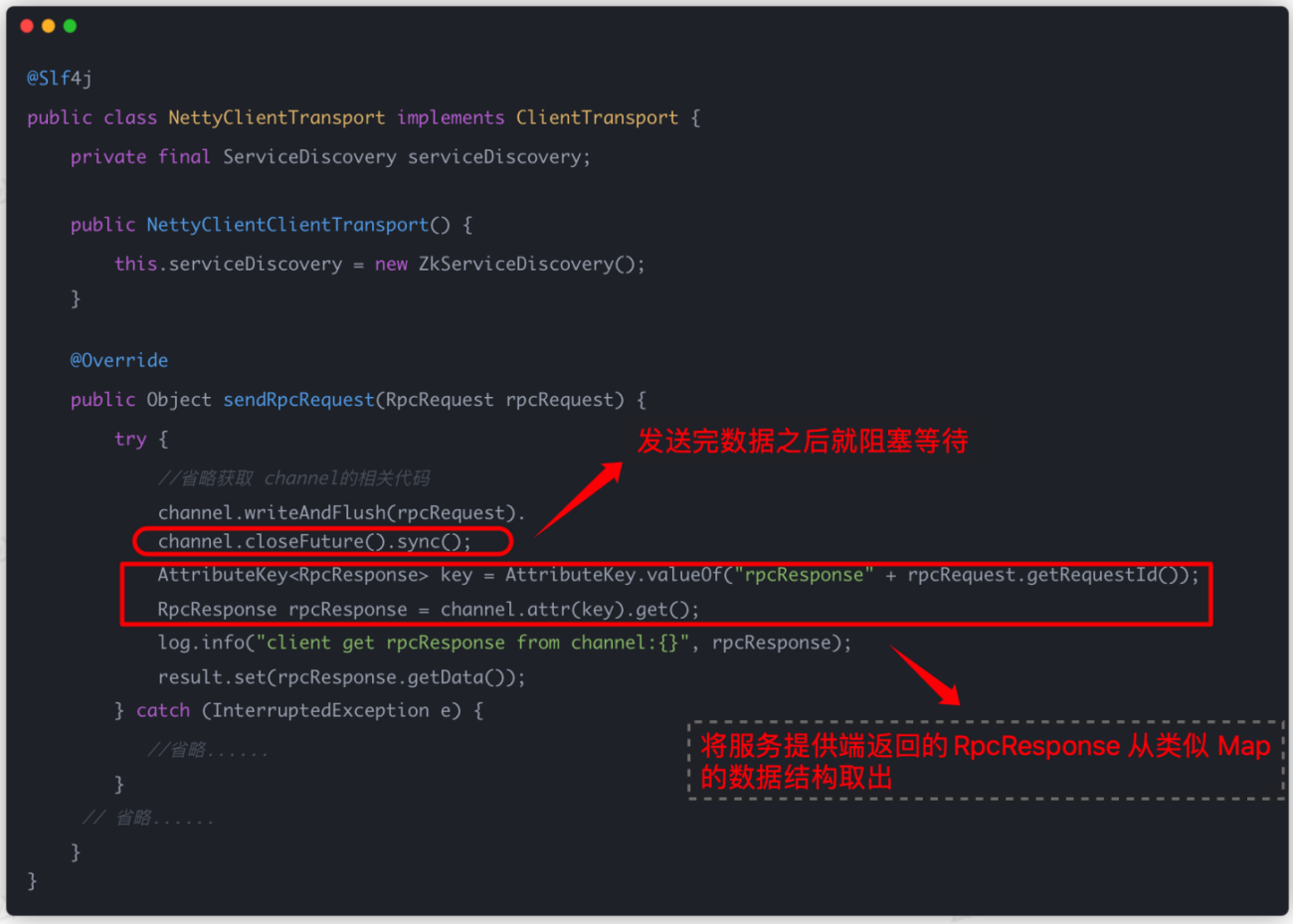
12 （优化）使用CompletableFuture优化接受服务提供端返回结果

12 （优化）使用CompletableFuture优化接受服务提供端返回结果

使用 AttributeMap 接受服务端返回结果

最开始的时候是通过 `AttributeMap` 绑定到Channel上实现的，相关代码如下：

`NettyClientTransport.java`（用来发送 RpcRequest 请求）




```
@Slf4j
public class NettyClientTransport implements ClientTransport {
    private final ServiceDiscovery serviceDiscovery;

    public NettyClientClientTransport() {
        this.serviceDiscovery = new ZkServiceDiscovery();
    }

    @Override
    public Object sendRpcRequest(RpcRequest rpcRequest) {
        try {
            //省略获取 channel的相关代码
            channel.writeAndFlush(rpcRequest).
            channel.closeFuture().sync();
            AttributeKey<RpcResponse> key = AttributeKey.valueOf("rpcResponse" + rpcRequest.getRequestId());
            RpcResponse rpcResponse = channel.attr(key).get();
            log.info("client get rpcResponse from channel:{}", rpcResponse);
            result.set(rpcResponse.getData());
        } catch (InterruptedException e) {
            //省略.....
        }
      // 省略.....
    }
}
```

`NettyClientHandler.java` （自定义客户端 ChannelHandler 来处理服务端发过来的数据）

```java
@Slf4j
public class NettyClientHandler extends ChannelInboundHandlerAdapter {

    /**
     * 读取服务端传输的消息
     */
    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg) {
        try {
            RpcResponse rpcResponse = (RpcResponse) msg;
            // 声明一个 AttributeKey 对象，类似于 Map 中的 key
            AttributeKey<RpcResponse> key = AttributeKey.valueOf("rpcResponse" + rpcResponse.getRequestId());
            /*
             * AttributeMap 可以看作是一个Channel的共享数据源
             * AttributeMap 的 key 是 AttributeKey，value 是 Attribute
             */
            // 将服务端的返回结果保存到 AttributeMap 上
            ctx.channel().attr(key).set(rpcResponse);
            ctx.channel().close();
        } finally {
            ReferenceCountUtil.release(msg);
        }
    }
}
```

服务端返回的 RpcResponse 被绑定在了这个 Channel 上了

这种是实现的缺点是不清晰，而且你每次都要调用 `channel.closeFuture().sync();` 阻塞来手动等待请求返回。

使用 CompletableFuture 进行优化

我使用 `CompletableFuture` 包装返回结果，对代码进行了重构，重要部分的代码如下：

`NettyClientTransport.java`（用来发送 RpcRequest 请求）

```java
@Slf4j
public class NettyClientTransport implements ClientTransport {
    private final ServiceDiscovery serviceDiscovery;
    private final UnprocessedRequests unprocessedRequests;

    public NettyClientTransport() {
        this.serviceDiscovery = new ZkServiceDiscovery();
        this.unprocessedRequests = SingletonFactory.getInstance(UnprocessedRequests.class);
    }

    @Override
    public CompletableFuture<RpcResponse> sendRpcRequest(RpcRequest rpcRequest) {
        // 构建返回值
        CompletableFuture<RpcResponse> resultFuture = new CompletableFuture<>();
        //省略获取 channel的相关代码
        if (channel != null && channel.isActive()) {
            // 放入未处理的请求
            unprocessedRequests.put(rpcRequest.getRequestId(), resultFuture);
            channel.writeAndFlush(rpcRequest);
            // 省略其他逻辑
        } else {
            throw new IllegalStateException();
        }

        return resultFuture;
    }

}
```


`NettyClientHandler.java` （自定义客户端 ChannelHandler 来处理服务端发过来的数据）

`UnprocessedRequests.java` 存放了未处理的请求（建议限制 map 容器大小，避免未处理请求过多 OOM

现在，在你只需要通过下面的方式就能成功接收到客户端返回的结果：

```java
CompletableFuture<RpcResponse> completableFuture =
(CompletableFuture<RpcResponse>)
clientTransport.sendRpcRequest(rpcRequest);
rpcResponse = completableFuture.get();
```