NX 奈学教育

Architecture

架构师训练营基础架构篇

主讲人：陈东

2020.08.03

个人简介

NX 奈学教育

奈学教育科技

联合创始人

转转

高级架构师
技术委员会核心成员
架构平台技术负责人

58集团

架构师
IM 后端技术负责人

人大金仓数据库

资深数据库内核研发工程师

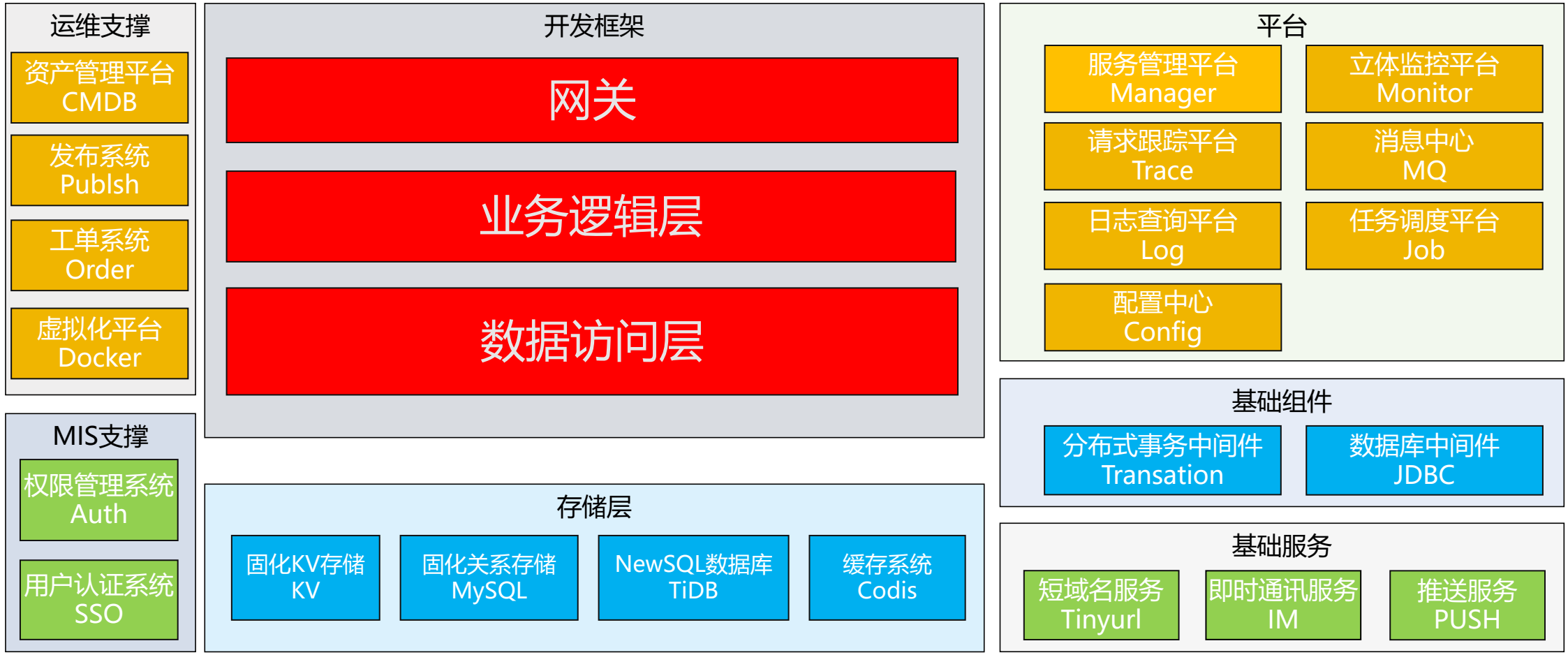
擅长领域

架构设计、基础服务、服务治理、数据存储等

对外分享

业界顶级大会
51CTO峰会
架构师峰会、TiDB峰会等

基础架构


运维支撑
资产管理平台 CMDB
发布系统 Publsh
工单系统 Order
虚拟化平台 Docker
MIS支撑
权限管理系统 Auth
用户认证系统 SSO
开发框架
网关
业务逻辑层
数据访问层
存储层
固化KV存储 KV
固化关系存储 MySQL
NewSQL数据库 TiDB
缓存系统 Codis
平台
服务管理平台 Manager
立体监控平台 Monitor
请求跟踪平台 Trace
消息中心 MQ
日志查询平台 Log
任务调度平台 Job
配置中心 Config
基础组件
分布式事务中间件 Transation
数据库中间件 JDBC
基础服务
短域名服务 Tinyurl
即时通讯服务 IM
推送服务 PUSH

NX 奈学教育

Architecture

RPC框架深入剖析与设计实践（上）

主讲人：陈东

2020.08.03

目录

NX 奈学教育

NX 奈学教育

01.RPC实现原理深入分析

01.RPC实现原理深入分析

奈学教育

RPC定义

RPC(Remote Procedure Call):远程过程调用，Remote Procedure Call Protocol它是一个计算机通信协议。它允许像调用本地方法一样调用远程服务。由于不在一个内存空间，不能直接调用，需要通过网络来表达调用的语义和传达调用的数据。

1

RPC作用

- 屏蔽组包解包
- 屏蔽数据发送/接收
- 提高开发效率
- 业务发展的必然产物

2

RPC核心组成

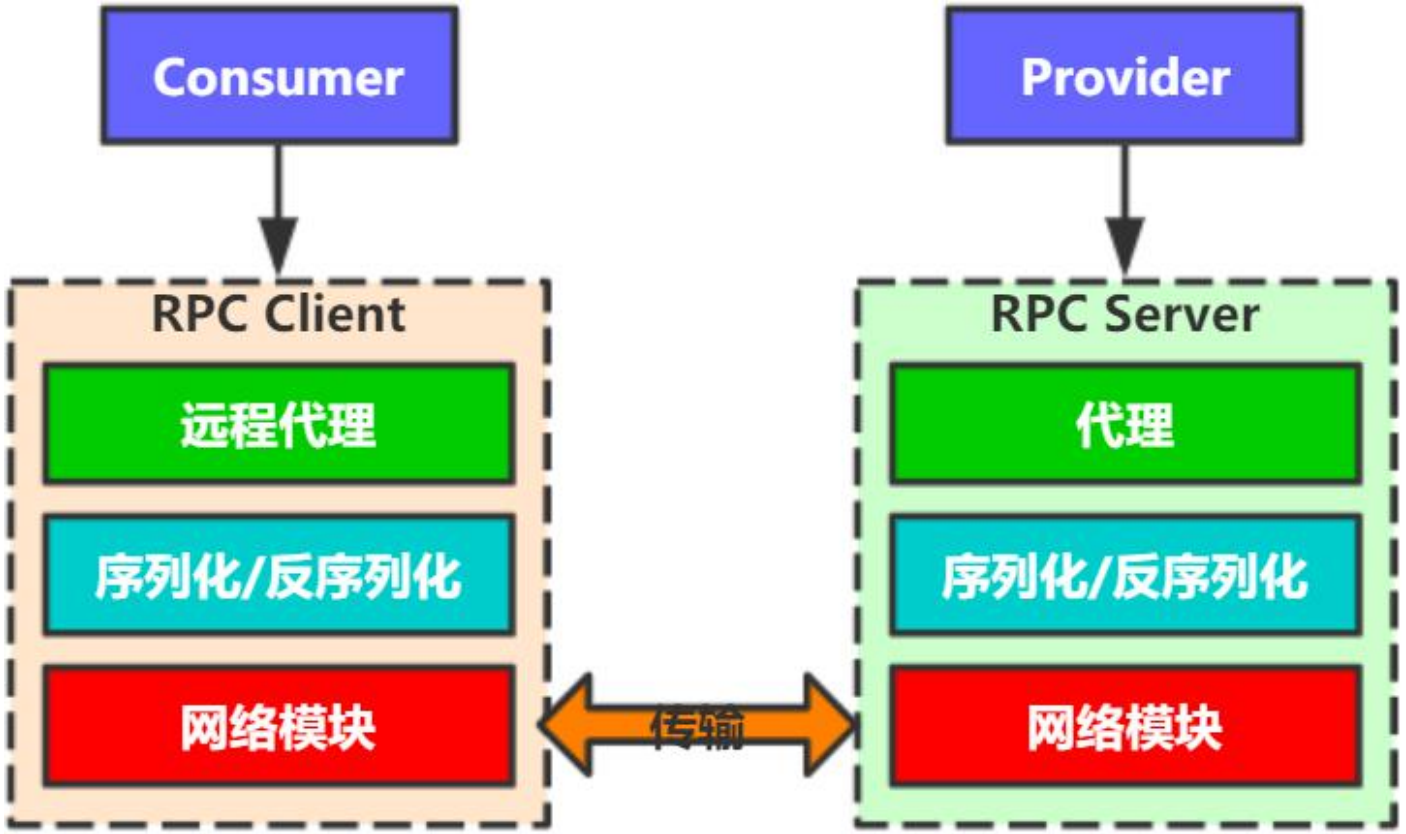
- 远程方法对象代理
- 连接管理
- 序列化/反序列化
- 寻址与负载均衡

3

RPC调用方式

- 同步调用
- 异步调用

01.RPC实现原理深入分析



RPC调用过程

- 远程代理
- 序列化
- 网络传输
- 反序列化

02.精简版RPC调用代码实现

02.精简版RPC调用代码实现

奈学教育

假如没有RPC

如果没有RPC框架支持，实现远程调用需要做哪些事?

Client 端工作

- 建立与Server的连接
- 组装数据
- 发送数据包
- 接收处理结果数据包
- 解析返回数据包

Server 端工作

- 监听端口
- 响应连接请求
- 接收数据包
- 解析数据包，调用相应方法
- 组装请求处理结果数据包
- 发送结果数据包

02.精简版RPC调用代码实现

奈学教育

设计“用户”服务

- 功能需求：用户信息管理—CRUD
- 调用方式：TCP长连接同步交互
- 协议：自定义协议

02.精简版RPC调用代码实现

奈学教育

接口设计

- **注册**

  bool addUser(User user)

- **更新**

  bool updateUser(long uid, User user)

- **注销**

  bool deleteUser(long uid)

- **查询**

  User Info getUser(long uid)

02.精简版RPC调用代码实现

奈学教育

序列化协议

远程调用涉及数据的传输，就会涉及组包和解包，需要调用方和服务方约定数据格式——序列化协议


```
public class RpcProtocol implements Serializable{
    public static int CMD_CREATE_USER = 1;
    private int version;
    private int cmd;
    private int magicNum;
    private int bodyLen = 0;
    private byte[] body;
    final public static int HEAD_LEN = 16;
```

02.精简版RPC调用代码实现

NX 奈学教育

序列化协议

远程调用涉及数据的传输，就会涉及组包和解包，需要调用方和服务方约定数据格式——序列化协议

创建用户

1	0	0x20191009	12	uid	age	sex

1	0	0x20191009	4	ret

```
public class RpcProtocol implements Serializable{
    public static int CMD_CREATE_USER = 1;
    private int version;
    private int cmd;
    private int magicNum;
    private int bodyLen = 0;
    private byte[] body;
    final public static int HEAD_LEN = 16;
```

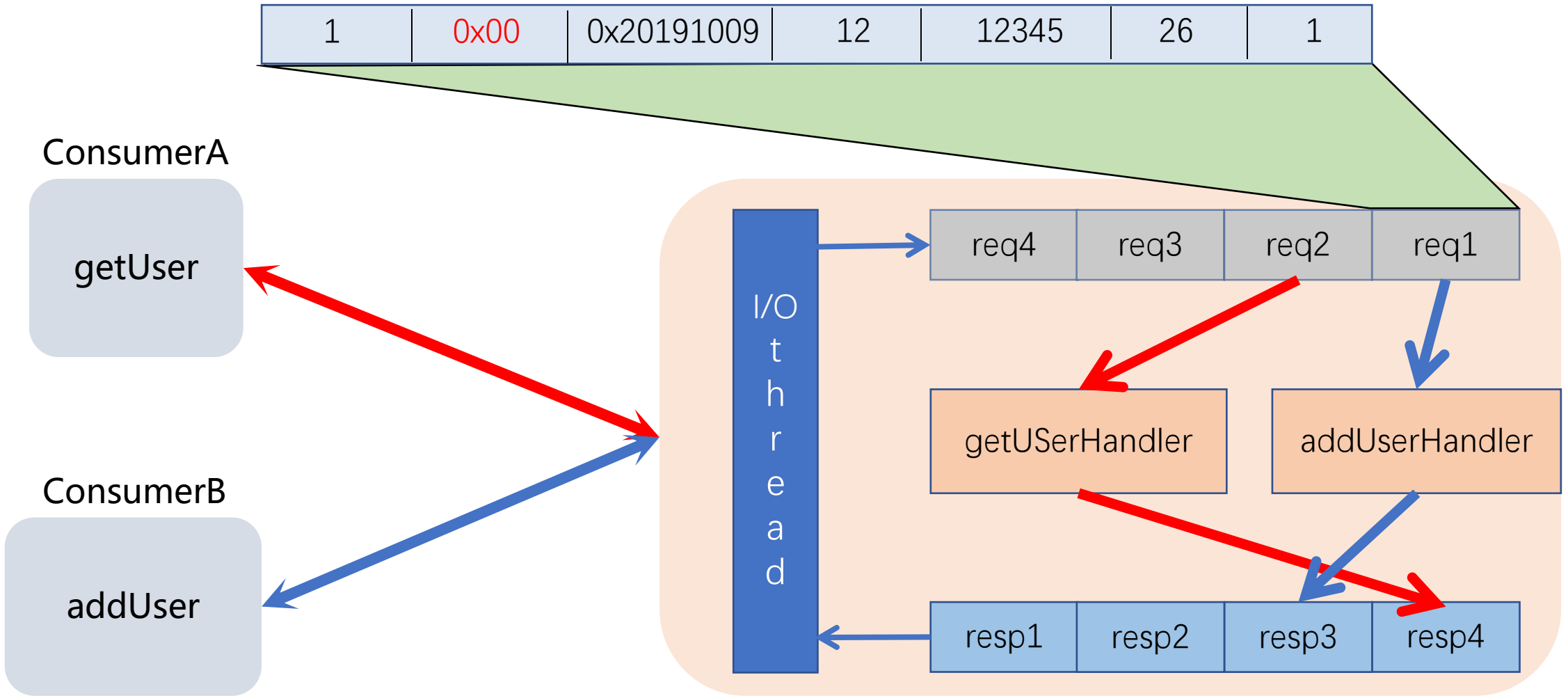
=1

=0

=0x20191009

=12

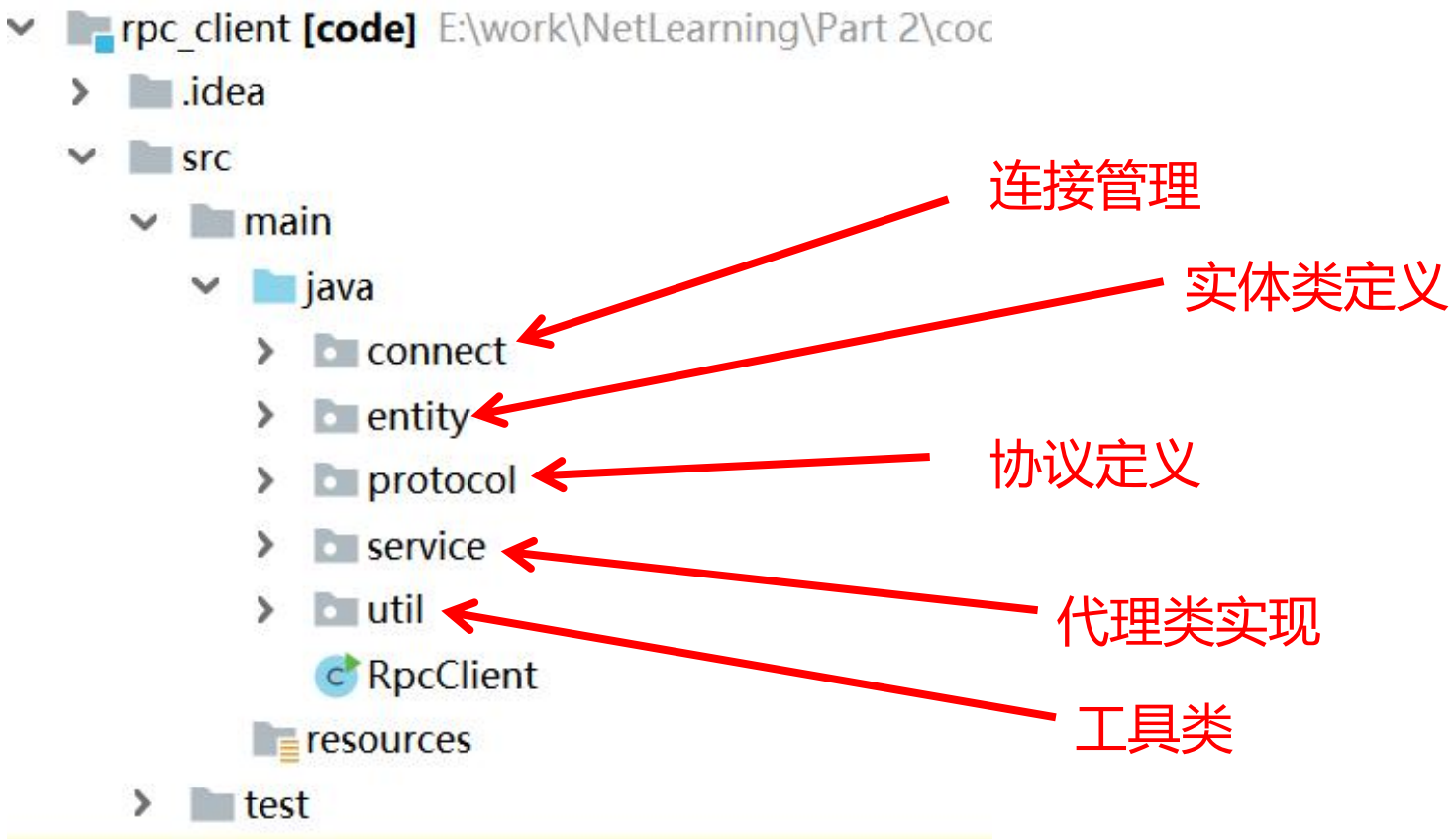
02.精简版RPC调用代码实现



02.精简版RPC调用代码实现

奈学教育

Consumer代码实现



02.精简版RPC调用代码实现

Consumer代码实现

- 创建代理类
- 构造请求数据
- 执行远程调用

```
public class RpcClient {
    public static void main(String[] args) throws Exception {
        UserService proxyUserService = new UserService();

        User user = new User();
        user.setAge((short) 26);
        user.setSex((short) 1);


        int ret = proxyUserService.addUser(user);
        if(ret == 0)
            System.out.println("调用远程服务创建用户成功！！！");
        else
            System.out.println("调用远程服务创建用户失败！！！");
    }
}
```

02.精简版RPC调用代码实现

奈学教育

addUser

- 初始化连接
- 组装协议数据
- 序列化数据
- 发送请求等待返回
- 反序列化返回数据

```
public int addUser (User userinfo) throws Exception {
    //初始化客户端连接
    TcpClient client = TcpClient.GetInstance();
    try {
        client.init();
    } catch (Exception e) {
        e.printStackTrace();
        logger.error("init rpc client error");
    }
```

02.精简版RPC调用代码实现

奈学教育

RpcProtocol

- 序列化
- 反序列化

序列化

```
public static byte[] intToBytes(int n) {
    byte[] buf = new byte[4];
    for (int i = 0; i < buf.length; i++) {
        buf[i] = (byte) (n >> (8 * i));
    }
    return buf;
}
```

反序列化

```
public static int bytesToInt(byte[] buf, int offset) {
    return buf[offset] & 0xff
            | ((buf[offset + 1] << 8) & 0xff00)
            | ((buf[offset + 2] << 16) & 0xff0000)
            | ((buf[offset + 3] << 24) & 0xff000000);
}
```

02.精简版RPC调用代码实现

奈学教育

RpcProtocol

- 序列化过程
  - 序列化请求参数到body
  - 序列化RpcProtocol

```
public byte[] generateByteArray()
{
    byte[] data = new byte[HEAD_LEN + bodyLen];
    int index = 0;
    System.arraycopy(ByteConverter.intToBytes(version), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(ByteConverter.intToBytes(cmd), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(ByteConverter.intToBytes(magicNum), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(ByteConverter.intToBytes(bodyLen), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(body, srcPos: 0, data, index, body.length);
    return data;
}
```

```
public static byte[] intToBytes(int n) {
    byte[] buf = new byte[4];
    for (int i = 0; i < buf.length; i++) {
        buf[i] = (byte) (n >> (8 * i));
    }
    return buf;
}
```

02.精简版RPC调用代码实现

NX 奈学教育

RpcProtocol

- 序列化过程
  - 序列化请求参数到body
  - 序列化RpcProtocol

```
public byte[] generateByteArray()
{
    byte[] data = new byte[HEAD_LEN + bodyLen];
    int index = 0;
    System.arraycopy(ByteConverter.intToBytes(version), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(ByteConverter.intToBytes(cmd), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(ByteConverter.intToBytes(magicNum), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(ByteConverter.intToBytes(bodyLen), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(body, srcPos: 0, data, index, body.length);
    return data;
}
```

```
public static byte[] intToBytes(int n) {
    byte[] buf = new byte[4];
    for (int i = 0; i < buf.length; i++) {
        buf[i] = (byte) (n >> (8 * i));
    }
    return buf;
}
```

02.精简版RPC调用代码实现

RpcProtocol

- 反序列化过程
  - 反序列化RpcProtocol
  - 反序列化body

```
public RpcProtocol byteArrayToRpcHeader(byte[] data)
{
    int index = 0;
    this.setVersion(ByteConverter.bytesToInt(data, index));
    index += Integer.BYTES;

    this.setCmd(ByteConverter.bytesToInt(data, index));
    index += Integer.BYTES;

    this.setMagicNum(ByteConverter.bytesToInt(data, index));
    index += Integer.BYTES;

    this.setBodyLen(ByteConverter.bytesToInt(data, index));
    index += Integer.BYTES;

    this.body = new byte[this.bodyLen];
    System.arraycopy(data, index, this.body, destPos: 0, this.bodyLen);

    return this;
}
```

```
public static int bytesToInt(byte[] buf, int offset) {
    return buf[offset] & 0xff
            | ((buf[offset + 1] << 8) & 0xff00)
            | ((buf[offset + 2] << 16) & 0xff0000)
            | ((buf[offset + 3] << 24) & 0xff000000);
}
```

02.精简版RPC调用代码实现

RpcProtocol

- 反序列化过程
  - 反序列化RpcProtocol
  - 反序列化body

```
public User byteArrayToUserInfo(byte[] data)
{
    User user = new User();
    int index = 0;

    user.setUid(ByteConverter.bytesToLong(data, index));
    index += Long.BYTES;

    user.setAge(ByteConverter.bytesToShort(data, index));
    index += Short.BYTES;

    user.setSex(ByteConverter.bytesToShort(data, index));
    index += Short.BYTES;
    return user;
}
```

```
public static int bytesToInt(byte[] buf, int offset) {
    return buf[offset] & 0xff
            | ((buf[offset + 1] << 8) & 0xff00)
            | ((buf[offset + 2] << 16) & 0xff0000)
            | ((buf[offset + 3] << 24) & 0xff000000);
}
```

02.精简版RPC调用代码实现

NX 奈学教育

Provider代码实现

02.精简版RPC调用代码实现

Provider代码实现

- 启动服务监听端口

```
public static void main(String[] args) throws Exception {
    Thread tcpServerThread = new Thread( name: "tcpServer") {
        public void  run() {
            TcpServer tcpServer = new TcpServer(SERVER_LISTEN_PORT);
            try {
                tcpServer.start();
            } catch (Exception e) {
                logger.info("TcpServer start exception: " + e.getMessage());
            }
        }
    };
    tcpServerThread.start();
    tcpServerThread.join();
}
```

02.精简版RPC调用代码实现

NX 奈学教育

Server启动

- 设置解码器
- 设置处理类
- 绑定端口

```
public void start() throws Exception {
    try {
        ServerBootstrap serverBootstrap = new ServerBootstrap();
        serverBootstrap.group(bossGroup, workerGroup);
        serverBootstrap.channel(NioServerSocketChannel.class);
        serverBootstrap.option(ChannelOption.SO_BACKLOG, value: 1024); //连接数
        serverBootstrap.localAddress(this.port);
        serverBootstrap.childOption(ChannelOption.SO_KEEPALIVE, value: true);
        serverBootstrap.childHandler(new ChannelInitializer<SocketChannel>() {
            @Override
            protected void initChannel(SocketChannel socketChannel) throws Exception {
                ChannelPipeline pipeline = socketChannel.pipeline();
                pipeline.addLast(new PkgDecoder());
                pipeline.addLast(new ServerHandler());
            }
        });

        ChannelFuture channelFuture = serverBootstrap.bind().sync();
        if (channelFuture.isSuccess()) {
            logger.info("rpc server start success!");
        } else {
            logger.info("rpc server start fail!");
        }
        channelFuture.channel().closeFuture().sync();
```

02.精简版RPC调用代码实现

奈学教育

请求包完整性校验

- 判断包头是否完整
- 判断Body是否完整

```
protected void decode(ChannelHandlerContext ctx, ByteBuf buffer, List<Object> out) throws Exception
{
    if (buffer.readableBytes() < RpcProtocol.HEAD_LEN) {
        return; //未读完足够的字节流，缓存后继续读
    }

    byte[] intBuf = new byte[4];
    buffer.getBytes( index: buffer.readerIndex() + RpcProtocol.HEAD_LEN - 4, intBuf);
    int bodyLen = ByteConverter.bytesToInt(intBuf);

    if (buffer.readableBytes() < RpcProtocol.HEAD_LEN + bodyLen) {
        return; //未读完足够的字节流，缓存后继续读
    }

    byte[] bytesReady = new byte[RpcProtocol.HEAD_LEN + bodyLen];
    buffer.readBytes(bytesReady);
    out.add(bytesReady);
}
```

02.精简版RPC调用代码实现

请求处理

- 反序列化
- 包校验
- 请求分发
- 返回结果构造

```
public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {
    byte[] recvData = (byte[]) msg;
    if (recvData.length == 0) {
        logger.warn("receive request from client, but the data length is 0");
        return;
    }

    logger.info("receive request from client, the data length is: " + recvData.length);

    //反序列化请求数据
    RpcProtocol rpcReq = new RpcProtocol();
    rpcReq.byteArrayToRpcHeader(recvData);

    if(rpcReq.getMagicNum() != RpcProtocol.CONST_CMD_MAGIC){
        logger.warn("request msgic code error");
        return;
    }
```

02.精简版RPC调用代码实现

NX 奈学教育

请求处理

- 反序列化
- 包校验
- 请求分发
- 返回结果构造

```
//解析请求，并调用处理方法
int ret = -1;
if(rpcReq.getCmd() == CMD_CREATE_USER){
    User user = rpcReq.byteArrayToUserInfo(rpcReq.getBody());
    UserService userService = new UserService();
    ret = userService.addUser(user);


    //构造返回数据
    RpcProtocol rpcResp = new RpcProtocol();
    rpcResp.setCmd(rpcReq.getCmd());
    rpcResp.setVersion(rpcReq.getVersion());
    rpcResp.setMagicNum(rpcReq.getMagicNum());
    rpcResp.setBodyLen(Integer.BYTES);
    byte[] body = rpcResp.createUserRespTobyteArray(ret);
    rpcResp.setBody(body);
    ByteBuf respData = Unpooled.copiedBuffer(rpcResp.generateByteArray());
    ctx.channel().writeAndFlush(respData);
}
```

02.精简版RPC调用代码实现

奈学教育

请求处理

- 反序列化
- 包校验
- 请求分发
- 返回结果构造

```
public byte[] createUserRespTobyteArray(int result)
{
    byte[] data = new byte[Integer.BYTES];
    int index = 0;
    System.arraycopy(ByteConverter.intToBytes(result), srcPos: 0, data, index, Integer.BYTES);
    return data;
}
```

```
public byte[] generateByteArray()
{
    byte[] data = new byte[HEAD_LEN + bodyLen];
    int index = 0;
    System.arraycopy(ByteConverter.intToBytes(version), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(ByteConverter.intToBytes(cmd), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(ByteConverter.intToBytes(magicNum), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(ByteConverter.intToBytes(bodyLen), srcPos: 0, data, index, Integer.BYTES);
    index += Integer.BYTES;
    System.arraycopy(body, srcPos: 0, data, index, body.length);
    return data;
}
```

```
public static byte[] intToBytes(int n) {
    byte[] buf = new byte[4];
    for (int i = 0; i < buf.length; i++) {
        buf[i] = (byte) (n >> (8 * i));
    }
    return buf;
}
```

NX 奈学教育

03.RPC服务消费方核心功能设计实现

RPC产品是什么样

仅仅实现远程调用是不够的，离产品化还有很长一段距离

数据传输　　序列化/反序列化　　客户端代理类实现　　请求映射分发

仅此而已？作为RPC产品还需要哪些功能

03.RPC服务消费方核心功能设计实现

奈学教育

RPC产品是什么样

仅仅实现远程调用是不够的，离产品化还有很长一段距离

Comsumer

- 连接管理
- 负载均衡
- 请求路由
- 超时处理
- 健康检查

Provider

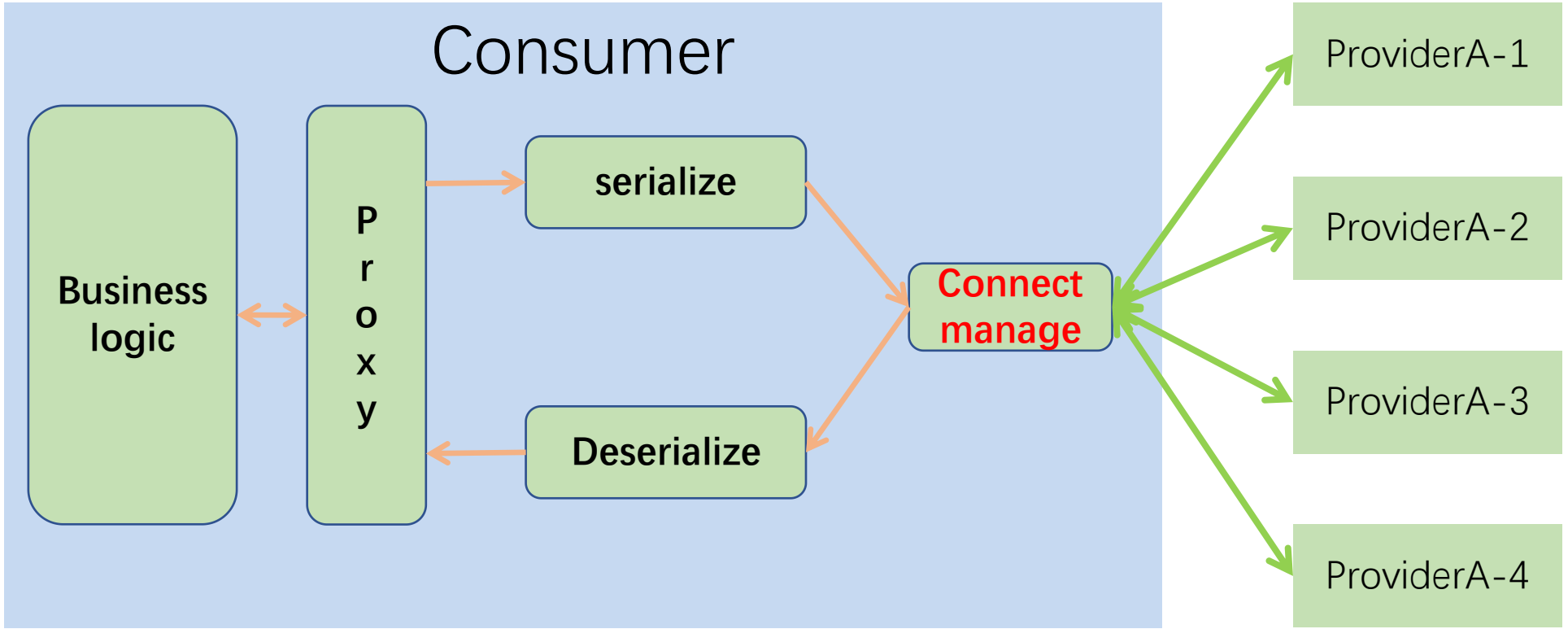
- 队列/线程池
- 超时丢弃
- 优雅关闭
- 过载保护

Consumer功能分析

- 连接管理
- 负载均衡
- 请求路由
- 超时处理

Consumer功能分析

- 连接管理
- 负载均衡
- 请求路由
- 超时处理



03.RPC服务消费方核心功能设计实现

奈学教育

连接管理

保持与服务提供方长连接，用于传输请求数据也返回结果

- 初始化时机
- 连接数维护
- 心跳/重连

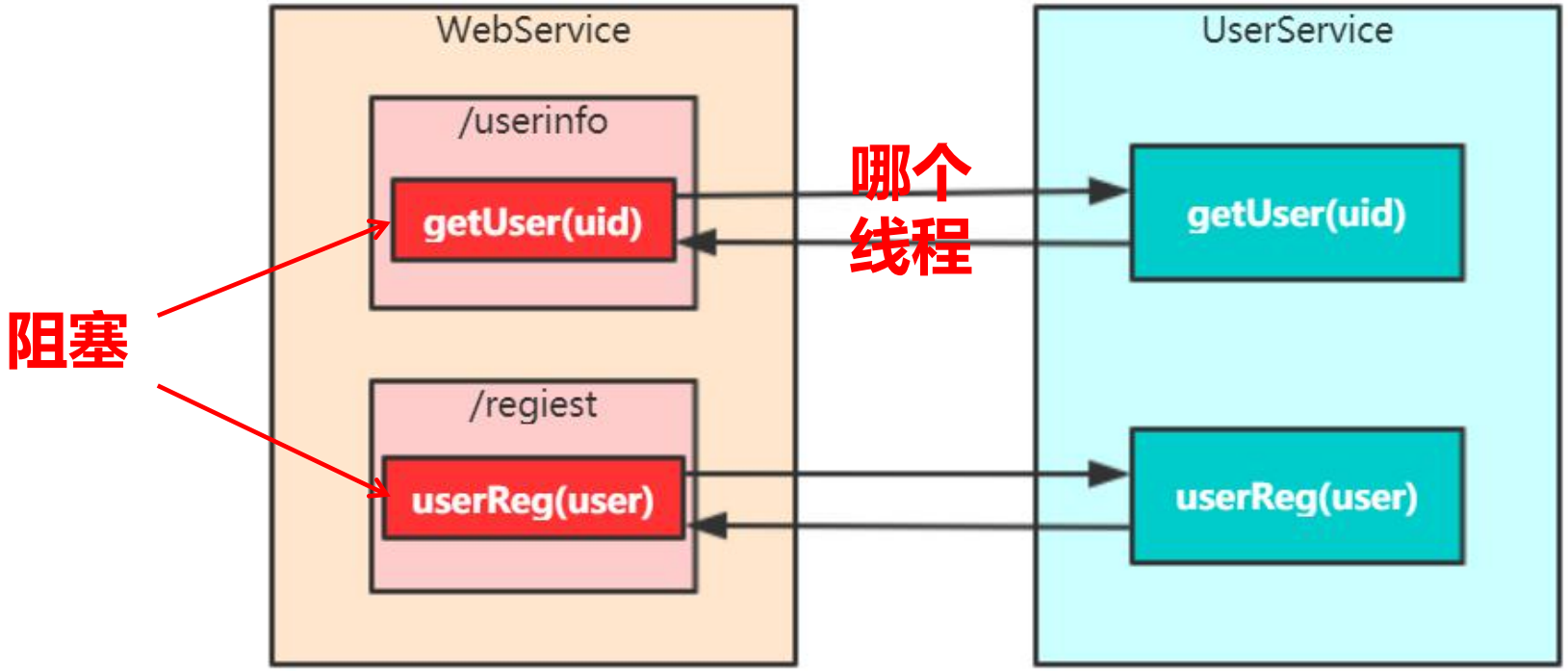
03.RPC服务消费方核心功能设计实现

NX 奈学教育

连接管理

保持与服务提供方长连接，用于传输请求数据也返回结果

客户端线程模型



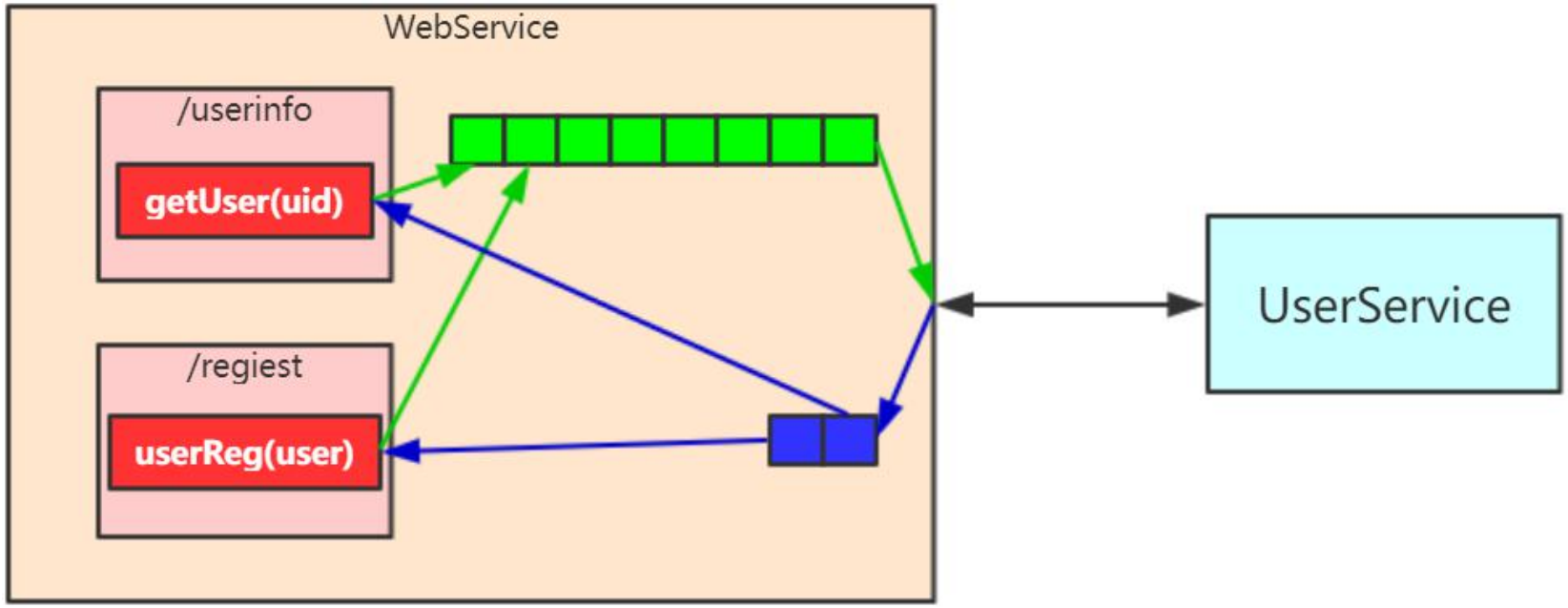
03.RPC服务消费方核心功能设计实现

NX 奈学教育

连接管理

保持与服务提供方长连接，用于传输请求数据也返回结果

客户端线程模型

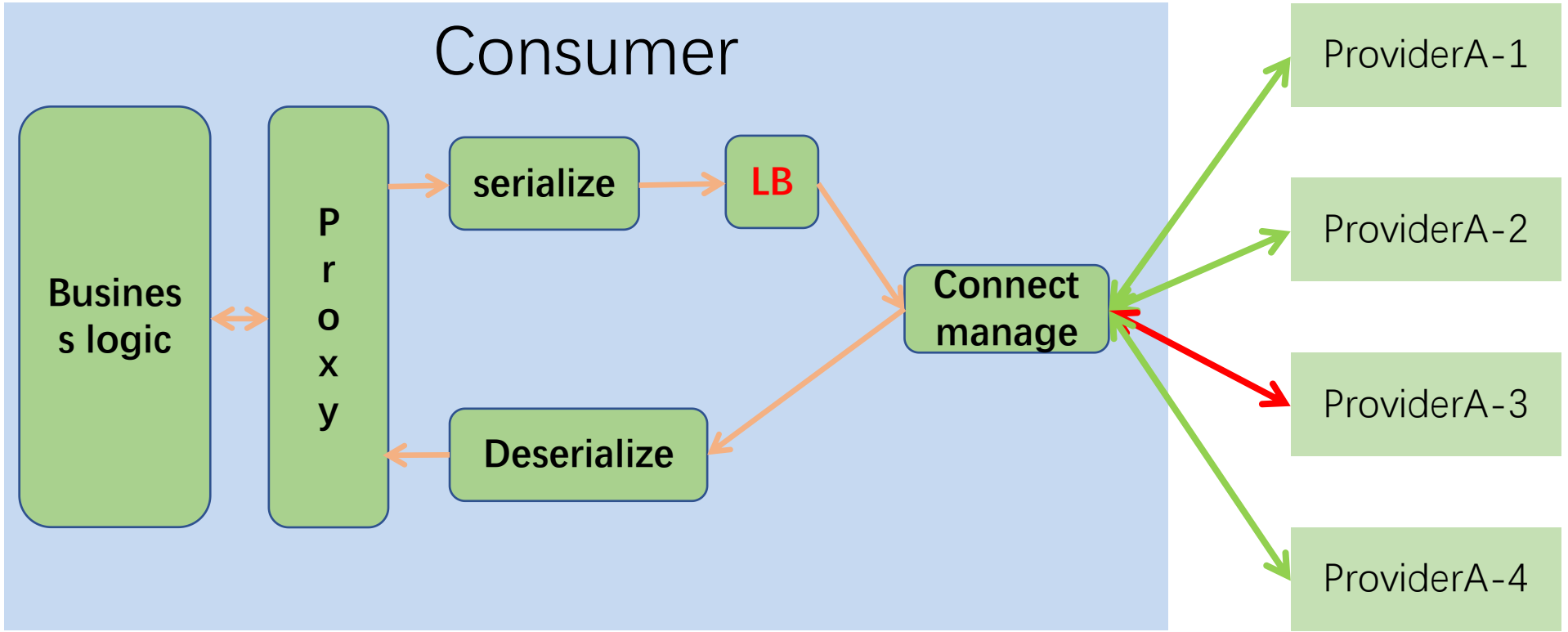


负载均衡

确保多个服务提供方节点流量均匀/合理，支持节点扩容与灰度发布

Consumer功能分析

- 连接管理
- 负载均衡
- 请求路由
- 超时处理



03.RPC服务消费方核心功能设计实现

负载均衡

确保多个服务提供方节点流量均匀/合理，支持节点扩容与灰度发布

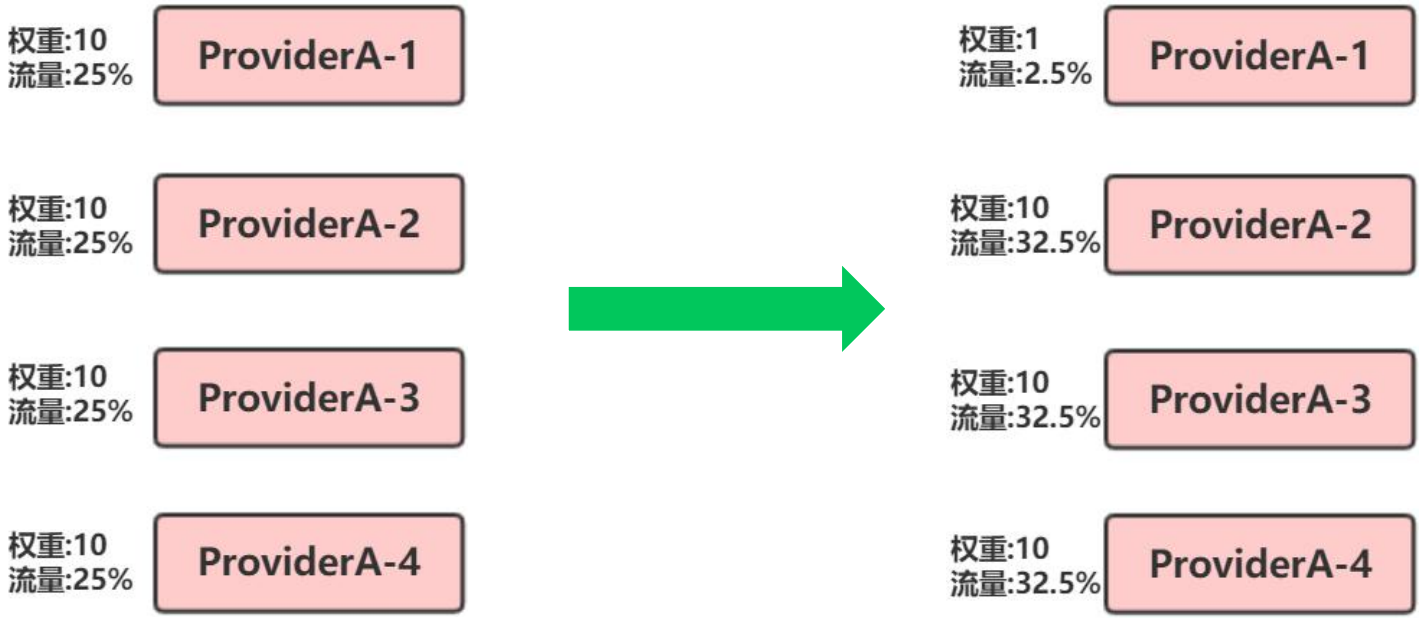
- 轮训
- 随机
- 取模
- 带权重
- 一致性Hash

03.RPC服务消费方核心功能设计实现

NX 奈学教育

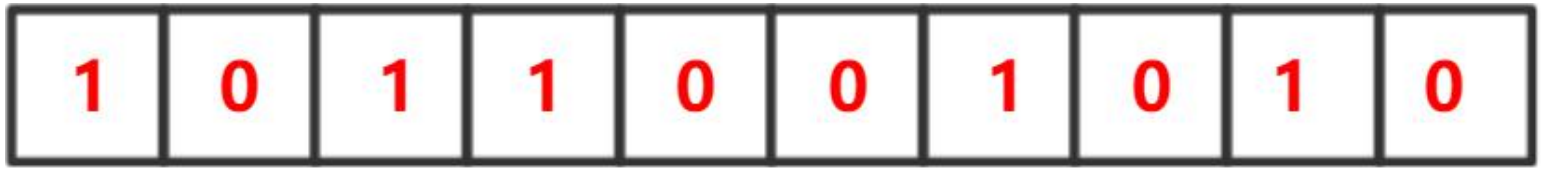
权重负载均衡设计

- 权重0~10范围内取值
- 值越大表示权重越高
- 权重高分配流量比例大



权重负载均衡设计

- **数据结构**
  - 数组，根据权重值填充
  - 0，1的位置随机打乱
- **算法描述**
  - 负载均衡选出一个结点
  - 生成0~9之间随机值
  - 对应数组中的值
    - 0 使用该节点
    - 1 不使用该节点



03.RPC服务消费方核心功能设计实现

奈学教育

轮询数组初始化

- num表示1的个数
- 随机填充数组

```
//  num 表示生成的数组中1的个数 在数组中1表示抛弃请求 0表示接受请求
public static byte[] randomGenerator(int limit, int num) {

    byte[] tempArray = new byte[limit];

    if (num <= 0) {...}
    if (num >= limit) {...}

    Random random = new Random();
    for (int i = 0; i < num; i++) {
        int temp = Math.abs(random.nextInt()) % limit;
        while (tempArray[temp] == 1) {
            temp = Math.abs(random.nextInt()) % limit;
        }
        tempArray[temp] = 1;
    }
    return tempArray;
}
```

03.RPC服务消费方核心功能设计实现

奈学教育

轮询+权重负载均衡实现

- 轮询到某一Server结点
- 根据权重再进行一次过滤
- 轮询到下一结点

```
for (int i = start; i < start + count; i++) {
    int index = i % count;
    Server server = servers.get(index);
    if (needChooseAnotherOne.test(server)) {
        requestCount.getAndIncrement();
        continue;
    }

    int requestTime = this.getRequestTimeCountAndSet(server, count);

    if (server.getWeight() < 10 && server.getWeight() > -1) {
        byte[] abandonArray = server.getAbandonArray();
        // abandonTimes[i] == 1表示server不接受该次请求
        if (abandonArray[requestTime % abandonArray.length] == 1) {
            requestCount.getAndIncrement();
            continue;
        }
    }

    if (ServerState.Normal == server.getState()) {
        result = server;
        break;
    }
    requestCount.getAndIncrement();
}
```

03.RPC服务消费方核心功能设计实现

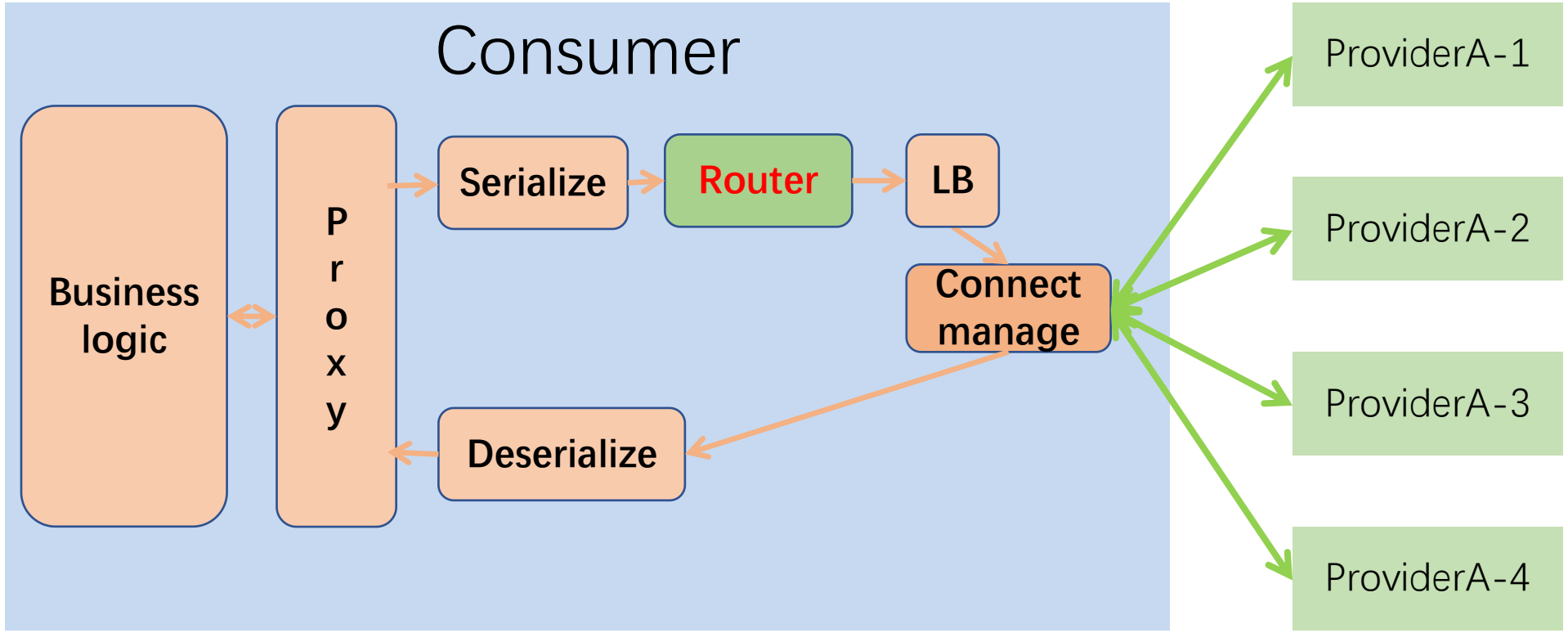
NX 奈学教育

请求路由

通过一系列规则过滤出可以选择的服务提供方节点列表，在应用隔离,读写分离,灰度发布中都发挥作用

Consumer功能分析

- 连接管理
- 负载均衡
- 请求路由
- 超时处理



03.RPC服务消费方核心功能设计实现

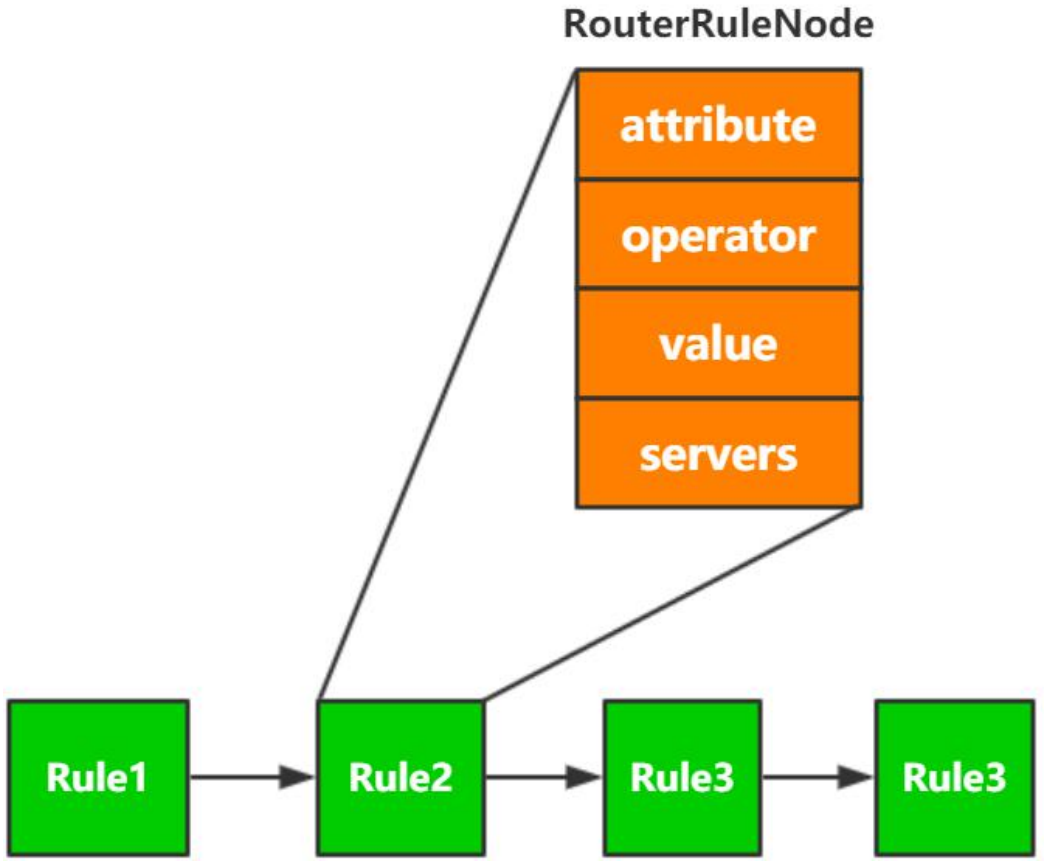
奈学教育

路由功能设计

- 匹配规则
- 行为
- 链表

路由功能设计实现

- 规则描述
  - 待比较属性
  - 运算符
  - 属性匹配值
  - 匹配结点
- 数据结构设计
  - 链表



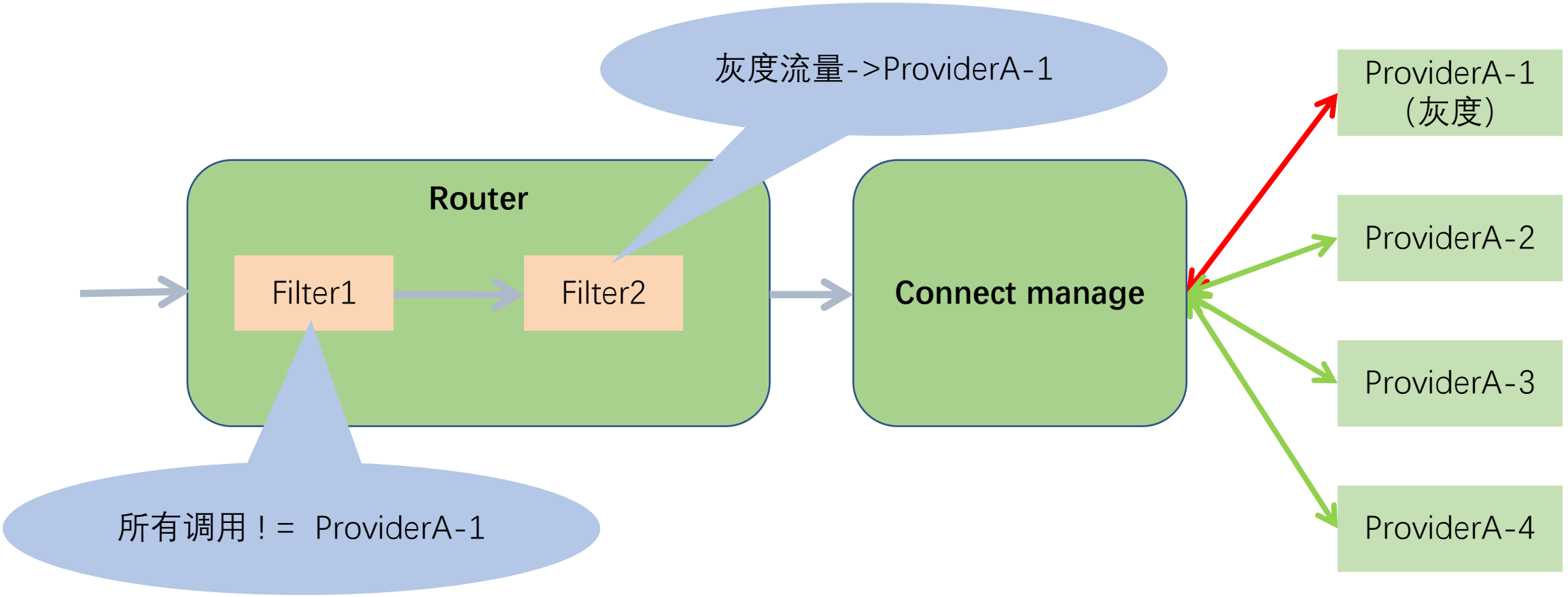
IP分流规则举例：attribute=IP，operator=IN，value=IP1，IP2，Servers：Node1，Node2

03.RPC服务消费方核心功能设计实现

NX 奈学教育

请求路由

通过一系列规则过滤出可以选择的服务提供方节点列表，在应用隔离,读写分离,灰度发布中都发挥作用

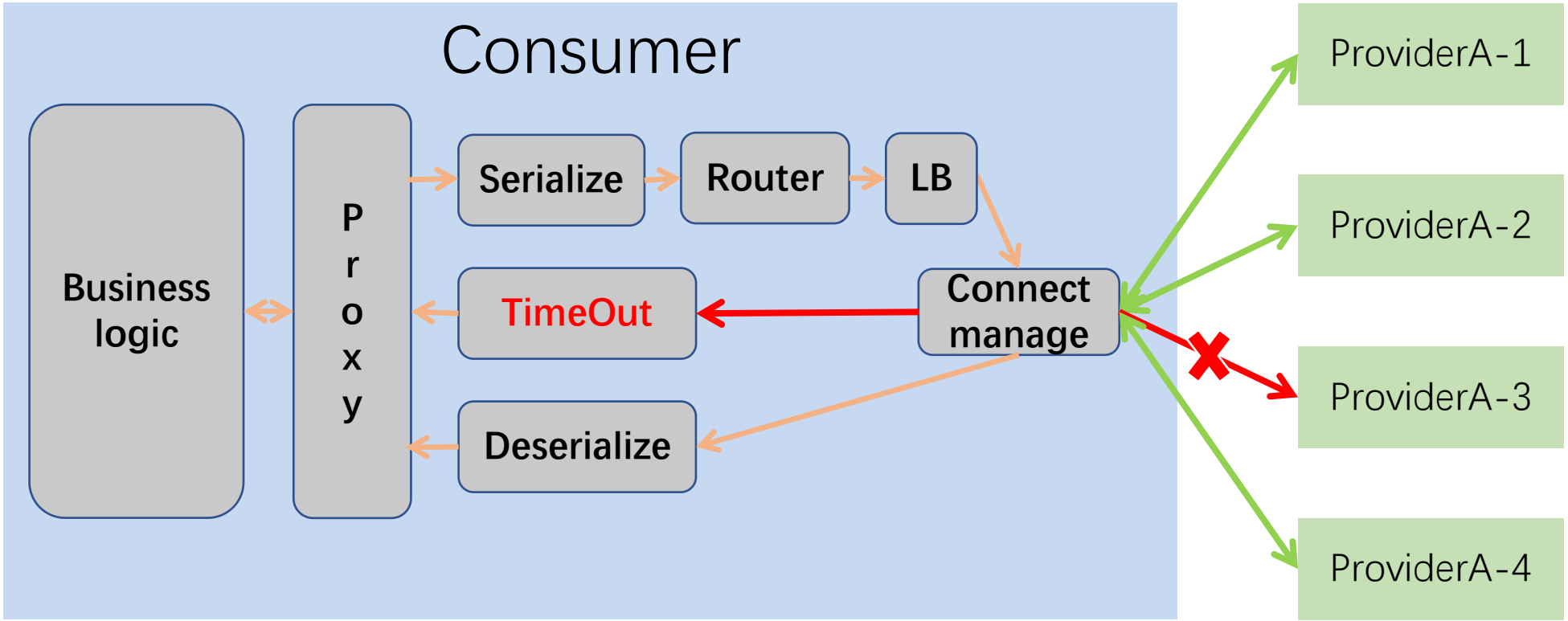


超时处理

对于长时间没有返回的请求，需要作出异常处理，及时释放资源

Consumer功能分析

- 连接管理
- 负载均衡
- 请求路由
- 超时处理



03.RPC服务消费方核心功能设计实现

调用方超时处理

- 工作线程阻塞位置
  - 等待回包通知
- 超时逻辑
  - 工作线程等待通知
  - 数据返回终止等待
  - 超时抛出异常
- 数据结构
  - Map：SessionID-WindowData

```
public Protocol request(Protocol requestProtocol) throws Exception {
    if (ServerState.Reboot == state || ServerState.Dead == state) {
        throw new RebootException();
    }
    increaseCU();
    CSocket socket = null;
    try {
        try {
            socket = socketPool.getSocket();
            byte[] data = requestProtocol.toBytes(socket.isRights(), socket.getDESKey());
            socket.registerRec(requestProtocol.getSessionID());
            socket.send(data);
        } catch (TimeoutException e) {
            timeout();
            throw e;
        } catch (UnresolvedAddressException e) {
            this.asDeath();
            logger.debug("server [{}] is dead", new Object[]{this.address});
```

03.RPC服务消费方核心功能设计实现

奈学教育

调用方超时处理

- 工作线程阻塞位置
  - 等待回包通知
- 超时逻辑
  - 工作线程等待通知
  - 数据返回终止等待
  - 超时抛出异常
- 数据结构
  - Map：SessionID-WindowData

```
public void registerRec(int sessionId) {
    AutoResetEvent event = new AutoResetEvent();
    WindowData wd = new WindowData(event);
    WaitWindows.put(sessionId, wd);
}
```

```
public void send(byte[] data) {
    try {
        if (null != transmitter) {
            TiresiasClientHelper.getInstance().setEndPoint(channel);
            TransmitterTask task = new TransmitterTask( socket: this, data);
            transmitter.invoke(task);
        }
    } catch (NotYetConnectedException ex) {
        _connecting = false;
        throw ex;
    }
}
```

03.RPC服务消费方核心功能设计实现

调用方超时处理

- 工作线程阻塞位置
  - 等待回包通知
- 超时逻辑
  - 工作线程等待通知
  - 数据返回终止等待
  - 超时抛出异常
- 数据结构
  - Map：SessionID-WindowData

```
public void invoke(TransmitterTask task) {
    int size = wqueue.size();
    if (size > 1024 * 64) {
        logger.warn(Version.ID + " send queue is to max size is:" + size);
    }
    wqueue.offer(task);
}
```

03.RPC服务消费方核心功能设计实现

奈学教育

调用方超时处理

- 工作线程阻塞位置
  - 等待回包通知
- 超时逻辑
  - 工作线程等待通知
  - 数据返回终止等待
  - 超时抛出异常
- 数据结构
  - Map：SessionID-WindowData

```
public void run() {
    int offset = 0;
    TransmitterTask[] elementData = new TransmitterTask[5];
    int waitTime = 0;
    for (; ; ) {
        try {
            TransmitterTask task = wqueue.poll(waitTime, TimeUnit.MILLISECONDS);
            if (null == task) {
                if (elementData.length > 0 && offset > 0) {
                    send(elementData, offset);
                    offset = 0;
                    arrayClear(elementData);
                }
                waitTime = 10;
                continue;
            }
            if (offset == 5) {
                //发送
                if (null != elementData) {
                    send(elementData, offset);
                }
                offset = 0;
                arrayClear(elementData);
            }
            elementData[offset] = task;
            waitTime = 0;
            ++offset;
```

03.RPC服务消费方核心功能设计实现

NX 奈学教育

调用方超时处理

- 工作线程阻塞位置
  - 等待回包通知
- 超时逻辑
  - 工作线程等待通知
  - 数据返回终止等待
  - 超时抛出异常
- 数据结构
  - Map：SessionID-WindowData

```
public byte[] receive(int sessionId, int queueLen) {
    WindowData wd = WaitWindows.get(sessionId);
    if (wd == null) {
        throw new RuntimeException("Need invoke 'registerRec' method before invoke 'receive' method!");
    }
    AutoResetEvent event = wd.getEvent();
    if (!event.waitOne(socketConfig.getReceiveTimeout())) {
        throw new TimeoutException("ServiceIP:[" + this.getServiceIP() + "],Receive data timeout or error!timeout:"
                + queueLen);
    }
    byte[] data = wd.getData();
    int offset = SFPStruct.Version;
    int len = ByteConverter.bytesToIntLittleEndian(data, offset);
    if (len != data.length) {
        throw new ProtocolException("The data length inconsistent!datalen:" + data.length + ",check len:" + len);
    }
    return data;
}
```

03.RPC服务消费方核心功能设计实现

调用方超时处理

- 工作线程阻塞位置
  - 等待回包通知
- 超时逻辑
  - 工作线程等待通知
  - 数据返回终止等待
  - 超时抛出异常
- 数据结构
  - Map：SessionID-WindowData

```
public class AutoResetEvent {
    CountDownLatch cdl;

    public AutoResetEvent() { cdl = new CountDownLatch(1); }

    public AutoResetEvent(int waitCount) { cdl = new CountDownLatch(waitCount); }

    public void set() {
        cdl.countDown();
    }

    public boolean waitOne(long time) {
        try {
            return cdl.await(time, TimeUnit.MILLISECONDS);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

03.RPC服务消费方核心功能设计实现

调用方超时处理

- 工作线程阻塞位置
  - 等待回包通知
- 超时逻辑
  - 工作线程等待通知
  - 数据返回终止等待
  - 超时抛出异常
- 数据结构
  - Map：SessionID-WindowData

```
public void decode(ByteBuffer receiveBuffer, byte[] receiveArray) throws Exception {
    try {
        int limit = receiveBuffer.limit();
        int num = 0;
        for (; num < limit; num++) {
            byte b = receiveArray[num];
            receiveData.write(b);
            if (b == ProtocolConst.P_END_TAG[index]) {
                index++;
                if (index == ProtocolConst.P_END_TAG.length) {
                    byte[] pak = receiveData.toByteArray(ProtocolConst.P_START_TAG.length, len: receiveData.size() -
                    int pSessionId = ByteConverter.bytesToIntLittleEndian(pak, offset: SFPStruct.Version + SFPStruct.
                    WindowData wd = WaitWindows.get(pSessionId);
                    if (wd != null) {
                        if (wd.getFlag() == 0) {
                            wd.setData(pak);
                            wd.getEvent().set();
                        } else if (wd.getFlag() == 1) {
                            /** 异步 */
                            if (null != unregisterRec(pSessionId)) {
                                wd.getReceiveHandler().notify(pak, wd.getInvokeCnxn());
                            }
```

NX 奈学教育

04.RPC服务提供方核心功能设计实现

04.RPC服务提供方核心功能设计实现

奈学教育

Provider功能分析

- 队列/线程池
- 超时丢弃
- 优雅关闭
- 过载保护

04.RPC服务提供方核心功能设计实现

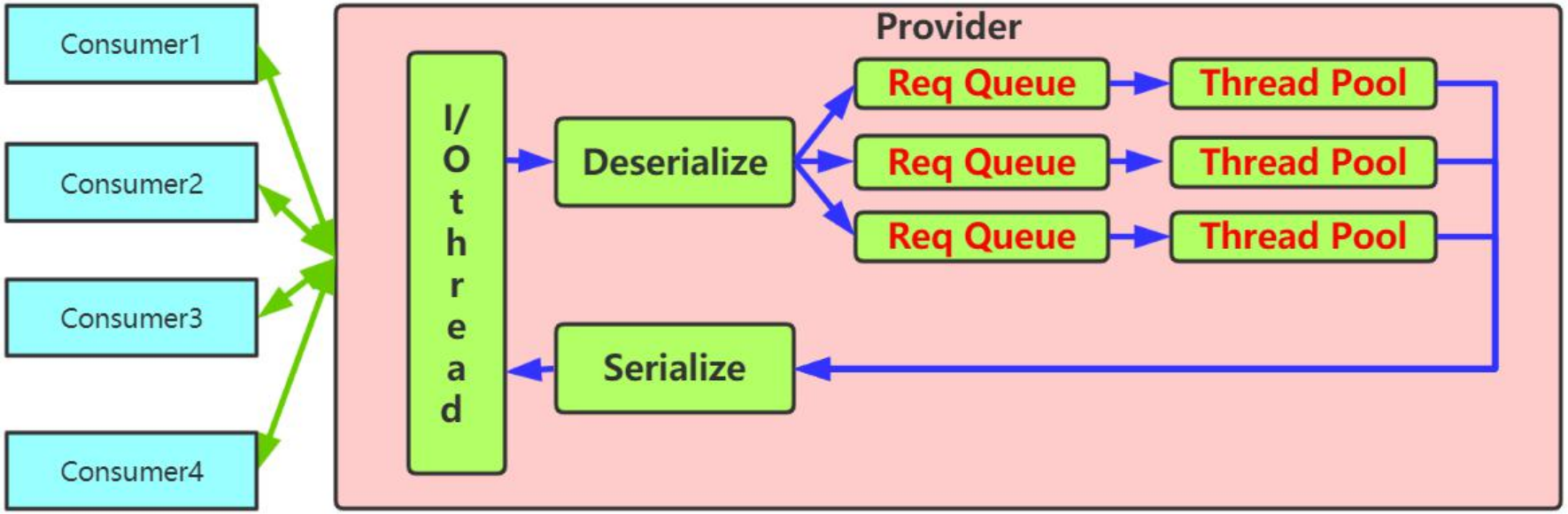
NX 奈学教育

队列&线程池

将不同类型的请求，放入各自的队列，每个队列分配独立的线程池，资源隔离

Provider功能分析

- 队列/线程池
- 超时丢弃
- 优雅关闭
- 过载保护



04.RPC服务提供方核心功能设计实现

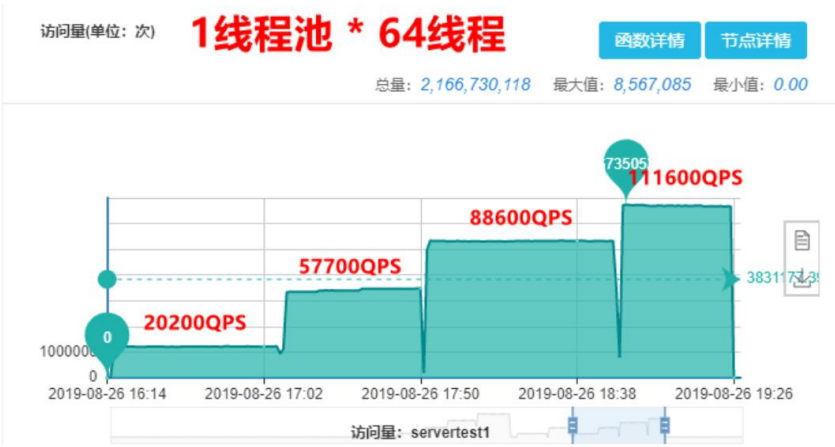
奈学教育

队列/线程池

队列数，线程池线程数如何选择?

线程池分配

- 单队列多线程 1*64
- 多队列单线程 64*1



NX 奈学教育

欢迎关注本人公众号
“架构之美”