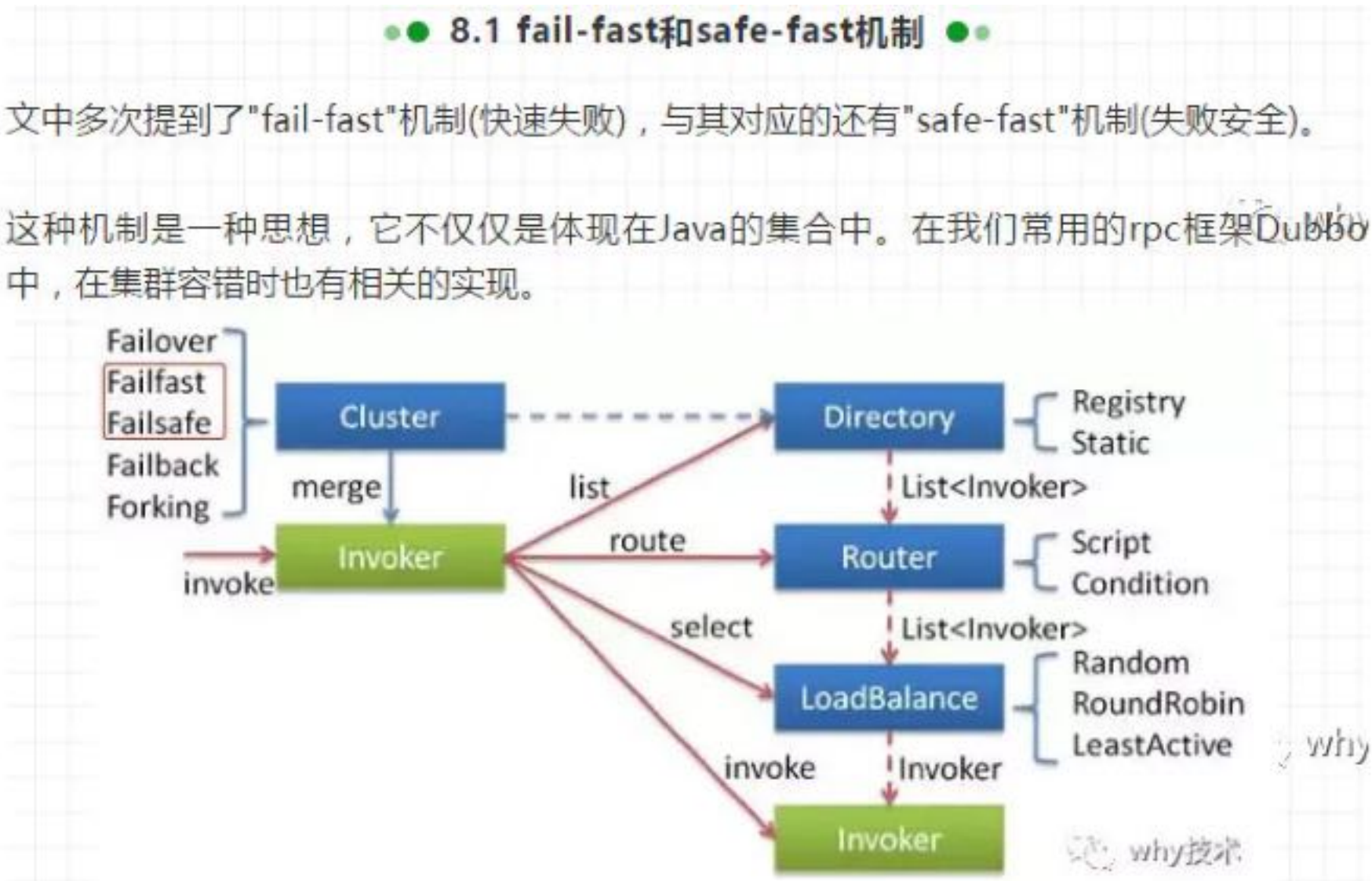
快速失败机制&失败安全机制

这是why技术的第29篇原创文章

之前在写**《这道Java基础题真的有坑！我求求你，认真思考后再回答。》**这篇文章时，我在8.1小节提到了快速失败和失败安全机制。



Dubbo 主要提供了这样几种容错方式：

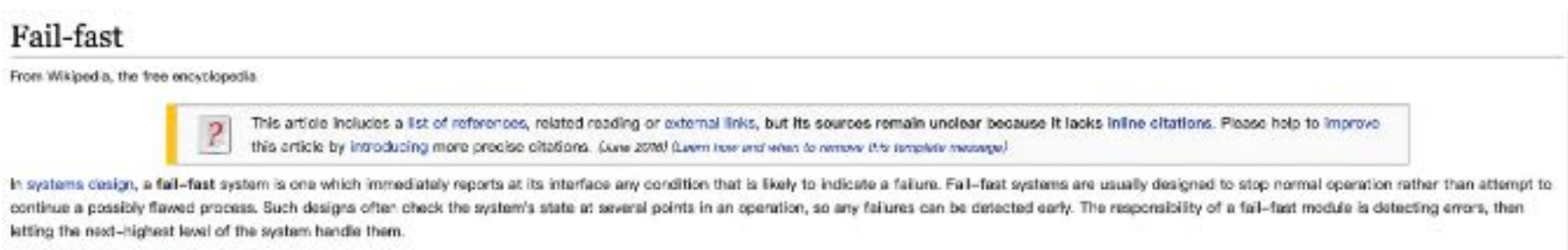
> Failover Cluster - 失败自动切换
> Failfast Cluster - 快速失败
> Failsafe Cluster - 失败安全
> Failback Cluster - 失败自动恢复
> Forking Cluster - 并行调用多个服务提供者

但是我发现当我搜索"快速失败"或"失败安全"的时候，检索出来的结果**百分之90以上**都是在说Java集合中是怎么实现快速失败或失败安全的。

在我看来，说到快速失败、失败安全时，我们首先想到的应该是这是一种机制、一种思想、一种模式，它属于**系统设计范畴**，其次才应该想到它的各种应用场景和具体实现。而不是立马想到了集合，这样就有点本末倒置的感觉了。

可以看一下wiki上对于快速失败和失败安全的描述：

快速失败：http://en.wikipedia.org/wiki/Fail-fast



失败安全：http://en.wikipedia.org/wiki/Fail-safe

简而言之：系统运行中，如果有错误发生，那么系统立即结束，这种设计就是快速失败。系统运行中，如果有错误发生，系统不会停止运行，它忽略错误（但是会有地方记录下来），继续运行，这种设计就是失败安全。

本文就对比一下Java集合中的快速失败、失败安全和Dubbo框架中的快速失败、失败安全。

读完之后，你就知道Java集合中实现和Dubbo中的实现就大不一样。

没有谁比谁好，只有结合场景而言，谁比谁更合适而已。

Java集合-快速失败

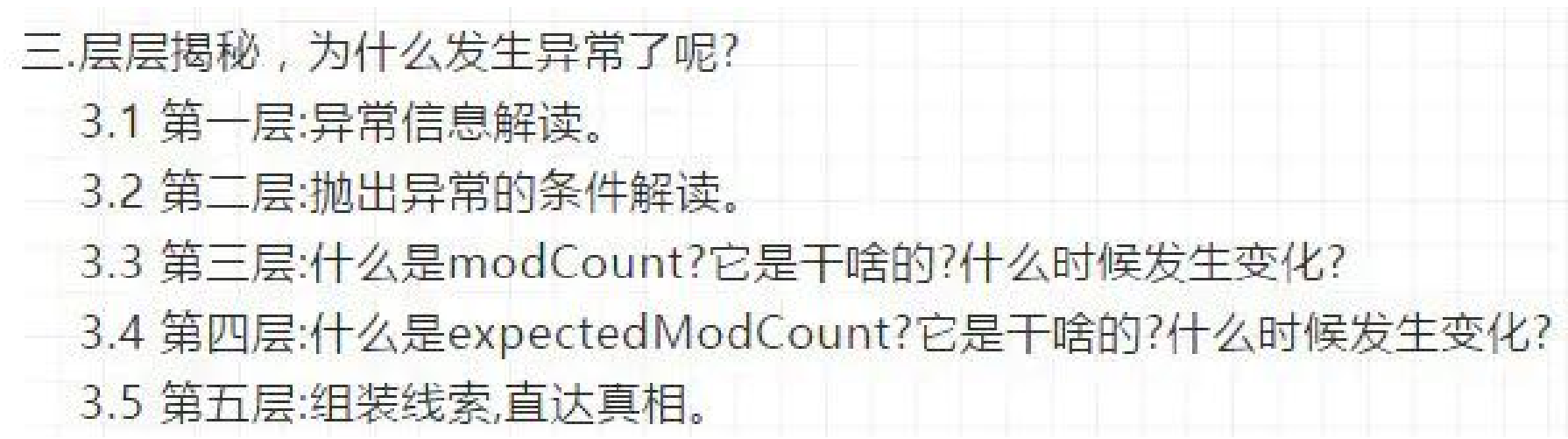
现象：在用迭代器遍历一个集合对象时，如果遍历过程中对集合对象的内容进行了增加、删除、修改操作，则会抛出ConcurrentModificationException。

原理：迭代器在遍历时直接访问集合中的内容，并且在遍历过程中使用一个 modCount 变量。集合在被遍历期间如果内容发生变化，就会改变modCount的值。每当迭代器使用hashNext()/next()遍历下一个元素之前，都会检测modCount变量是否为expectedmodCount值，是的话就返回遍历；否则抛出ConcurrentModificationException异常，终止遍历。

注意：这里异常的抛出条件是检测到 modCount！=expectedmodCount 这个条件。如果集合发生变化时修改modCount值刚好又设置为了expectedmodCount值，则异常不会抛出。因此，不能依赖于这个异常是否抛出而进行并发操作的编程，这个异常只建议用于检测并发修改的bug。

场景：java.util包下的集合类都是快速失败的，不能在多线程下发生并发修改（迭代过程中被修改）。

上面的知识点我在**《这道Java基础题真的有坑！我求求你，认真思考后再回答。》**这篇文章中第三小节已经抽丝剥茧般的详细说明了，有兴趣的可以阅读一下：



Java集合-失败安全

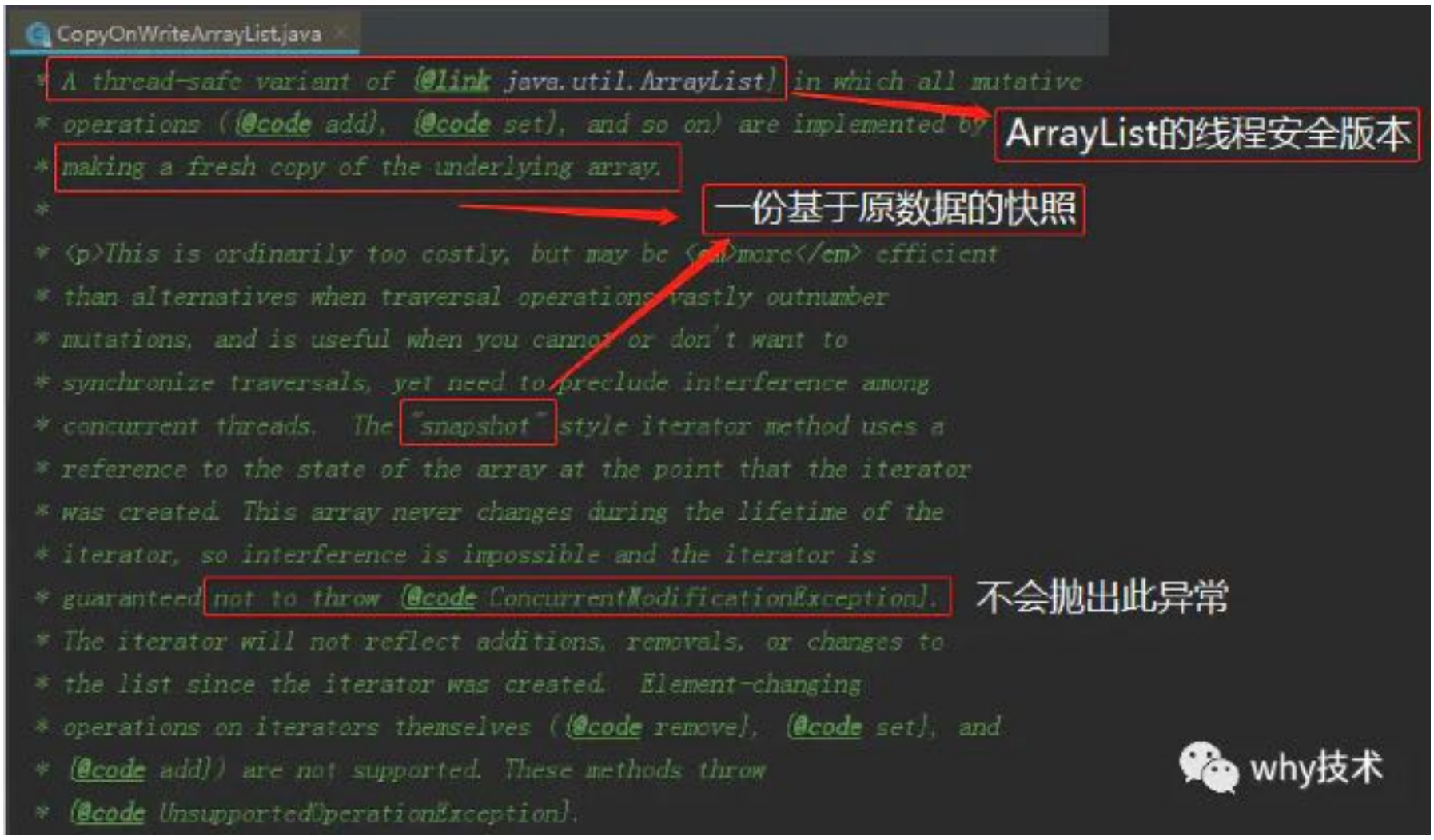
现象：采用失败安全机制的集合容器，在遍历时不是直接在集合内容上访问的，而是先复制原有集合内容，在拷贝的集合上进行遍历。

原理：由于迭代时是对原集合的拷贝进行遍历，所以在遍历过程中对原集合所作的修改并不能被迭代器检测到，所以不会触发ConcurrentModificationException。

缺点：基于拷贝内容的优点是避免了ConcurrentModificationException，但同样地，迭代器并不能访问到修改后的内容，即：迭代器遍历的是开始遍历那一刻拿到的集合拷贝，在遍历期间原集合发生的修改迭代器是不知道的。这也就是他的缺点，同时，由于是需要拷贝的，所以比较吃内存。

场景：java.util.concurrent包下的容器都是安全失败，可以在多线程下并发使用，并发修改。

比如之前文章说到的CopyOnWriteArrayList：

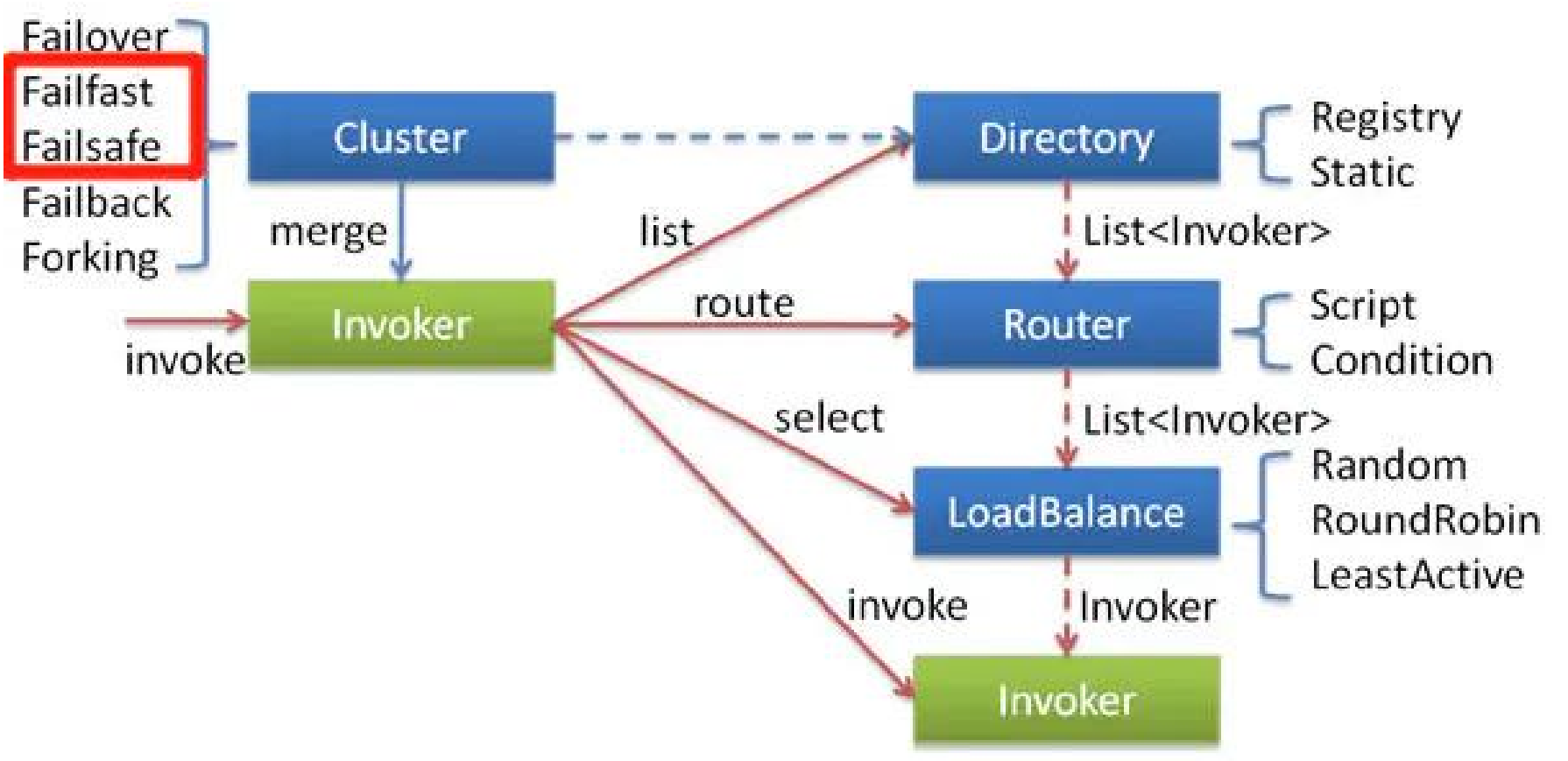


集合部分涉及到的参考链接：https://www.cnblogs.com/ygj0930/p/6543350.html

Dubbo中的集群容错

在描述快速失败和失败安全在Dubbo中的体现之前，我们必须先说说Dubbo中的集群容错机制，因为**快速失败和失败安全是其容错机制中的一种。**

还是看之前出现的图片：



其中，集群 Cluster 用途是将多个服务提供者合并为一个 Cluster Invoker，并将这个 Invoker 暴露给服务消费者。这样一来，服务消费者只需通过这个 Invoker 进行远程调用即可，至于具体调用哪个服务提供者，以及调用失败后如何处理等问题，现在都交给集群模块去处理。

集群模块是服务提供者和服务消费者的中间层，为服务消费者屏蔽了服务提供者的情况，这样服务消费者就可以专心处理远程调用相关事宜。

对于容错方式官网上是这样的说的：

以上就是集群工作的整个流程，这里并没介绍集群是如何容错的。Dubbo 主要提供了这样几种容错方式：

- Failover Cluster - 失败自动切换
- Failfast Cluster - 快速失败
- Failsafe Cluster - 失败安全
- Failback Cluster - 失败自动恢复
- Forking Cluster - 并行调用多个服务提供者

注意哦，官网说的是主要提供了这样几种，并没有完全列举，通过查看源码你可以看到：

org.apache.dubbo.rpc.cluster.support.AbstractClusterInvoker

有九个实现类，说明官方提供了九种容错方式供你选择：

而本文主要讨论的是：

org.apache.dubbo.rpc.cluster.support.FailfastClusterInvoker org.apache.dubbo.rpc.cluster.support.FailsafeClusterInvoker

搭建Demo项目

为了方便演示快速失败和失败安全在Dubbo中的体现，我们需要先搭建一个简单的Demo项目，搭建过程我就不演示了，这一小节仅对Demo的关键地方进行说明：

服务端接口如下:

```
public interface DemoService {
    String sayHello(String name);
}
```

服务端接口实现如下:

```
@Service(value = "demoService")
public class DemoServiceImpl implements DemoService {
    @Override
    public String sayHello(String name) {
        try {
            TimeUnit.SECONDS.sleep(3);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return "Hello " + name;
    }
}
```

休眠3秒，模拟业务超时。

服务端Dubbo xml配置如下:

```xml
<dubbo:registry address="zookeeper://127.0.0.1:2181" timeout="1000"/>

<!-- 用dubbo协议在20880端口暴露服务 -->
<dubbo:protocol name="dubbo" port="20883"/>

<!-- 声明需要暴露的服务接口 -->
<dubbo:service interface="com.example.dubbo.cluster.provider.service.DemoService"
               ref="demoService">
</dubbo:service>
```

消费端Dubbo xml配置如下：

```xml
<dubbo:registry address="zookeeper://127.0.0.1:2181" timeout="1000"/>

<!-- 生成远程服务代理，可以和本地bean一样使用demoService -->
<dubbo:reference id="demoService"
                 interface="com.example.dubbo.cluster.provider.service.DemoService"
                 cluster="failfast"
/>
```

这里就是配置集群容错机制，本文中配置为failfast或者failsafe

消费端在Test类中消费如下：

```java
@Resource
private DemoService demoService;

@Test
public void testCluster() {
    try {
        String returnStr = demoService.sayHello( name: "why技术");
        System.out.println("returnStr = " + returnStr);
    } catch (Exception e) {
        System.out.println("抓到异常啦！");
        e.printStackTrace();
    }
}
```

Dubbo中的快速失败

快速失败对应的实现类是：

org.apache.dubbo.rpc.cluster.support.FailfastClusterInvoker

启用该实现类，只需要在Dubbo xml中指定cluster属性为failfast：

```xml
<dubbo:registry address="zookeeper://127.0.0.1:2181" timeout="1000"/>

<!-- 生成远程服务代理，可以和本地bean一样使用demoService -->
<dubbo:reference id="demoService"
                 interface="com.example.dubbo.cluster.provider.service.DemoService"
                 cluster="failfast"
/>
```

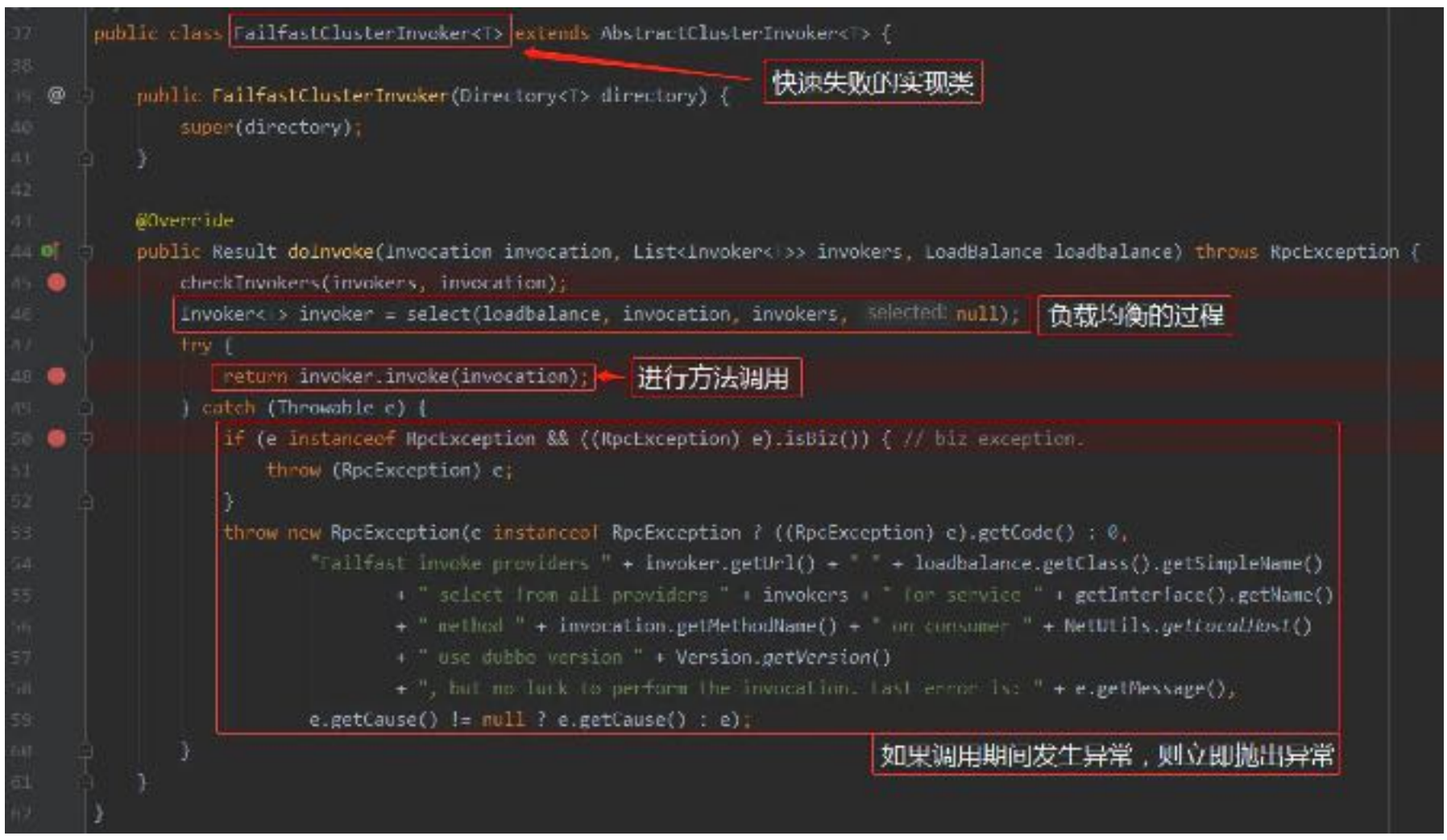
先看一下实现类上的注释是怎么写的：

```java
/**
 * Execute exactly once, which means this policy will throw an exception immediately in case of an invocation error.
 * Usually used for non-idempotent write operations
 *
 * <a href="http://en.wikipedia.org/wiki/Fail-fast">Fail-fast</a>
 *
 */
public class FailfastClusterInvoker<T> extends AbstractClusterInvoker<T> {
```

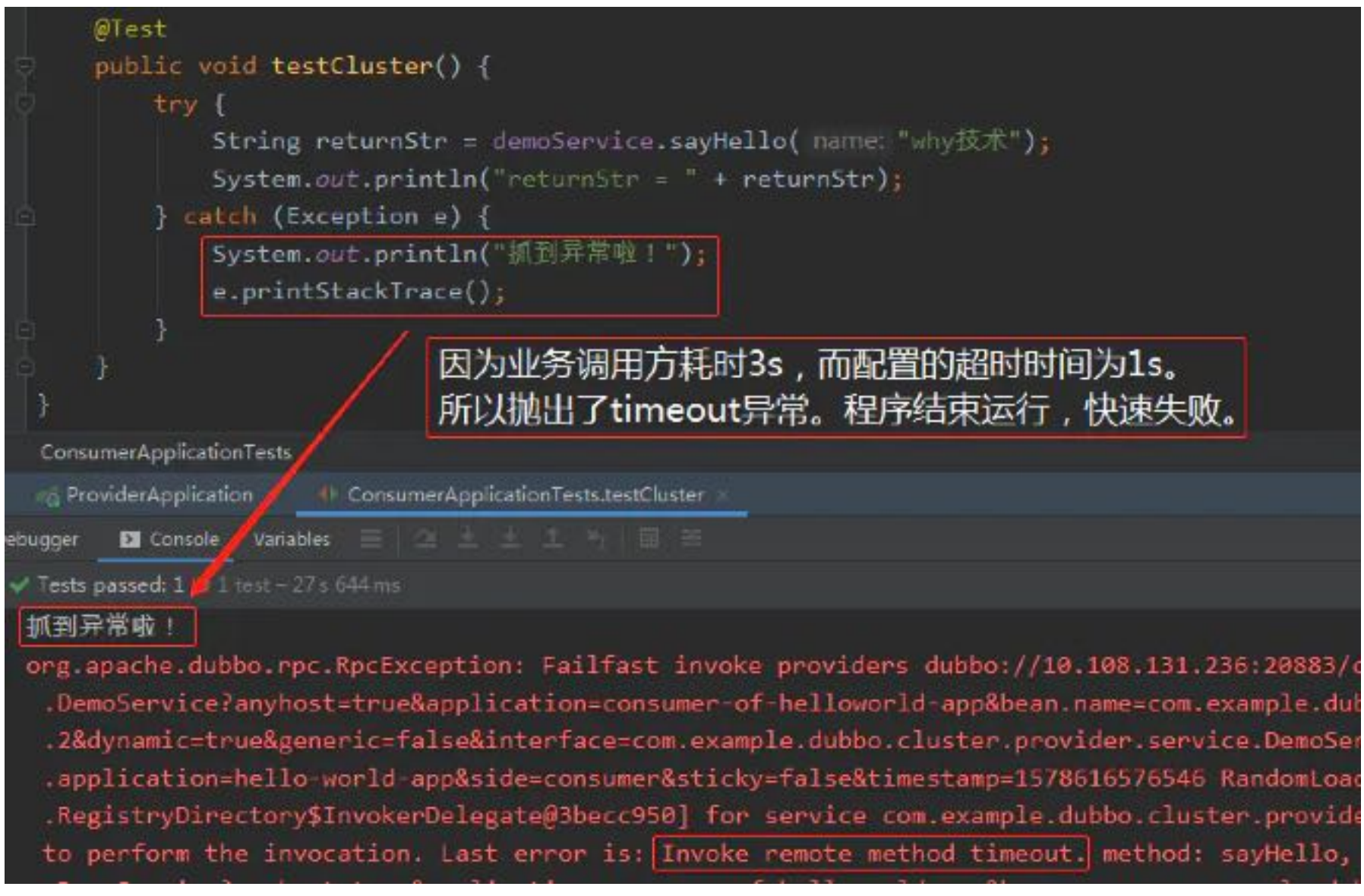
Execute exactly once, which means this policy will throw an exception immediately in case of an invocation error. Usually used for non-idempotent write operations。

FailfastClusterInvoker 只会进行一次调用，失败后立即抛出异常。适用于幂等操作，比如新增记录。

实现类的源码如下：



执行结果如下：



Dubbo中的失败安全

失败安全对应的实现类是：

org.apache.dubbo.rpc.cluster.support.FailsafeClusterInvoker

启用该实现类，只需要在Dubbo xml中指定cluster属性为failsafe：

```
<dubbo:registry address="zookeeper://127.0.0.1:2181" timeout="1000"/>

<!-- 生成远程服务代理，可以和本地bean一样使用demoService -->
<dubbo:reference id="demoService"
                 interface="com.example.dubbo.cluster.provider.service.DemoService"
                 cluster="failsafe"
/>
```

先看一下实现类上的注释是怎么写的：

```
/**
 * When invoke fails, log the error message and ignore this error by returning an empty Result.
 * Usually used to write audit logs and other operations
 *
 * <a href="http://en.wikipedia.org/wiki/Fail-safe">Fail-safe</a>
 *
 */
public class FailsafeClusterInvoker<T> extends AbstractClusterInvoker<T> {
```

When invoke fails, log the error message and ignore this error by returning an empty Result. Usually used to write audit logs and other operations

FailsafeClusterInvoker 是一种失败安全的 Cluster Invoker。所谓的失败安全是指，当调用过程中出现异常时，FailsafeClusterInvoker 仅会打印异常，而不会抛出异常。适用于写入审计日志等操作。

实现类的源码如下：

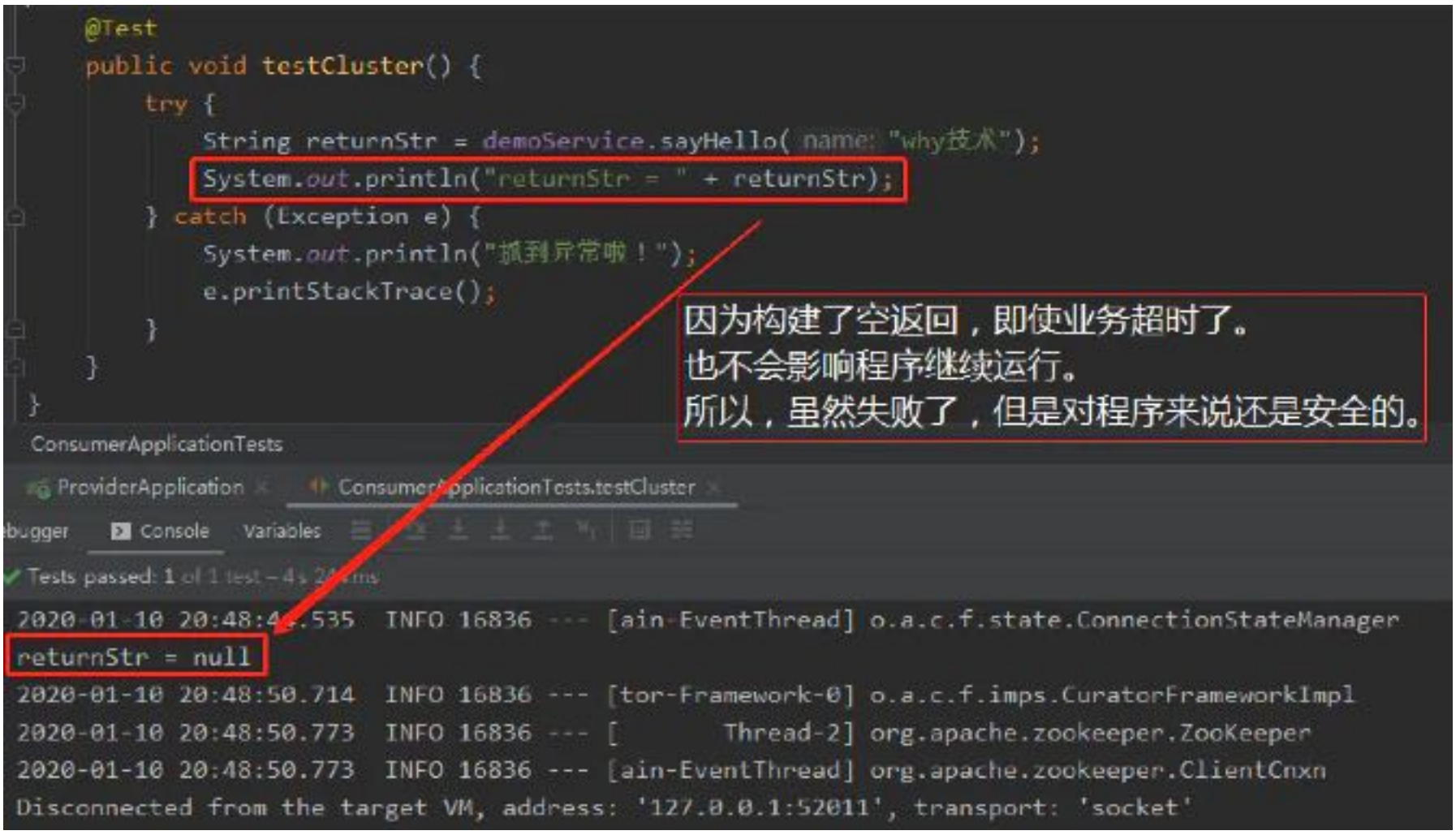
```
public class FailsafeClusterInvoker<T> extends AbstractClusterInvoker<T> {
    private static final Logger logger = LoggerFactory.getLogger(FailsafeClusterInvoker.class);

    public FailsafeClusterInvoker(Directory<T> directory) {
        super(directory);
    }

    @Override
    public Result doInvoke(Invocation invocation, List<Invoker<T>> invokers, LoadBalance loadbalance) throws RpcException {
        try {
            checkInvokers(invokers, invocation);
            Invoker<T> invoker = select(loadbalance, invocation, invokers, selected: null);
            return invoker.invoke(invocation);
        } catch (Throwable e) {
            Logger.error( msg: "Failsafe ignore exception: " + e.getMessage(), e);
            return AsyncRpcResult.newDefaultAsyncResult( value: null, t: null, invocation); // ignore
        }
    }
}
```

异常导致调用失败后，仅仅是打印error日志，然后构建一个空结果返回给调用方

执行效果如下，首先可以看到超时异常被捕获：

```
    @Override
    public Result doInvoke(Invocation invocation, List<Invoker<T>> invokers, LoadBalance loadbalance) throws RpcException {
        try {
            checkInvokers(invokers, invocation);
            Invoker<T> invoker = select(loadbalance, invocation, invokers, selected: null);
            return invoker.invoke(invocation);
        } catch (Throwable e) {
            logger.error( msg: "Failsafe ignore exception: " + e.getMessage(), e);
            return AsyncRpcResult.newDefaultAsyncResult( value: null, t: null, invocation); // ignore
        }
    }
}
```

所以虽然超时了，但是在Test类中，还是打印出了returnStr：



文章背后的故事

本周输出这篇文章实属不易。

由于周六公司年会与运动会，我有幸当选了某队队长，所以本周周一到周五工作之外的时间都在忙碌着运动会的物质筹备、数据统计等相关工作。仅仅有早起的一小会空挡时间能随手翻翻书，但是写文章时间是不够的。

周六活动开始，从早上6点30分起床，到晚上23点做完最后的收尾工作，忙碌了整整一天。所幸的是我所在的小队在10支队伍中脱颖而出，勇夺第一。

在战况异常激烈的拔河环节，在所有拉拉队员整齐划一，嘶声呐喊的口号声中，参赛队员体现出的那种坚韧不拔、咬牙死撑的精神，深深的打动了我。一共有两次轮空的机会，我们队伍都与之擦肩而过，但是我们一次又一次的击败了实力强劲的对手，还是握住了总决赛拔河的绳子。由于对手轮空一轮，所以体力充沛，我们遗憾告负。但是在裁判没有吹哨之前，我们永不言弃。

所有项目比完之后，我所在队伍最终的总积分排名第一。在最终结果没有宣布之前，我们永不言弃。

周日，和我从小一起长大的表姐结婚，早上7点多去现场帮忙。中午吃饭时我都没敢喝太多酒，因为我想着这周的文章还没写，我得保持清醒。但是看到姐姐和姐夫站在大荧幕下的那一刻，我还是感动的热泪盈眶。下午陪家人玩了一下午。晚上到家之后已经非常疲倦了，但是我还是输出了这篇文章。

女朋友问我这周能不能不写，我想了一下说：不能。因为那一瞬间我想到了，路遥先生在《早晨从中午开始》中的一句话：**只有初恋般的热情和宗教般的意志，人才有可能成就某种事业。**