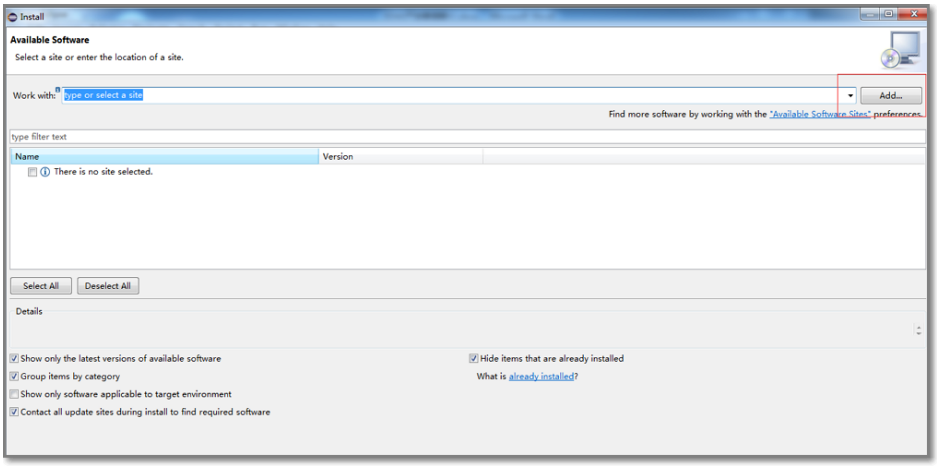
配置 Eclipse 需要下载 Tomcat4E 插件，并存放在 安装 Ali-Tomcat 时Pandora 容器的保存路径中，配置之后开发者可以直接在 Eclipse 中发布、调试本地代码。具体步骤如下：

下载 Tomcat4E 插件，并解压至本地（如：d:\work\tomcat4e\）。

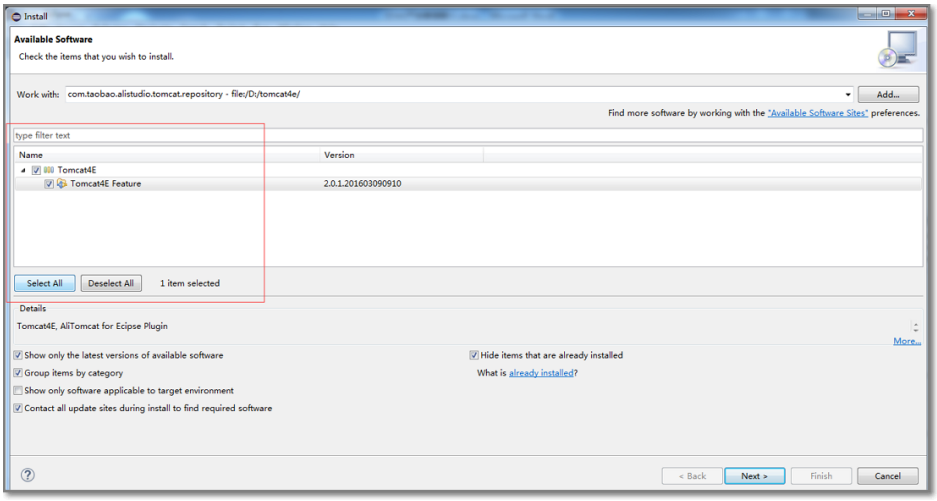
压缩包内容如下：



打开 Eclipse，在菜单栏中选择 Help > Install New Software，弹出如下对话框。依次单击 Add > Local，选中解压好的 Tomcat4E 插件目录（d:\work\tomcat4e\）>，单击 OK。

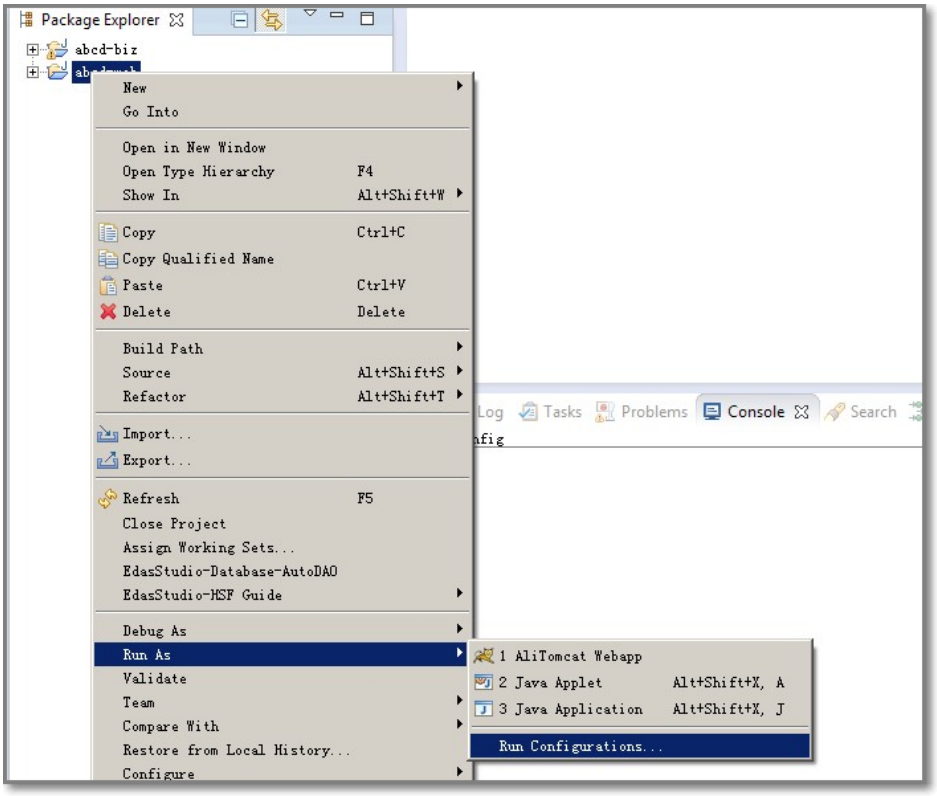


单击 **Select All** > **Next** 按钮即完成插件安装。



重启 Eclipse。

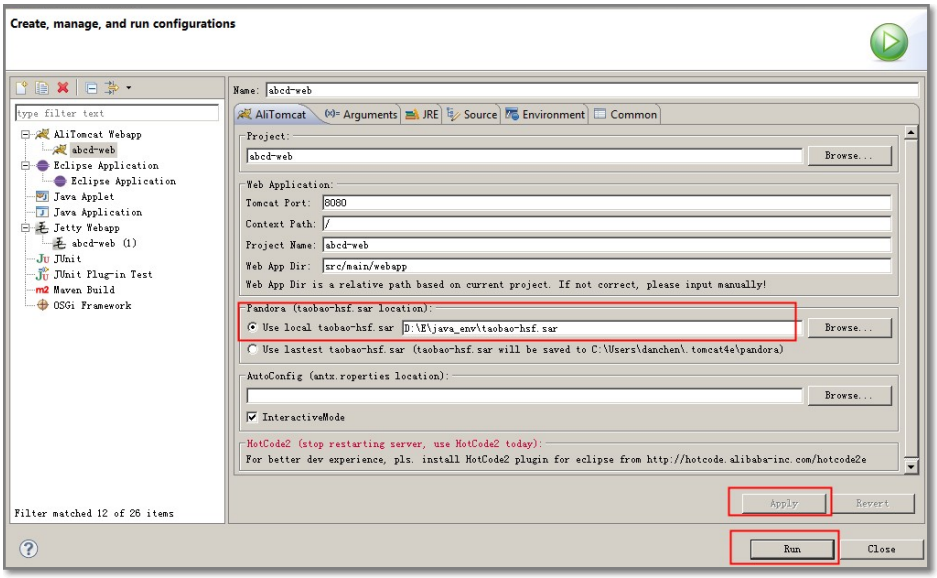
要使 Tomcat4E 生效，需要针对相应的 Eclipse 工程进行单独设置。右键单击相应的 Eclipse 工程，在弹出的菜单中选择 **Run As** > **Run Configurations**。



选择左侧导航选项中的 **AliTomcat Webapp**，单击上方的 **新增启动配置**。

在弹出的界面中，选择 **AliTomcat** 选项卡，在 Pandora (taobao-hsf.sar location) 区域，选择

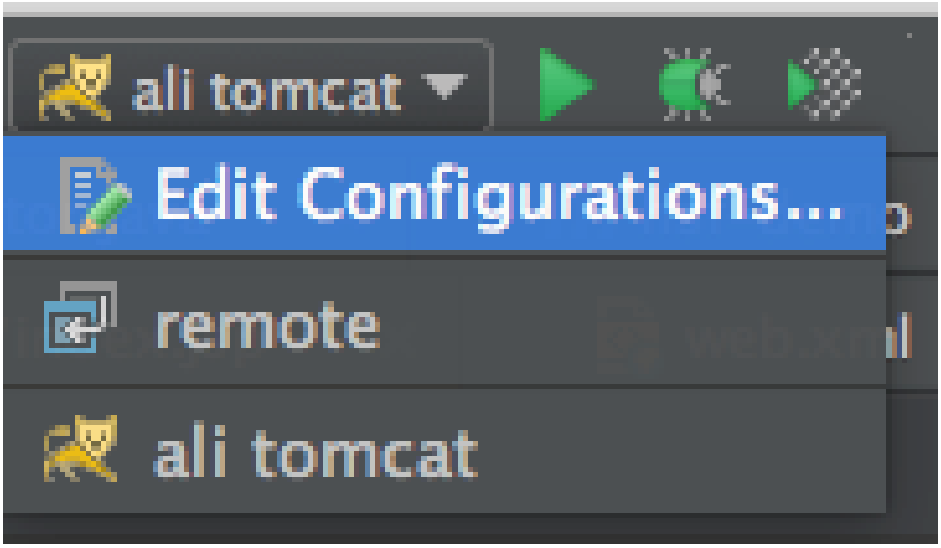
Use local taobao-hsf.sar，并单击旁边的 **Browse** 以选择本地的 Pandora 路径，如：d:\work\tomcat\deploy\taobao-hsf.sar。



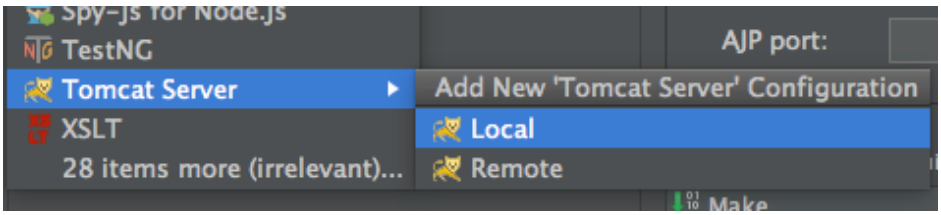
单击 **Apply** 或 **Run**，完成设置。一个工程只需配置一次，下次可直接启动。

IDEA 的配置不需要依赖额外的插件，而是通过JVM启动参数 -Dpandora.location 来不过目前 IDEA 仅仅支持商业版，社区版本暂不支持此种方式。具体步骤如下：

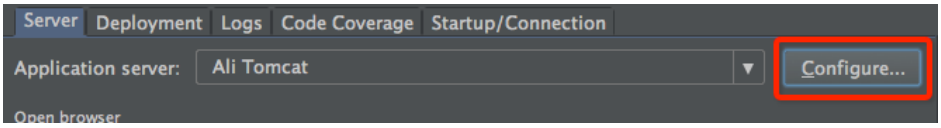
从菜单中或者工具栏中点击 **Run**，选择 **Edit Configuration**。



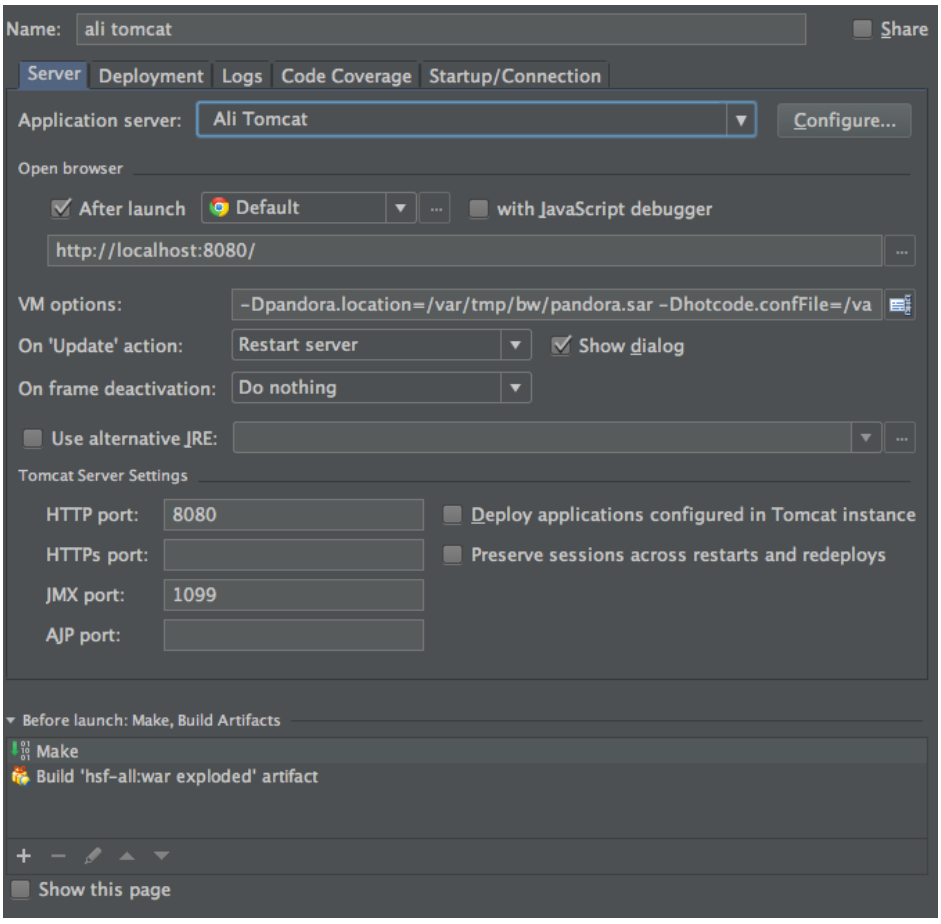
单击界面的加号（ + ），选择 Tomcat Server，单击 **Local**，以增加一个本地 Tomcat 的启动配置。



配置 AliTomcat：在右侧页面选择 **Server** 选项卡，在 **Application Server** 区域，点击 **Configure**。然后在弹出的页面中选择在 **安装 Ali-Tomcat** 时 AliTomcat 的保存路径（如：d:\work\tomcat\）。



配置好 AliTomcat 后，在 Application Server 区域的下拉菜单中，选择刚刚配置好的 AliTomcat 实例。



在 VM Options 区域，增加 JVM 启动参数指向 Pandora 的路径，如：-Dpandora.location=d:\work\tomcat\deploy\taobao-hsf.sar

单击 Apply 或 OK 完成配置。

轻量配置中心给开发者提供在开发、调试、测试的过程中的服务发现、注册和查询功能。此模块不属于 EDAS 正式环境中的服务，使用时请下载安装包进行安装。

在一个公司内部，通常只需要在一台机器上安装轻量配置中心服务，并在其他开发机器上绑定特定的 Host 即可。具体安装和使用的步骤请参见下文。

安装轻量配置中心

确认环境是否达到要求。

正确配置环境变量 JAVA_HOME，指向一个 1.6 或 1.6 以上版本的 JDK。

确认 8080 和 9600 端口未被使用。

由于启动 EDAS 配置中心将会占用此台机器的 8080 和 9600 端口，因此推荐您找一台 **专门的** 机器启动 EDAS 配置中心，比如某台测试机器。如果您是在同一台机器上进行测试，请将 Web 项目的端口修改为其它未被占用的端口。

启动 EDAS 配置中心。

下载 EDAS 配置中心安装包 并解压。

如有需要，可以下载历史版本：

i. 2017年07月版本
ii. 2017年03月版本
iii. 2016年12月版本

进入 edas-config-center 目录启动配置中心：

i. Windows 操作系统：请双击 startup.bat。
ii. Unix 操作系统：请在当前目录下执行 sh startup.sh 命令。

轻量配置中心的使用

对于需要使用轻量配置中心的开发机器，请在本地 DNS（hosts 文件）中，将 jmenv.tbsite.net 域名指向启动了 EDAS 配置中心的机器 IP。

hosts 文件的路径如下：

Windows 操作系统：C:\Windows\System32\drivers\etc\hosts

Unix 操作系统：/etc/hosts

示例：

如果您在 IP 为 192.168.1.100 的机器上面启动了 EDAS 配置中心，则所有开发者只需要在机器的 hosts 文件里加入如下一行即可。

192.168.1.100 jmenv.tbsite.net

绑定 Host 解析后，HSF 服务将基于此注册中心进行服务注册与发现。

服务开发

本章中介绍的代码均可以通过官方 Demo 获取。

下载 Demo 工程 。

将下载下来的压缩包解压，可以看到 itemcenter-api，itemcenter 和 detail 三个 Maven 工程。

其中：

- itemcenter-api 工程提供接口定义
- detail 工程是消费者应用
- Itemcenter 工程是服务提供者应用。

注意：请使用 JDK 1.7 及以上版本。

HSF 的服务基于接口实现，当接口定义好之后，生产者将通过该接口以实现具体的服务，消费者也是基于此接口作为服务去订阅。

在 Demo 的 itemcenter-api 工程中，定义了一个服务接口 com.alibaba.edas.carshop.itemcenter.ItemService，内容如下：

```
public interface ItemService {
public Item getItemById(long id);
public Item getItemByName(String name);
}
```

此接口将提供两个方法，getItemById 与 getItemByName，也可以理解成为该服务 com.alibaba.edas.carshop.itemcenter.ItemService 将提供两个方法。

生产者将实现服务接口以提供具体实现，除了代码实现的工作之外，由于 HSF 是基于 Spring 框架来实现的，所以还需要再定义服务发布的 XML 文件。

代码实现服务接口

在 Demo 工程的 itemcenter 中可看到具体的示例：

```
package com.alibaba.edas.carshop.itemcenter;
public class ItemServiceImpl implements ItemService {

    @Override
    public Item getItemById( long id ) {
        Item car = new Item();
        car.setItemId( 1l );
        car.setItemName( "Mercedes Benz" );
        return car;
    }
    @Override
    public Item getItemByName( String name ) {
        Item car = new Item();
        car.setItemId( 1l );
        car.setItemName( "Mercedes Benz" );
        return car;
    }
}
```

服务配置

上述例子主要实现了 com.alibaba.edas.carshop.itemcenter.ItemService，并在两个方法中返回了一个 Item 对象，代码开发好之后，除了必要的 web.xml 中的 Spring 常规配置，我们还需要增加相应的 Maven 依赖，同时在 Spring 配置文件使用 <hsf /> 标签注册并发布该服务。具体内容如下：

在 pom.xml 中添加如下 Maven 依赖的内容：

```
<dependencies>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>com.alibaba.edas.carshop</groupId>
        <artifactId>itemcenter-api</artifactId>
        <version>1.0.0-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>2.5.6(及其以上版本)</version>
    </dependency>
```

```
    <dependency>
      <groupId>com.alibaba.edas</groupId>
      <artifactId>edas-sdk</artifactId>
      <version>1.5.0</version>
    </dependency>
</dependencies>
```

增加 Spring 关于 HSF 服务的配置，Demo 工程 HSF 配置文件 /resources/hsf-provider-beans.xml 内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:hsf="http://www.taobao.com/hsf"
    xmlns="http://www.springframework.org/schema/beans"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.taobao.com/hsf
    http://www.taobao.com/hsf/hsf.xsd" default-autowire="byName">
    <!-- 定义实现该服务的具体实现 -->
    <bean id="itemService" class="com.alibaba.edas.carshop.itemcenter.ItemServiceImpl" />
    <!-- 用 hsf:provider 标签表明提供一个服务生产者 -->
    <hsf:provider id=“itemServiceProvider"
      <!-- 用 interface 属性说明该服务为此类的一个实现 -->
      interface=“com.alibaba.edas.carshop.itemcenter.ItemService"
      <!-- 此服务具体实现的 spring 对象 -->
      ref=“itemService"
      <!-- 发布该服务的版本号，可任意指定，默认为 1.0.0 -->
      version=“1.0.0"
      <!-- 服务分组 -->
      group=“testHSFGroup-09-04 “>
    </hsf:provider>
</beans>
```

生产者配置属性清单

关于 HSF 生产者的属性配置，除了上述内容提到的之外，还有如下的内容可供选择：

属性	描述
interface	interface 必须配置 [String]，为服务对外提供的接口。
version	version 为可选配置 [String]，含义为服务的版本，默认为 1.0.0。
group	serviceGroup 为可选配置 [String]，含义为服务所属的组别，以便按组别来管理服务的配置，默认为 HSF。
clientTimeout	该配置对接口中的所有方法生效，但是如果客户端通过 MethodSpecial 属性对某方法配置了超时时间，则该方法的超时时间以客户端配置为准，其他方法不受影响，还是以服务端配置为准。
serializeType	serializeType 为可选配置

	[String(hessian | java)]，含义为序列化类型，默认为 hessian。
corePoolSize	单独针对这个服务设置核心线程池，是从公用线程池这个大蛋糕里切一块下来。
maxPoolSize	单独针对这个服务设置线程池，是从公用线程池这个大蛋糕里切一块下来。
enableTXC	开启分布式事务 GTS。
ref	ref 必须配置 [ref]，为需要发布为 HSF 服务的 Spring Bean ID。
methodSpecials	methodSpecials 为可选配置，用于为方法单独配置超时(单位 ms)，这样接口中的方法可以采用不同的超时时间，该配置优先级高于上面的 clientTimeout 的超时配置，低于客户端的 methodSpecials 配置。

标签配置示例：

```
<bean id="impl" class="com.taobao.edas.service.impl.SimpleServiceImpl" />
   <hsf:provider id="simpleService" interface="com.taobao.edas.service.SimpleService"
      ref="impl" version="1.0.1" group="test1" clientTimeout="3000" enableTXC="true"
      serializeType="hessian">
      <hsf:methodSpecials>
         <hsf:methodSpecial name="sum" timeout="2000" />
      </hsf:methodSpecials>
   </hsf:provider>
```

pom 中添加edas-sdk依赖

```
<dependency>
   <groupId>com.alibaba.edas</groupId>
   <artifactId>edas-sdk</artifactId>
   <version>1.6.1</version>
</dependency>
```

开发环境发布服务

完成代码和配置的开发任务之后，在 Eclipse 和 IDEA 中，可直接以 Ali-Tomcat 运行该服务（具体请参照文档 IDE 运行时启动配置，运行成功后，可在轻量配置中心查询到所发布的服务，具体请参考服务查询文档。

服务提供者额外的 JVM 启动参数

在服务的提供者中，有一些额外的启动参数来改变 HSF 的行为，具体如下：

属性	描述
-Dhsf.server.port	指定 HSF 的启动服务绑定端口，默认为 12200。
-Dhsf.serializer	指定 HSF 的序列化方式，默认值为 hessian。

-Dhsf.server.max.poolsize	指定 HSF 的服务端最大线程池大小，默认值为600。
-Dhsf.server.min.poolsize	指定 HSF 的服务端最小线程池大小。默认值为50。

消费者的订阅从代码编写的角度分为两个部分：首先 Spring 的配置文件使用标签<hsf:consumer/>定义好一个 Bean；然后在使用的时候从 Spring 的 context 中将 Bean 取出来即可。Demo 工程的 detail 为消费者的示例，示例代码说明如下。

消费者配置

与生产者一样消费者的配置文件分为 Maven 依赖配置与 Spring 的配置，且 Maven 的依赖与生产者依赖一样，详情请参考文档 生产者实现服务 的 **服务配置** 小节。

除了必要的 Spring 所定义的配置之外，还需要在 Spring 的配置文件中增加消费者的定义，HSF 框架将根据该配置文件去服务中心订阅所需的服务，配置文件 /resource/hsf-consumer-beans.xml 内容与释义如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:hsf="http://www.taobao.com/hsf"
xmlns="http://www.springframework.org/schema/beans"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.taobao.com/hsf
http://www.taobao.com/hsf/hsf.xsd" default-autowire="byName">
<!-- 消费一个服务示例 -->
<hsf:consumer
<!-- Bean ID，在代码中可根据此ID进行注入并使用 -->
id="item"
<!-- 服务名，与服务提供者的相应配置对应，HSF 将根据 interface + version + group 查询并订阅所需服务 -->
interface="com.alibaba.edas.carshop.itemcenter.ItemService"
<!-- 版本号，与服务提供者的相应配置对应，HSF 将根据 interface + version + group 查询并订阅所需服务 -->
version="1.0.0"
<!-- 分组名 -->
group="testHSFGroup">
</hsf:consumer>
</beans>
```

消费者使用服务

Demo 中的示例代码如下：

```
public class StartListener implements ServletContextListener{

@Override
public void contextInitialized( ServletContextEvent sce ) {
ApplicationContext ctx = WebApplicationContextUtils.getWebApplicationContext( sce.getServletContext() );
// 根据 Spring 配置中的Bean ID “item” 获取订阅到的服务
final ItemService itemService = ( ItemService ) ctx.getBean( "item" );
```

```
.....
// 调用服务 ItemService 的 getItemById 方法
System.out.println( itemService.getItemById( 1111 ) );
// 调用服务 ItemService 的 getItemByName 方法
System.out.println( itemService.getItemByName( "myname is le" ) );
.....
}
}
```

消费者配置属性清单

除了示例代码中体现的属性(interface，group，version)之外，还有下表的属性配置以供选择：

属性	描述
interface	interface 必须配置 [String]，为需要调用的服务的接口。
version	version 为可选配置 [String]，含义为需要调用的服务的版本，默认为1.0.0。
group	group 为可选配置 [String]，含义为需要调用的服务所在的组，以便按组别来管理服务的配置，默认为 HSF，建议配置。
methodSpecials	methodSpecials 为可选配置，含义为为方法单独配置超时(单位 ms)，这样接口中的方法可以采用不同的超时时间，该配置优先级高于服务端的超时配置。
target	主要用于单元测试环境和开发环境中在消费者端指定 -Dhsf.run.mode=0 的情况下，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。
connectionNum	connectionNum 为可选配置，含义为支持设置连接到 server 连接数，默认为1，在小数据传输，要求低延迟的情况下设置多一些，会提升 TPS。
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 。
asyncallMethods	asyncallMethods 为可选配置 [List]，含义为调用此服务时需要采用异步调用的方法名列表以及异步调用的方式。默认为空集合，即所有方法都采用同步调用。
maxWaitTimeForCsAddress	配置该参数，目的是当服务进行订阅时，会在该参数指定时间内，阻塞线程等待地址推送，避免调用该服务时因为地址为空而出现地址找不到的情况。若超过该参数指定时间，地址还是没有推送，线程将不再等待，继续初始化后续内容。注意，仅仅限于应用初始化时需要同步调用某个服务后，进行初始化数据时，才使用。其他情况请勿使用。该功能会延长启动启动时间。

标签配置示例:

```
<hsf:consumer id="service" interface="com.taobao.edas.service.SimpleService"
version="1.1.0" group="test1" clientTimeout="3000"
target="10.1.6.57:12200?_TIMEOUT=1000" maxWaitTimeForCsAddress="5000">
<hsf:methodSpecials>
<hsf:methodSpecial name="sum" timeout="2000" ></hsf:methodSpecial>
</hsf:methodSpecials>
</hsf:consumer>
```

pom 中添加 edas-sdk 依赖

```
<dependency>
<groupId>com.alibaba.edas</groupId>
<artifactId>edas-sdk</artifactId>
<version>1.6.1</version>
</dependency>
```

开发环境消费服务

完成代码和配置的开发任务之后，在 Eclipse 和 IDEA 中，可直接以 Ali-Tomcat 运行该服务（具体请参照文档 IDE 运行时启动配置。运行成功后，请参考 开发环境搭建 文档，在搭建好的 ConfigCenter 中查询到所需要消费的服务。

目前 EDAS 支持 Dubbo 与 HSF 的服务注册，此篇文档仅说明 HSF 的服务查询方式。如果您的 Dubbo 的服务还是发布到了原有的注册中心(如：ZooKeeper)的话，目前 EDAS 的后台无法进行查看。

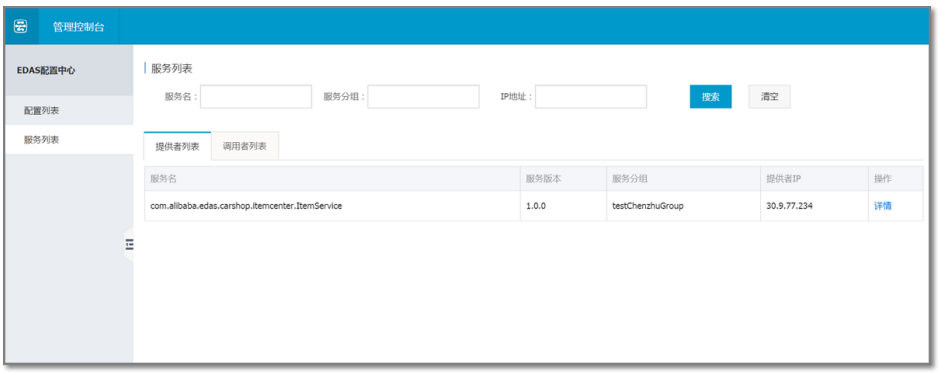
开发环境查询 HSF 服务

在开发调试的过程中，如果您的服务是通过轻量配置中心进行服务注册与发现，就可以通过轻量配置中心的后台查询某个应用提供或调用的服务。

假设您在一台 IP 为 192.168.1.100 的机器上启动了 EDAS 配置中心。

进入http://192.168.1.100:8080/ 。

在左侧菜单栏单击**服务列表**，输入服务名、服务组名或者 IP 地址进行搜索，查看对应的服务提供者以及服务调用者。



注意：配置中心启动之后默认选择第一块网卡地址做为服务发现的地址，如果开发者所在的机器有多块网卡的情况，可设置启动脚本中的 SERVER_IP 变量进行显式的地址绑定。

常见查询案例

提供者列表页

在搜索条件里输入 IP 地址，点击**搜索**即可查询该 IP 地址的机器提供了哪些服务。

在搜索条件里输入服务名或服务分组，即可查询哪些 IP 地址提供了这个服务。

调用者列表页

在搜索条件里输入 IP 地址，点击**搜索**即可查询该 IP 地址的机器调用了哪些服务。

在搜索条件里输入服务名或服务分组，即可查询哪些 IP 地址调用了这个服务。

线上环境查询 HSF 服务

开发好的服务打包并在 EDAS 后台部署完毕之后，确认完应用正常启动的情况下，用户可从 EDAS 后台查询相应的服务列表信息，具体步骤如下：

登录 EDAS 控制台，在左侧菜单栏选择 **应用管理** 。

在应用列表页，单击部署的应用进入应用详情页。

在左侧菜单栏选择 **服务列表** 选项，可看到 **发布的服务** 与 **消费的服务** 两个 TAB 标签。**发布的服务** 即应用配置的 Provider， **消费的服务** 为应用配置的 Consumer。

注意：如您是通过子账号登录，请确认有无查看**服务列表**页的权限。您可以在控制台左侧菜单栏选择

账号管理>**所有权限**，在权限管理页查看**应用管理**下有无服务列表的查看权限。

如果在相应的服务列表中没有相应的服务，可按照以下步骤排除可能存在的问题：

- 请确认服务配置在代码中是否配置正确。
- 请确认服务的 Tomcat 进程正常启动，且在 log 中没有报错 (查看 TOMCAT_HOME/logs/catalina.out 和 $TOMCAT_HOME/logs/localhost.log.$DATE_FORMAT)。
- 请确认是否是最新的软件版本(具体在对应服务信息界面左侧菜单栏的 **软件版本** 处查看), 如果不是最新版本，需确认对应的 HSF 分组是否创建。
- 请确认对应机器的 Host 是否有特殊的网络绑定。正常情况下线上机器无须绑定任何 Host。
- 请确认机器的网络与 ECS 安全组配置是否有明显的限制。

本文介绍 HSF 一些特性的使用方法及注意事项。

所有特性的实例 Demo 请在这里下载：Demo 下载

前提条件

使用 HSF 相关特性，请在 POM 文件中加入以下 edas-sdk 依赖。

```
<dependency>
<groupId>com.alibaba.edas</groupId>
<artifactId>edas-sdk</artifactId>
<version>1.5.1</version>
</dependency>
```

隐式传参（目前仅支持字符串传输）

隐式传参一般用于传递一些简单 KV 数据，又不想通过接口方式传递，类似于 Cookie。

单个参数传递

服务消费者：

RpcContext.getContext().setAttachment("key", "args test");

服务提供者：

String keyVal=RpcContext.getContext().getAttachment("key");

多个参数传递

服务消费者：

```
Map<String,String> map=new HashMap<String,String>();
map.put("param1", "param1 test");
map.put("param2", "param2 test");
map.put("param3", "param3 test");
map.put("param4", "param4 test");
map.put("param5", "param5 test");
RpcContext rpcContext = RpcContext.getContext();
rpcContext.setAttachments(map);
```

服务提供者：

```
Map<String,String> map=rpcContext.getAttachments();
Set<String> set=map.keySet();
for (String key : set) {
System.out.println("map value:"+map.get(key));
}
```

异步调用

支持异步调用 callback 和 future 两种方式。

callback 调用方式

客户端配置为 callback 方式调用时，需要配置一个实现了 HSFResponseCallback 接口的 listener ，结果返回之后， HSF 会调用 HSFResponseCallback 中的方法。

注意：这个 HSFResponseCallback 接口的 listener 不能是内部类，否则 Pandora 的 classloader 在加载时就会报错。

XML 中的配置：

```
<hsf:consumer id="demoApi" interface="com.alibaba.demo.api.DemoApi"
version="1.1.2" group="test" >
<hsf:asyncallMethods>
<hsf:method name="ayncTest" type="callback"
listener="com.alibaba.ifree.hsf.consumer.AsynABTestCallbackHandler" />
</hsf:asyncallMethods>
</hsf:consumer>
```

其中 AsynABTestCallbackHandler 类，是实现了 HSFResponseCallback 接口。DemoApi 接口中有一个方法是 ayncTest 。

代码示例

```
public void onAppResponse(Object appResponse) {
//这里获取到异步调用后的值
String msg = (String)appResponse;
```

```
System.out.println("msg:"+msg);
}
```

注意：

- 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。
- 不支持在 call 里再发起 HSF 调用。这种做法可能导致 IO 线程挂起，无法恢复。

future 调用方式

客户端配置为 future 方式调用时，发起调用之后，通过 HSFResponseFuture 中的 public static Object getResponse(long timeout) 来获取返回结果。

XML 中的配置：

```
<hsf:consumer id="demoApi" interface="com.alibaba.demo.api.DemoApi" version="1.1.2"  group="test"
>
<hsf:asyncallMethods>
<hsf:method name="ayncTest" type="future" />
</hsf:asyncallMethods>
</hsf:consumer>
```

代码示例如下。

单个业务异步处理：

```
//发起调用
demoApi.ayncTest();
// 处理业务
...
//直接获得消息（若无需获得结果，可以不用操作该步骤）
String msg=(String) HSFResponseFuture.getResponse(3000);
```

多个业务需要并发处理：

若是多个业务需要并发处理，可以先获取 future，进行存储起来，等调用完毕后在使用。

```
//定义集合
List<HSFFuture> futures = new ArrayList<HSFFuture>();
```

方法内进行并行调用：

```
//发起调用
demoApi.ayncTest();
//第一步获取future对象
```

```
HSFFuture future=HSFResponseFuture.getFuture();
futures.add(future);
//继续调用其他业务(同样采取异步调用)
HSFFuture future=HSFResponseFuture.getFuture();
futures.add(future);

// 处理业务
...

//获得数据并做处理
for (HSFFuture hsfFuture : futures) {
String msg=(String) hsfFuture.getResponse(3000);
//处理相应数据
...
}
```

泛化调用

通过泛化调用可以组合接口、方法、参数进行RPC调用,无需依赖任何业务API。

操作步骤

在消费者XML配置中加入泛化属性。

```
<hsf:consumer id="demoApi" interface="com.alibaba.demo.api.DemoApi"  group="unittest" generic="true"/>
```

说明：generic 代表泛化参数，true 表示支持泛化，false 表示不支持，默认为 false。

DemoApi 接口方法：

```
public String dealMsg(String msg);
public GenericTestDO dealGenericTestDO(GenericTestDO testDO);
```

获取demoApi进行强制转换为泛化服务

导入泛化服务接口

```
import com.alibaba.dubbo.rpc.service.GenericService
```

获取泛化对象

- XML加载方式

```
//若WEB项目中，可通过Spring bean进行注入后强转，这里是单元测试，所以采用加载配置文件方式
ClassPathXmlApplicationContext consumerContext = new ClassPathXmlApplicationContext("hsf-generic-consumer-beans.xml");
```

```
//强转接口接口为 GenericService
GenericService svc = (GenericService) consumerContext.getBean("demoApi");
```

代码订阅方式

```
HSFApiConsumerBean consumerBean = new HSFApiConsumerBean();
consumerBean.setInterfaceName("com.alibaba.demo.api.DemoApi");
consumerBean.setGeneric("true"); // 设置 generic 为 true
consumerBean.setGroup("unittest");
consumerBean.setVersion("1.0.0");
consumerBean.init();
// 强转接口接口为 GenericService
GenericService svc = (GenericService) consumerBean.getObject();
```

泛化接口

```
Object $invoke(String methodName, String[] parameterTypes, Object[] args) throws GenericException;
```

说明：

methodName：需要调用的方法名称。

parameterTypes：需要调用方法参数的类型。

args：需要传输的参数值。

泛化调用(String类型参数)

```
svc.$invoke("dealMsg", new String[] { "java.lang.String" }, new Object[] { "hello" })
```

泛化调用（对象参数，服务端和客户端需要保证相同的对象）

```
// 第一步构造实体对象GenericTestDO，该实体有id、name两个属性
GenericTestDO genericTestDO = new GenericTestDO();
genericTestDO.setId(1980l);
genericTestDO.setName("genericTestDO-tst");
// 使用 PojoUtils 生成二方包pojo的描述
Object comp = PojoUtils.generalize(genericTestDO);
// 服务泛化调用
svc.$invoke("dealGenericTestDO",new String[] { "com.alibaba.demo.generic.domain.GenericTestDO" }, new Object[] {
comp });
```

在测试环境中，有两种方式做单元测试。

Demo 下载

方式一 、通过 LightApi 代码发布和订阅服务

在 Maven 中添加 LightApi 依赖。

```
<dependency>
<groupId>com.alibaba.hsf</groupId>
<artifactId>LightApi</artifactId>
<version>1.0.0</version>
</dependency>
```

创建 ServiceFactory。

这里需要设置 Pandora 的地址，参数是 SAR 包所在目录。如果 SAR 包地址是 /Users/Jason/Work/AliSoft/PandoraSar/DevSar/taobao-hsf.sar，则参数如下：

```
private static final ServiceFactory factory =
ServiceFactory.getInstanceWithPath("/Users/Jason/Work/AliSoft/PandoraSar/DevSar");
```

通过代码进行发布和订阅服务。

```
// 进行服务发布（若有发布者，无需再这里写）
factory.provider("helloProvider")// 参数是一个标识，初始化后，下次只需调用provider("helloProvider")即可拿出对应服务
.service("com.alibaba.edas.unit.service.UnitTestService")// 接口全类名
.version("1.0.0")// 版本号
.group("light")// 组别
.impl(new UnitTestServiceImpl())// 对应的服务实现
.publish();// 发布服务，至少要调用service()和version()才可以发布服务

// 进行服务消费
factory.consumer("helloConsumer")// 参数是一个标识，初始化后，下次只需调用consumer("helloConsumer")即可直接拿出对应服务
.service("com.alibaba.edas.unit.service.UnitTestService")// 接口全类名
.version("1.0.0")// 版本号
.group("light")// 组别
.subscribe();
factory.consumer("helloConsumer").sync();// 同步等待地址推送，最多6秒。
UnitTestService log4jService = (UnitTestService) factory.consumer("helloConsumer").subscribe();// 用ID取出对应服务，subscribe()方法返回对应的接口
// 调用服务方法
System.out.println("bean -> msg rec success:-"+log4jService.print());
```

方式二、通过 XML 配置发布订阅服务。

编写好 HSF 的 XML 配置。

通过代码方式加载配置文件。

```
 //XML方式加载服务提供者
new ClassPathXmlApplicationContext("hsf-provider-beans.xml");
//XML方式加载服务消费者
ClassPathXmlApplicationContext consumerContext=new ClassPathXmlApplicationContext("hsf-
consumer-beans.xml");
//获取Bean
UnitTestXMLConsumer unitTestXMLConsumer=(UnitTestXMLConsumer)
consumerContext.getBean("unitTestConsumer");
//服务调用
unitTestXMLConsumer.testUnitProvider();
```

本文主要针对使用 Spring Boot 开发应用的用户，介绍如何快速让 HSF 在 Spring Boot 中运行，并提供了完整的 Demo下载。

开发在 EDAS 中运行的 Spring Boot 项目

开发 Spring Boot 项目需要使用 Servlet 3.0 及以上版本，同时保持生成环境和 Spring Boot 编译环境 JDK 版本一致。具体开发步骤参见下文。

创建 Web 项目，引入 Spring Boot 依赖包。

pom.xml

```
<!-- 打包方式由JAR 改为 WAR -->
<packaging>war</packaging>

<!-- 添加 spring-boot-starter-parent 依赖 -->
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>1.4.1.RELEASE</version>
</parent>

<!-- Spring Boot 依赖，这里必须排除嵌入式的 Tomcat ，否则在 Tomcat4E 会出现问题；若使用 Main 函数启
动，则不用排除 -->
<dependencies>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
      <exclusions>
         <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
         </exclusion>
      </exclusions>
   </dependency>
</dependencies>
```

```
<!-- 添加 Spring Boot 插件 -->

<plugins>
  <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
  </plugin>
</plugins>
```

注意：

在 pom.xml 里面引入的 Spring Boot 依赖，若要以 WAR 包形式在 Tomcat 中运行，需要排除 spring-boot-starter-web 嵌入式 Tomcat 依赖，否则会和 Tomcat 默认的 lib 内容形成冲突，导致项目无法启动。如果在 Main 函数中启动则不用排除嵌入式 Tomcat，但是在打包发布时，请一定记住需要排除。

创建 ServletInitializer 类，继承 SpringBootServletInitializer。

类似初始化 Spring 上下文，标记该项目打包成 WAR 项目在 Tomcat 中运行。

继承 SpringBootServletInitializer 类，实现 configure 方法 ServletInitializer.class。

```
public class ServletInitializer extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(StartupDemoApplication.class);
    }
}
```

创建 Spring Boot 启动类。

类名：StartupDemoApplication.class

```
@SpringBootApplication
public class StartupDemoApplication {

    public static void main(String[] args) {
        //启动服务
        SpringApplication.run(StartupDemoApplication.class, args);
    }
}
```

删除 webapp 下面的 web.xml 文件。

若该文件存在，在容器加载 Web 的时候，就会直接加载该配置文件，但是由于该配置文件没有做任

何配置，因此会造成 Spring Boot 加载不成功。

排除方式：

若发现发布在 EDAS 中的 WAR 没有执行您自己的代码，那么有可能就是该问题导致，请及时检查是否存在 web.xml 文件。

打包 WAR，上传 EDAS 启动即可。

如果测试阶段，需要直接在 Main 方法中运行，请把 pom.xml 文件中的 spring-boot-starter-tomcat 排除取消。

Spring Boot 结合 HSF 开发

添加 HSF 服务提供者配置文件（hsf-provider-beans.xml）。

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

    <!-- 服务处理 bean -->
    <bean id="memberServiceImpl"
        class="com.alibaba.edas.springboot.service.impl.MemberServiceImpl" />

    <!-- 提供一个 HSF 服务示例 -->
    <hsf:provider id="memberService"
        interface="com.alibaba.edas.springboot.service.MemberService" ref="memberServiceImpl"
group="test" />

</beans>
```

导入 HSF 服务提供者配置。

```
@ImportResource(locations={"classpath:hsf-provider-beans.xml"})
@SpringBootApplication
public class StartupDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(StartupDemoApplication.class, args);
    }
}
```

打包上传 EDAS ，具体请参考服务上线。

添加 HSF 消费者配置文件（hsf-consumer-beans.xml）。

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

    <!-- 消费一个 HSF 服务示例 -->
    <hsf:consumer id="memberService"  interface="com.alibaba.edas.springboot.service.MemberService"
group="test" />

</beans>
```

导入 HSF 消费者配置。

```
@ImportResource(locations="classpath:hsf-consumer-beans.xml")
@SpringBootApplication
public class StartupDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(StartupDemoApplication.class, args);
    }
}
```

打包上传 EDAS ，具体请参考服务上线。

目前 EDAS 已经支持 Spring Cloud 和 Spring Boot 开发 EDAS 微服务应用了，通过打包成一个 FatJar 并发布到 EDAS 平台中。点击 此处下载最新的基于 Spring Boot 开发 HSF 服务的实践 ，点击此处下载最新的基于 Spring Cloud 开发 RESTful 服务的实践。开发环境部署时需要使用到轻量级配置中心。

- 私服配置及依赖 Maven 私服配置Maven 依赖 通用依赖HSF 服务依赖RESTful 服务依赖
- HSF 服务开发 全局配置启动代码服务发布服务订阅单元测试异步调用
- RESTful 服务开发 全局配置服务端启动消费端启动服务发布服务订阅 服务发现服务消费
- 部署 本地部署EDAS 中部署

私服配置及依赖

Maven 私服配置

注意： Maven 版本要求 3.x 及以上请在你的 Maven 配置文件 Setting 中，加入中间件私服地址。样例文件下载。

```
<?xml version="1.0"?>
<settings>
<!-- 设置用户自己的本地库 -->
<!-- <localRepository>/Users/../.m2/repository</localRepository> -->
<profiles>
<profile>
<id>nexus</id>
<repositories>
<repository>
<id>central</id>
<url>http://repo1.maven.org/maven2</url>
<releases>
<enabled>true</enabled>
</releases>
<snapshots>
<enabled>true</enabled>
</snapshots>
</repository>
</repositories>
<pluginRepositories>
<pluginRepository>
<id>central</id>
<url>http://repo1.maven.org/maven2</url>
<releases>
<enabled>true</enabled>
</releases>
<snapshots>
<enabled>true</enabled>
</snapshots>
</pluginRepository>
</pluginRepositories>
</profile>

<profile>
<id>edas.oss.repo</id>
<repositories>
<repository>
<id>edas-oss-central</id>
<name>taobao mirror central</name>
<url>http://edas-public.oss-cn-hangzhou.aliyuncs.com/repository</url>
<snapshots>
<enabled>true</enabled>
</snapshots>
<releases>
<enabled>true</enabled>
</releases>
</repository>
</repositories>
<pluginRepositories>
<pluginRepository>
<id>edas-oss-plugin-central</id>
<url>http://edas-public.oss-cn-hangzhou.aliyuncs.com/repository</url>
```

```
<snapshots>
<enabled>true</enabled>
</snapshots>
<releases>
<enabled>true</enabled>
</releases>
</pluginRepository>
</pluginRepositories>
</profile>

</profiles>

<activeProfiles>
<activeProfile>nexus</activeProfile>
<activeProfile>edas.oss.repo</activeProfile>
</activeProfiles>
</settings>
```

Maven 依赖

以下列出了几种依赖关系，不同的依赖针对代码编写也不一样。关于以下依赖的版本信息详见 Demo 工程中的父 pom 文件。

通用依赖

最少依赖，**必选**。

```
 <dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-pandora</artifactId>
<version>1.0</version>
</dependency>
```

FatJar 插件依赖(该插件用于生成 FatJar ,并且默认排除 taobao-hsf.sar 这个 JAR 包，该 JAR 包在生产环境部署时会通过 -D 参数自动指定，无需打包，避免资源浪费)，**必选**。

```
 <plugin>
<groupId>com.taobao.pandora</groupId>
<artifactId>pandora-boot-maven-plugin</artifactId>
<version>2.1.6.3</version>
<executions>
<execution>
<phase>package</phase>
<goals>
<goal>repackage</goal>
</goals>
</execution>
</executions>
</plugin>
```

HSF 服务依赖

使用 Spring Boot 编程模型结合 Pandora Boot开发HSF服务时，需要使用到的依赖（若使用XML配置文件方式，可以不添加），**可选**。

```
 <dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-hsf</artifactId>
</dependency>
```

RESTful 服务依赖

为 Spring Cloud 开发的 RESTful 应用在 EDAS 中添加服务注册与发现的支持，**必选**。 EDAS 中 Spring Cloud 服务注册与发现功能目前只支持自研的 VIPServer。

```
 <dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-vipclient</artifactId>
</dependency>
```

为 Spring Cloud 开发的 RESTful 应用在 EDAS 中服务添加链路跟踪的依赖。**可选，添加后才能使用服务监控 功能。**

```
 <dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eagleeye</artifactId>
</dependency>
```

为 Spring Cloud 开发的 RESTful 应用在 EDAS 中服务添加限流支持的依赖。**可选，添加后才能使用限流降级 功能。**

```
 <dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-sentinel</artifactId>
</dependency>
```

为 Spring Cloud 开发的 RESTful 应用在 EDAS 中服务添加服务调用鉴权的依赖，**可选**。

```
 <dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-dauth</artifactId>
</dependency>
```

注意：若应用没有依赖 spring-boot-starter-web ，将不会引入嵌入式 Tomcat ，导致 EDAS 页面上面的一些 Tomcat 相关参数及健康检查无法使用。

HSF 服务开发

全局配置

注意： 全局配置只针对于采用 Spring Boot 编程模型在 Pandora Boot 应用中使用 EDAS 服务时有用，针对 Spring XML 配置文件发布/订阅服务方式无效。

使用方式：在 application.properties 添加 EDAS 服务的版本信息和客户端超时信息, 样例配置如下（**建议都默认添加**）：

```
spring.hsf.group=HSF
spring.hsf.version=1.0.0
spring.hsf.timeout=3000
```

若未添加，则默认属性：group=HSF, version=1.0.0. DAILY, timeout=3000

启动代码

```
@SpringBootApplication
public class HSFBootDemoProviderApplication {
public static void main(String[] args) {
// 启动Pandora Boot 用于加载 Pandora 容器
PandoraBootstrap.run(args);
// 启动Spring Boot
SpringApplication springApplication=new SpringApplication(HSFBootDemoProviderApplication.class);
springApplication.run(args);
// 标记服务启动完成,并设置线程 wait。防止用户业务代码运行完毕退出后，导致容器退出。
PandoraBootstrap.markStartupAndWait();
}
}
```

说明：这里主要添加了 PandoraBootstrap 相关代码，主要用途是为了启动 Pandora 容器相关插件。

服务发布

配置发布

编写服务配置

```
  <!-- 提供一个服务示例，interface 表示需要公布的服务接口，ref 是具体该接口实现的服务-->
<hsf:provider id="hSFBootDemoApi"
interface="com.aliware.edas.hsfboot.api.VersionInfoApi" ref="versionInfoApiImpl" />
```

导入配置（在 Spring Boot Application 的上添加 @ImportResource ）

```
@SpringBootApplication
@ImportResource(locations = {"classpath:hsf-beans.xml"})
public class HSFBootDemoProviderApplication
```

注解发布

注解方式符合 Spring Boot 编程风格，只需要通过简单几个注解即可完成服务发布。

编写 HSF 服务, 在发布的服务实现上添加 @HSFProvider, 其中 serviceInterface 是要发布服务的接口，如下

```
@HSFProvider(serviceInterface =
VersionInfoApi.class,serviceGroup="HSF",clientTimeout=3000,serviceVersion="1.0.0")
public class VersionInfoApiImpl implements VersionInfoApi
```

服务订阅

配置订阅

编写服务配置

```
<!-- 提供一个服务示例 interface 表示需要订阅的服务接口 -->
<hsf:consumer id="versionInfoApi"
interface="com.aliware.edas.hsfboot.api.VersionInfoApi"></hsf:consumer>
```

导入配置（在 Spring Boot Application 的上添加 @ImportResource ）

```
@SpringBootApplication
@ImportResource(locations = {"classpath:hsf-beans.xml"})
public class HSFBootDemoConsumerApplication
```

使用

```
@Autowired
private VersionInfoApi versionInfoApi;
```

注解订阅

注解方式符合 Spring Boot 编程风格，只需要通过简单几个注解即可完成服务订阅。

通过 @HSFConsumer 注入需要消费的 interface，如下

```
@HSFConsumer(serviceGroup="HSF",clientTimeout=3000,serviceVersion="1.0.0")
private VersionInfoApi versionInfoApi;
```

最佳实践：在 Config 类里配置一次 @HSFConsumer ，然后在多处 @Autowired 注入使用通常一个 HSF Consumer 需要在多个地方使用，但并不需要在每次使用的地方都用 @HSFConsumer 来标记。只需要写一个统一个 Config 类，然后在其它需要使用的地方，直接 @Autowired 注入即可。

@Configuration public class HsfConfig {

```
/**
* 版本接口，这里只需要一个注解，就完成了服务订阅，在使用的地方，直接@Autowired即可使用。
* 注意：配置方式订阅服务方式不需要写该注解
*/
@HSFConsumer(serviceGroup="HSF",clientTimeout=3000,serviceVersion="1.0.0")
private VersionInfoApi versionInfoApi;
}
```

说明：

- 如果在 application.proeprteis 中定义了全局配置, 那么 serviceGroup 和 serviceVersion 是不需要指定的。

@HSFProvider 、@HSFConsumer 中的 serviceGroup 和 serviceVersion 以及其他 String 类型的配置都可以支持 Spring 的 property placeholder, 如:

```
@HSFProvider(serviceInterface = VersionInfoApi.class, serviceGroup = "${service.group.name}")
@HSFConsumer(serviceGroup="${service.group.name}",clientTimeout=3000)
```

单元测试

Pandora Boot 应用的单元测试可以通过 PandoraBootRunner 启动，并与 SpringJUnit4ClassRunner 无缝集成

代码编写

```
@RunWith(PandoraBootRunner.class)
@DelegateTo(SpringJUnit4ClassRunner.class)
// 加载测试需要的类，一定要加入Spring Boot的启动类，其次需要加入本类
@SpringBootTest(classes = { HSFBootDemoProviderApplication.class, VersionInfoTest.class })
```

```
@Component
public class VersionInfoTest {
//当这里使用 @HSFConsumer 时，一定要在 @SpringBootTest 类加载中，加载本类，通过本类来注入对象，否则当做泛化时，会报类转换异常

@HSFConsumer(generic = true)
VersionInfoApi versionApi;

@Test
public void testInvoke() {
System.out.println("#####:"+versionApi.getVersionInfo());
TestCase.assertEquals("1.0.0", versionApi.getVersionInfo());
}

@Test
public void testGenericInvoke() {
GenericService service = (GenericService) versionApi;
Object result = service.$invoke("getVersionInfo", new String[] {}, new Object[] {});
System.out.println("####:"+result);
TestCase.assertEquals("1.0.0", result);
}

@Test
public void testMock() {
VersionInfoApi mock = Mockito.mock(VersionInfoApi.class, AdditionalAnswers.delegatesTo(versionApi));
Mockito.when(mock.getVersionInfo()).thenReturn("V1.9.beta");
TestCase.assertEquals("V1.9.beta", mock.getVersionInfo());
}
}
```

异步调用

HSF 异步回调支持

通过实现 HSFResponseCallback 以及在实现类上标注 @AsyncOn 可以很方便地使用 HSF Consumer 的 callback listener。

```
    @AsyncOn(interfaceName=VersionInfoApi.class,methodName="getVersionInfo")
public class VersionInfoResponseListener implements HSFResponseCallback {
@Override
public void onAppException(Throwable t) {
}

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

@Override
public void onHSFException(HSFException hsfEx) {
}
```

```
}
```

HSF Future 调用支持

通过在 @HSFConsumer 中包含 futureMethods (以 , 分隔)在指定希望哪些方法走 Future 调用。

```
@HSFConsumer(serviceGroup = "HSF", clientTimeout = 3000, serviceVersion = "1.0.0", futureMethods = "countVersionInfo")
private VersionInfoApi versionInfoApi;
```

调用执行后，可以通过以下的代码获取到执行结果。更详细的用法请参阅 HSF Future 调用文档

```
HSFFuture hsfFuture = HSFResponseFuture.getFuture();
Object result = future.getResponse(5000);
```

RESTful 服务开发

全局配置

使用方式：在 application.properties 添加 EDAS 服务的相关配置，样例配置如下:

```
spring.application.name=spring-cloud-provider
server.port=18080
```

其他相关参数 （ 可选 ）

```
spring.edas.dauth.enable=true #接口是否开启鉴权,默认为true
spring.edas.dauth.whitelist.paths=/appwl #白名单，不做鉴权的路径，只判断第一层路径
spring.sentinel.filter.urlPatterns=/* #需要进行限流的url，这里添加所有都进行限流
vipserver.register.doms=my-provider,sc-provider,service-name #vipserver注册的服务名
vipserver.register.enabled=false #关闭服务注册的功能
```

说明：

spring.edas.dauth.enable： 鉴权相关的配置，默认为开启，可不填。

spring.sentinel.filter.urlPatterns：限流降级相关的配置，在这里可以指定需要监控的URL后，可以通过EDAS控制台对这些URL进行限流降级规则配置。

vipserver.register.doms：服务名的配置，用英文逗号隔开。样例中的三个服务名是对等的关系，订阅者可以使用其中任意一个调用。若不填写，默认为应用名称。

vipserver.register.enabled：服务发布的开关，如果此工程只有服务消费者，没有服务提供者，应该将此值设置为false。

服务端启动

```
    @EnableDiscoveryClient
@SpringBootApplication
public class ProviderApplication {
public static void main(String[] args) {
// 启动Pandora Boot 用于加载 Pandora 容器
PandoraBootstrap.run(args);
// 启动Spring Boot
SpringApplication.run(ProviderApplication.class, args);
// 标记服务启动完成,并设置线程 wait。防止用户业务代码运行完毕退出后，导致容器退出。
PandoraBootstrap.markStartupAndWait();
}
}
```

消费端启动

```
    @EnableDiscoveryClient
@EnableFeignClients
@SpringBootApplication
public class ConsumerApplication {

public static void main(String[] args) {
PandoraBootstrap.run(args);
SpringApplication.run(ConsumerApplication.class, args);
PandoraBootstrap.markStartupAndWait();
}
}
```

可以看出，传统的 Spring Cloud 应用部署到 EDAS 上在代码层面只需要做一点小的修改，在SpringApplication.run()前后加上PandoraBootStrap启动相关的动作即可。

服务发布

首先使用注解@EnableDiscoveryClient激活服务发布和订阅的功能，发布的服务名在application.properties中配置

```
vipserver.register.doms=my-provider,sc-provider,service-name
```

注意@EnableDiscoveryClient既是服务提供者进行发布的开关，也是服务消费者订阅的开关。如果此工程只有服务消费者，没有服务提供者，应该在application.properties配置文件中将发布功能关闭。

```
vipserver.register.enabled=false
```

服务订阅

服务发现

VipServerClient已经自动集成到FeignClient、RestTemplate和AsyncRestTemplate中，自动实现了服务列表

维护和自动更新等功能，使用者无需关心，直接使用即可。

服务消费

FeignClient模式

在FeignClient中，如果想使用某服务，只需在@FeignClient的name字段指定服务名即可。

```
 @FeignClient(name = "my-provider")
public interface IDemoApi {

@RequestMapping(value = "/test", method = RequestMethod.GET)
public String echo(@RequestParam("content")String content);

@RequestMapping(value = "/test", method = RequestMethod.POST)
public String pay();

}
```

注意此name应该为vipserver.register.doms中的一个，其他配置与原生的 Spring Cloud 服务无差别。

RestTemplate模式

使用RestTemplate 访问微服务

直接将域名改为服务名即可

```
    @RequestMapping(method = RequestMethod.GET)
public String consumerEchoMethod(String content) {
String msg = restTemplate.getForObject("http://my-provider/test?content=" + content, String.class);
LOGGER.info("returnMsg:" + msg + ",content:" + content);
return msg;
}
```

部署

本地部署

本地部署时，必须先启动轻量级配置中心，同时，将hosts中的jmenv.tbsite.net地址，设为轻量级配置中心所在的ip地址，详情见安装轻量配置中心

使用**本地轻量级配置中心**时，**必须**通过在启动时添加JVM参数来指定如下配置：

- -Daddress.server.domain=127.0.0.1 (轻量级配置中心的ip地址)
- -Daddress.server.port=8080 (地址服务器端口)

- -Dvipserver.client.port=8080 (Vipserver服务的监听端口)

在 IDE 中，通过 main 方法直接启动。

本地打包 FatJar ，通过 JAVA 命令启动。

排除 taobao-hsf.sar 依赖启动方式（加入-D 指定 SAR 位置）

```
#java -jar -Dpandora.location=/Users/yizhan/.m2/repository/com/taobao/pandora/taobao-hsf.sar/dev.3.5/taobao-hsf.sar-dev.3.5.jar spring-cloud-provider-0.0.1-SNAPSHOT.jar
```

注意 ： -Dpandora.location 指定的路径必须是全路径

不排除 taobao-hsf.sar 依赖启动方式（通过插件设置）通过 pandora-boot-maven-plugin 插件，把 excludeSar 设置为 false ，默认是 true ，打包时就会自动包含该 SAR 包。

```
<plugin>
<groupId>com.taobao.pandora</groupId>
<artifactId>pandora-boot-maven-plugin</artifactId>
<version>2.1.6.3</version>
<configuration>
<excludeSar>false</excludeSar>
</configuration>
<executions>
<execution>
<phase>package</phase>
<goals>
<goal>repackage</goal>
</goals>
</execution>
</executions>
</plugin>
```

直接启动

```
java -jar spring-cloud-provider-0.0.1-SNAPSHOT.jar
```

EDAS 中部署

在应用管理中，创建的应用需要选择支持 FatJar 功能的容器（3.2.5及以上）。选择部署应用，上传 FatJar 应用即可部署。那么以后默认该应用只能上传 FatJar 了，不再支持 WAR 。

线上服务发布

以当前 Demo 为例，打包流程如下：

使用命令提示符(Win 环境)或 SHELL 终端（ *nix 环境 ）进入到示例工程目录。

打包 itemcenter-api：cd itemcenter-api && mvn clean install && cd ../

打包 detail 工程：cd detail && mvn clean package && cd ../

打包 itemcenter 工程：cd itemcenter && mvn clean package && cd ../

完成了生产者和消费者开发并通过测试之后，先进行服务打包，然后需要将服务发布到线上。下面以当前 Demo 为例，简述线上发布的流程。

注意： EDAS RPC服务需要使用12200端口，因此请确保服务所在机器该端口可以被服务消费者访问。

发布服务

由于有两个应用（ detail.war 和 itemcenter.war ）要发布，所以至少需准备两台机器，且两台机器上均需要 **安装 EDAS Agent**，假设两台机器的名字分别为 (edas-detail和edas-itemcenter)，部署应用的步骤为：

登录 EDAS **控制台**，在左侧导航栏中选择 **服务市场** > **服务分组**，在界面右上角点击 **创建服务组**。

在弹出的对话框中，填入要发布服务组别 (Spring 配置文件中标签<hsf:provider/> 中的 group 属性)。

选择左侧菜单栏的 **应用管理**，在页面右上角点击 **创建应用**。在弹出的界面中选择应用所在区域并填入正确的应用名后，单击 **下一步**。

在机器列表中选择相应的机器；在这里我们创建两个应用， app-detail 和 app-itemcenter，对应部署的机器分别为 edas-detail 和 edas-itemcenter。

创建好应用后，进入 **应用管理**，分别进入应用 app-detail 和 app-itemcenter。单击右上角的 **部署应用**，在弹出的对话框中，选择在 **服务打包** 中打好的 WAR 包上传并部署。

部署完毕后，单击页面右上角的 **启动应用**。

应用在机器上启动完毕之后，在应用基本信息页面的 **实例信息** 区域，可以看到对应的机器状态，当机器的实时状态为 **正常** 且任务状态为 **运行中** 时，说明应用在机器上已经启动成功。

查看发布的服务

1. 两个应用均启动成功后，在 **应用管理** 界面选择应用 app-temcenter，进入应用。

   在应用界面的左侧菜单栏中，选择 **服务列表**，然后再选择 **发布的服务** 选项卡，可以看到 Spring 配置文件中所定义的发布的服务：com.alibaba.edas.carshop.itemcenter.ItemService

   同上，选择 app-detail 进入应用，可以看到 Spring 配置文件中所定义的消费的服务：com.alibaba.edas.carshop.itemcenter.ItemService

   进入机器 edas-detail，用 admin 身份进入到 AliTomcat 的 logs 目录下（假设路径为：/home/admin/taobao-tomcat/logs/）时，可以在 catalina.out 的输出中看到有如下的 log 信息：Item[id: 1, nam: Mercedes Benz] ，这正是从服务提供者 (app-itemcenter) 处返回的 Item 对象。

EDAS 中的 Dubbo 开发

Dubbo 是阿里巴巴集团开源的一个分布式 RPC 框架，具体的介绍和开发指南请参阅官方文档。本节主要介绍 Dubbo 在 EDAS 中的使用流程及相关说明。

主要包括以下内容：

JAR 转换 WAR。

配置 Dubbo。

（可选）多注册中心兼容

使用 Dubbo 发布应用。

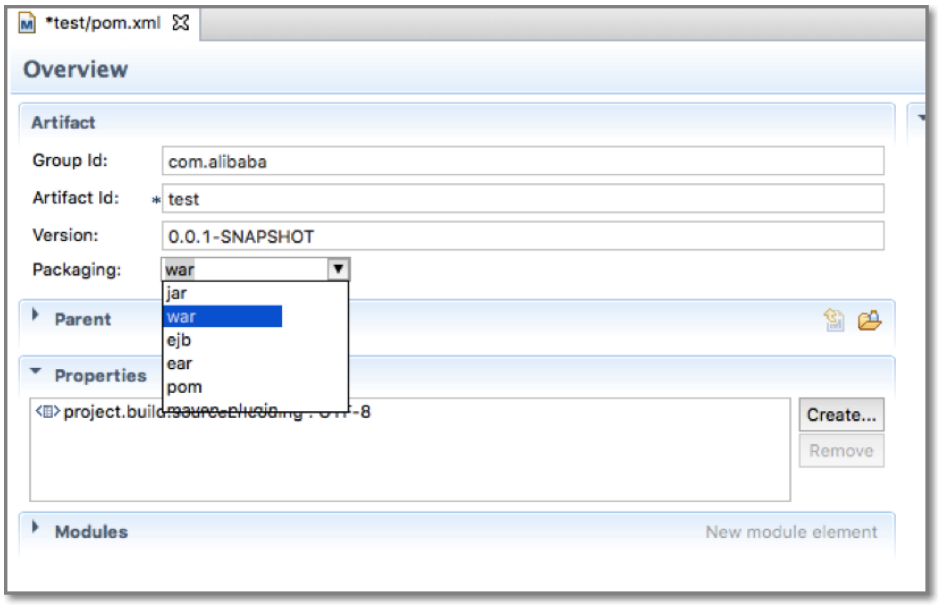
检查和 HSF 的兼容性。

开始在 EDAS 中开发 Dubbo 服务之前，请确保您已经了解并掌握了 Dubbo 项目的开发，且对 Dubbo 相关的参数和属性有一定的了解。

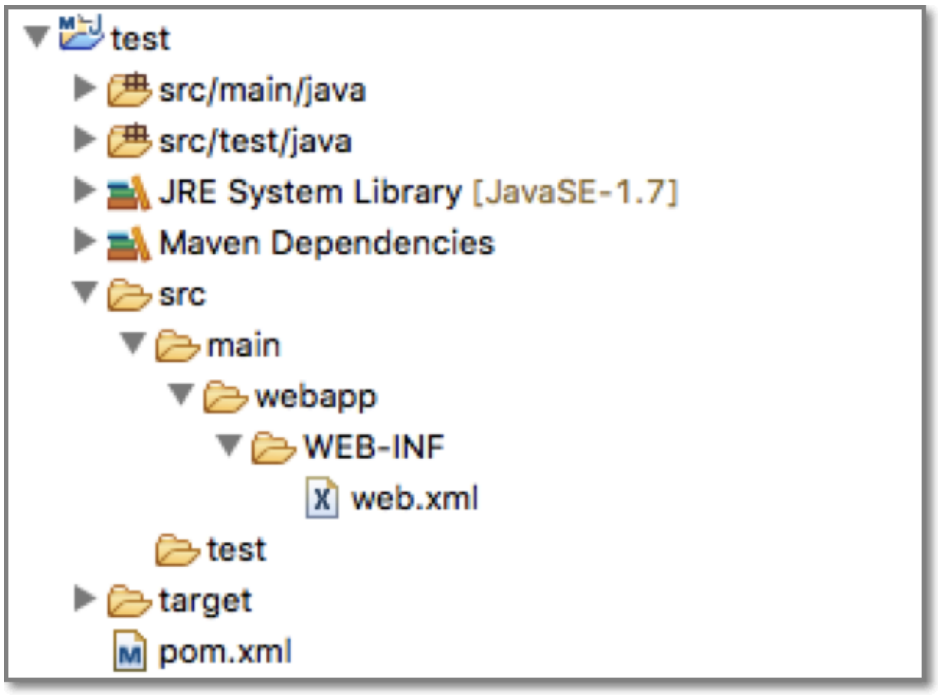
目前 EDAS 产品只支持 WAR 形式的 Web 项目，所以如果你的项目是 JAR 方式发布的，需要先进行转换。本文主要基于 Maven 项目来做示例。

步骤如下：

修改 POM 文件 JAR 为 WAR。



如果没有 web.xml，则需要增加一个 web.xml 文件配置。



配置 web.xml 加载配置文件。

```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>classpath:hsf-provider-beans.xml</param-value>
</context-param>

<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
<listener>
<listener-class>
org.springframework.web.context.request.RequestContextListener
</listener-class>
</listener>
```

目前 Dubbo 在 EDAS 中运行支持两种配置服务提供者和服务消费者的方式：XML 配置、注解配置。本文档提供这两种方式的配置示例。

XML 文件配置方式

以下是 Dubbo XML 配置示例，设置正确则不需要做修改即可直接放入 EDAS 中运行。

服务生产者 XML 配置

```
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
<dubbo:application name="edas-dubbo-demo-provider" ></dubbo:application>
<bean id="demoProvider" class="com.alibaba.edas.dubbo.demo.provider.DemoProvider" ></bean>
<dubbo:registry address="zookeeper://127.0.0.1:2181" ></dubbo:registry>

<dubbo:protocol name="dubbo" port="20880" threadpool="cached"
threads="100" ></dubbo:protocol>

<dubbo:service delay="-1" interface="com.alibaba.edas.dubbo.demo.api.DemoApi"
ref="demoProvider" version="1.0.0" group="dubbogroup" retries="3" timeout="3000"></dubbo:service>

</beans>
```
```

注意：

- 可选配置包括 threadpool、threads、delay、version、retries、timeout，其他均为必选配置。配置项可以任意调换位置。
- Dubbo 的 RPC 协议支持多种方式，如 RMI， hessian 等，但是目前 EDAS 的适配方案只支持了 Dubbo 协议，如：<dubbo:protocol name="dubbo" port=“20880” >，否则会引起类似于：” com.alibaba.dubbo.config.ServiceConfig service [xx.xx.xxx] contain xx protocal，HSF not supportted” 的错误发生。

服务消费者 XML 配置

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
<dubbo:application name="edas-dubbo-consumer" />
<dubbo:registry address="zookeeper://127.0.0.1:2181" />
<dubbo:reference id="demoProviderApi"
interface="com.alibaba.edas.dubbo.demo.api.DemoApi" version="1.0.0" group="dubbogroup" lazy="true"
loadbalance="random">
<!-- 指定某个方法不用等待返回值 -->
<dubbo:method name="sayMsg" async="true" return="false" />
</dubbo:reference>
<bean id="demoConsumer" class="com.alibaba.edas.dubbo.demo.consumer.DemoConsumer"
init-method="reviceMsg">
<property name="demoApi" ref="demoProviderApi"></property>
</bean>
</beans>
```

注意：

- 可选配置包括 version、group、lazy、loadbalance、async、return，其他选项为必须。配置项可以任意调换位置。
- 注册中心在 EDAS 中是不生效的，所有 Dubbo 的服务会自动注册到 EDAS 的配置中心，用户无需关心。
- 由于 Dubbo 配置文件消费者可以指定多个分组，而 EDAS 目前只能通过 group 属性配置一个分组，无法指定多个分组。
- 当有业务需要在程序启动过程中加载服务，则需要设置lazy=true，进行延迟加载

注解配置方式

从 EDAS 容器 V3.0 版本开始，已经对 Dubbo 原生注解进行支持了，用户无需进行注解转换 XML 即可使用 EDAS 服务。

兼容说明：

- 服务发布注解： @Service
- 服务订阅注解： @Reference

支持属性： group、version、timeout

使用方式： 在创建容器的时候，选择最新版本容器 V3.0 即可。

多注册指 Dubbo/HSF 应用可以同时注册服务到 EDAS、ZooKeeper 注册中心，为其他消费者提供服务。

多订阅指 Dubbo/HSF 应用去消费一个服务时，可以同时订阅 EDAS、ZooKeeper 注册中心中的服务。

使用方式

在当前应用中加入不低于1.5.1的 edas-sdk 依赖。

```
<dependency>
<groupId>com.alibaba.edas</groupId>
<artifactId>edas-sdk</artifactId>
<version>1.5.1</version>
</dependency>
```

指定 ZooKeeper 注册/订阅中心地址。指定方式主要包含以下两种：

环境变量指定（支持 HSF 、Dubbo 应用）：

-Dhsf.registry.address=zookeeper://IP地址:端口

XML 指定方式（只支持 HSF 应用）：

<hsf:registry address="zookeeper://IP地址:端口" />

指定 ZooKeeper 地址后 Dubbo 应用默认会启动双注册和订阅。HSF 应用若需要启用双注册/订阅，还需要设置调用参数 invokeType 。

- 只注册/订阅 ConfigServer 中的服务：invokeType="hsf"
- 只注册/订阅 ZooKeeper 中的服务: invokeType="dubbo"
- 双订阅/注册: invokeType="hsf，dubbo"

创建应用时，需要选择不低于3.0版本的容器，然后上传启动即可。

使用 Dubbo 发布 Web 项目有两种方式。

通过右键直接启动 Tomcat4E 插件来启动 Web 项目。

这种方式常用于测试环境中，直接在 IDE 中运行项目，比较简单，无需过多配置，如果是多个项目只

需要保证 Tomcat 的端口不要重复即可。Tomcat4E 插件配置请参考文档 Ali-Tomcat 安装 介绍。

通过 EDAS 控制台 来发布 Web 项目（WAR 包）。

对照下表，检查 Dubbo 配置文件中的服务属性与 HSF 的兼容情况。检查完配置兼容性之后，即可按照前面文档介绍的内容进行应用的调试与发布。

功能特性	Dubbo 配置参数	兼容情况说明	错误提示	EDAS是否支持
超时	timeout			支持
延迟暴露	delay			支持
线程模型	dispatcher= “all” threadpool= “fixed” threads= “100”			支持
回声测试				支持
延迟	lazy= “true”	默认开启		支持
连接				
本地调用	protocol= “injvm”			支持
隐式传参				支持
并发控制	actives= “10” executes= “10 ”	已经实现 EDAS 控制台可视化配置，无需配置		支持
连接控制	accepts= “10” connections=” 2”	已经实现 EDAS 控制台可视化配置，无需配置		支持
服务降级		已经实现 EDAS 控制台可视化配置，无需配置		支持
集群容错	retries/cluster	支持 retries	无报错	部分支持
负载均衡	loadbalance	默认 random	无报错	部分支持
服务分组	group	不支持 * 配置	java.lang.IllegalStateException: hsf2 不支持同时消费多个分组!	部分支持
多版本	version	不支持 * 配置	[HSF-Consumer] 未找到需要调用的服务的目标地址	部分支持
异步调用	async= “true”	return 参数无效	无报错	部分支持

	return=”false”			
启动时检查	check	EDAS 默认是启动不检查	无报错	默认支持启动不检查
多协议		只支持 Dubbo 协议	com.alibaba.dubbo.config.ServiceConfig服务[com.alibaba.demo.api.DemoApi] 配置了 RMI 协议， HSF2 不支持	部分支持
路由规则		已经实现 EDAS 控制台可视化配置，无需配置		支持
配置规则		已经实现 EDAS 控制台可视化配置，无需配置		支持
多注册中心				不支持
分组聚合	group=“aaa,bbb” merger=“true”	报错	java.lang.IllegalStateException: hsf2 不支持同时消费多个分组!	不支持
上下文信息		报错	Caused by: java.lang.UnsupportedOperationException: not support getInvocation method in hsf2	不支持

使用 Jenkins 实现 EDAS 持续集成

使用 Jenkins 可以构建 EDAS 应用的持续集成方案。该方案涉及下面的计算机语言或开发工具，阅读本文需要对下述的语言或工具有一定的理解。

工具	**说明**
Maven	Maven 是一个项目管理和构建自动化工具。
Jenkins	Jenkins 是一个可扩展的持续集成引擎。
GitLab	GitLab 是一个利用 Ruby on Rails 开发的开源应用程序，实现一个自托管的Git 项目仓库，可通过 Web 界面进行访问公开的或者私人项目。 它拥有与 GitHub 类似的功能，能够浏览源代码，管理缺

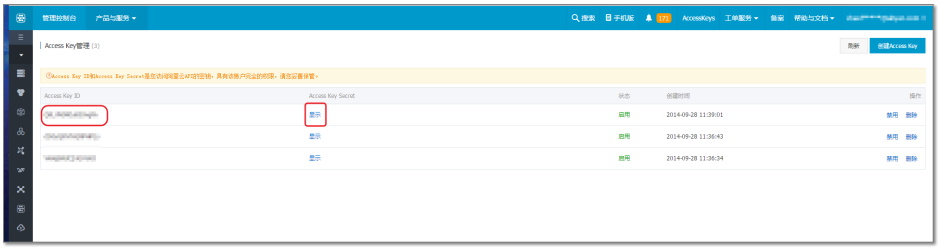
	陷和注释。

在开始持续集成之前，需要完成下述的准备工作。

获取阿里云的 Access Key ID 和 Access Key Secret。

使用已经开通了 EDAS 服务的 **主账号** 登录阿里云官网。

进入 Access Key 控制台，创建 Access Key ID 和 Access Key Secret。



在 EDAS 控制台创建应用。

在使用 Jenkins 自动部署应用之前，需要先在 EDAS 控制台中创建一个可以部署的应用。

登录 EDAS控制台。

参考 发布应用 ，创建应用。

如果已经创建了应用，请忽略此步。

在左侧导航栏中单击 **应用管理**。找到您在上一步中创建的应用并单击进入详情页面，获取应用 ID 的字段内容。

使用 GitLab 托管您的代码。

可以自行搭建 Gitlab 或者使用 阿里云 Code。

本文使用通过自行搭建的GitLab做演示，关于Gitlab的更多信息请参考 GitLab。

了解并使用 Jenkins。

关于 Jenkins 的更多详细信息请参考 Jenkins官网。

目前，阿里云还没有合适的产品替代 Jenkins，不过即将推出基于Jenkins 的 DevOps 平台，请持续关注。

创建持续集成主要包含以下三个步骤：

[TOC]

安装和配置Jenkins

安装 Jenkins。

安装Jenkins，请参考 Jenkins。 如已安装则请忽略此步。

在 Jenkins 服务器安装 Python 运行环境（仅支持2.7及以上版本，不支持 Python3）。

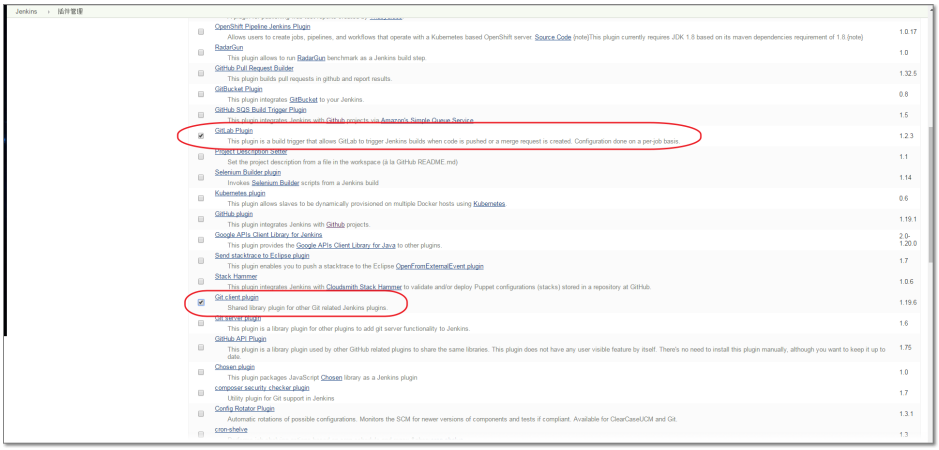
安装 Python，请参考 Python。 如已安装请忽略此步。

在 Jenkins 中安装 Git 和 GitLab 插件。

在 Jenkins 控制台的菜单栏中选择 **系统管理** > **插件管理**，安装插件。

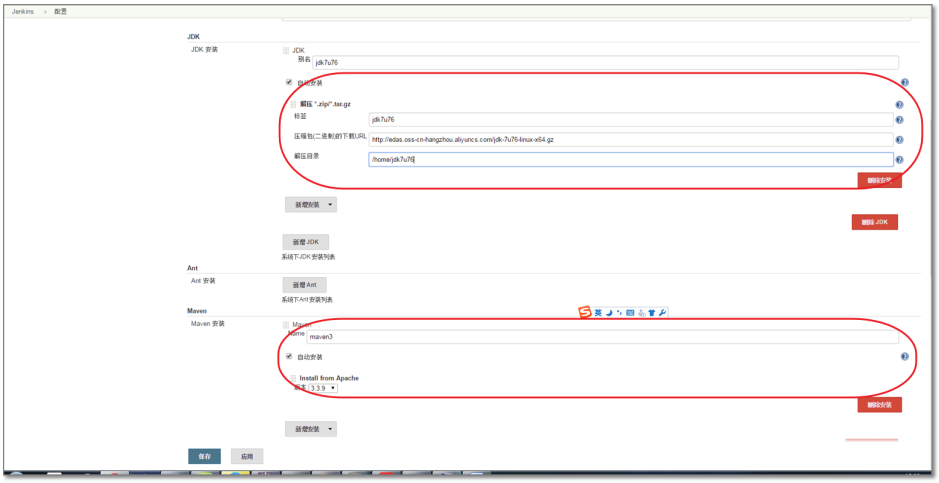
安装 GIT Client Plugin 和 GIT Plugin 插件可以帮助 Jenkins 拉取 Git 仓库中的代码。

安装 Gitlab Hook Plugin 插件可以帮助Jenkins在收到Gitlab发来的Hook后触发一次构建。



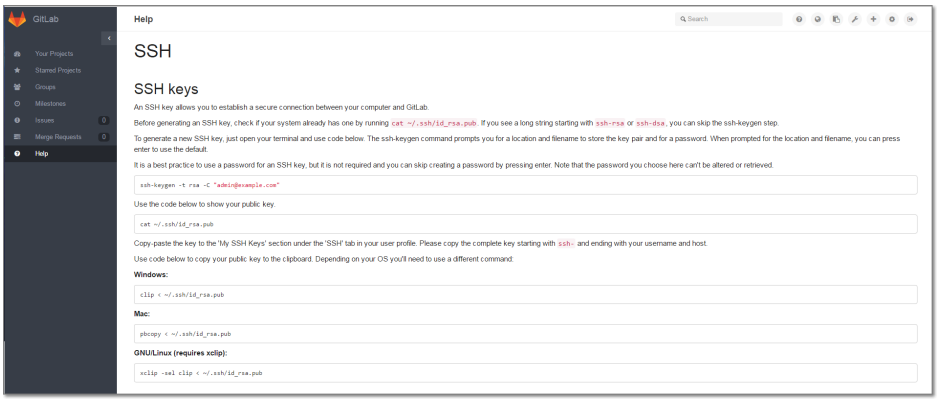
安装 JDK 和 Maven。

在 Jenkins 控制台的菜单栏中选择 **系统管理** > **系统设置**，参考下图中的标示为 Jenkins 安装 JDK 和 Maven。

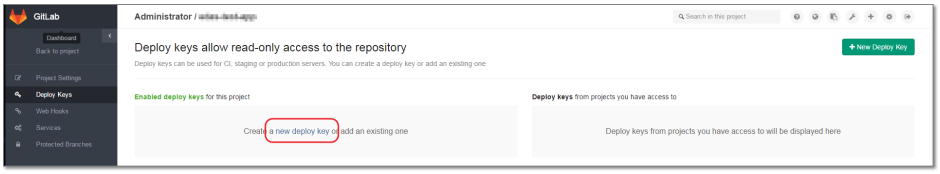


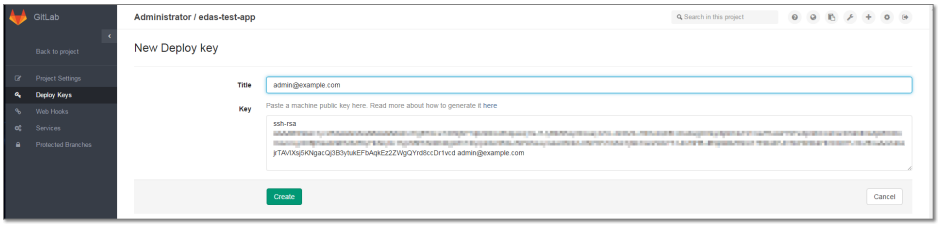
生成 RSA 密钥对，导入 GitLab 和 Jenkins。 实现 Jenkins 拉取 GitLab代码时的认证。

参考 GitLab 文档，创建 RSA 密钥对。



进入您的项目的 GitLab 首页，在菜单栏选择 **Settings** > **Deploy Keys**。然后单击 **new deploy key** 添加key，导入密钥。



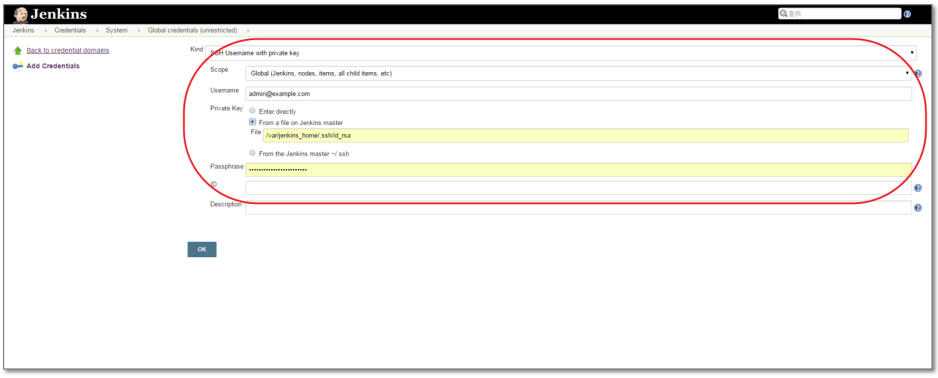


通过 RSA 私钥添加 Jenkins 认证。

在 Jenkin s首页单击 **Credentials** 菜单。 单击 **Add credentials**，在下图页面输入相关信息，单击 **OK**。

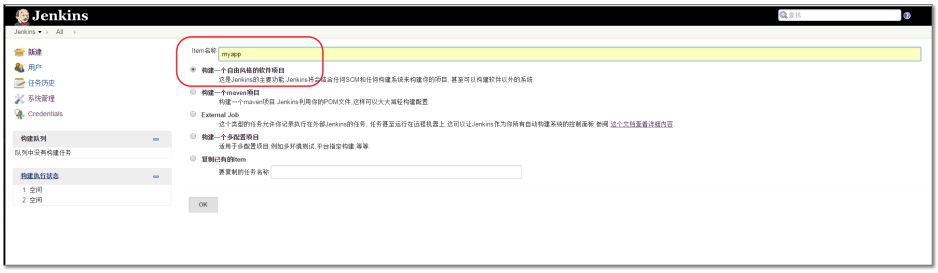
选择 **SSH Username with private key** 的认证方式。

按照图例填写 **Scope**，**UserName**， **Private Key** 等配置， 将第一步生成 RSA 密钥对中生成的私钥文件拷贝到"/var/jenkins_home/.ssh/id_rsa"文件。 图例中的 Scope、UserName、Private Key 可以根据您的需要来填写。

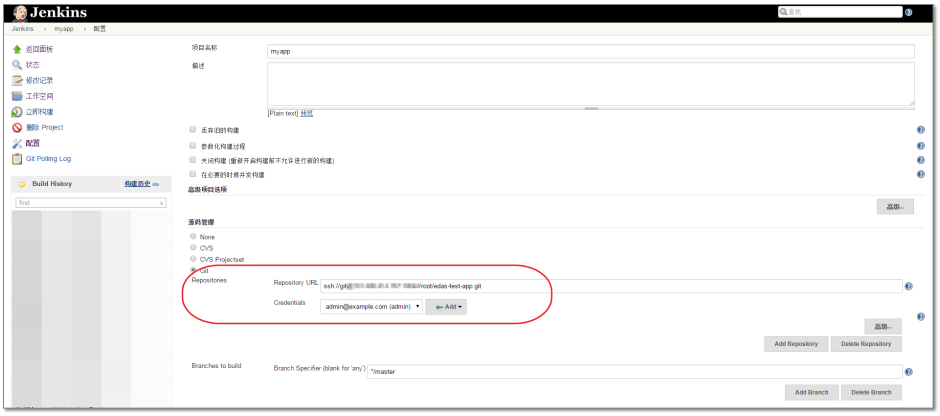


创建 Jenkins 项目。

在 Jenkins 首页左侧单击 **新建**，创建 Jenkins 项目。



配置 Git 项目地址时，勾选上一步中通过 RSA 创建的认证方式。正确配置后如下图所示。“Poll SCM” 必须勾选。



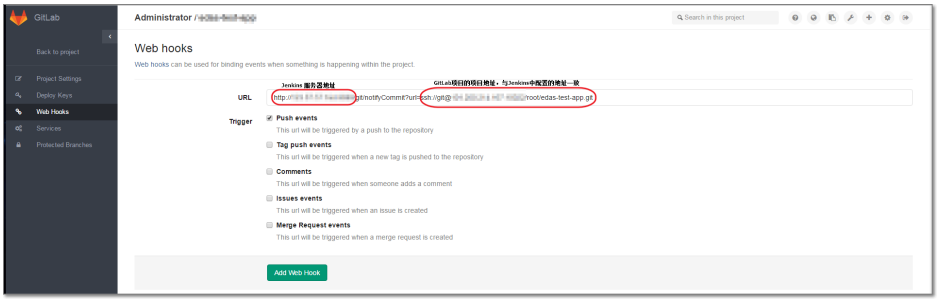
配置 Maven 构建和自定义构建动作 **Execute shell** （本文示例通过调用 Shell 命令完成构建后的自动部署，如果您的 Jenkins 是在 Windows 服务器上搭建的，则需要选择 **Execute Windows batch command** ）。

上图中配置的*"python $WORKSPACE/build/build.py"*是一个 EDAS 提供的示例脚本，该脚本主要完成 WAR 包的上传和应用的部署工作。具体如何使用，在接下来的第三步、第四步中有详细说明。

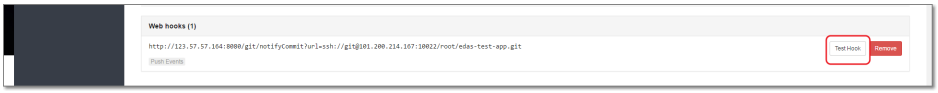
注：如果部署的是 Docker 应用，脚本"python $WORKSPACE/build/build.py" 请使用 "python $WORKSPACE/build/dockerbuild.py"

配置Gitlab的Web Hook，实现自动构建

点击 GitLab 工程进入配置（"Settings"）页面，参考下图进行配置。 图中表示的 Jenkins 服务器地址为您的 Jenkins 服务器的 Web 访问地址如 http://localhost:8080/。



配置完成后可以点击页面中的 "Test Hook" 进行测试。



调用 EDAS Open API 进行部署

下载示例工程demo.zip。

拷贝示例中的 build 目录到您的 Git 工程中。

打开 build 目录中的.osscredentials 文件，配置为您在准备工作中获取的 Access Key ID 和 Access Key Secret。

```
[OSSCredentials]
accessid = 您的Access Key ID
accesskey = 您的Access Key Securet
```

由于部署分为普通应用和 Docker 应用，由于 Open API 不相同，配置方式不同，分别介绍：

普通应用

打开 build 目录中的 config.json 文件，配置 WAR 包地址，应用 ID 等属性。 配置文件格式满足 JSON 格式。

```
{
    "edas": {
        "host": "edas.console.aliyun.com",
        "port": 80
    },
    "apps": [
        {
            "appName": "jenkins-consumer",
            "appId": "b8ce22a3-4554-4385-bb89-f43848eedea1",
            "userId": [redacted],
            "target": "edas-demo-portal/target/portal.war",
            "deployGroupId":"",
            "batch":1
        },
        {
            "appName": "jenkins-provider",
            "appId": "faf4ffb1-91a7-42bf-9ea1-4a764106695d",
            "userId": [redacted],
            "target": "edas-demo-service/target/service.war",
            "deployGroupId":"",
            "batch":1
        }
    ]
}
```

"host"配置项代表了您访问的 EDAS 的控制台的域名或者IP，如果您使用的是公有云的 EDAS 服务，则无需修改"edas"的"host"和"port"属性。

"apps"配置项中可以配置多个应用，也可以只配置一个。 各配置项的含义及获取方式如下：

- appName：应用名称，准备工作中创建，通过 EDAS 控制台可以取到。
- appId：应用 ID，准备工作中创建，通过 EDAS 控制台可以取到。
- userId；您登录阿里云的用户 ID。
- target：Maven 编译后打出来的 WAR 的本地路径，以 Git 项目为根目录的相对目录。
- deployGroupId：应用分组 ID。
- batch：分批发布。

Docker 应用

Docker 应用支持 WAR 部署和镜像部署（其中镜像部署只需要在 Jenkins 项目中配置 Maven 构建和自定义构建动作 **Execute shell**，内容为"python $WORKSPACE/build/dockerbuild.py"。 ）

打开 build 目录中的 dockerconfig.json 文件，配置 WAR 包地址，应用 ID 等属性。 配置文件格式满足 JSON 格式。

```
{
    "edas": {
        "host": "edas.console.aliyun.com",
        "port": 80
    },
    "apps": [
        {
            "appName": "doc-jenkins-consumer",
            "appId": "6e941be8-8c9e-4cb9-9ac8-6b3263e99173",
            "userId": [redacted],
            "target": "edas-demo-portal/target/portal.war",
            "type": "upload",
            "deployToStr": "all",
            "packageVersion":"1.3",
            "description": "",
            "imageUrl":"",
            "regionId":"cn-hangzhou"
        },
        {
            "appName": "doc-jenkins-provider",
            "appId": "9b670366-9ba0-4336-88da-1d8a9bad5773",
            "userId": [redacted],
            "target": "",
            "type": "image",
            "deployToStr": "all",
            "packageVersion":"1.3",
            "description": "",
            "imageUrl":"registry.cn-hangzhou.aliyuncs.com/edas_test1/onenight:1",
            "regionId":"cn-hangzhou"
        }
    ]
}
```

"apps"配置项中可以配置多个应用，上图配置了两个应用，第一个为 Docker 应用 WAR 包部署方式，第二个为 Docker 应用镜像部署方式，各配置项的含义及获取方式如下：

- appName：应用名称，准备工作中创建，通过 EDAS 控制台可以取到。
- appId：应用 ID，准备工作中创建，通过 EDAS 控制台可以取到。
- userId；您登录阿里云的用户 ID。
- type：部署方式类型。upload 为 WAR 包部署，image 为镜像部署。
- target：Maven 编译后打出来的 WAR 的本地路径，WAR 部署不能为空。
- imageUrl：镜像地址。image部署时，不能为空。
- packageVersion：部署包的版本号。
- description：描述信息。
- deployToStr：应用分组 ID。 为"all"时，代表该应用所有应用实例。
- regionId：区域 ID，应用所在的区域ID。

配置正确后，提交变更到 GitLab。

如果上述步骤配置正确，这次提交会触发一次 GitLab Hook。 Jenkins 在接受到这个 Hook 后会构建您的 Maven 项目，并在构建结束时调用 Open API 触发部署。

说明：如为 Docker 镜像部署方式，配置完成后，可手动触发 jenkins 项目。

开发中遇到的很多问题都可以通过查看相关日记进行定位，进而解决问题。下表是 EDAS 相关日记路径汇总。

日志文件名	日志文件含义
/home/admin/taobao-tomcat-production-xxxx/logs/catalina.out	最重要的日志，可以看到应用和 Tomcat 应用服务器的异常，这是开发最应该关注的日志。

/home/admin/taobao-tomcat-production-xxxx/logs/localhost.log.xxx	如果通过 catalina.out 看到的错误很模糊或者没有错误，可以结合 localhost 来一起看，保证应用正常启动，再看下面的日志继续排查问题。
/home/admin/configclient/logs/config.client.log	可以通过此日记查看服务发布订阅是否成功，“Register-ok”、“Publish-ok” 等关键字。
/home/admin/logs/hsf/hsf.log	HSF 服务的日志，有 HSF 服务调用过程的一些详细信息。如果 HSF 调用中出现异常，可以看看这个日志。

作为应用的容器，简单的排查问题的思路就是：

先查看 /home/admin/taobao-tomcat-production-xxxx/logs/catalina.out

再查看 /home/admin/taobao-tomcat-production-xxxx/logs/localhost.log.xxx

找到 Tomcat 相关的第一个错误，解决第一个错误后重启观察, 看是否还有其他错误。