**Ikasoa入门使用手册**

*Version: 0.1-BETA3*

1. **概述**

Ikasoa是一款高性能轻量级的RPC框架,基于apache thrift开发,抛弃了原有的idl定义接口方式.客户端可以像调用本地接口那样去调用远程接口,并支持负载均衡,简化了服务定义,降低了学习成本.

*(源码地址: http://svn.ikamobile.com/sme\_ikasoa/trunk)*

1. **环境搭建**
2. Maven配置

需要配置Ikamobile的Nexus私服,并添加ikasoa的依赖:

pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

*......*

*<!-- 配置Nexus私服 -->*

<repositories>

<repository>

<id>ikamobile-nexus</id>

<name>ikamobile Nexus Repository</name>

<url>http://repo.ikamobile.cn:8081/nexus/content/groups/public/</url>

</repository>

<repository>

<snapshots>

<enabled>true</enabled>

</snapshots>

<id>public</id>

<name>Public Repositories</name>

<url>http://repo.ikamobile.cn:8081/nexus/content/repositories/sulei-snapshots/</url>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>ikamobile-nexus</id>

<name>ikamobile Nexus Repository</name>

<url>http://repo.ikamobile.cn:8081/nexus/content/groups/public/</url>

</pluginRepository>

<pluginRepository>

<id>public</id>

<name>Public Repositories</name>

<url>http://repo.ikamobile.cn:8081/nexus/content/repositories/sulei-snapshots/</url>

</pluginRepository>

</pluginRepositories>

<distributionManagement>

<repository>

<id>ikamobile-nexus</id>

<name>Internal Releases</name>

<url>http://repo.ikamobile.cn:8081/nexus/content/repositories/releases/</url>

</repository>

<snapshotRepository>

<id>ikamobile-nexus</id>

<name>ikamobile Snapshots</name>

<url>http://repo.ikamobile.cn:8081/nexus/content/repositories/snapshots</url>

</snapshotRepository>

</distributionManagement>

*......*

<dependencies>

*......*

*<!-- 在这里添加对ikasoa的依赖 -->*

<dependency>

<groupId>com.ikamobile</groupId>

<artifactId>ikasoa</artifactId>

<version>0.1-BETA3</version>

</dependency>

*......*

</dependencies>

</project>

1. 导入工程&编译代码

工程目录下命令行执行”mvn eclipse:eclipse”,并导入eclipse.(如果IDE非eclipse,则使用相对应的方式导入)

执行命令”mvn clean package”打包.

1. **“helloworld”**
2. 新建例子接口(ExampleService.java),对象(ExampleVO.java)和实现 (ExampleServiceImpl.java)类:

ExampleService.java

public interface ExampleService {

*// 查询对象*

public ExampleVO findVO(int id);

}

ExampleVO.java

public class ExampleVO {

private int id;

private String string;

public ExampleVO() {

}

public ExampleVO(int id, String string) {

this.id = id;

this.string = string;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getString() {

return string;

}

public void setString(String string) {

this.string = string;

}

}

ExampleServiceImpl.java

public class ExampleServiceImpl implements ExampleService {

@Override

public ExampleVO findVO(int id) {

return new ExampleVO(id, “helloworld”);

}

}

1. 创建执行类

HelloWorld.java

public class HelloWorld {

public static void main(String[] args) {

IkasoaFactory ikasoaFactory = new DefaultIkasoaFactory();

try {

*// 获取Ikasoa服务*

IkasoaServer ikasoaServer = ikasoaFactory.getIkasoaServer(ExampleServiceImpl.class, 9999);

*// 启动服务*

ikasoaServer.run();

*// 客户端获取远程接口实现*

ExampleService es = ikasoaFactory.getIkasoaClient(ExampleService.class, "localhost", 9999);

*// 客户端输出结果*

System.out.println(es.findVO(1).getString());

*// 停止服务*

ikasoaServer.stop();

} catch (IkasoaException e) {

e.printStackTrace();

}

}

}

1. 执行HelloWorld.java

如输出”helloword”则表示执行成功.

1. **使用实例**

例子程序需要使用到Spring框架.

1. 服务端例子

SpringBean.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:context="http://www.springframework.org/schema/context" xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.1.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.1.xsd">

*......*

*<!-- ikasoa服务端配置 -->*

<bean id="soaServer" class="org.sulei.example.ikasoa.SoaServer" init-method="run" destroy-method="stop">

<constructor-arg index="0" ref="ikasoaFactory"/>

<constructor-arg index="1">

<value>9993</value>*<!-- 设置服务开放端口 -->*

</constructor-arg>

</bean>

<bean id="ikasoaFactory" class="com.ikamobile.ikasoa.DefaultIkasoaFactory"/>

*......*

</beans>

SoaServer.java

package org.sulei.example.ikasoa;

import com.ikamobile.ikasoa.IkasoaException;

import com.ikamobile.ikasoa.IkasoaFactory;

import com.ikamobile.ikasoa.IkasoaServer;

public class SoaServer {

private IkasoaServer server;

public SoaServer(IkasoaFactory ikasoaFactory, int serverPort) throws IkasoaException {

*// 实现类必须最终类,不能是抽象类*

this.server = ikasoaFactory.getIkasoaServer(ExampleServiceImpl.class, serverPort);

*// 如果已有实例化后的对象(例如通过Spring注入的对象),则可以通过ImplClsCon类进行封装,Ikasoa将会直接引用该对,而不会重新实例化.例子如下:*

*// this.server = ikasoaFactory.getIkasoaServer(new ImplClsCon(ExampleServiceImpl.class, exampleServiceImpl), serverPort);*

*// 如有多个接口实现,可以传入List.例子如下:*

*// List<ImplClsCon> sList = new ArrayList<ImplClsCon>();*

*// sList.add(new ImplClsCon(ExampleServiceImpl.class));*

*// sList.add(new ImplClsCon(Example2ServiceImpl.class));*

*// IkasoaServer ikasoaServer = ikasoaFactory.getIkasoaServer(sList, port);*

System.out.println("服务端口:" + serverPort);

for (String key : this.server.getIkasoaServiceKeys()) {

System.out.println("加载服务:" + key);

}

}

public void run() {

server.run();

}

public void stop() {

server.stop();

}

}

1. 客户端例子

SoaClient.java

import com.ikamobile.ikasoa.DefaultIkasoaFactory;

public class SoaClient {

public static void main(String[] args) {

*// 如果接口之间有继承关系,则只需要配置子接口类*

*// 设置服务器地址为”hocalhost”,端口为9993*

ExampleService es = new DefaultIkasoaFactory().getIkasoaClient(ExampleService.class, "localhost", 9993);

*// 如果有多个服务提供者,服务地址也可以传入List,系统将自动执行负载均衡.例子如下:*

*// List<String> hostList = new ArrayList<String>();*

*// hostList.add("localhost");*

*// hostList.add("192.168.1.41");*

*// ExampleService es = new DefaultIkasoaFactory().getIkasoaClient(ExampleService.class, hostList, 9993);*

System.out.println(es.findVO(1).getString());

}

}

1. 执行SoaClient.java

如输出”helloword”则表示执行成功.

1. **ThriftIDL实例**
2. 客户端调用Thrift服务端例子

ThriftClientDemo.java

import org.apache.thrift.transport.TTransport;

import org.apache.thrift.transport.TTransportFactory;

import org.sulei.core.thrift.client.ThriftClient;

import org.sulei.core.thrift.client.ThriftClientConfiguration;

import com.ikamobile.ikasoa.DefaultIkasoaFactory;

import com.ikamobile.tmcs.controller.thrift.server.acceptor.GeneralThriftAcceptor;

public class ThriftClientDemo {

public static void main(String[] args) {

ThriftClientConfiguration configuration = new ThriftClientConfiguration();

configuration.setTransportFactory(new TTransportFactory()); *// 协议需要与服务端匹配*

ThriftClient thriftClient = new DefaultIkasoaFactory(configuration).getThriftClient("121.40.119.240", 9201); *// 配置Thrift的服务器地址和端口*

TTransport transport = null;

try {

transport = thriftClient.getTransport();

transport.open();

*// GeneralThriftAcceptor为IDL中配置的service*

GeneralThriftAcceptor.Client client = new GeneralThriftAcceptor.Client(

thriftClient.getProtocol(transport, "GeneralThriftAcceptor")); *// 参数"GeneralThriftAcceptor"为服务的key,如果没有则可以不传*

*// 打印结果*

System.out.println(client.getTmc(1));

} catch (Exception e) {

e.printStackTrace();

} finally {

transport.close();

}

}

}

1. **服务提供类型的选择**

Ikasoa默认使用Thrift作为服务类型的实现,但也提供了Netty以供选择.

1. 使用Thrift服务

*……*

IkasoaFactory ikasoaFactory = new DefaultIkasoaFactory();

*……*

1. 使用Netty服务

*……*

IkasoaFactory ikasoaFactory = new NettyIkasoaFactory();

*……*

1. **序列化方式的选择**

Ikasoa提供了3种序列化方式,分别为fastjson,kryo,xml,默认使用fastjson.

1. 选择fastjson作为序列化方式(默认)

*……*

IkasoaFactory ikasoaFactory = new DefaultIkasoaFactory();

*// 也可以写为如下方式:*

*// IkasoaFactory ikasoaFactory = new DefaultIkasoaFactory(ProtocolType.JSON);*

*……*

1. 选择kryo作为序列化方式

*……*

IkasoaFactory ikasoaFactory = new DefaultIkasoaFactory(ProtocolType.KRYO);

*……*

*(需要注意在0.1-BETA2版本中kryo序列化方式暂未对异常对象进行处理.)*

1. 选择xml作为序列化方式

*……*

IkasoaFactory ikasoaFactory = new DefaultIkasoaFactory(ProtocolType.XML);

*……*

1. 各序列化方式的简单性能测试对比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序列化方式** | **序列化后的字符串长度** | **100次连续序列化和反序列化耗时** | **1000次连续序列化和反序列化耗时** | **10000次连续序列化和反序列化耗时** | **适用场景** |
| fastjson | 45 | 422ms | 484ms | 609ms | 适合对象结构简单,访问频次较高的场景. |
| kryo | 28 | 172ms | 359ms | 969ms | 适合对象结构复杂,对响应时间要求较高的场景. |
| xml | 580 | 1407ms | 7812ms | 60859ms | 适合开发和测试场景. |

1. **功能对比**
2. 功能对比

|  |  |  |  |
| --- | --- | --- | --- |
|  | **thrift** | **ikasoa** | **dubbo** |
| *规模* | 中量级 | 轻量级 | 重量级 |
| *性能* | 非常好 | 非常好 | 非常好 |
| *开发复杂度* | 高 (需要学习ThriftIDL语法,并生成相应的代码才能进行调用) | 低 (配置简单,可像调用本地接口一样调用远程接口,同时兼容ThriftIDL) | 中 (可像调用本地接口一样调用远程接口,但配置相对复杂) |
| *跨语言* | 支持 | 支持 *(可通过ThriftIDL兼容方式支持跨语言,或者进行针对性的二次开发)* | 不支持 *(dubbox通过restful可支持跨语言)* |
| *序列化方式* | thrift | json(fastjson) *(默认)* kryo  xml | hessian *(默认)* json(fastjson) json(dubbo json) java serializable  *等* |
| *集群和软负载均衡* | 不支持 | 支持 | 支持 |
| *软负载均衡方式* | - | 轮询 *(默认)* 权重轮询 随机 *(并支持扩展)* | 随机 *(默认)* 权重轮询 最少活跃调用数 一致性hash *(并支持扩展)* |
| *协议* | thrift | thrift *(默认)* netty (nifty) | netty+hessian *(默认)* rmi http redis thrift *等,(并支持协议扩展)* |
| *协调服务支持* | 不支持 | 暂不支持 *(正式版计划将支持zookeeper)* | zookeeper multicast *等* |

[*sulei@ikamobile.com*](mailto:sulei@ikamobile.com) *| 2016-04*