HARPC简介

北京百分点-杭州研发中心
2015年09月18日

目录

- HARPC是什么
- 要解决的问题
- 设计思路
- 如何使用
- 可视化管理
- 我们的保证
- 使用案例
- 相关资料

HARPC是什么

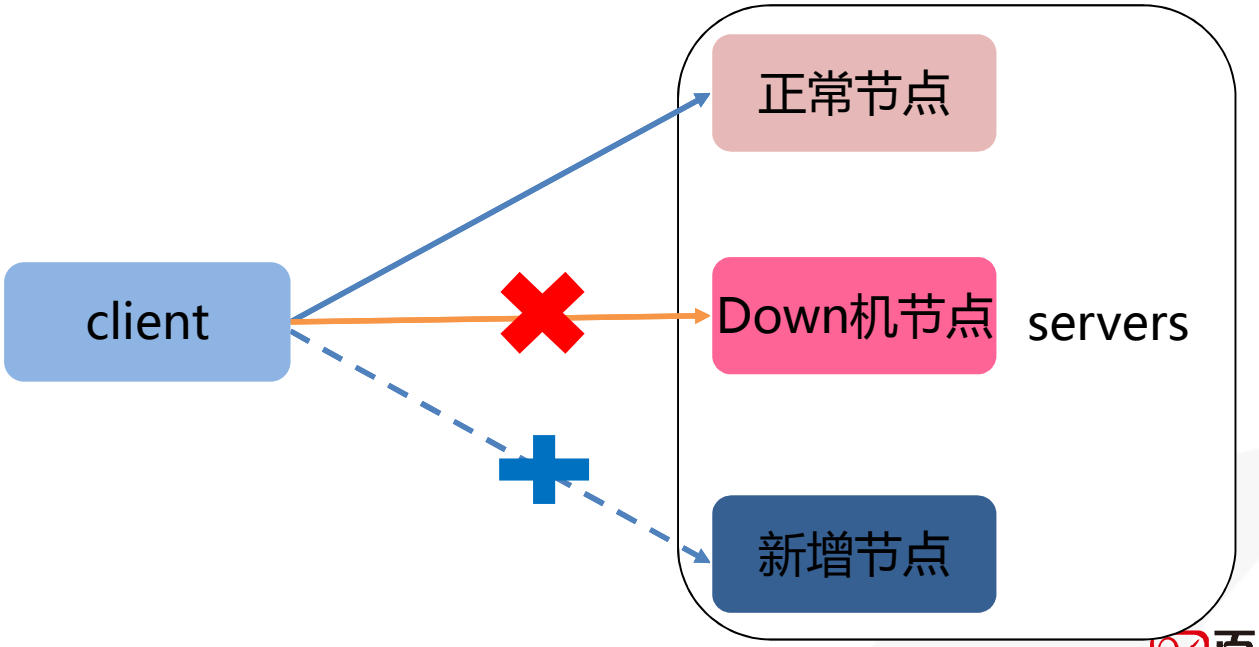
HARPC（High Availability RPC）是基于thrift的跨语言、高可用的RPC框架

要解决的问题

- 问题1：不同语言之间的通信问题
  - 实际场景中服务端和客户端往往使用不同的编程语言

要解决的问题

- 问题2：负载均衡和容灾处理
  - 如何实现任务的分布式处理？
  - 如何实现水平扩展，自动发现新的服务节点？
  - 如何处理节点的异常down机？



要解决的问题

- 问题3：服务维护成本高
  - 服务进程还在吗？
  - 服务请求量和性能怎么样？
  - 我们的服务有谁在调用？

```
2012-08-26 15:50:46,907 - __main__ - INFO - Add wait request for
2012-08-26 15:50:47,085 - __main__ - INFO - Handing write request
364373265636364363437363361303138306264373863666663373939, qname=
, flush=None
2012-08-26 15:50:47,087 - __main__ - INFO - Write 78 byte, msg_id
656363643634373633613031383062643738636666663373939, qname=demo-id
2012-08-26 15:50:51,666 - __main__ - INFO - Handing write request
432386437616436623462646138353862373238623237616363306563, qname=
 flush=None
2012-08-26 15:50:51,668 - __main__ - INFO - Write 16 byte, msg_id
376164366234626461383538623732386232376163306563, qname=demo-cp
2012-08-26 15:50:51,669 - __main__ - INFO - Notify 1 requests wai
```

设计思路

- 针对不同的需求，我们提供了两种方式，用户只需要通过配置来区分：
  1. 注册中心方式（**推荐**）

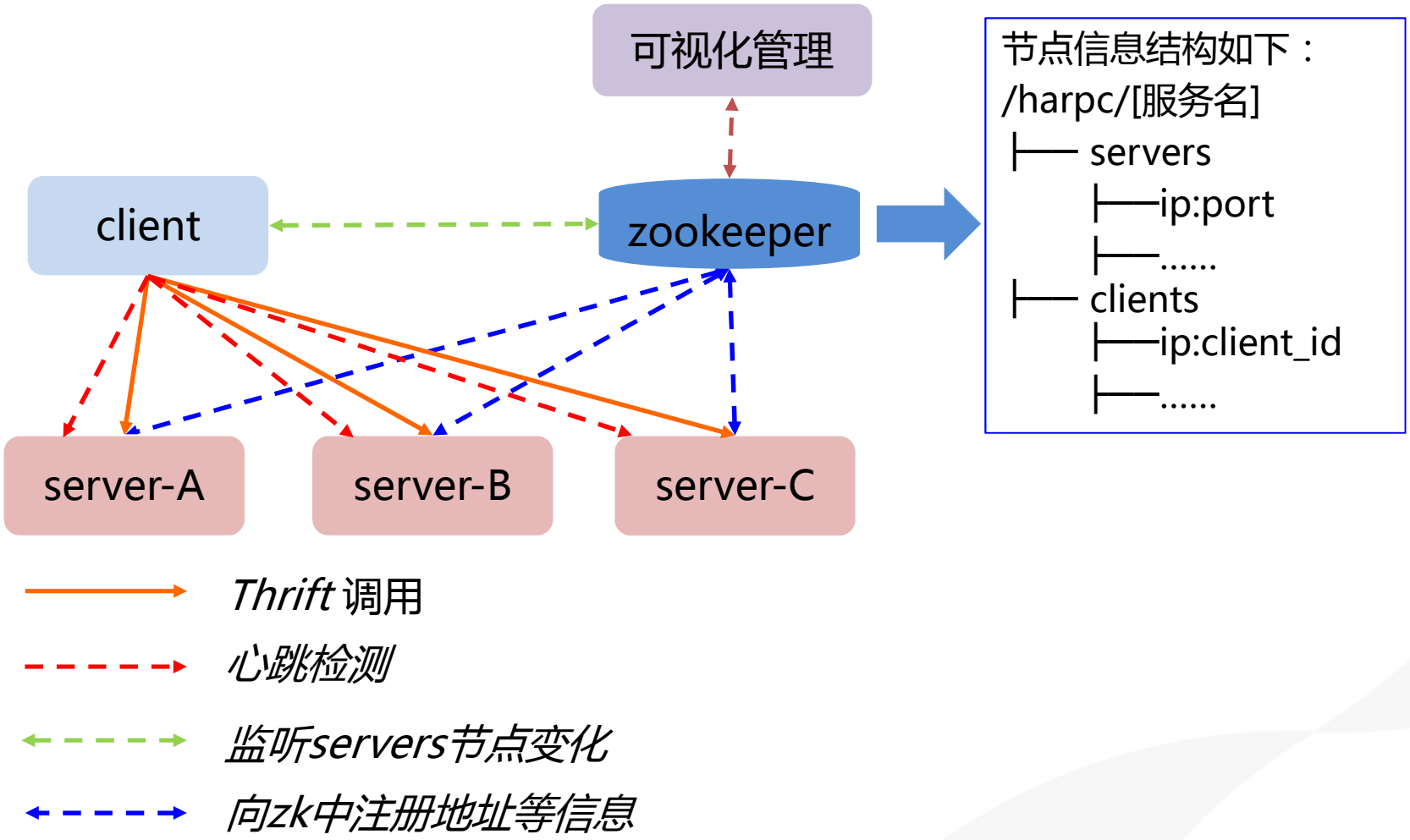
     这是推荐的方式，支持服务端自动注册、客户端服务自动发现的功能

  2. 直连方式

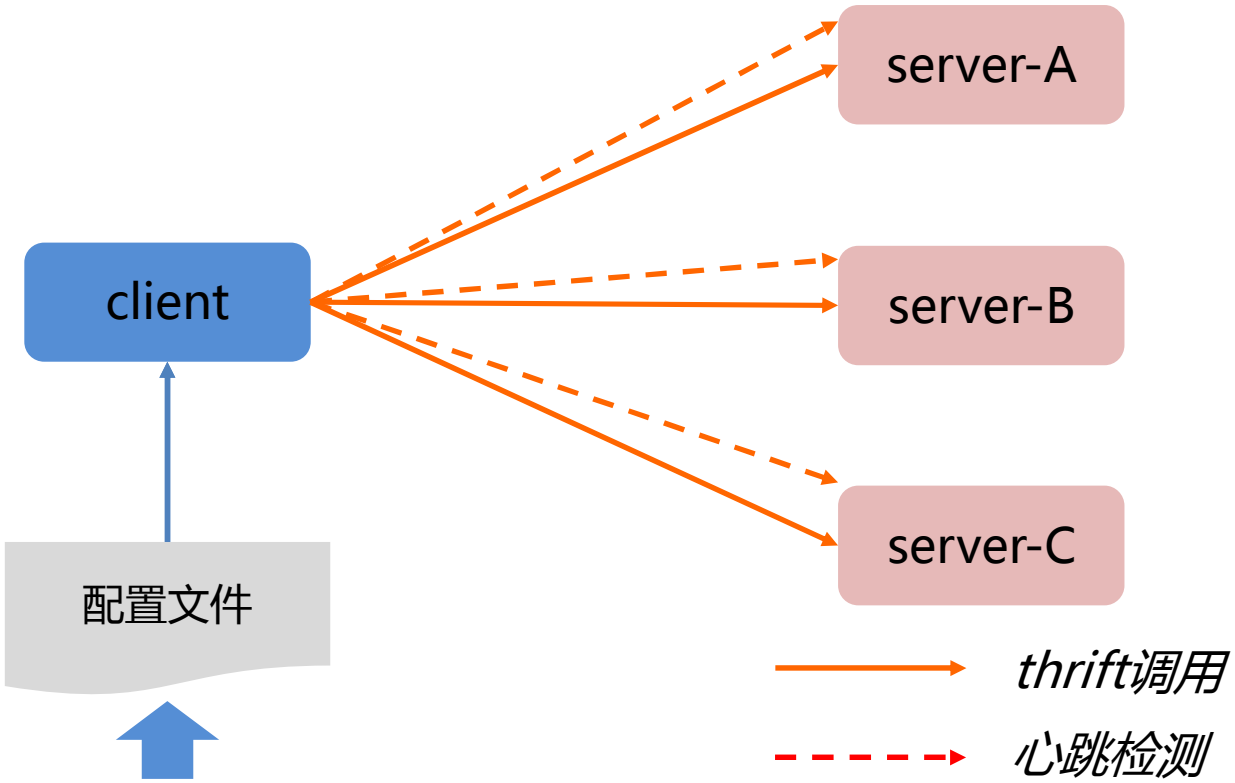
     适用于**第三方**的thrift服务或没有集成HARPC的服务

     注意：该方式**不支持自动发现**新的服务节点

设计思路-注册中心方式

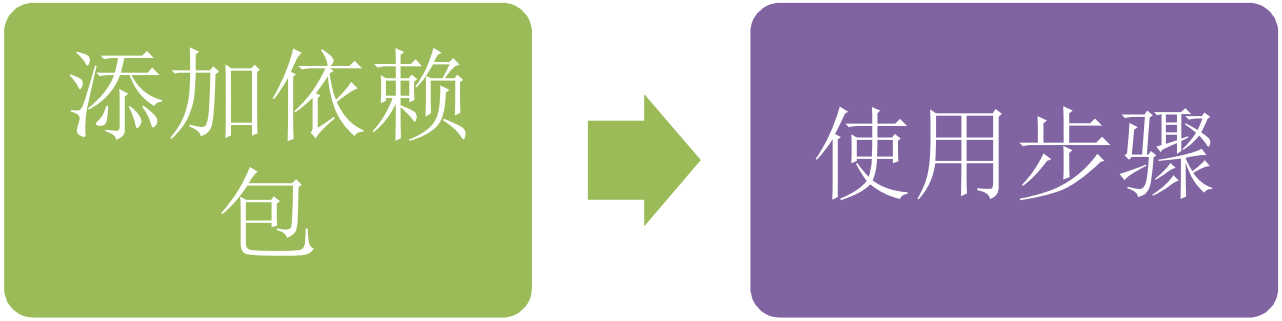

可视化管理
client
zookeeper
server-A
server-B
server-C
节点信息结构如下：
/harpc/[服务名]
├── servers
│ ├──ip:port
│ ├──.....
├── clients
│ ├──ip:client_id
│ ├──.....
Thrift 调用
心跳检测
监听servers节点变化
向zk中注册地址等信息

设计思路-直连方式



配置服务地址列表（ip:port;ip:port;...）,如：
address=172.18.1.1:19090;172.18.1.2:19090;172.18.1.3:19090

Java用户指南



添加依赖包

- 新建一个maven工程

  这里**推荐**使用maven工程，否则添加依赖会非常麻烦

- 选择最新的HARPC的版本

  通过下列地址查看最新的稳定版本：

  https://github.com/baifendian/harpc/releases

- 在pom文件中添加相关版本的依赖

```
<dependency>
    <groupId>com.baifendian</groupId>
    <artifactId>harpc</artifactId>
    <version>${version}</version>
</dependency>
```

使用步骤

（1）thrift 文件定义

```
# file: demo.thrift
namespace java com.bfd.harpc.demo.gen
service EchoService {
    string echo(1: string msg);
}
```

（2）thrift --gen java demo.thrift

将生成的gen-java文件夹下的文件拷贝到maven工程的src/main/java目录下

注意： Thrift程序的版本建议使用**0.9.2**及以上

使用步骤

（3）项目目录结构

```
src/main/java
├── com
│   ├── bfd
│       ├── harpc
│           ├── demo
│               ├── ServerDemo.java          #服务端
│               ├── ClientDemo.java          #客户端
│               ├── EchoServiceImpl.java     #接口实现类
│               ├── gen                      #thrift生成文件
│                   ├── EchoService.java
src/main/resources
├── server.properties                        #server配置文件
├── client.properties                        #client配置文件
pom.xml
```

使用步骤

（4） server端配置

```
#zookeeper连接字符串
registry.connectstr = 172.18.1.22:2181, 172.18.1.23:2181, 172.18.1.24:2181
#授权字符串，格式为：用户名:密码
registry.auth = admin:admin123
#zookeeper会话超时时间，单位ms
registry.timeout = 3000

#服务名(全称)：命名空间$服务名简称
server.service = com.bfd.harpc.demo$EchoService
#服务端口
server.port = 19090
#服务名
server.name = harpc-demo-server
#服务负责人
server.owner = dongsheng.fan@baifendian.com
#是否发送统计信息到zk
server.monitor = true
#发送的时间间隔，单位为s
server.interval = 60
……
```

使用步骤

（5） server代码示例

```
String[] configs = new String[] { “classpath:server.properties” };  // 配置文件路径
EchoServiceImpl impl = new EchoServiceImpl();

try {
        Server server = new Server(configs, impl);
        server.start(); // 启动服务，非阻塞

        // 阻塞主线程
        synchronized (ServerDemo.class) {
                while (running) {
                        try {
                                ServerDemo.class.wait();
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                        }
                }
        }
} catch (Exception e) {
        e.printStackTrace();
}
```

使用步骤

（6）client端配置

```
#zookeeper连接字符串
registry.connectstr = 172.18.1.22:2181, 172.18.1.23:2181, 172.18.1.24:2181
#zookeeper会话超时时间，单位ms
registry.timeout = 3000
#直连到不同的server,多个server以分号隔开（若配置address，则使用直连方式）
#address=172.18.1.22:19090;172.18.1.23:19090

#服务名
client.name = harpc-demo-client
#服务负责人
client.owner = dongsheng.fan@baifendian.com
#服务名(全称)：命名空间$服务名简称
client.service = com.bfd.harpc.demo$EchoService
#thrift生成文件的Iface接口
client.iface = com.bfd.harpc.demo.gen.EchoService$Iface
#client到server的超时时间，单位为ms
client.timeout = 10000
#重试次数
client.retry = 1
……
```

使用步骤

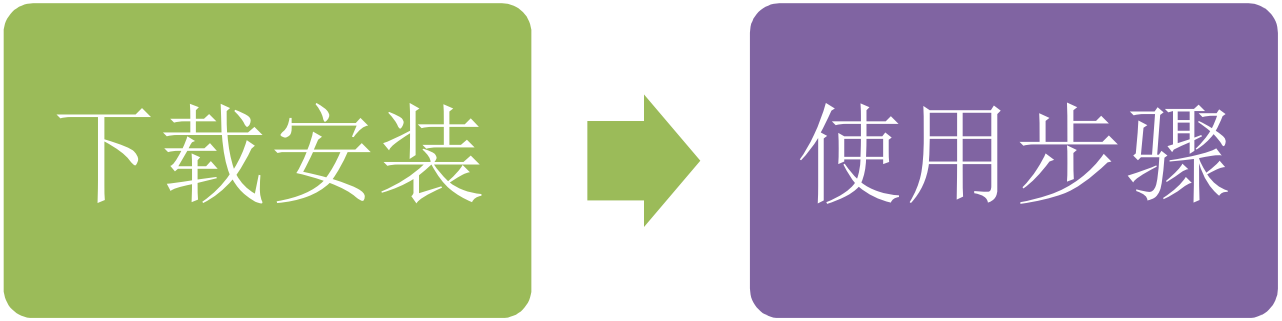
（7）client代码示例

```
String[] configs = new String[] { “classpath:client.properties” };

try {
        Client<Iface> client = new Client<Iface>(configs);
        // 注意:代理内部已经使用连接池，所以这里只需要创建一个实例，多线程共享
        // 特殊情况下，可以允许创建多个实例，但严禁每次调用前都创建一个实例
        Iface echoIface = client.createProxy();

        for (int i = 0; i < 1000; i++) {
                try {
                        System.out.println(echoIface.echo("world"));
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
} catch (Exception e) {
        e.printStackTrace();
}
```

Python用户指南



下载安装

- git clone -b master https://github.com/baifendian/harpc.git
  说明: -b 对应不同分支

- cd ha-rpc/rpc-python　sudo python setup.py install
  说明: 下载依赖 安装到python库中

- 版本依赖
  1. gevent>=1.0
  2. kazoo>=2.0
  3. thrift>=0.9

使用步骤

(1) thrift 文件定义

(2) thrift --gen py tutorial.thrift

(3) 示例项目的目录结构

```
#file tutorial.thrift
namespace py tutorial
service TutorialService {
    string echo(1:string str)
}
```

```
demo
├── client.py                          #客户端
├── direct_cliet.py                    #客户端直连
├── etc
│   ├── client.conf
│   ├── direct_client.conf
│   └── server.conf
├── gen-py                             #thrift生成的代码
│   ├── __init__.py
│   └── tutorial
│       ├── constants.py
│       ├── __init__.py
│       ├── __init__.pyc
│       ├── ttypes.py
│       ├── ttypes.pyc
│       ├── TutorialService.py
│       ├── TutorialService.pyc
│       └── TutorialService-remote
└── server.py                          #服务端
```

使用步骤

(4) server端配置

```
[server]
#服务名(全称)：命名空间$服务名简称, client端请求server的标识
service=python_test$EchoService
#server服务端口
port=9095
#zookeeper的连接字符串
zk_connect_str=172.18.1.22:2181,172.18.1.23:2181,172.18.1.23:2181
#server授权用户字符串 避免其他服务也注册此服务
auth_user=test
#server授权用户字符串 避免其他服务也注册此服务名
auth_password=test
#是否监控server的状态和请求数等信息， 默认值是False
monitor=True
#server服务的名称
name=EchoServiceServerDemo
#server 服务负责人
owner=wenting.wang@baifendian.com
#server 开启的进程数 默认值10个
process_num=10
#server 每个进程中的协程数目，默认值100个
coroutines_num=100
.....
```

使用步骤

(5) server代码示例

```
#thrift 接口实现
class EchoServiceHandler:
    def echo(self, msg):
        return msg

#配置加载
conf = config.Config("./etc/server.conf")

#TutorialService.Processor thrift生成的Processor
server_demo = server.GeventProcessPoolThriftServer(TutorialService.Processor,
                                                   EchoServiceHandler(), conf)
#服务启动
server_demo.start()
```

使用步骤

(6) client端配置

```
[client]
#是否使用zk，默认值是True，如果使用直连方式连接server，需要设置成False
use_zk=True
#直连到不同的server,多个server以分号隔开
direct_address=172.18.1.101:19999;172.18.1.102:19999
#zookeeper连接字符串
zk_connect_str=172.18.1.22:2181,172.18.1.23:2181,172.18.1.24:2181
#服务名(全称)：命名空间$服务名简称,client端请求server的标识
service=python_test$EchoService
#client服务的名称
name=EchoServiceClientDemo
#client服务负责人
owner=wenting.wang@baifendian.com
#client请求失败时，重试次数
retry=3
#client负载均衡策略，默认轮询
balance=bfd.harpc.loadbalancing_strategy.round_robin_strategy.RoundRobinStrategy
[loadbalancer]
#心跳的interval时间，默认值10秒
heartbeat_interval=10
[connection_pool]
#连接池大小
pool_size=10
```

使用步骤

(5) client代码示例

```
#读取配置文件
conf = config.Config( “./etc/client.conf” )

#创建实例
 manager = client.Client(TutorialService.Client, conf)

#创建代理实例
 proxy_client = manager.create_proxy()

#接口调用
 proxy_client.echo("hello world!")

#关闭
 manager.close()
```

C++用户指南

下载安装

- 简要说明

  c++ 目前不依赖zk，client端通过直连方式，稳定性高，经过了线上**两年**考验

- 下载

  git clone https://github.com/baifendian/harpc.git

- 编译工具

  cmake > 3.01

- 版本依赖

  1. boost>=1.55
  2. g++ >= 4.8.1
  3. thrift>=0.9 (建议0.92)

使用步骤

(1) thrift 文件定义

```
#file  echo.thrift
namespace cpp bfd.harpc.demo
service EchoService {
   string echo(1:string msg)
}
```

(2) thrift --gen py echo.thrift

将生成的gen-cpp文件夹下的文件拷贝到demo/thrift 中

注意： Thrift程序的版本建议使用0.9.2及以上

使用步骤

(3) 示例项目目录结构

```
demo
├── client                          # client示例
│   ├── conf
│   │   └── conf.xml
│   └── cpp
│       └── src
│           └── main.cpp
├── cmake                           #cmake 编译文件
│   ├── client.cmake
│   ├── server.cmake
│   ├── start.cmake
│   └── Tools.cmake
├── harpc                           #harpc 基础库文件
│   ├── config.h
│   ├── myconnect.h
│   └── ThriftClientPool.h
├── release                         #编译目录
│   └── CMakeLists.txt
├── server                          # server示例
│   └── cpp
│       └── src
│           ├── DemoService.h
│           └── main.cpp
└── thrift                          #thrift 生成的代码
    ├── echo_constants.cpp
    ├── echo_constants.h
    ├── EchoService.cpp
    ├── EchoService.hp
    ├── echo_types.cpp
    └── echo_types.h
```

使用步骤

(4) server端示例

```
// 创建hander类
class DemoService : virtual public bfd::harpc::demo::EchoServiceIf {
public:
    void echo(std::string &_return, const std::string &msg) {
        _return = "Hello, you are client";
    }
};

// 原生thrift server 加载
boost::shared_ptr<DemoService> handler(new DemoService());
boost::shared_ptr < TProcessor> processor(new EchoServiceProcessor(handler));
boost::shared_ptr<TServerTransport> serverTransport(new TServerSocket(port));
boost::shared_ptr<TProtocolFactory> protocolFactory(new TBinaryProtocolFactory());
boost::shared_ptr<TTransportFactory> transportFactory(new TBufferedTransportFactory());
TSimpleServer server(processor, serverTransport, transportFactory, protocolFactory);

// 服务启动
server.serve();
```

使用步骤

(5) client端配置文件

```
<configuration>
    <thrift>
        <!– 所有server的ip和port-->
        <connect>127.0.0.1:11212,127.0.0.1:11213</connect>
        <connTimeOut>500</connTimeOut>
        <sendTimeOut>400</sendTimeOut>
        <recvTimeOut>400</recvTimeOut>
    </thrift>
    <threshold>
        <!– 针对每个server端的最大连接池大小 -->
        <maxPoolSize>50</maxPoolSize>
        <!– 针对每个server端的初始化连接池大小-->
        <initPoolSize>10</initPoolSize>
        <!—检查连接是否有效的间隔时间-->
        <checkInterval>15</checkInterval>
    </threshold>
</configuration>
```

使用步骤

(6) client端示例代码

```
std::string confPath = "../conf/conf.xml";

//bfd::harpc::demo::EchoServiceClient 是 thrift 生成的 client 接口
ThriftClientPool<bfd::harpc::demo::EchoServiceClient> * ms_client_pool =
        new ThriftClientPool<bfd::harpc::demo::EchoServiceClient>(confPath);

//函数代理
ms_client_pool->invoke([&](bfd::harpc::demo::EchoServiceClient *clintPtr) {
    clintPtr->echo(result, "Hello, service.");
  });
```

编译

1. 修改cmake/start.cmake 文件

```
#添加demo thrift 动态库
ADD_DC_SERVICE("thrift/" "demo_thrift" thrift)
# 加入自己项目的cmake 文件
INCLUDE( ${RESEARCH_CMAKE_DIR}/client.cmake )
INCLUDE( ${RESEARCH_CMAKE_DIR}/server.cmake )
```

2. 修改cmake/server.cmake 文件

```
# 添加 server 执行文件
ADD_SERVICE_EXEC("server/cpp/src" "demo_server"
        demo_thrift thrift boost_program_options boost_system)
```

3. 修改cmake/client.cmake 文件

```
# 添加 client执行文件
ADD_SERVICE_EXEC("client/cpp/src" "demo_client" demo_thrift thrift boost_system)
```

4. 编译

```
cd release/
cmake .
#编译server
make  demo_server
make  demo_client
```

可视化管理

- 线上服务信息查看
  1. 展示出所有server和client
  2. 查找服务地址以及端口信息
- 线上服务请求状态跟踪
  1. 查看服务请求量
  2. 查看服务性能

服务总览

服务列表

server 列表

client 列表

操作功能

服务名	servers列表	clients列表	操作
bfd.harpc$HbaseHa	192.168.1.2:9091,192.168.1.1:9091(共2个)		查看
com.bfd.harpc.demo$EchoService	172.18.1.23:19092,172.18.1.24:19091(共2个)		查看
com.bfd.harpc.demo$EchoService1			查看
com.bfd.kafkalib.consumer$KafkaConsumerServic			查看
com.bfd.kafkalib.consumer$KafkaConsumerService	172.18.1.22:19090,172.18.1.22:19092,172.18.1.22:19...(共3个)	192.168.1.155:0:i_0000013340(共1个)	查看
com.bfd.kafkalib.consumer$KafkaConsumerService.test			查看
com.bfd.kafkalib.consumer$KafkaConsumerService1			查看
com.bfd.kafkalib.producer$KafkaProducerService	172.18.1.22:12106,172.18.1.22:12104,172.18.1.22:12...(共4个)	172.18.1.22:0:i_0000000660,172.18.1.23:0:i_0000000...(共4个)	查看
com.bfd.kafkalib.producer$KafkaProducerService.test			查看
com.bfd.kafkalib.producer$KafkaProducerService1			查看

服务详情

显示 10 项结果 搜索:

server地址	名称	负责人	底层协议	创建时间	操作
172.18.1.22:12100	kafka producer thrift service	qifeng.dai@baifendian.com	thrift	2015-08-28 18:30:04	查看
172.18.1.22:12102	kafka producer thrift service	qifeng.dai@baifendian.com	thrift	2015-09-11 16:40:06	查看
172.18.1.22:12104	kafka producer thrift service	qifeng.dai@baifendian.com	thrift	2015-08-27 16:51:14	查看
172.18.1.22:12106	kafka producer thrift service	qifeng.dai@baifendian.com	thrift	2015-08-27 20:53:38	查看

服务 address

服务描述

负责人

支持协议

服务创建时间

测试结论

- Java/Python版本容错性测试

序号	测试点	是否正常
1	zookeeper异常关闭的情况	正常
2	client、server与zookeeper之间网络故障的情况	正常
3	server异常关闭的情况	正常
4	server不稳定，即部分请求处理超时的情况	正常
5	server假死，新的请求全部超时，一段时间后恢复	正常
6	7*24小时稳定运行无异常	正常

从上述的测试结果来看，harpc服务在各种异常情况下均能正常使用或恢复，具备**高可用**的特性。

C++版本经过了线上两年实际考验，稳定性和性能极佳。

测试结论

- Java版本性能测试情况

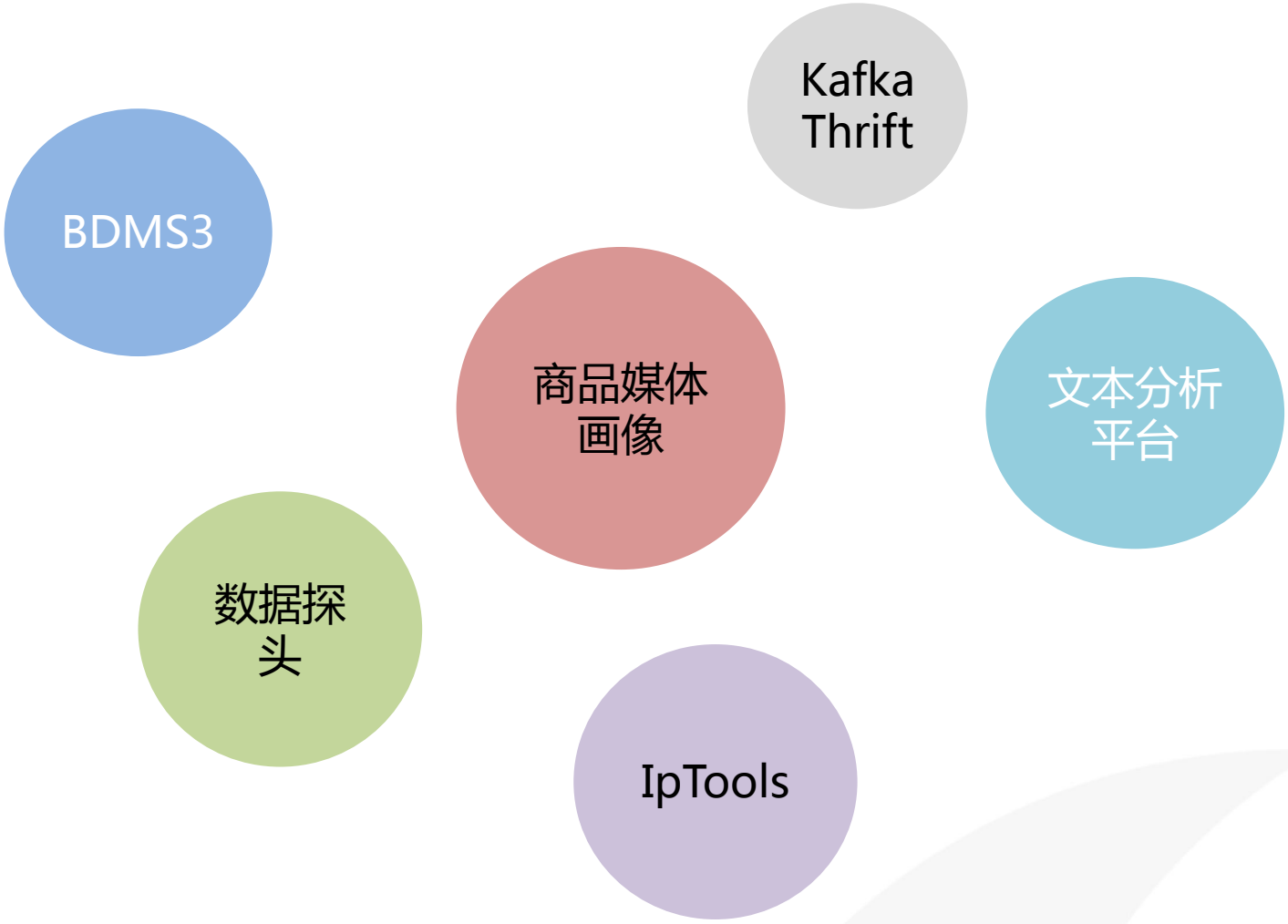
数据包大小（KB）	并发数	响应时间（ms）	TPS（ops/s）	网络带宽占用（MB/s）
1	1	0.31	3245.70	6.34
	5	0.23	21204.41	41.41
	50	0.38	128899.2	251.76
	500	3.86	17875.63	247.60
10	1	0.57	1739.74	33.98
	5	0.57	8758.10	171.06
	50	1.86	26160.21	510.94
	500	21.39	3256.34	451.05

测试结论

- Python版本性能测试情况

数据包大小（KB）	进程数	线程数	响应时间（ms）	TPS（ops/s）	网络带宽占用（MB/s）
1	1	1	0.46	2164.72	4.23
	24	1	0.74	32526.19	63.53
	24	5	1.60	74865.40	146.22
	24	10	2.88	83174.80	162.45
10	1	1	0.85	1170.62	22.86
	24	1	1.63	14689.14	286.89
	24	5	5.9	20294.60	396.37
	24	10	13.8	17388.56	339.62

使用案例



相关资料

- 源码

  https://github.com/baifendian/harpc

  （目前包括Java、Python、C++版本源码、管理系统源码）

- 交流

  QQ群： 398091913

Big Data Practitioner

百分点
BAIFENDIAN.COM