ι门总结（三）RMI+Zookeeper实现远程调用框架

ɔbby 最后发布于2017-11-20 15:04:13 阅读数 1114 收藏

RMI + ZooKeeper 实现远程调用框架

弊端

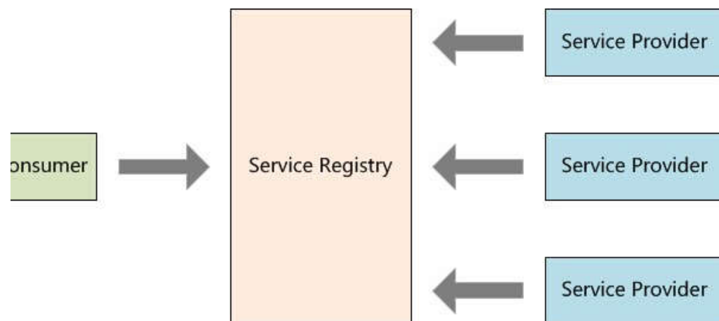
Java语言中的RPC框架，除了其语言局限之外，其实现上还有其他的一些弊端。

模型使用**BIO模型**（伪异步I/O），使用BIO和线程池的方式在大数据量、多连接情况下存在性能瓶颈。

了 **Java 默认的序列化方式**，对于性能要求比较高的系统，可能需要使用其它序列化方案来解决（例如：**Protobuf**）。
E运行时难免会存在出故障，例如，如果 RMI 服务无法连接了，就会导致客户端无法响应的现象。
ƛ下，Java 默认的序列化方式确实已经足以满足我们的要求了，如果性能方面如果不是问题的话，我们需要解决的实际上是第三点，也就是说，让使系统具备 HA（High Availability，**高可用性**）。

ookeeper

Il 服务的高可用性问题，我们需要利用 ZooKeeper 充当一个 **服务注册表**（Service Registry），让多个 服务提供者（Service Provider）形成一个集群，让 服务消费者（Service Consumer）**通过服务访问地址**（也就是 RMI 服务地址）去访问具体的服务提供者。如下图所示：



ɪ，服务注册表并不是 Load Balancer（负载均衡器），提供的不是“反向代理”服务，而是“**服务注册**”与“**心跳检测**”功能。

]表来注册 RMI 地址，这个很好理解，那么“心跳检测”又如何理解呢？说白了就是通过服务中心定时向各个服务提供者发送一个请求（实际上建立的是一个**Socket 长连接**），如果长期没有响应，
是供者已经“挂了”，只会从还“活着”的服务提供者中选出一个做为当前的服务提供者。
f虑到，服务中心可能会出现单点故障，如果服务注册表都坏掉了，整个系统也就瘫痪了。看来要想实现这个架构，必须保证**服务中心也具备高可用性**。ZooKeeper 正好能够满足我们上面提到的所有
(eeper 的临时性 ZNode 来存放服务提供者的 RMI 地址，一旦与服务提供者的 Session 中断，会自动清除相应的 ZNode。
者去监听这些 ZNode，一旦发现 ZNode 的数据（RMI 地址）有变化，就会**重新获取一份有效数据的拷贝**。
ɛr 与生俱来的**集群能力**（例如：数据同步与领导选举特性），可以确保服务注册表的高可用性。

ookeeper的实现

i

ˋ ServiceProvider 类，来发布 RMI 服务，并将 RMI 地址注册到 ZooKeeper 中（实际存放在 ZNode 上）：

```
age com.king.zkrmi;

rt org.apache.zookeeper.*;
rt org.slf4j.Logger;
rt org.slf4j.LoggerFactory;

rt java.io.IOException;
rt java.net.MalformedURLException;
rt java.rmi.Naming;
rt java.rmi.Remote;
rt java.rmi.RemoteException;
rt java.rmi.registry.LocateRegistry;
rt java.util.concurrent.CountDownLatch;

MI服务提供者

ic class ServiceProvider {

private static final Logger LOGGER = LoggerFactory.getLogger(ServiceProvider.class);
```

举报

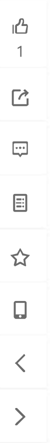
```
// 用于等待 SyncConnected 事件触发后继续执行当前线程23 |     private CountDownLatch latch = new CountDownLatch(1);

// 发布 RMI 服务并注册 RMI 地址到 ZooKeeper 中
public void publish(Remote remote, String host, int port) {
    String url = publishService(remote, host, port); // 发布 RMI 服务并返回 RMI 地址
    if (url != null) {
        ZooKeeper zk = connectServer(); // 连接 ZooKeeper 服务器并获取 ZooKeeper 对象
        if (zk != null) {
            createNode(zk, url); // 创建 ZNode 并将 RMI 地址放入 ZNode 上
        }
    }
}

// 发布 RMI 服务
private String publishService(Remote remote, String host, int port) {
    String url = null;
    try {
        url = String.format("rmi://%s:%d/%s", host, port, remote.getClass().getName());
        LocateRegistry.createRegistry(port);
        Naming.rebind(url, remote);
        LOGGER.debug("publish rmi service (url: {})", url);
    } catch (RemoteException | MalformedURLException e) {
        LOGGER.error("", e);
    }
    return url;
}

// 连接 ZooKeeper 服务器
private ZooKeeper connectServer() {
    ZooKeeper zk = null;
    try {
        zk = new ZooKeeper(Constant.ZK_CONNECTION_STRING, Constant.ZK_SESSION_TIMEOUT, new Watcher() {
            @Override
            public void process(WatchedEvent event) {
                if (event.getState() == Event.KeeperState.SyncConnected) {
                    latch.countDown(); // 唤醒当前正在执行的线程
                }
            }
        });
        latch.await(); // 使当前线程处于等待状态
    } catch (IOException | InterruptedException e) {
        LOGGER.error("", e);
    }
    return zk;
}

// 创建 ZNode
private void createNode(ZooKeeper zk, String url) {
    try {
        byte[] data = url.getBytes();
        String path = zk.create(Constant.ZK_PROVIDER_PATH, data, ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.EPHEMERAL_SEQUENTIAL);    // 创建一个临时性且有序的 ZN
        LOGGER.debug("create zookeeper node ({} => {})", path, url);
    } catch (KeeperException | InterruptedException e) {
        LOGGER.error("", e);
    }
}
```



nstant 常量，见如下代码：

```
age com.king.zkrmi;

K 常量

ic interface Constant {

String ZK_CONNECTION_STRING = "localhost:2181";
int ZK_SESSION_TIMEOUT = 5000;
String ZK_REGISTRY_PATH = "/registry";
String ZK_PROVIDER_PATH = ZK_REGISTRY_PATH + "/provider";
```

j先需要使用 ZooKeeper 的客户端工具创建一个持久性 ZNode，名为“/registry”，该节点是不存放任何数据的，可使用如下命令：

```
registry null
```

i

要在创建的时候连接 ZooKeeper，同时监听 **/registry** 节点的**NodeChildrenChanged** 事件，也就是说，一旦该节点的子节点有变化，就需要重新获取最新的子节点。这里提到的子 就是**存**
I 地址。需要强调的是，这些子节点都是临时性的，当服务提供者与 ZooKeeper 服务注册表的 Session 中断后，该临时性节会被自动删除。

```
kage com.king.zkrmi;

ort org.apache.zookeeper.KeeperException;
ort org.apache.zookeeper.WatchedEvent;
ort org.apache.zookeeper.Watcher;
ort org.apache.zookeeper.ZooKeeper;
ort org.slf4j.Logger;
ort org.slf4j.LoggerFactory;

ort java.io.IOException;
ort java.net.ConnectException;
ort java.net.MalformedURLException;
ort java.rmi.Naming;
ort java.rmi.NotBoundException;
ort java.rmi.Remote;
ort java.rmi.RemoteException;
ort java.util.ArrayList;
ort java.util.List;
ort java.util.concurrent.CountDownLatch;
ort java.util.concurrent.ThreadLocalRandom;


RMI服务消费者

lic class ServiceConsumer {

 private static final Logger LOGGER = LoggerFactory.getLogger(ServiceConsumer.class);

 // 用于等待 SyncConnected 事件触发后继续执行当前线程
 private CountDownLatch latch = new CountDownLatch(1);

 // 定义一个 volatile 成员变量，用于保存最新的 RMI 地址（考虑到该变量或许会被其它线程所修改，一旦修改后，该变量的值会影响到所有线程）
 private volatile List<String> urlList = new ArrayList<>();

 // 构造器
 public ServiceConsumer() {
     ZooKeeper zk = connectServer(); // 连接 ZooKeeper 服务器并获取 ZooKeeper 对象
     if (zk != null) {
         watchNode(zk); // 观察 /registry 节点的所有子节点并更新 urlList 成员变量
     }
 }

 // 查找 RMI 服务
 public <T extends Remote> T lookup() {
     T service = null;
     int size = urlList.size();
     if (size > 0) {
         String url;
         if (size == 1) {
             url = urlList.get(0); // 若 urlList 中只有一个元素，则直接获取该元素
             LOGGER.debug("using only url: {}", url);
         } else {
             url = urlList.get(ThreadLocalRandom.current().nextInt(size)); // 若 urlList 中存在多个元素，则随机获取一个元素
             LOGGER.debug("using random url: {}", url);
         }
         service = lookupService(url); // 从 JNDI 中查找 RMI 服务
     }
     return service;
 }

 // 连接 ZooKeeper 服务器
 private ZooKeeper connectServer() {
     ZooKeeper zk = null;
     try {
         zk = new ZooKeeper(Constant.ZK_CONNECTION_STRING, Constant.ZK_SESSION_TIMEOUT, new Watcher() {
             @Override
             public void process(WatchedEvent event) {
                 if (event.getState() == Event.KeeperState.SyncConnected) {
```

举报

```
                latch.countDown(); // 唤醒当前正在执行的线程 70 |                    }
            }
        });
        latch.await(); // 使当前线程处于等待状态
    } catch (IOException | InterruptedException e) {
        LOGGER.error("", e);
    }
    return zk;
}

// 观察 /registry 节点下所有子节点是否有变化
private void watchNode(final ZooKeeper zk) {
    try {
        List<String> nodeList = zk.getChildren(Constant.ZK_REGISTRY_PATH, new Watcher() {
            @Override
            public void process(WatchedEvent event) {
                if (event.getType() == Event.EventType.NodeChildrenChanged) {
                    watchNode(zk); // 若子节点有变化，则重新调用该方法（为了获取最新子节点中的数据）
                }
            }
        });
        List<String> dataList = new ArrayList<>(); // 用于存放 /registry 所有子节点中的数据
        for (String node : nodeList) {
            byte[] data = zk.getData(Constant.ZK_REGISTRY_PATH + "/" + node, false, null); // 获取 /registry 的子节点中的数据
            dataList.add(new String(data));
        }
        LOGGER.debug("node data: {}", dataList);
        urlList = dataList; // 更新最新的 RMI 地址
    } catch (KeeperException | InterruptedException e) {
        LOGGER.error("", e);
    }
}

// 在 JNDI 中查找 RMI 远程服务对象
@SuppressWarnings("unchecked")
private <T> T lookupService(String url) {
    T remote = null;
    try {
        remote = (T) Naming.lookup(url);
    } catch (NotBoundException | MalformedURLException | RemoteException e) {
        if (e instanceof ConnectException) {
            // 若连接中断，则使用 urlList 中第一个 RMI 地址来查找（这是一种简单的重试方式，确保不会抛出异常）
            LOGGER.error("ConnectException -> url: {}", url);
            if (urlList.size() != 0) {
                url = urlList.get(0);
                return lookupService(url);
            }
        }
        LOGGER.error("", e);
    }
    return remote;
}
```

1

] ServiceProvider 的 publish() 方法来发布 RMI 服务，发布成功后也会自动在 ZooKeeper 中注册 RMI 地址:

```
age com.king.zkrmi;


服务发布

ic class Server {
public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("please using command: java Server <rmi_host> <rmi_port>");
        System.exit(-1);
    }

    String host = args[0];
    int port = Integer.parseInt(args[1]);

    ServiceProvider provider = new ServiceProvider();

    HelloService helloService = new HelloServiceImpl();
    provider.publish(helloService, host, port);
```

举报

```
        Thread.sleep(Long.MAX_VALUE);
}
```

1

Ť Server 类的 main() 方法时，一定要使用**命令行参数来指定 host 与 port**，例如：

```
 Server localhost 1099
 Server localhost 2099
```

/a 命令可在本地运行两个 Server 程序，当然也可以同时运行更多的 Server 程序，只要 port 不同就行。

·viceConsumer 的 **lookup**() 方法来查找 RMI 远程服务对象。我们使用一个“死循环”来模拟每隔 3 秒钟调用一次远程方法。

```
age com.king.zkrmi;

MI客户端

ic class Client {

public static void main(String[] args) throws Exception {
    ServiceConsumer consumer = new ServiceConsumer();

    while (true) {
        HelloService helloService = consumer.lookup();
        String result = helloService.sayHello("Jack");
        System.out.println(result);
        Thread.sleep(3000);
    }
}
```

§验证 RMI 服务的高可用性：
Server 程序，一定要确保 port 是不同的。
Client 程序。
一个 Server 程序，并观察 Client 控制台的变化（停止一个 Server 不会导致 Client 端调用失败）。
刂才关闭的 Server 程序，继续观察 Client 控制台变化（新启动的 Server 会加入候选）。
斤有的 Server 程序，还是观察 Client 控制台变化（Client 会重试连接，多次连接失败后，自动关闭）。

们尝试使用 ZooKeeper 实现了一个简单的 RMI 服务高可用性解决方案，通过 ZooKeeper 注册所有服务提供者发布的 RMI 服务，让服务消费者监听 ZooKeeper 的 Znode，从而获取当前可用的
F RMI 服务，对于任何形式的服务（比如：WebService），也提供了一定参考。
ZooKeeper 自身的集群，那才是一个相对完美的解决方案，对于 ZooKeeper 的集群，请读者自行实践。

转载：使用 RMI + ZooKeeper 实现远程调用框架

供的RPC框架，虽然在I/O上使用简单的BIO，在服务管理上使用Registry（JNDI）的方式完成，其实现较为简单，但是在理解RPC框架时当仁不让的会是年轻人的第一个RPC框架。通过对框架的学
「发现，RPC框架的技术核心点就在以下几条（见：RPC入门总结（一）RPC定义和原理）

亍式：本文中的服务管理使用了Zookeeper，实际上某些成熟RPC也使用**Zookeeper**作为服务管理和注册工具（dubbo）

E成：本文中的代理对象生成实际上采用常规方式，由客户端和服务端自行创建和持有连接对象，而在成熟框架中可以采用**Spring**框架进行依赖注入（IoC），便于管理和编程。

本文中使用的RMI采用其底层BIO实现，在成熟框架中往往采用**NIO/Netty/Mina**等成熟的I/O多路复用方式实现（dubbo

S文中使用的RMI采用其底层字节方式进行Marshell（序列化），在对性能要求较高的场合一般可以采用**自定义私有序列化方式**（Thrift）或**JSON/ProtoBuf**等成熟序列化框架完成（[illegible]）。

举报

收藏　分享　…

ıhobby

ʒ了17 篇原创文章 · 获赞 174 · 访问量 56万+

1

他的留言板

SDK
ɛ现音视频通话能力，免费测试不限时！

ʒ科技

ɛ者说点什么

用及常用框架之Hessian 阅读数 134
ɪ及常用框架从语言兼容上的rpc框架有 thrift zeroC-ICE protbuf从服务治理角度的rpc架... 博文 来自： FOREVER_DO

据格式以及基于HTTP2的RPC框架---gRPC的使用 阅读数 8812
ɛ非常高效的数据传输格式框架ProtoBuffer。基于该数据传输的RPC框架会非常高效，如... 博文 来自： 不能说的秘密...

用及常用框架之ICE 阅读数 111
ɪ及常用框架之ICE 博文 来自： FOREVER_DO

ɛ理 阅读数 16
PC是什么RPC（Remote Procedure Call Protocol）——远程过程调用协议，它是一种通... 博文 来自： weixin_30872...

框架 阅读数 2万+
ɪ用RPC？ RPC（remoteprocedurecall）是指远程过程调用，比如两台服务器A和B，A服... 博文 来自： u013952133...

ʒ--dubbo其实很简单，就是一个远程服务调用的框架 阅读数 53
ɑ? 博文 来自： 拉着空气去兜...

结（一）RPC定义和原理 阅读数 4万+
ɛ出RPC-浅出篇转载：RPC框架与Dubbo完整使用转载：深入浅出RPC-深入篇一、RPC1.R... 博文 来自： Invoker's Tower

远程服务调用框架RSF 阅读数 1763
ɛ程服务调用框架RSF　苏宁的系统间交互最初使用中心化 ESB 架构，但随着系统拆分工作... 博文 来自： helios1988的...

C远程过程调用框架 阅读数 163
程调用框架简介gRPC是由Google公司开源的高性能RPC框架。 gRPC支持多语言gRPC原... 博文 来自： limengshi138...

ɛ程调用框架原理 阅读数 63
ɪ码农都做不了架构师？ >>> ...... 博文 来自： weixin_34019...

RPC?远程调用有什么好处?
ɪ是何. 我了解了一下dubbo框架,很多的术语搞得是更加模糊不清. 顺便提一点, 为什么深奥的东... 论坛

ʒ林杀手.简
ɪ文章
ɪ:千里之外

不能说的秘密go
201篇文章
排名:2000+

weixin_30872337
4662篇文章
排名:千里之外

Simple_Yang92
60篇文章
排名:千里之外

+ ZooKeeper 实现远程调用框架 阅读数 459
/my.oschina.net/xianggao/blog/645015在 Java 世界里，有一种技术可以实现“跨虚拟... 博文 来自： zqftisson的专栏

框架】如何实现一个简单的RPC框架（一）想法与设计 阅读数 7741
一个简单的RPC框架】系列文章：【远程调用框架】如何实现一个简单的RPC框架（一）想... 博文 来自： 吃货小跟班的...

ɛ解决方法 阅读数 6万+
ɛ器（Network File System，简称NFS），是分布式计算系统的一个组成部分，可实现在... 博文 来自： 小小外星人的...

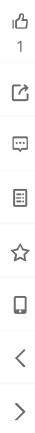
用框架设计与实现 阅读数 7098
ɪ框架设计与实现Auther: cenwenchuEmail: wenchu.cenwc@alibaba-inc.comVersion: ... 博文 来自： 放翁（文初）...

ɛ务框架 11-15
ɪ框架。 封装了xfire、rmi、hessian、httpinvoker 客户端可以统一调用，省略了远程调用的编码。 服务通过... 下载

举报

中常用的RPC框架介绍 阅读数 13万+

!程调用的简称，广泛应用在大规模分布式应用中，作用是有助于系统的垂直拆分，使系统... 博文 来自： Vince的修炼之路



1

程调用通讯协议的比较 阅读数 5011

:比较了RMI，Hessian，Burlap，Httpinvoker，web service等5种通讯协议的在不同的... 博文 来自： 一个码农的博客

框架】如何实现一个简单的RPC框架（二）实现与使用 阅读数 5043

一个简单的RPC框架】系列文章：【远程调用框架】如何实现一个简单的RPC框架（一）想... 博文 来自： 吃货小跟班的...

框架】如何实现一个简单的RPC框架（四）优化二：改变底层通信框架 阅读数 1553

一个简单的RPC框架】系列文章：【远程调用框架】如何实现一个简单的RPC框架（一）想... 博文 来自： 吃货小跟班的...

用框架-DWR入门介绍 阅读数 920

VR全称:Direct Web Remoting，直接Web远程调用。是JAVA EE领域中的一个ajax框架，... 博文 来自： qq_27905183...

+的进程间通信 阅读数 2951

!程调用框架）是一个C ++ IPC框架，提供了一种在C ++程序中实现进程间通信的简单而... 博文 来自： swartz_lubel的...

架--Spring与远程方法调用 阅读数 2262

amework.remoting.httpinvoker.HttpInvokerServiceExporter 实现远程服务调用（1）h... 博文 来自： 非淡泊无以明...

用框架-neptune-rpc 阅读数 279

调用框架-neptune-rpc```项目背景：最近一直在看dubbo相关的源码，以及一些dubbo... 博文 来自： RoySaliencyD...

架的原理、核心技术点概念以及用java socket实现简单的rpc 阅读数 167

;框架原理与实践》一、RPC架构 rpc，全称remote procedure call，即远程过程调用... 博文 来自： LZJWXJ树袋熊

ubbo很简单，其实就是一个远程服务调用的框架 阅读数 243

i分钟快速入门之dubbo，为后面的dubbo实战以及dubbo源码分析做铺垫。一、dubbo... 博文 来自： Java进阶架构师

几种方式 阅读数 34

式服务框架中。最基础的问题就是远程服务是怎么通讯的。首先来看看计算机系统网络通... 博文 来自： weixin_34363...

呈调用框架dubbo原理 阅读数 15

几个分布式框架，主要有：进行远程调用(类似于RMI的这种远程调用)的(dubbo、hsf)，j... 博文 来自： weixin_30326...

结（二）RMI的原理和使用 阅读数 4295

计：系统间通信（8）——通信管理与RMI 上篇转载：一、RMIRMI（Remote Method I... 博文 来自： Invoker's Tower

结（四）RPC IO基础：Netty原理和使用 阅读数 2024

扁程思想（七） BIO/NIO/AIO的区别(Reactor和Proactor的区别)转载：Java 编程思想（... 博文 来自： Invoker's Tower

结（七）Thrift+Zookeeper实现服务治理 阅读数 2491

ɔokeeper、连接池、Failover/LoadBalance等改造Thrift 服务化转载：基于ZooKeeper和... 博文 来自： Invoker's Tower

: 框架 阅读数 7万+

隹能用通俗的语言解释一下什么是 RPC 框架？ - 远程过程调用协议RPC（Remote Proced... 博文 来自： 乐乐的博客

X中安装启动zookeeper 阅读数 3万+

:装和启动zookeeper 博文 来自： whereismatrix...

的简单实现-zookeeper 阅读数 357

ʞage demo.zookeeper.remoting.client;package demo.zookeeper.remoting.client;im... 博文 来自： MY WORLD

与Zookeeper 阅读数 586

:M I 介绍 Java RMI 指的是远程方法调用 (Remote Method Invocation)。它是一种... 博文 来自： chbxw

结（六）Thrift的介绍和用法 阅读数 3319

入深了解Thrift（一）——Thrift介绍与用法转载：Thrift源码分析（八）--总结加一个完整... 博文 来自： Invoker's Tower

结（五）RPC IO基础：Netty高性能并发关键技术点 阅读数 903

系列之Netty百万级推送服务设计要点转载：转载：一、 博文 来自： Invoker's Tower

之 妹子图网MM批量下载 阅读数 21万+

27报网MM批量下载 点击看了本文，相信大家对爬虫一定会产生强烈的兴趣，激励自己去... 博文 来自： Nick.Peng 的...

举报

]正确打开方式 阅读数 29万+

对于入门级学习java的最佳学习方法莫过于视频+博客+书籍+总结，前三者博主将淋漓尽... 博文 来自： 程序员宜春的...

掌握的核心算法有哪些? 阅读数 41万+

-直强调数据结构以及算法学习的重要性，所以就有一些读者经常问我，数据结构与算法应... 博文 来自： 帅地

son java mysql pycharm android linux json格式 c#影院售票系统有哪些 c#鼠标相对窗体的坐标 c#
:和 c# 界面设计 c#窗口隐藏 c# 动态注入il 测试c#程序的软件 加入队列c# c# 模型验证取消 c# 小数点后保

©2019 CSDN 皮肤主题: 编程工作室 设计师: CSDN官方博客

ıobby
个人主页 >

⺦	获赞	评论	访问
!	174	50	56万+

周排名: **9万+**
总排名: **8779**

私信

包不停
:低跑步受傷風險而設
:定・全新NIKE REAC
TY RUN升級登場

Works学软开（一）敏捷之于
, 甜到发腻胸到忧伤
]到实战（八）Solr的层级
]到实战（七）Solr查询规则
（三）自动化构建工具Gradle
（二）包管理工具yarn与npm

ıoughtWorks... 1篇
40篇
6篇
ıid 2篇
64篇
展开

1篇
2篇
22篇
2篇
23篇

1
举报

24篇
22篇
12篇

展开

（一）消息队列概念和使用场

（一）RPC定义和原理

不通到入门（五）
表、分库、分片和分区

O表示法/小o表示法/Ω/Θ

读写方法总结

&）的用法和...
经常看经常忘

：分治、动态规划、贪心...
总结简直是宝藏，感谢~

接池的实现
[reply]huahangwanghao[/repl

（一）RPC定义和原理
请问一下,客户端编码后把请求发
务端解码后处理完返回数据给 ...

不通到入门（五）S...
转载的，但是文章质量很高。谢谢

kefu@csdn.net
400-660-0108
-22:00
招聘 广告服务 网站地图
658号 经营性网站备案信息
号 11010502030143
北京创新乐知网络技术有限
0报警服务
去和不良信息举报中心
报中心 家长监护 版权申诉

1

举报