# 集群容错：

各种ClusterInvoker：

进行远程调用时，AbstractClusterInvoker的invoke方法会被调用，列举Invoker，再调用子类实现的doInvoke方法进行后续操作（模板方法模式）

不同的子类：

FailoverClusterInvoker在调用失败时，会自动切换Invoker进行重试，默认配置下，Dubbo使用这个类作为缺省ClusterInvoker。doInvoke逻辑：获取重试次数，然后循环调用，失败重试，再每次重试之前都重新列举Invoker，然后通过负载均衡选择Invoker(调用select方法)，调用Invoker的invoke方法。

Select方法：获取调用的方法名，获取sticky配置，sticky代表粘滞连接，指的是调用服务消费者尽可能调用同一个服务提供者，除非该提供者挂了再进行切换。如果stickyInvoker为空，或者不可用，继续调用doSelect选择Invoker。

DoSelect方法：通过负载均衡组件选择Invoker，，需要检查可用性等等，如果不可用继续reselect方法

FailbackClusterInvoker在调用失败后，返回一个空结果给服务提供者。并通过定时任务对失败的调用进行重传，适合执行消息通知等操作。通过ScheduledExecutorService线程池来对失败的调用进行重传

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

FailSafeClusterInvoker是一种失败安全的Cluster Invoker。当调用过程中出现异常时，仅会打印异常，而不是抛出异常，适用于写入日志等操作。

ForkingClusterInvoker会在运行时通过线程池创建多个线程，并发调用多个服务提供者，只要有一个服务提供者成功返回了结果，doInvoke就会立刻结束执行。应用场景是对实时性要求比较高的读操作。首先读取forks配置，超时配置，然后循环选出Invoker。通过newCachedThreadPool来为每个Invoker创建一个执行线程进行远程调用，将结果放到BlockingQueue阻塞队列中，如果出现异常会计数，仅在value>selected.size()时才将异常独享存入阻塞对象。（为什么？因为调用多个服务提供者，只要有一个提供者能够正常返回结果就可以）。最后从阻塞队列中取出远程调用对象。

BroadcastClusterInvoker会逐个调用每个服务提供者，如果其中一台报错，在循环调用结束后，BroadcastClusterInvoker会抛出异常。该类通常用于通知所有提供者更新缓存或日志等本地资源信息。

# LoadBalance

所有负载均衡类均继承自AbstractLoadBalance，这个类实现了LoadBalance接口。负载均衡的入口方法select对invokers列表确认，如果如果还只有一个invoker直接返回，然后调用子类实现的方法doSelect(模板方法模式)。除此之外，还封装了一些公共逻辑，比如服务提供者的权重计算逻辑getWeight。(运行时间/预热时间)\*weight。也就是当服务运行时间时间小于服务预热时间时，对服务进行降权，避免让服务在启动之初就处于高负载状态。

RandomLoadBalance是加权随机算法的具体实现。检测所有权重是否都相同，如果如果都相同随机返回一个就好，else：累加权重，然后获取随机数，用获得的值减去服务提供者的权重值，当offset小于0时，返回相应的invoker

LeastActiveLoadBalance 最小活跃数负载均衡（基于加权）。每个服务提供者活跃数均为0，每收到一个请求活跃数加一，完成请求后活跃数减一。如果活跃数最小的有多个，按加权去随机返回

ConsisitentHashLoadBalance 一致性哈希算法。引入虚拟节点，避免数据倾斜问题。doSelect方法主要做了一些前置工作，比如检测invokers表是不是变动过，以及创建ConsistentHashSelctor，这些工作做完后，调用ConsisitentHashSelector的select方法选择invoker。

ConsistentHashSelector使用 TreeMap存储Invoker虚拟节点。先完成虚拟节点到invoker的映射，再讲虚拟节点存储到TreeMap中。

RoundRobinLoadBalance 加权轮训负载均衡 平滑加权算法

Dubbo 服务调用过程

通过Javaassist框架为服务接口生成动态代理类，调用invocationHandler通过反射执行执行方法，InvokerInvocationHandler内有invoker的引用，它的invoke方法首先拦截在Object勒种定义的方法（未被子类重写or被子类重写），没有拦截到就将method和args封装到RpcInvocation中，并执行后续的调用。

InvokerInvocationHandler中的invoker成员变量类型为MockClusterInvoker，MockClusterInvoker内部封装了服务降级的逻辑。

它的invoke方法首先获取mock配置值如果没有mock逻辑，直接调用invoker对象的invoke方法。或者是force：xxx, 直接执行mock逻辑，不发起远程调用，如果是fail:xxx表示用户调用服务失败后再执行mock逻辑。

Invoke方法也是模板方法模式，父类AbstractInvoker的invoke方法用于添加信息到invocation的attachment变量。

DubboInvoker的doInvoke方法获取异步配置，决定同步调用还是异步调用，Dubbo实现同步和异步调用的关键在于谁调用了ResponseFuture的get方法。同步是框架自身调用get方法，异步是将future设置到上下文中，返回一个空结果，由用户调用该方法。

调用ExchangeClient的类（这里有很多有引用，形成了一个栈），负责封装request和response，最后调用NettyClient的send方法发送，编码工作。

服务提供方接受请求后，首先执行解码逻辑，通过反序列化将诸如path、version、调用方法名、参数列表等信息一次解析出来，并设置到相应字段中，最终得到一个具有完整调用信息的DecodeableRpcInvocation对象，封装在request对象中，NettyHandler的messageReceived方法会受到这个对象，继续传递，最后封装到Runnable实现类中，放入线程池。 //Dubbo的线程派发策略，针对不同的消息有选择性的派发至线程池（I/O线程，请求消息等等）

ChannelEventRunnable不执行处理逻辑，将请求信息传递给其他handler处理，解码，最后传递给HeaderExchangeHandler，HeaderExchangHandler首先向后调用，得到调用结果，然后将调用结果封装到Response对象中，最后再将该记过返回给服务消费方。

ExchangeHandler通过getInvoker获得invoker，调用invoker的invoke方法。Invoke实例是运行时运行时通过JavassistProxyPactory动态生成的。

返回结果后，对客户端对响应数据解码，解码完成后，Dubbo会将响应对象派发到线程池上，通过调用编号从Futures集合中查找指定线程

# Dubbo SPI机制

SPI全称为Service Provider Interface，是一种服务发现机制，本质是将接口实现类的全限定名配置在文件中，并由服务加载器读取配置配置而文件，加载实现类。

Dubbo并未使用Java SPI，而是重新实现，用ExtensisonLoader类加载，Dubbo SPI通过键值对的方式类配置，可以按需加载指定类。

入口方法为ExtensionLoader的getExtension方法，首先检查缓存，缓存未命中则调用createExtension方法创建拓展对象。

createExtension方法从配置文件中加载所有的拓展类，然后通过反射创建实例，向实例中助于依赖，循环创建Wrapper实例并注入依赖

Dubbo IOC通过setter注入依赖

# Dubbo自适应拓展机制

在Dubbo中有些拓展不想在框架启动阶段被加载，而是希望在拓展方法被调用时，根据运行参数进行加载，通过自适应扩展机制实现。首先Dubbo会为拓展接口生成具有代理功能的代码，然后通过javaassist或jdk编译这段代码，得到Class类，在通过反射创建代理类。

@Adaptive可注解在类或方法上，当注解在类上时，不会为该类生成代理类，注解在方法上时，Dubbo则会为该方法生成代理逻辑。

入口方法： getAdaptiveExtension方法，首先检查缓存，缓存未命中则调用createAdaptiveExtension方法创建自适应拓展。

createAdaptiveExtension调用getAdaptiveExtensionClass方法获取自适应拓展的Class类对象，通过反射进行实例化，调用injectExtension方法注入依赖

构建类对象逻辑： 构建自适应拓展代码（具体Adaptive的方法实现是通过SPI加载，再调用方法），获得编译器实现类，编译代码，生成Class

# 服务导出

Dubobo服务导出过程始于Spring容器发布刷新时间，Dubbo在接受到时间后，会激励执行服务导出逻辑。整个逻辑大致可分为三个部分，第一部分是前置工作，主要用于检查参数，组装URL，第二部分是导出服务，包含导出服务到本地（JVM），和导出服务到远程两个过程。第三部分是向注册中心注册服务，用于服务发现。

服务导出的入口方法是ServiceBean 对onApplicaitonEvent方法，在Spring上下文刷新事件后执行服务导出操作。根据条件决定是否导出，是否延迟导出。

前置工作主要包含两个部分，分别是配置检查以及URL装配。export()方法对export和delay配置检查，然后调用doExport方法。

首先检测<dubbo:service>标签的interface属性合法性

检测核心配置类对象是否为空，那个获取相应实例

检测并处理泛化服务类型和普通服务类  
 处理本地存根类

分局配置和协议组装URL

导出服务，分为导出到本地和导出到远程

Invoker通过ProxyFactory创建而来，Dubbo默认使用JavaassistProxyFactory，建立class的包装类