重试机制

重试设计

重试条件

什么情况开启重试机制。由于网络原因导致异常情况发生，开启重试

重试时间

1. 固定重试间隔，比如隔1秒重试1次
2. 指数退避重试，比如第一次是1秒，第二次是2秒，第三次是4秒，第四次是8秒
3. 随机间隔重试，随机隔多少秒重试
4. 可变延迟重试，根据上一次的响应调整重试时间

停止重试

1. 重试几次后停止
2. 重试多长时间后停止

重试后做什么

1. 通知告警
2. 降级容错

开发实现

1. 编写重试策略接口

```java
package com.yupi.yurpc.fault.retry;

import com.yupi.yurpc.model.RpcResponse;

import java.util.concurrent.Callable;

/**
 * 重试策略
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">鱼皮的编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航学习圈</a>
 */
public interface RetryStrategy {

    /**
     * 重试
     *
     * @param callable
     * @return
     * @throws Exception
     */
    RpcResponse doRetry(Callable<RpcResponse> callable) throws Exception;
}

```

2. 重试策略实现

Java

```xml
<!-- https://github.com/rholder/guava-retrying -->
<dependency>
    <groupId>com.github.rholder</groupId>
    <artifactId>guava-retrying</artifactId>
    <version>2.0.0</version>
</dependency>
```

不重试 - 重试策略 Java

```java
package com.yupi.yurpc.fault.retry;

import com.yupi.yurpc.model.RpcResponse;
import lombok.extern.slf4j.Slf4j;

import java.util.concurrent.Callable;

/**
 * 不重试 - 重试策略
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">鱼皮的编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航学习圈</a>
 */
@Slf4j
public class NoRetryStrategy implements RetryStrategy {

    /**
     * 重试
     *
     * @param callable
     * @return
     * @throws Exception
     */
    public RpcResponse doRetry(Callable<RpcResponse> callable) throws Exception {
        return callable.call();
    }

}

```

固定时间间隔 – 重试策略 Java

```java
package com.yupi.yurpc.fault.retry;

import com.github.rholder.retry.*;
import com.yupi.yurpc.model.RpcResponse;
import lombok.extern.slf4j.Slf4j;

import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;

/**
 * 固定时间间隔 – 重试策略
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">鱼皮的编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航学习圈</a>
 */
@Slf4j
public class FixedIntervalRetryStrategy implements RetryStrategy {

    /**
     * 重试
     *
     * @param callable
     * @return
     * @throws ExecutionException
     * @throws RetryException
     */
    public RpcResponse doRetry(Callable<RpcResponse> callable) throws ExecutionException, RetryException {
        Retryer<RpcResponse> retryer = RetryerBuilder.<RpcResponse>newBuilder()
                .retryIfExceptionOfType(Exception.class)
                .withWaitStrategy(WaitStrategies.fixedWait(3L, TimeUnit.SECONDS))
                .withStopStrategy(StopStrategies.stopAfterAttempt(3))
                .withRetryListener(new RetryListener() {
                    @Override
                    public <V> void onRetry(Attempt<V> attempt) {
                        log.info("重试次数 {}", attempt.getAttemptNumber());
                    }
                })
                .build();
        return retryer.call(callable);
```

```
    }

}
```

上述代码中，重试策略如下：

- 重试条件：使用 retryIfExceptionOfType 方法指定当出现 Exception 异常时重试。
- 重试等待策略：使用 withWaitStrategy 方法指定策略，选择 fixedWait 固定时间间隔策略。
- 重试停止策略：使用 withStopStrategy 方法指定策略，选择 stopAfterAttempt 超过最大重试次数停止。
- 重试工作：使用 withRetryListener 监听重试，每次重试时，除了再次执行任务外，还能够打印当前的重试次数。

支持配置和扩展重试策略

1. 定义扩展策略常量

Java

```java
package com.yupi.yurpc.fault.retry;

/**
 * 重试策略键名常量
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">鱼皮的编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航学习圈</a>
 */
public interface RetryStrategyKeys {

    /**
     * 不重试
     */
    String NO = "no";

    /**
     * 固定时间间隔
     */
    String FIXED_INTERVAL = "fixedInterval";

}

```

2. 使用工厂模式+spi机制

Java

```java
package com.yupi.yurpc.fault.retry;

import com.yupi.yurpc.spi.SpiLoader;

/**
 * 重试策略工厂（用于获取重试器对象）
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航知识星球</a>
 */
public class RetryStrategyFactory {

    static {
        SpiLoader.load(RetryStrategy.class);
    }

    /**
     * 默认重试器
     */
    private static final RetryStrategy DEFAULT_RETRY_STRATEGY = new NoRetryStrategy();

    /**
     * 获取实例
     *
     * @param key
     * @return
     */
    public static RetryStrategy getInstance(String key) {
        return SpiLoader.getInstance(RetryStrategy.class, key);
    }

}

```

3. 在rpcconfig全局配置中新增配置

```java
@Data
public class RpcConfig {
    /**
     * 重试策略
     */
    private String retryStrategy = RetryStrategyKeys.NO;
}

```

应用重试功能

Java

```java
// 使用重试机制
RetryStrategy retryStrategy = RetryStrategyFactory.getInstance(rpcConfig.getRetryStrategy());
RpcResponse rpcResponse = retryStrategy.doRetry(() ->
        VertxTcpClient.doRequest(rpcRequest, selectedServiceMetaInfo)
);
/**
 * 服务代理（JDK 动态代理）
 *
 * @author <a href="https://github.com/liyupi">程序员鱼皮</a>
 * @learn <a href="https://codefather.cn">编程宝典</a>
 * @from <a href="https://yupi.icu">编程导航知识星球</a>
 */
public class ServiceProxy implements InvocationHandler {

    /**
     * 调用代理
     *
     * @return
     * @throws Throwable
     */
    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        // 指定序列化器
        final Serializer serializer = SerializerFactory.getInstance(RpcApplication.getRpcConfig().getSerializer());

        // 构造请求
        String serviceName = method.getDeclaringClass().getName();
        RpcRequest rpcRequest = RpcRequest.builder()
                .serviceName(serviceName)
                .methodName(method.getName())
                .parameterTypes(method.getParameterTypes())
                .args(args)
                .build();
        try {
            // 从注册中心获取服务提供者请求地址
            RpcConfig rpcConfig = RpcApplication.getRpcConfig();
            Registry registry = RegistryFactory.getInstance(rpcConfig.getRegistryConfig().getRegistry());
            ServiceMetaInfo serviceMetaInfo = new ServiceMetaInfo();
            serviceMetaInfo.setServiceName(serviceName);
            serviceMetaInfo.setServiceVersion(RpcConstant.DEFAULT_SERVICE_VERSION);
```

```
            List<ServiceMetaInfo> serviceMetaInfoList = registry.serviceDiscovery(serviceMetaInfo.getServiceKey());
            if (CollUtil.isEmpty(serviceMetaInfoList)) {
                throw new RuntimeException("暂无服务地址");
            }

            // 负载均衡
            LoadBalancer loadBalancer = LoadBalancerFactory.getInstance(rpcConfig.getLoadBalancer());
            // 将调用方法名（请求路径）作为负载均衡参数
            Map<String, Object> requestParams = new HashMap<>();
            requestParams.put("methodName", rpcRequest.getMethodName());
            ServiceMetaInfo selectedServiceMetaInfo = loadBalancer.select(requestParams, serviceMetaInfoList);

            // rpc 请求
            // 使用重试机制
            RetryStrategy retryStrategy = RetryStrategyFactory.getInstance(rpcConfig.getRetryStrategy());
            RpcResponse rpcResponse = retryStrategy.doRetry(() ->
                    VertxTcpClient.doRequest(rpcRequest, selectedServiceMetaInfo)
            );
            return rpcResponse.getData();
        } catch (Exception e) {
            throw new RuntimeException("调用失败");
        }
    }
}
```