```
接口定义
                                                                                                                       注册中心的接口 NameService URI lookupService(String serviceName) throws IOException;
                                                                                                                                                           注册服务具体范星、RPC原李服务解RpcRequestHandler维护一个map,Key 就是 服务名、Value 就是服务具件方。 也就是服务实现来的实例,注册服务的论程就是同
服务名、Value 就是服务量件方,也就是服务实现来的实例,注册服务的论程就是同
                                                                                                                                                           服务提供者 ⊙
                                                                                            流程
                                                                                                                                                            PHIIoService helloService = rpcAccessPoint.getRemoteService(uri, HelloService.class)。 P芫素tep2:根据从注册中心获得的服务地址,利用RPC 框架提供的服务 RpcAccessPoint,获得还理服务的本地实例,即"驻"
                                                                                                                                                         客户端在调用 RPC 框架提供的服务时,实际调用的就是"性"提供的方法。在桩的实现
方法中、它会发源求的服务名的参数显形资源。服务编数 RPC 框架收置填束后,器析
出版务名和参数后,调用在 RPC 框架中注册的"真正的服务提供者",然后将结束返回
给客户端
                                                                                                                   依赖倒置的设计原则 ② 注册中心和RequestHandler按口使用依赖侧置原则解耦调用者和实现。

SPI(Service Provider Interface),在 SPI 中,每个接口在目录 META-INF/
services/下层有一个配置文件,文件名就是以这个提口的协会。 文件的内容就是它的
实现实的形式。 建设 5PI 的方式未进行成。 加敏岭教训是这个接口的(class 对象,
返回做就是这个接口的所有实现实的实例
                                                                                                                                                                                                                                                                                                                                                                                  RequestHandler handler = requestHandlerRegistry.get(request.getHeader().getType());
                                                                                                                                                                                                                                                                                                                                                                                 requestransaction public class RequestHandlerRegistry (
private static final Logger logger =
LoggerFactory_SetLogger(RequestHandlerRegistry.class);
private Map < integer, RequestHandler> handlerMap = new HashMap <> 0;
private Map < integer, RequestHandler>
                                                                                                                                                                                                                                                                                                                                                                                 private RequestHandler-Registry() {
Collection RequestHandler> requestHandlers =
ServiceSupport LoadAl(RequestHandler.class);
for (RequestHandler requestHandler.irequestHandlers) {
handlerMapp intequestHandler; repeatestHandlers);
logger-info("Load request handler, type (), class: 0;
requestHandler, type), (requestHandler);
grequestHandler, type), (requestHandler);
                                                                                                                                                                                                 在 RequestInvocation 美中,根据请求命令中的请求类型
(command.getHeader().getType()),分发到对应的请求处理器 RequestHandler
RPC框架设计思想
                                                                                                                                                                                                                                                                                                                                                                                    @Singleton public class RpcRequestHandler implements RequestHandler, 
ServiceProviderRegistry ( 
private static final Logger logger = 
LoggerFactory,getLogger(RpcRequestHandler.class);
                                                                                                                                                                                                                                                                                                                                                                                        /**
* Key 就是服务名,Value 就是服务提供方,也就是服务实现类的实例
                                                                                                                                                                                                                                                                                                                                                                                      */
private Map<String/*service name*/, Object/*service provider*/>
serviceProviders = new HashMap<>();
                                                                                                                                                                                                                                                                                                                                                                                   服务端分两层做了两次请求分发(
                                                                                                                                                                                                                                                                                                                                                                                   header_getVersion(), header_getRequestid(), SerializeSupport.serialize(result)

// 如果技術。返回NO_PROVIDER描译和应
loggerwam(*No service Provider of (##)(String)**,
rpcRequest_getInterfaceName(), rpcRequest_getMethodName());
retrum new Command(new ResponseHeader(type), header_getVersion(),
header_getRequestid(), Code NO_PROVIDER_getCode(). *No provider(*), new
bytest filternowable 1 {
// DC#生异果。返回UNINOWN ERRORI描读响应。
loggerwam(*Exception: *);
retrum new Command(new ResponseHeader(type(), header_getVersion(),
header_getRequestid(), Code UNINOWN_ERROR_getCode(), t.getMessage()),
new byte([0]);
}
                                                                                                                                                                                                                                                                                                                                                                                      @Override public int type() { return ServiceTypes.TYPE_RPC_REQUEST;
                                                                                 服务端
                                                                                                                                                                                                                                                                                                                                                                                       /**
*注册 RPC 服务
*@param serviceClass
*@param serviceProvide
*@param <T>
                                                                                                                                                                                                                                                                                                                                                                                     "/
Override
public syndronized <1> void addServiceProvider(Class <1 extends T>
serviceClass, T serviceProvider()
[serviceClass, T serviceProvider)
[serviceClass, T serviceProvider)
[serviceClass, Class of ServiceProvider)
[serviceClass of ServiceClass of ServiceProvider)
[serviceClass of ServiceClass of ServiceProvider getClass()
[serviceClass of ServiceClass of ServiceProvider getClass()
[serviceClass of ServiceClass of ServiceProvider getClass()
[serviceProvider getClass(
                                                                                                                                                                                               RpcRequestHandler 类中,根据 RPC 请求中的服务名,把 RPC 请求分发到对应的 服务实现类的实例中去
                                                                                                                                                                                                                                                                      public class InFlightRequests implements Closeable { private final static long TIMEOUT_SEC = 10L;
                                                                                                                                                                                                                                                                      private final Semaphore semaphore = new Semaphore(10);
                                                                                                                                                                                                                                                                         /**
* 用于维护所有的在途请求,key为请求id,value为返回值
                                                                                                                                                                                                                                                                     "/
private final Map<Integer, ResponseFuture> futureMap = new
ConcurrentHashMap <>):
private final ScheduledExecutorService scheduledExecutorService =
Executors.newSingleTineadScheduledExecutor();
private final ScheduledFuture scheduledFuture;
public InFlightRequests() {
                                                                                                                                                                                                                                                                     scheduledFuture = scheduledExecutorService.scheduleAtFixedRate(this::removeTimeoutFutures, TIMEOUT_SEC, TIMEOUT_SEC, TimeUnit.SECONDS);
                                                                                                                                                                                                                                                                     public void put(ResponseFuture responseFuture) throws Interrupte 
TimeoutException { if(semaphore.tryAcquire(TIMEOUT_SEC, TimeUnit.SECONDS)) { futureMap.put(responseFuture.getRequestId(), responseFuture);
                                                                                                                                                                                                                                                                             } else {
throw new TimeoutException();
                                                                                                                                                                                                                                                                       private sold removeTimeoutEutures ( | LuturalBapa minySell) removeIlfermy > ( | If, System Jano Timel) - entry .getValue().getTimestamp() > TIMEOUT_SEC + 10000000001. ( ) | semaphore.release(); return true; | Jeise ( ) | return true; | Jeise ( ) |
                                                                                                                                                                                                                                                                       public ResponseFuture remove(int requestid) {
ResponseFuture future = futureMap.remove(requestid);
iff(null = future) {
semaphore.release();
                                                                                                                                                                                                                                                                              eturn future;
                                                                                                                InFlightRequests使用Semaphore和线程池实现服务端的背压机制 〇
```