容错机制

容错策略

容错实现方式

开发实现

- 容错策略接口
- 快速失败容错策略
- 静默处理容错
- 故障恢复策略
- 支持可配置可扩展
  - 容错策略常量
  - 容错策略工厂+spi机制
  - rpcConfig全局配置增加容错策略配置
  - 应用容错

容错策略

1. Fail-over：故障转移，当调用某个服务异常时，可换另一个节点进行调用。
2. Fail-back：失败自动恢复，一旦发现故障，系统会尝试通过备用组件来继续提供服务，这通常涉及到Fail-over（故障转移）过程。
3. Fail-Safe：静默处理，当调用发生异常时，直接忽略掉，不做任何处理，就像没发生过一样。
4. Fail-fast：快速失败，当调用发生异常时，快速抛出异常。交给调用层处理。

容错实现方式

1. 重试：系统错误后重试；
2. 限流：当系统压力过大，出现部分错误时，限制请求的频率或数量，对系统进行保护；
3. 降级：当系统出现错误时，改为执行其他更稳健的操作，兜底服务，不至于让服务彻底不能用；

4. 熔断：当调用某个接口时，接口异常，此时不再调用此接口，避免引发连锁反应；
5. 超时控制：如果请求超时，暂时中断对该接口的请求。

开发实现

容错策略接口

Java

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

import com.yupi.yurpc.model.RpcResponse;

import java.util.Map;

/**
 * 容错策略
 *
 * @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 TolerantStrategy {

    /**
     * 容错
     *
     * @param context 上下文，用于传递数据
     * @param e       异常
     * @return
     */
    RpcResponse doTolerant(Map<String, Object> context, Exception e);
}

```

快速失败容错策略

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

import com.yupi.yurpc.model.RpcResponse;

import java.util.Map;

/**
 * 快速失败 - 容错策略（立刻通知外层调用方）
 *
 * @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 class FailFastTolerantStrategy implements TolerantStrategy {

    @Override
    public RpcResponse doTolerant(Map<String, Object> context, Exception e) {
        throw new RuntimeException("服务报错", e);
    }
}

```

静默处理容错

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

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

import java.util.Map;

/**
 * 静默处理异常 - 容错策略
 *
 * @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 FailSafeTolerantStrategy implements TolerantStrategy {

    @Override
    public RpcResponse doTolerant(Map<String, Object> context, Exception e) {
        log.info("静默处理异常", e);
        return new RpcResponse();
    }
}

```

故障恢复策略

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

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

import java.util.Map;

/**
 * 降级到其他服务 - 容错策略
 *
 * @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 FailBackTolerantStrategy implements TolerantStrategy {

    @Override
    public RpcResponse doTolerant(Map<String, Object> context, Exception e) {
        // todo 可自行扩展，获取降级的服务并调用
        return null;
    }
}

```

支持可配置可扩展

容错策略常量

Java

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

/**
 * 容错策略键名常量
 *
 * @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 TolerantStrategyKeys {

    /**
     * 故障恢复
     */
    String FAIL_BACK = "failBack";

    /**
     * 快速失败
     */
    String FAIL_FAST = "failFast";

    /**
     * 故障转移
     */
    String FAIL_OVER = "failOver";

    /**
     * 静默处理
     */
    String FAIL_SAFE = "failSafe";

}

```

容错策略工厂+spi机制

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

import com.yupi.yurpc.spi.SpiLoader;

/**
 * 容错策略工厂（工厂模式，用于获取容错策略对象）
 *
 * @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 class TolerantStrategyFactory {

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

    /**
     * 默认容错策略
     */
    private static final TolerantStrategy DEFAULT_RETRY_STRATEGY = new FailFastTolerantStrategy();

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

}

```

rpcConfig全局配置增加容错策略配置

```java
@Data
public class RpcConfig {

    /**
     * 容错策略
     */
    private String tolerantStrategy = TolerantStrategyKeys.FAIL_FAST;
}

```

应用容错

Java

```java
/**
 * 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 {
        // 构造请求
        String serviceName = method.getDeclaringClass().getName();
        RpcRequest rpcRequest = RpcRequest.builder()
                .serviceName(serviceName)
                .methodName(method.getName())
                .parameterTypes(method.getParameterTypes())
                .args(args)
                .build();

        // 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());
```

```
40          // 将调用方法名（请求路径）作为负载均衡参数
41          Map<String, Object> requestParams = new HashMap<>();
42          requestParams.put("methodName", rpcRequest.getMethodName());
43          ServiceMetaInfo selectedServiceMetaInfo = loadBalancer.select(requ
    estParams, serviceMetaInfoList);
44          // rpc 请求
45          // 使用重试机制
46          RpcResponse rpcResponse;
47          try {
48              RetryStrategy retryStrategy = RetryStrategyFactory.getInstance
    (rpcConfig.getRetryStrategy());
49              rpcResponse = retryStrategy.doRetry(() ->
50                      VertxTcpClient.doRequest(rpcRequest, selectedServiceMe
    taInfo)
51              );
52          } catch (Exception e) {
53              // 容错机制
54              TolerantStrategy tolerantStrategy = TolerantStrategyFactory.ge
    tInstance(rpcConfig.getTolerantStrategy());
55              rpcResponse = tolerantStrategy.doTolerant(null, e);
56          }
57          return rpcResponse.getData();
58      }
59  }
60
```