链滴

Motan_ 高可用（HA）

作者：Lord-X

原文链接：https://ld246.com/article/1588142205734

来源网站：链滴

Motan系列文章

- Motan如何完成与Spring的集成
- Motan的SPI插件扩展机制
- Motan服务注册
- Motan服务调用
- Motan心跳机制
- Motan负载均衡策略
- Motan高可用策略

HA即高可用，也可以理解为容错策略。在Motan中，HA作用于Client端，其含义是当RPC调用失败，采取的策略。目前有以下两种策略：

- Failover：失效转移（默认）

配置方式如下：

```
<motan:protocol ... haStrategy="failover"/>
```

其含义是，当RPC调用失败时，自动重试其他服务器。

- Failfast：快速失败

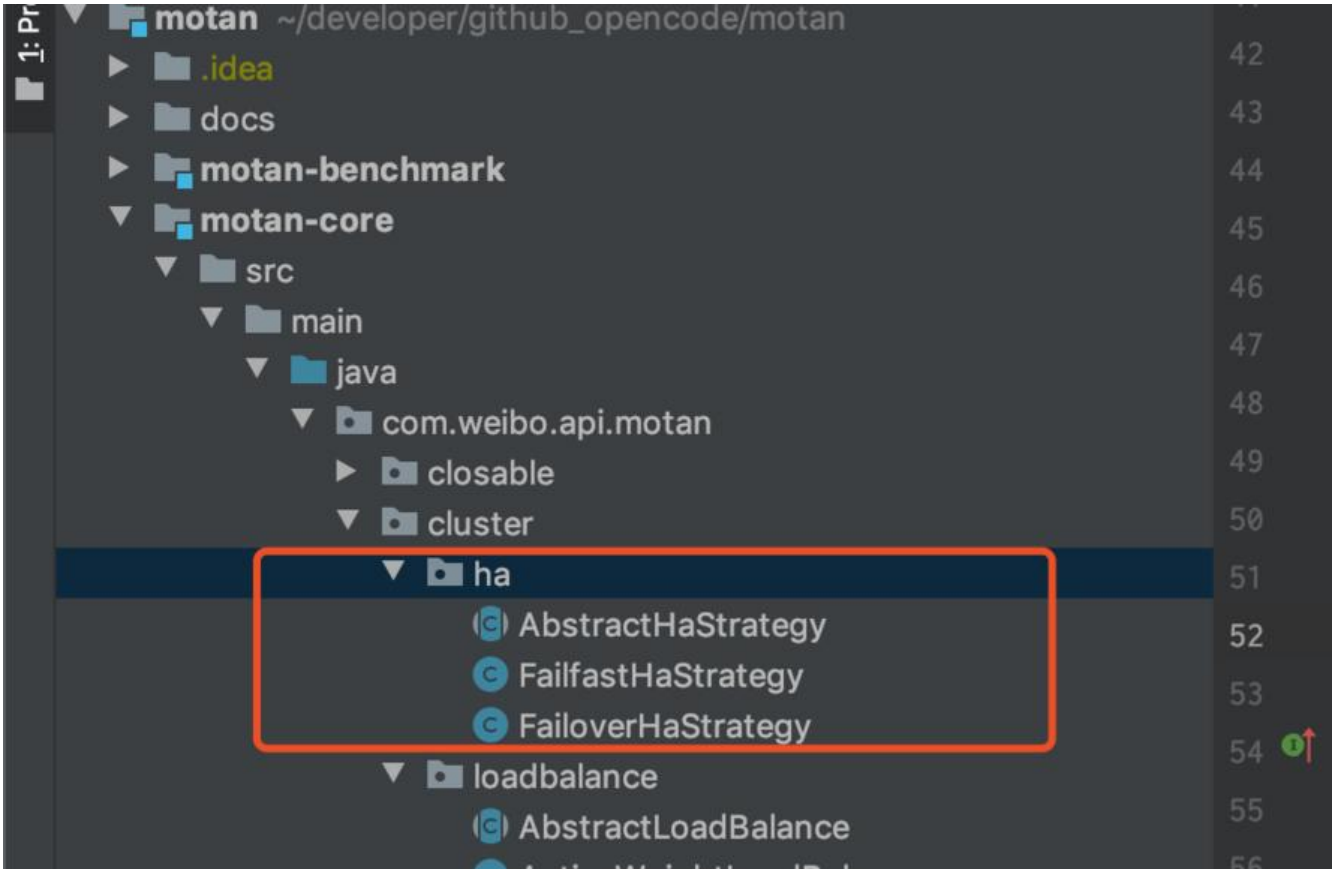
配置方式如下：

```
<motan:protocol ... haStrategy="failfast"/>
```

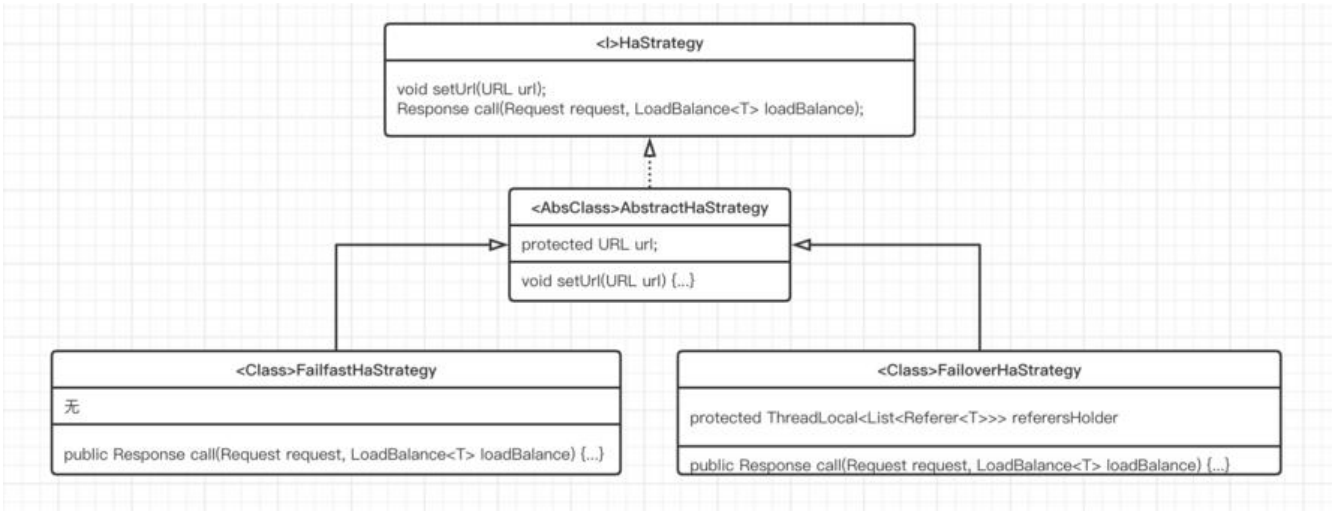
其含义是，只发起一次调用，失败立即报错。

0 工程结构及继承关系

下图展示了HA源码在工程中的位置：



下面来看一下HA的体系结构：



这里最核心的是 call() 方法。这个方法在 HaStrategy 接口定义，由具体实现类实现，用于完成HA策逻辑。

1 HA策略的选取及设置

Motan会在Cluster初始化阶段设置HA策略。HA策略可通过上述配置来定义，如果不配置，默认使用ailover，即失效转移策略。

在实现上，HA采用插件化开发（即SPI），每个HA的具体实现都是一个SPI，其实现类都标注了 @SpMeta 注解，在setHaStrategy时，根据具体的HA名称即可找到具体的实现。以 Failover 为例：

```
@SpiMeta(name = "failover")
public class FailoverHaStrategy<T> extends AbstractHaStrategy<T> {

}
```

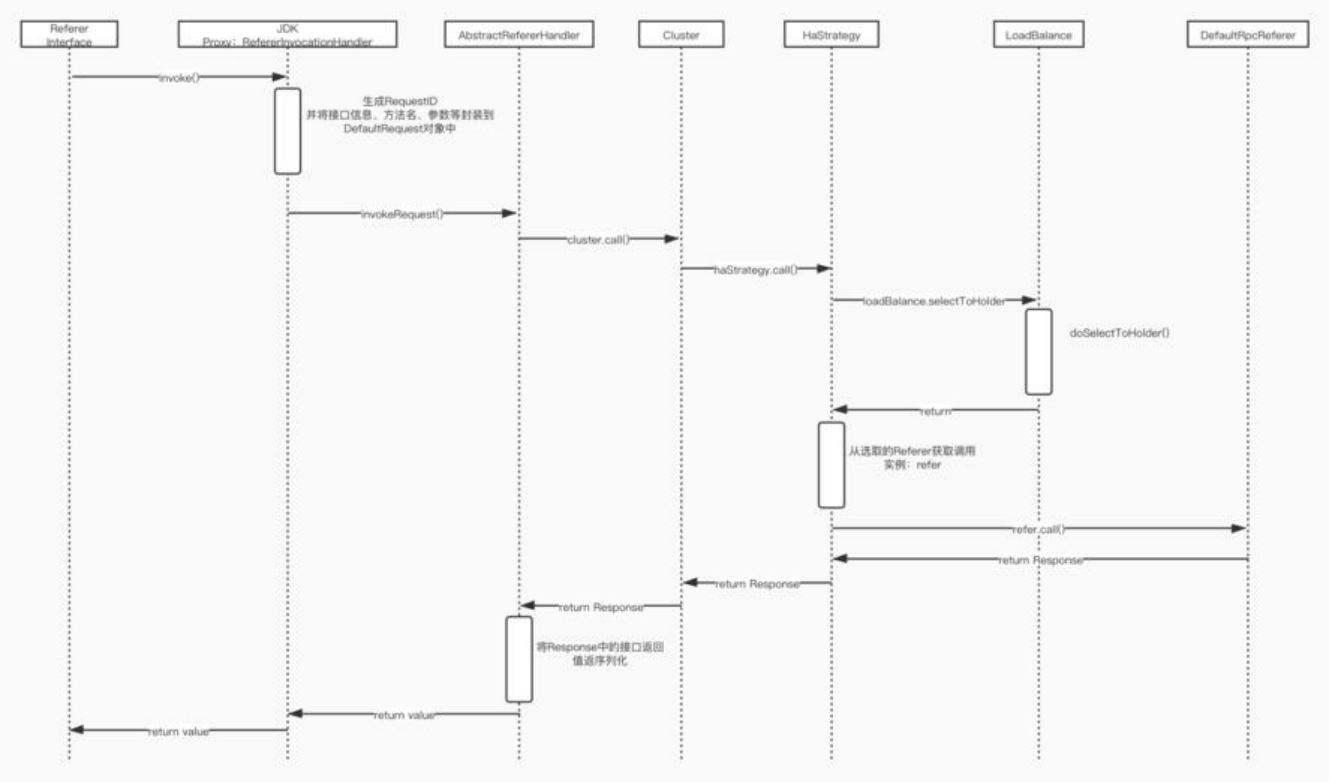
然后在 ClusterSupport 中通过SPI获取HA的具体实现，并setHaStrategy。

```
private void prepareCluster() {
    String clusterName = url.getParameter(URLParamType.cluster.getName(), URLParamType.custer.getValue());
    String loadbalanceName = url.getParameter(URLParamType.loadbalance.getName(), URLParamType.loadbalance.getValue());
    // 获取HA策略，优选获取用户配置的，如果没有配置，取URLParamType配置的默认值，即 failoer
    String haStrategyName = url.getParameter(URLParamType.haStrategy.getName(), URLParmType.haStrategy.getValue());

    cluster = ExtensionLoader.getExtensionLoader(Cluster.class).getExtension(clusterName);
    LoadBalance<T> loadBalance = ExtensionLoader.getExtensionLoader(LoadBalance.class).geExtension(loadbalanceName);
    // 通过SPI的方式获取具体HA实现
    HaStrategy<T> ha = ExtensionLoader.getExtensionLoader(HaStrategy.class).getExtension(hStrategyName);
    ha.setUrl(url);
    cluster.setLoadBalance(loadBalance); // setLoadBalance
    cluster.setHaStrategy(ha);
    cluster.setUrl(url);
}
```

2 调用时序

配置好HA以后，再来看一下调用时序。以下时序图说明了一个RPC请求的调用时序，主要关注HaStraegy的调用时机。



至此，本文已说明HA的设置以及HA的调用时机，下面来介绍一下两种HA的具体实现。

3 HA的实现

下面介绍一下两种HA的具体源码实现：

- Failover：失效转移
- Failfast：快速失败

3.1 Failover

Failover是默认的HA策略。

```
@SpiMeta(name = "failover")
public class FailoverHaStrategy<T> extends AbstractHaStrategy<T> {

    // 用ThreadLocal存储每次调用中 可用的集群备选节点
    protected ThreadLocal<List<Referer<T>>> referersHolder = new ThreadLocal<List<Refere
<T>>>() {
        @Override
        protected java.util.List<com.weibo.api.motan.rpc.Referer<T>> initialValue() {
            return new ArrayList<Referer<T>>();
        }
    };

    @Override
    public Response call(Request request, LoadBalance<T> loadBalance) {

        List<Referer<T>> referers = selectReferers(request, loadBalance);
```

```
        if (referers.isEmpty()) {
            throw new MotanServiceException(String.format("FailoverHaStrategy No referers for r
quest:%s, loadbalance:%s", request,
                    loadBalance));
        }
        URL refUrl = referers.get(0).getUrl();
        // 获取重试次数配置，默认等于0，即不重试
        int tryCount =
                refUrl.getMethodParameter(request.getMethodName(), request.getParamtersDesc(),
URLParamType.retries.getName(),
                        URLParamType.retries.getIntValue());
        // 如果有问题，则设置为不重试
        if (tryCount < 0) {
            tryCount = 0;
        }

        for (int i = 0; i <= tryCount; i++) {
            // 失效转移，当tryCount大于0时，会挨个尝试referers中的节点。
            Referer<T> refer = referers.get(i % referers.size());
            try {
                // 设置重试次数
                request.setRetries(i);
                return refer.call(request);
            } catch (RuntimeException e) {
                // 对于业务异常，直接抛出
                if (ExceptionUtil.isBizException(e)) {
                    throw e;
                // 当重试次数未达到tryCount时，继续下一次循环，否则抛出异常
                } else if (i >= tryCount) {
                    throw e;
                }
                LoggerUtil.warn(String.format("FailoverHaStrategy Call false for request:%s error=%
", request, e.getMessage()));
            }
        }

        throw new MotanFrameworkException("FailoverHaStrategy.call should not come here!");
    }

    // 根据LoadBalance策略选取备选节点，并存在ThreadLocal中
    protected List<Referer<T>> selectReferers(Request request, LoadBalance<T> loadBalance)
{
        List<Referer<T>> referers = referersHolder.get();
        referers.clear();
        loadBalance.selectToHolder(request, referers);
        return referers;
    }

}
```

由上述代码可知，失效转移机制和 retries 这个参数有关，当这个参数为0时，只会调用一次，如果失
就直接抛出异常了，当这个参数大于0时，会挨个尝试LoadBalance策略返回的可用集群节点。

3.2 Failfast

Failfast会在RPC调用失败时直接抛出异常，其实现很简单：

```
@SpiMeta(name = "failfast")
public class FailfastHaStrategy<T> extends AbstractHaStrategy<T> {

    @Override
    public Response call(Request request, LoadBalance<T> loadBalance) {
        Referer<T> refer = loadBalance.select(request);
        return refer.call(request);
    }
}
```

由于快速失败策略不用重试，所以调用LoadBalance的 select 方法选取一个节点即可（Failover因为要满足失效转移，所以需要多个备选节点，referersHolder 就是获取多个节点的方法），如果节点调失败，则抛出 RuntimeException。

参考

- Motan_Github