阿里再开源！基于JAVA的模块化开发框架JarsLink

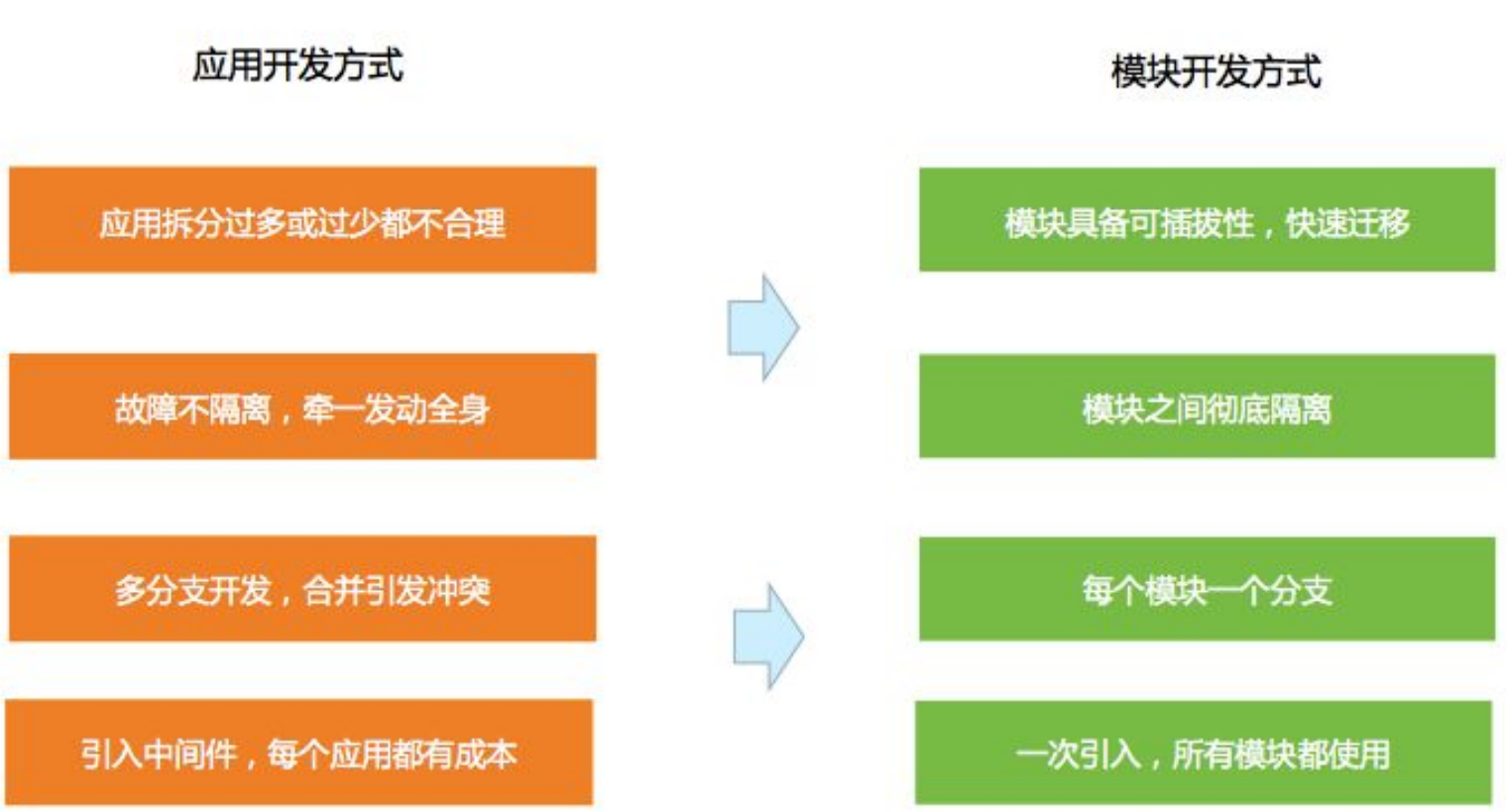
阿里妹导读：JarsLink是一个基于JAVA的模块化开发框架，它提供在运行时动态加载模块（JAR包）、卸载模块和模块间调用的API，它能够帮助你进行模块化开发，也能帮助你的系统在运行时动态添加新功能，减少编译、打包和部署带来的发布耗时，同时它也是阿里巴巴的开源项目之一，目前在蚂蚁金服微贷事业群各团队广泛使用。

开源地址：https://github.com/alibaba/jarslink

需求背景

- 应用拆分的多或少都有问题。多则维护成本高，每次发布一堆应用。少则拆分成本高，无用功能很难下线。
- 故障不隔离。当一个系统由多人同时参与开发时，修改A功能，可能会影响B功能，引发故障。
- 多分支开发引发冲突。多分支开发完之后合并会产生冲突。
- 牵一发动全身。一处核心代码的改动，或一个基础Jar的升级需要回归整个系统。
- 升级和迁移成本高。中间件升级每个应用都有升级成本。

模块化开发的好处



- 可插拔，一个应用由多个模块组成，应用里的模块可拆和合，模块可快速在多个系统中迁移和部署。
- 模块化开发，模块之间互相隔离，实现故障隔离。
- 一个模块一个分支，不会引发代码冲突。

- 在模块中增加或修改功能，只会影响当前模块，不会影响整个应用。
- 动态部署，在运行时把模块部署到应用中，快速修复故障，提高发布效率。
- 多版本部署，可以在运行时同时部署某个模块的新旧版本，进行AB TEST。
- 减少资源消耗，通过部署模块的方式减少应用数量和机器数量。

JarsLink的应用场景

- 数据管理中心，如果你需要开发一个数据管理系统，这个系统需要去不同的异构系统采集数据，这些系统会提供不同类型的接口，如RPC，HTTP等。并且数据采集的数据源多，每种数据源都需要对接和开发，数据质量比较差，需要经常修改代码进行发布。在这种场景下，通过模块化开发，实现一个数据源使用一个模块进行对接，上线新数据源只需要新增模块，修改BUG只需要修改某个模块，并能快速上线。
- 后台管理系统，互联网应用发展到一定阶段会出现很多后台需求，如客服查询用户的信息帮助解答问题，开发查后台数据排查系统BUG，运营使用后台功能发送运营活动等。这些功能发布频率会大于核心系统，如果放在核心系统里会影响其稳定性，所以我们必须要建一个后台系统来开发后台功能，但是这样又带来一个新的问题，很多开发都会来这个系统进行开发，拉多分支造成代码冲突，A业务的BUG影响到B业务。所以如果每个业务线一个模块，每个模块使用一个单独的分支进行开发，就能进行隔离开发，提高开发速度，开发完后在运行时加载到系统中。
- 微服务集成测试, 目前一个微服务是一个FAT JAR,如果有几十个微服务,则需要启动很多进程,DEBUG端口会很多,使用JarsLink框架合并FAT JAR,再路由请求到其他JAR,就可以只启动一个进程进行DEBUG测试。
- 指标计算系统，可以把消息转发到不同的模块中进行处理，并输出指标。

目前蚂蚁金服微贷事业部几个系统和几十个模块已经使用JarsLink框架。

JarsLink的特性

隔离性

- 类隔离：框架为每个模块的Class使用单独的ClassLoader来加载，每个模块可以依赖同一种框架的不同的版本。
- 实例隔离：框架为每个模块创建了一个独立的Spring上下文，来加载模块中的BEAN，实例化失败不会影响其他模块。
- 资源隔离：后续会支持模块之间的资源隔离，每个模块使用独立的CPU和内存资源。

动态性

- 动态发布：模块能在运行时动态加载到系统中，实现不需要重启和发布系统新增功能。支持突破双亲委派机制，在运行时加载父加载器已经加载过的类，实现模块升级依赖包不需要系统发布。
- 动态卸载：模块能在运行时被动态卸载干净，实现快速下线不需要功能。

易用性

提供了通用灵活的API让系统和模块进行交互。

实现原理

模块加载

JarsLink为每个模块创建一个新的URLClassLoader来加载模块。并且支持突破双亲委派，设置了overridePackages的包将由子类加载进行加载，不优先使用父类加载器已加载的。

模块的卸载

卸载模块需要满足三个条件：

- 模块里的实例对象没有被引用
- 模块里的Class没有被引用
- 类加载器没有被引用

所以需要做到三点卸载实例，卸载类和卸载类加载器，整个模块的卸载顺序如下：

- 关闭资源：关闭HTTP连接池或线程池。
- 关闭IOC容器：调用applicationContext.close()方法关闭IOC容器。
- 移除类加载器：去掉模块的引用。

- 卸载JVM租户（开发中）：卸载该模块使用的JVM租户，释放资源。

模块间隔离

模块化开发需要解决隔离性问题，否则各模块之间会互相影响。模块之间的隔离有三个层次：

- 类隔离：为每个模块创建一个类加载器来实现类隔离。
- 实例隔离：为每个模块创建一个新的IOC容器来加载模块里面的BEAN。
- 资源隔离：对每个模块只能使用指定的CPU和内存。

目前JarsLink实现了类隔离和实例隔离，资源隔离准备引入ALIJVM多租户来解决。

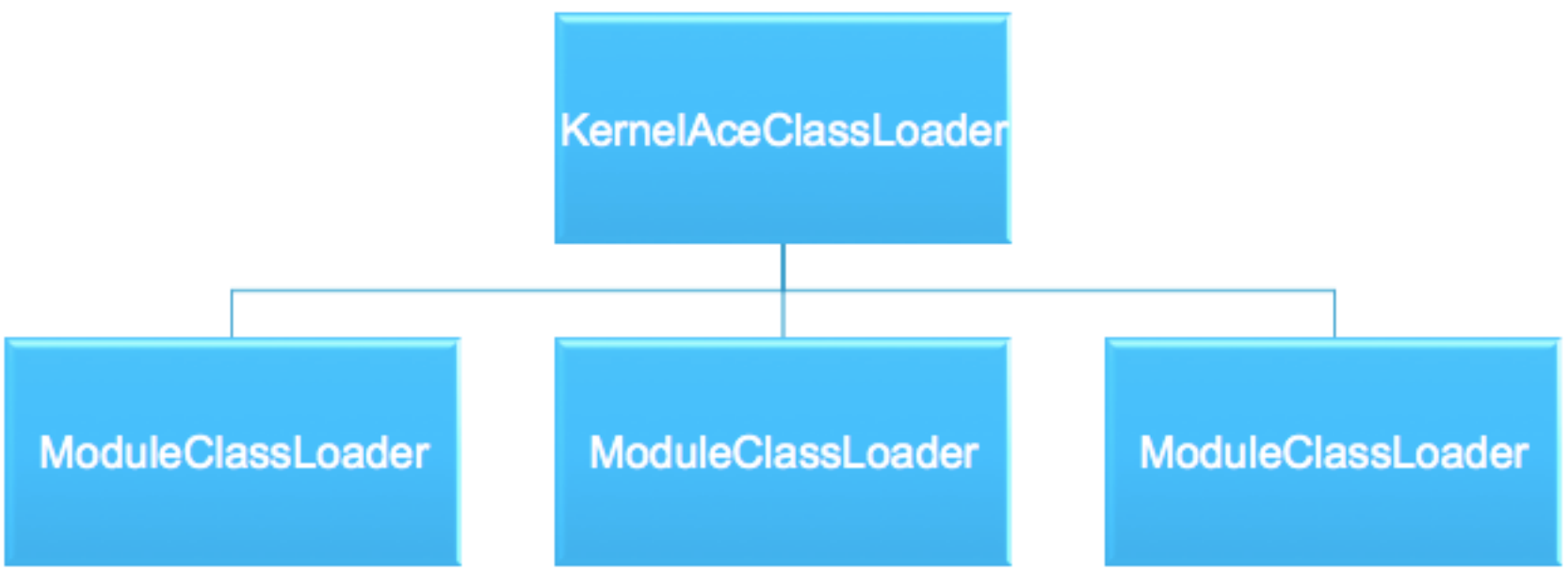
模块间通讯

模块之间的通讯也有三种方式，RPC，本地调用，深克隆/反射。

- 本地调用：目前JarsLink的doAction就是使用的这种通讯方式，这种方式要求模块的类加载器是父子关系，且IOC容器也是父子容器。
- RPC调用：用于跨JVM的模块之间调用，利用SOFA 4动态API在模块中发布和引用TR服务来实现。
- 深克隆/反射：深克隆其他模块的入参，反射其他模块的方法实现调用。

类加载机制

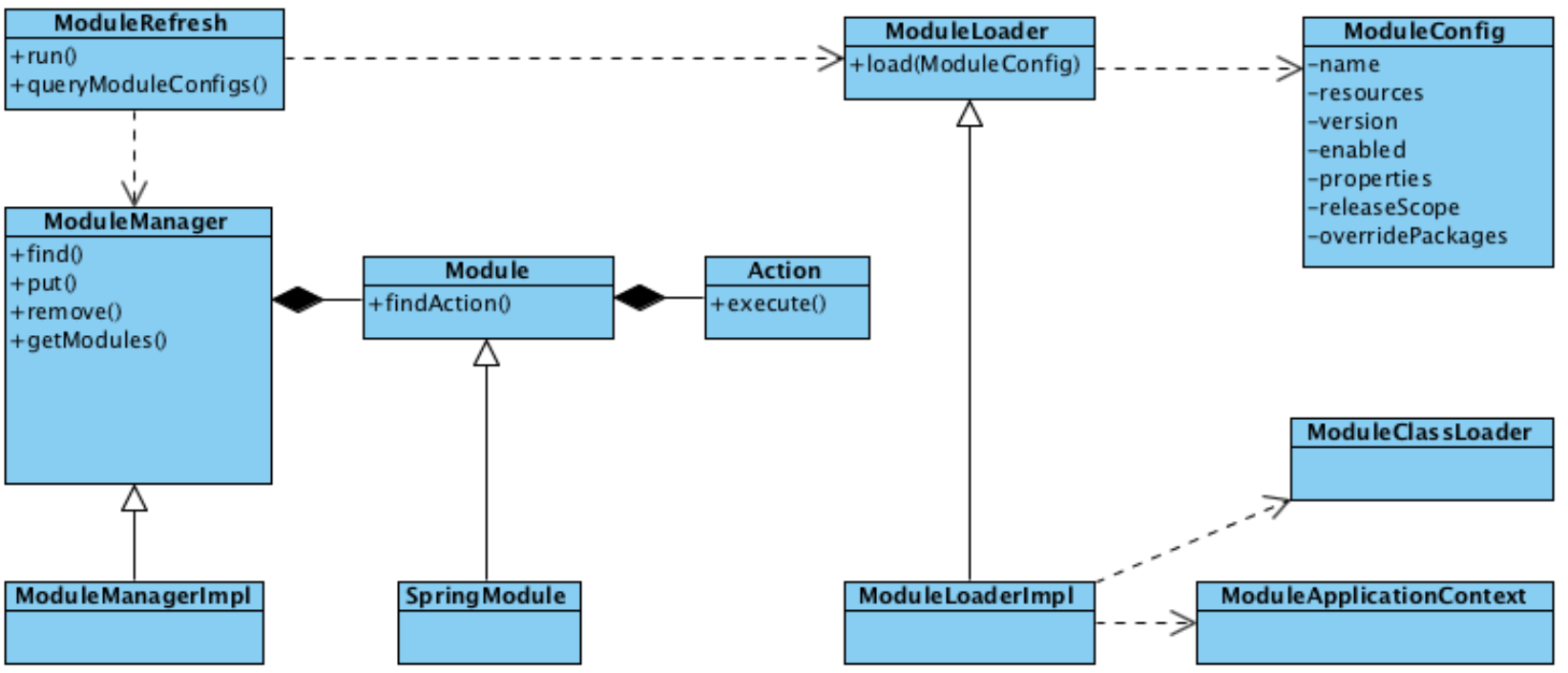
OSGI类加载机制的关系采用的是网状结构，每个模块通过 Export-Package 来声明我要给别人用哪些类，通过 Import-Package来声明我要用别人的哪些类。JarsLink采用扁平化管理，每个模块都有一个共同的父类，这个父类加载器就是加载ModuleLoader类的加载器，如果是SOFA应用，模块的父加载器是KernelAceClassLoader，类加载器关系如下：



如果所有模块都需要使用的类，可以通过KernelAceClassLoader加载，如果是SOFA系统可以通过POM引入。

JarsLink框架类图

JarsLink框架的类图如下：



- AbstractModuleRefreshScheduler：入口类，负责定期扫描本地和内存中的模块是否发生变更，如果变更，则更新模块。
- ModuleLoader：模块加载引擎，负责模块加载。
- ModuleManager：模块管理者，负责在运行时注册，卸载，查找模块和执行Action。
- Module：模块，一个模块有多个Action。
- Action：模块里的执行者。

如何使用?

1：引入POM

JarsLink Maven Repo

```
<dependency>
  <groupId>com.alipay.jarslink</groupId>
  <artifactId>jarslink-api</artifactId>
  <version>1.5.0.20180213</version>
</dependency>
```

JarsLink依赖的POM也需要引入

```
<properties>
    <slf4j.version>1.7.7</slf4j.version>
    <apache.commons.lang.version>2.6</apache.commons.lang.version>
    <apache.commons.collections.version>3.2.1</apache.commons.collections.version>
</properties>


<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>${org.springframework.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${org.springframework.version}</version>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>1.1.3</version>
    </dependency>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.3.2</version>
    </dependency>
    <dependency>
        <groupId>commons-lang</groupId>
        <artifactId>commons-lang</artifactId>
        <version>${apache.commons.lang.version}</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>${slf4j.version}</version>
    </dependency>
    <dependency>
        <groupId>commons-collections</groupId>
        <artifactId>commons-collections</artifactId>
        <version>${apache.commons.collections.version}</version>
    </dependency>
    <dependency>
        <groupId>com.google.guava</groupId>
        <artifactId>guava</artifactId>
        <version>17.0</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>
</dependencies>
```

2：引入jarslink BEAN

在系统中引入以下两个BEAN。

```
<!-- 模块加载引擎 -->
<bean name="moduleLoader" class="com.alipay.jarslink.api.impl.ModuleLoaderImpl"></bean>
<!-- 模块管理器 -->
<bean name="moduleManager" class="com.alipay.jarslink.api.impl.ModuleManagerImpl"></bean>
```

3：集成JarsLink API

使用JarsLink API非常简单，只需要继承AbstractModuleRefreshScheduler，并提供模块的配置信息，代码如下：

```
public class ModuleRefreshSchedulerImpl extends AbstractModuleRefreshScheduler {

    @Override
    public List<ModuleConfig> queryModuleConfigs() {
        return ImmutableList.of(ModuleManagerTest.buildModuleConfig());
    }

    public static ModuleConfig buildModuleConfig() {
        URL demoModule = Thread.currentThread().getContextClassLoader().getResource("META-INF/spring/demo-1.0.0.jar");
        ModuleConfig moduleConfig = new ModuleConfig();
        moduleConfig.setName("demo");
        moduleConfig.setEnabled(true);
        moduleConfig.setVersion("1.0.0.20170621");
        moduleConfig.setProperties(ImmutableMap.of("svnPath", new Object()));
        moduleConfig.setModuleUrl(ImmutableList.of(demoModule));
        return moduleConfig;
    }
```

这个调度器在bean初始化的时候会启动一个调度任务，每分钟刷新一次模块，如果模块的版本号发生变更则会更新模块。实现这个方法时，必须把模块（jar包）下载到机器本地，模块的配置信息说明如下：

- name：全局唯一，建议使用英文，忽略大小写。
- enabled：当前模块是否可用，默认可用，卸载模块时可以设置成false。
- version：模块的版本，如果版本号和之前加载的不一致，框架则会重新加载模块。
- Properties：spring属性配置文件。
- moduleUrl：模块的本地存放地址。
- overridePackages：需要突破双亲委派的包名,一般不推荐使用，范围越

小越好，如com.alipay.XX。

把ModuleRefreshSchedulerImpl类注册成Spring的bean。

```
<bean id="moduleRefreshScheduler"
      class="com.alipay.**.ModuleRefreshSchedulerImpl">
    <property name="moduleManager" ref="moduleManager" />
    <property name="moduleLoader" ref="moduleLoader" />
</bean>
```

JarsLink API 暂时不提供模块可视化管理能力，所以需要使用其他系统来管理和发布模块。目前可以通过com.alipay.jarslink.api.ModuleManager#getModules获取运行时所有模块的信息。

你也可以使用API来加载并注册模块，详细使用方式可以参考ModuleManagerTest，代码如下。

```
//1:加载模块
Module module = moduleLoader.load(buildModuleConfig());

//2:注册模块
ModuleManager moduleManager = new ModuleManagerImpl();
moduleManager.register(module);
```

3：开发模块

在模块中只需要实现并开发Action，代码如下：

```
public class HelloWorldAction implements Action<ModuleConfig, ModuleConfig> {

    @Override
    public ModuleConfig execute(ModuleConfig actionRequest) {
        ModuleConfig moduleConfig = new ModuleConfig();
        moduleConfig.setName(actionRequest.getName());
        moduleConfig.setEnabled(actionRequest.getEnabled());
        moduleConfig.setVersion(actionRequest.getVersion());
        moduleConfig.setModuleUrl(actionRequest.getModuleUrl());
        moduleConfig.setProperties(actionRequest.getProperties());
        moduleConfig.setOverridePackages(actionRequest.getOverridePackages());
        return moduleConfig;
    }

    @Override
    public String getActionName() {
        return "helloworld";
    }

}
```

5：调用接口

开发者需要利用JarsLink API把请求转发给模块，先根据模块名查找模块，再根据aciton name查找Action，最后执行Action。

```
//查找模块
Module findModule = moduleManager.find(module.getName());
Assert.assertNotNull(findModule);

//查找和执行Action
String actionName = "helloworld";
ModuleConfig moduleConfig = new ModuleConfig();
moduleConfig.setName("h");
moduleConfig.setEnabled(true);
ModuleConfig result = findModule.doAction(actionName, moduleConfig)
```

其他特性

Spring配置

通过moduleConfig的Properties属性可以设置Spring bean变量的配置信息。

1：定义变量

```
<bean id="userService" class="com.alipay.XX.UserService">
 <property name="url" value="${url}" />
</bean>
```

2：配置变量信息

```
Map<String, Object> properties = new HashMap<String, Object>();
properties.put("url", "127.0.0.1");
moduleConfig.setProperties(properties);
```

3：排除spring配置文件

```
Map<String, Object> properties = new HashMap<String, Object>();
properties.put("exclusion_confige_name", "text.xml");
moduleConfig.setProperties(properties);
```

排除多个文件用逗号分隔。

最佳实践

HTTP请求转发

可以把HTTP请求转发给模块处理。

```
private ModuleManager moduleManager;

@RequestMapping(value = "module/{moduleName}/{actionName}/process.json", method = { RequestMethod.GET,RequestMethod.POST })
public Object process(HttpServletRequest request, HttpServletResponse response) {

    Map<String, String> pathVariables = resolvePathVariables(request);

    String moduleName = pathVariables.get("moduleName").toUpperCase()
    String actionName = pathVariables.get("actionName").toUpperCase()
    String actionRequest = XXX;
    return moduleManager.doAction(moduleName,
            actionName, actionRequest);
}


private Map<String, String> resolvePathVariables(HttpServletRequest request) {
        return (Map<String, String>) request
            .getAttribute(HandlerMapping.URI_TEMPLATE_VARIABLES_ATTRIBUTE);
    }
```

消息请求转发

可以把消息转发给模块进行处理。遵循默认大于配置的方式，你可以把

TOPIC当做模块名，EventCode当做ActionName来转发请求。

接口说明

JarsLink框架最重要的两个接口是ModuleManager和ModuleLoader。

ModuleManager接口

ModuleManager负责注册，卸载，查找模块和执行Action。

```
import java.util.List;
import java.util.Map;

/**
 * 模块管理者, 提供注册,移除和查找模块能力
 *
 * @author tengfei.fangtf
 * @version $Id: ModuleManager.java, v 0.1 2017年05月30日 2:55 PM tengfei.fangtf Exp $
 */
public interface ModuleManager {

    /**
     * 根据模块名查找Module
     * @param name
     * @return
     */
    Module find(String name);

    /**
     * 获取所有已加载的Module
     *
     * @return
     */
    List<Module> getModules();

    /**
     * 注册一个Module
     *
     * @param module 模块
     * @return 新模块
     */
    Module register(Module module);

    /**
     * 移除一个Module
     *
     * @param name 模块名
     * @return 被移除的模块
     */
    Module remove(String name);

    /**
     * 获取发布失败的模块异常信息
     *
     * @return
     */
    Map<String, String> getErrorModuleContext();

}
```

ModuleLoader接口

ModuleLoader只负责加载模块。

```
public interface ModuleLoader {

    /**
     * 根据配置加载一个模块，创建一个新的ClassLoadr加载jar里的class
     *
     * @param moduleConfig 模块配置信息
     *
     * @return 加载成功的模块
     */
    Module load(ModuleConfig moduleConfig);

}
```

近期，JarsLink会支持多版本加载，并陆续支持模块间调用、资源隔离等特性。我们也希望更多的童鞋参与进来，让JarsLink帮助更多开发者提升效率。

End

你可能还喜欢

点击下方图片即可阅读

xMedia来了！支付宝客户端的智能化“武器”

知识图谱数据构建的“硬骨头”，

阿里工程师如何拿下？

如何用架构师思维解读区块链技术？

关注「阿里技术」

把握前沿技术脉搏