1、dubbo是一个分布式、高性能、透明化的**RPC服务框架。无论是不是微服务架构，我们都可以通过将服务交给dubbo供其他系统调用。优缺点如下：**

Dubbo优缺点

优点：

1. 透明化的远程方法调用
   - 像调用本地方法一样调用远程方法；只需简单配置，没有任何API侵入。
2. 软负载均衡及容错机制
   1. 可在内网替代nginx lvs等硬件负载均衡器。
3. 服务注册中心自动注册 & 配置管理
   -不需要写死服务提供者地址，注册中心基于接口名自动查询提供者ip。
   使用类似zookeeper等分布式协调服务作为服务注册中心，可以将绝大部分项目配置移入zookeeper集群。
4. 服务接口监控与治理
   -Dubbo-admin与Dubbo-monitor提供了完善的服务接口管理与监控功能，针对不同应用的不同接口，可以进行 多版本，多协议，多注册中心管理。

缺点：

- 只支持JAVA语言

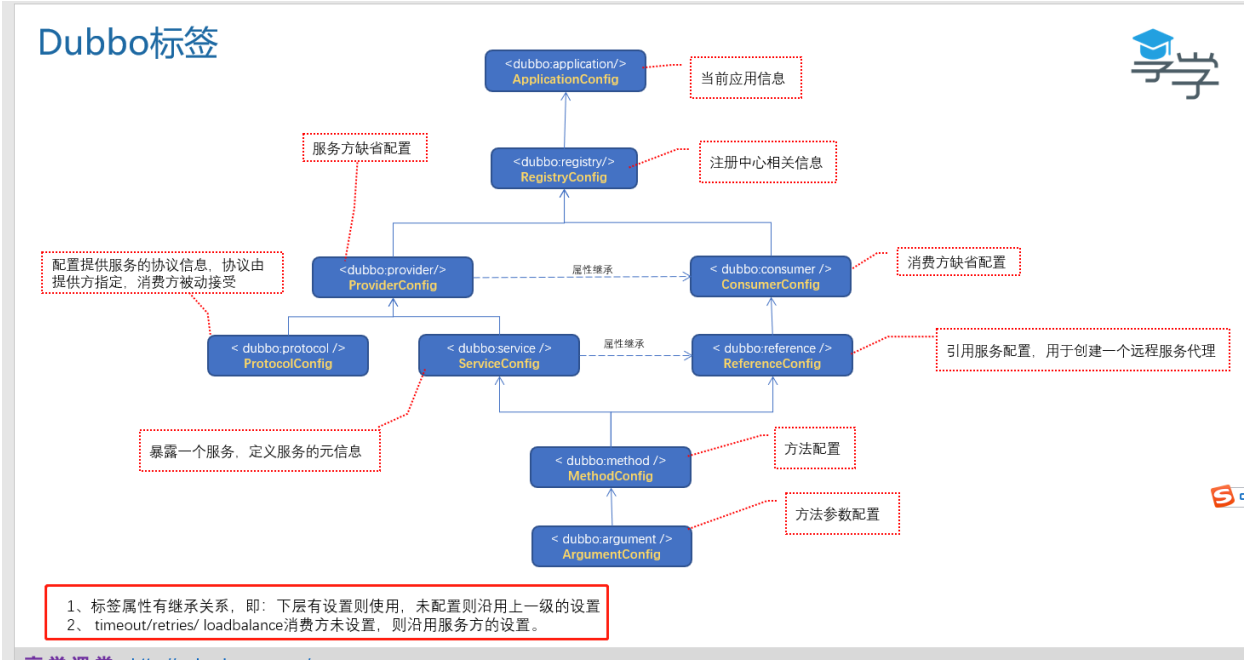
1、dubbo只能支持spring管理的服务；

2、dubbo上层标签中设置的属性能够被下层标签自动继承，比如provider标签中设置一些公共的属性如timeout、retries等，则对下层标签如service等都起作用，如下图：

```
                accepts="${dubbo.protocol.dubbo.accepts}"/>
    <dubbo:provider retries="0" timeout="${dubbo.provider.timeout}" />

    <dubbo:service interface="com.itfin.cb.gl.facade.GlService" ref="glService"/>

    <dubbo:service interface="com.itfin.cb.gl.batch.facade.RunGlJobService" ref="runGlJobService">
    </dubbo:service>

    <dubbo:service interface="com.itfin.cb.gl.batch.facade.GetGlJobExecutionListService" ref="getGlJob
```

3、消费服务方标签不设置属性的话会自动继承消费提供方的标签属性。

4、标签属性继承关系：



5、dubbo的service注解有两功能，第一注册服务到spring容器，第二能暴露为rpc服务---容器有两个实例。即在本工程中通过@Autowired引到，也可以在其它工程中通过@Reference引用。

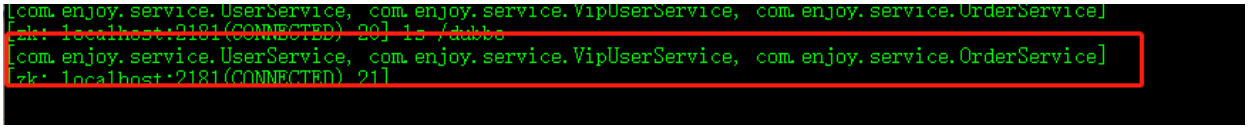
6、dubbo的分包原则，一般将接口拆分成独立模块（接口、请求实体类、响应实体类）提供给服务调用者，例如我们的facade包就是这样做的，如图：

7、将facade包独立出来以后，同工程的其它模块中引入facade包并实现接口然后通过dubbo向外提供服务就可以了，如图：

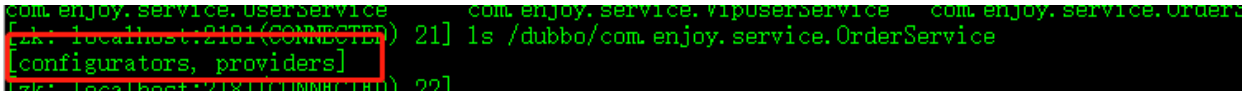
8、对于dubbo服务启动成功以后前台线程自动关闭的问题，解决方案是需要将线程阻塞住，如图：

```java
public class SofaSpringBootApplication {
    public static void main(String[] args) throws Exception {
        SpringApplication springApplication = new SpringApplication(SofaSpringBootApplication.class);
        // a non-web application
        // springApplication.setWebEnvironment(false);
        ApplicationContext applicationContext = springApplication.run(args);

        System.out.println("Current Server Application Context : " + applicationContext + "\n"
                + "==============启动成功=============");
        // 阻塞主线程退出以达到常驻后台的目的
        new CountDownLatch(1).await();
    }
```

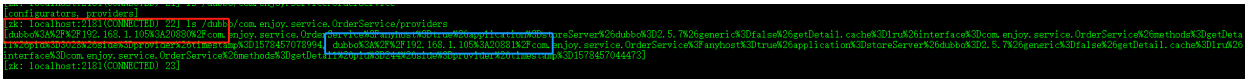
9、hivegl是一个应用，hivegl应用包含多个服务需要向外提供（例如GlService、ExtractData，此处的服务指一个应用向外提供的接口有几个），当我们在不同机器启动hivegl应用并注册dubbo服务成功以后，zk中注册的服务目录如下：



进入每个服务目录节点以后可以查看服务的详细提供者信息：

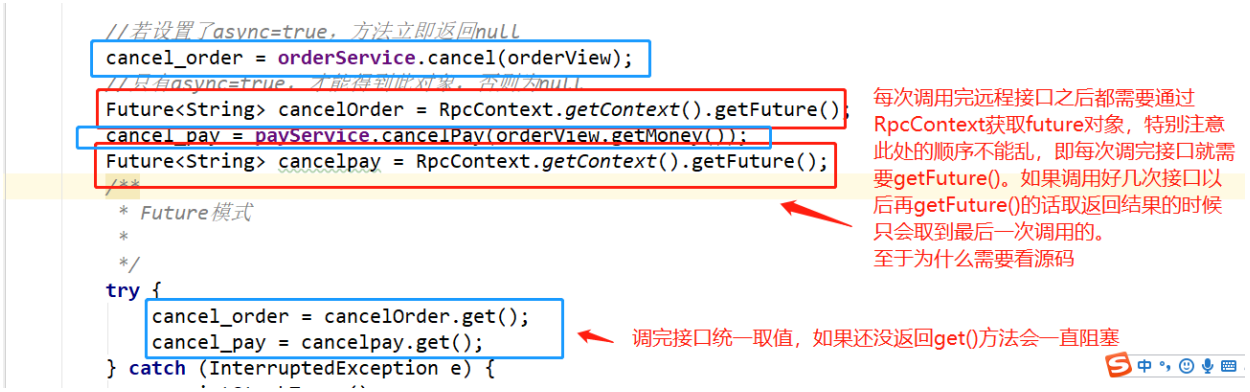


进入providers，可以看到OrderService有两个提供者（协议为dubbo，其他的有rmi等）：



10、dubbo异步并发调用（调用多个接口的时候并发执行，然后统一在某个点获取执行结果，好处在于执行时间的长短取决于最慢返回接口调用时间【木桶效应】，调用A需2s，调用B需4s，如果同步的话一共执行时间大于6s，而异步的话执行时间大于4s）：

1. future模式如图（很少用）：



2、异步回调（oninvoke、onreturn、onthrow），方法第一个参数为返回值，其余为接口参数：

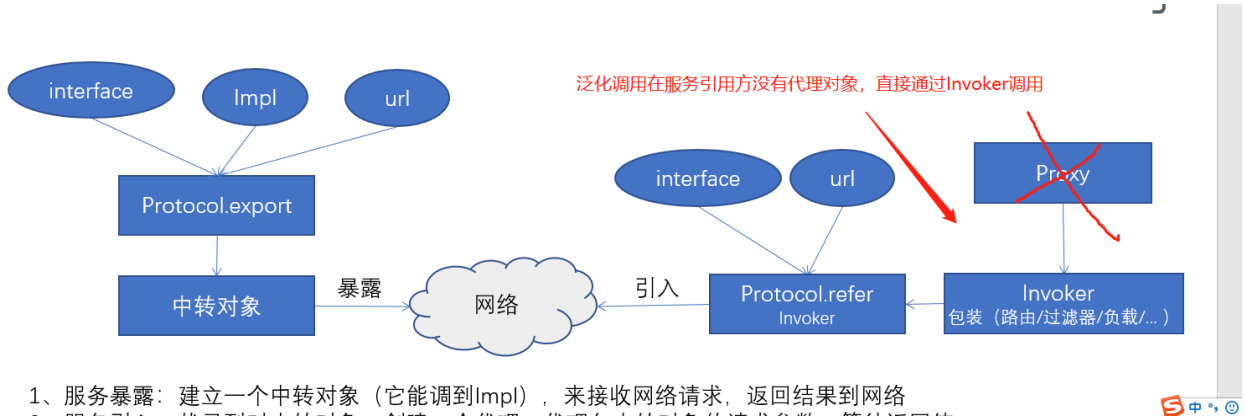
```
 */
public class CallBack {

    //第一个参数，为返回结果值，后续参数是入参
    public void onOrderSubmit(OrderEntiry result,OrderEntiry form){
        System.out.println("生成了一单，金额："+ result.getMoney());
    }

    //第一个参数，为返回结果值，后续参数是入参
```

调用接口返回的结果值

后边全是调用接口时传的参数，图片中只有一个参数

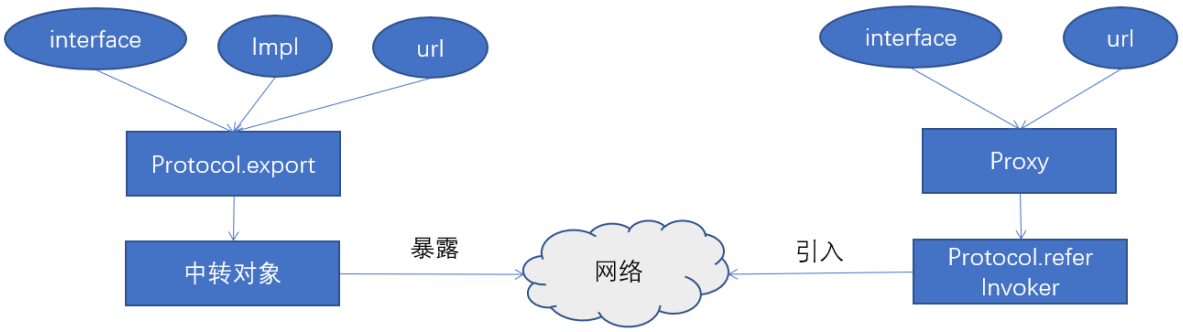
11、回声测试。

12、泛化调用，实际是将class名传过去，通过反射调用。泛化调用一般是应急用的，假如A应用没有引B应用的facade（接口包），此时A可以先通过泛化调用调B的服务，等后边应用升级的时候在将泛化调用平滑抹掉。泛化调用相比正常调用是在服务引用方少了代理对象（因为没有接口信息）。



13、dubboRpc调用流程。在提供方建立一个中转对象（不同协议有不同的中转对象，相同协议共用），它可以直接调用接口实现类。服务引用方通过服务接口创建代理对象，并通过socket将服务名、方法名、参数传给服务方中转对象，中转对象通过反射调用实现类。具体看rpc和spring rmi实现。

Dubbo的RPC服务暴露和引入



1、服务暴露：建立一个中转对象（它能调到Impl），来接收网络请求，返回结果到网络
2、服务引入：找寻到对中转对象，创建一个代理，代理向中转对象传请求参数，等待返回值
3、URL总线：中转对象---URL信息，一一对应，通过URL来找寻中转对象
3、URL包含完整rpc信息： rmi://192.168.56.1:20881/com.enjoy.service.ProductService?anyhost=true&application=storeServer&dubbo=2.5.7&generic=false&interface=com.enjoy.service.ProductService&methods=modify,getDetail,status&pid=2476&side=provider&timestamp=1542267315993

14、dubbo中不是一个服务一个端口，而是一个协议一个端口，服务器为dubbo中使用到的不同协议开通不同端口供通信使用。

15、http其实也是rpc调用的一种，只不过使用的协议不同，所以在网络上传输的效率不同。http是应用层协议，而TCP、UDP是传输层协议，那么直接通过TCP、UDP调用肯定比http快。http在最顶层，它最终也会经过传输层（tcp）。

16、当一个接口有多个实现时我们可以通过service标签的group属性分组。

17、有多少个<dubbo:service>、<dubbo:reference>标签就会创建多少个ServiceBean、ReferenceBean并把它们交给IOC容器扩展。

```
    }

    @Override
    public void init() {
        registerBeanDefinitionParser( elementName: "application", new DubboBeanDefinitionParser(ApplicationConfig.class, required: true));
        registerBeanDefinitionParser( elementName: "module", new DubboBeanDefinitionParser(ModuleConfig.class, required: true));
        registerBeanDefinitionParser( elementName: "registry", new DubboBeanDefinitionParser(RegistryConfig.class, required: true));
        registerBeanDefinitionParser( elementName: "config-center", new DubboBeanDefinitionParser(ConfigCenterBean.class, required: true));
        registerBeanDefinitionParser( elementName: "metadata-report", new DubboBeanDefinitionParser(MetadataReportConfig.class, required: true));
        registerBeanDefinitionParser( elementName: "monitor", new DubboBeanDefinitionParser(MonitorConfig.class, required: true));
        registerBeanDefinitionParser( elementName: "metrics", new DubboBeanDefinitionParser(MetricsConfig.class, required: true));
        registerBeanDefinitionParser( elementName: "provider", new DubboBeanDefinitionParser(ProviderConfig.class, required: true));
        registerBeanDefinitionParser( elementName: "consumer", new DubboBeanDefinitionParser(ConsumerConfig.class, required: true));
        registerBeanDefinitionParser( elementName: "protocol", new DubboBeanDefinitionParser(ProtocolConfig.class, required: true));
        registerBeanDefinitionParser( elementName: "service", new DubboBeanDefinitionParser(ServiceBean.class, required: true));
        registerBeanDefinitionParser( elementName: "reference", new DubboBeanDefinitionParser(ReferenceBean.class, required: false));
        registerBeanDefinitionParser( elementName: "annotation", new AnnotationBeanDefinitionParser());
    }

}
```

只有这三个标签对应解析出来的实例才会加载IOC容器中即都是spring bean，其他的都只是存储配置属性而已，不会加载到IOC容器中

这三个通过实现ApplicationContextAware接口向容器中加载bean的

```
 * @export
 */
public class ServiceBean<T> extends ServiceConfig<T> implements InitializingBean, DisposableBean,
        ApplicationContextAware, ApplicationListener<ContextRefreshedEvent>, BeanNameAware,
        ApplicationEventPublisherAware {


    private static final long serialVersionUID = 213195494150089726L;

    private final transient Service service;
```

18、dubbo中所有配置信息最终都会反应到它的URL总线。

19、服务发布流程：

20何谓SPI以及SPI思想（https://mp.weixin.qq.com/s/bQc_tASkfsojlcd897kLtA）

20、jdk spi中实现类的路径是写死的，必须放在 META-INF/services/ 目录下：

```
 */

public final class ServiceLoader<S>                    路径是定死的
    implements Iterable<S>
{

    private static final String PREFIX = "META-INF/services/";

    // The class or interface representing the service being loaded
    private final Class<S> service;

    // The class loader used to locate, load, and instantiate providers
    private final ClassLoader loader;
```

21、jdk spi与dubbo spi（spi文件名是接口的全路径名称，不是实现类的）

- **jdk spi在指定调用者具体使用接口的哪个实现类时需要在services中写死，并且当有多个实现类的时候需要通过迭代器去遍历**

3.JDK SPI缺点

- 需要遍历所有的实现，并实例化，然后我们在循环中才能找到我们需要的实现。
- 配置文件中只是简单的列出了所有的扩展实现，而没有给他们命名。导致在程序中很难去准确的引用它们。
- 扩展如果依赖其他的扩展，做不到自动注入和装配
- 不提供类似于Spring的AOP功能
- 扩展很难和其他的框架集成，比如扩展里面依赖了一个Spring bean，原生的Java SPI不支持

所以Java SPI应付一些简单的场景是可以的，但对于Dubbo，它的功能还是比较弱的。Dubbo对原生SPI机制进行了一些扩展。接下来，我们就更深入地了解下Dubbo的SPI机制。

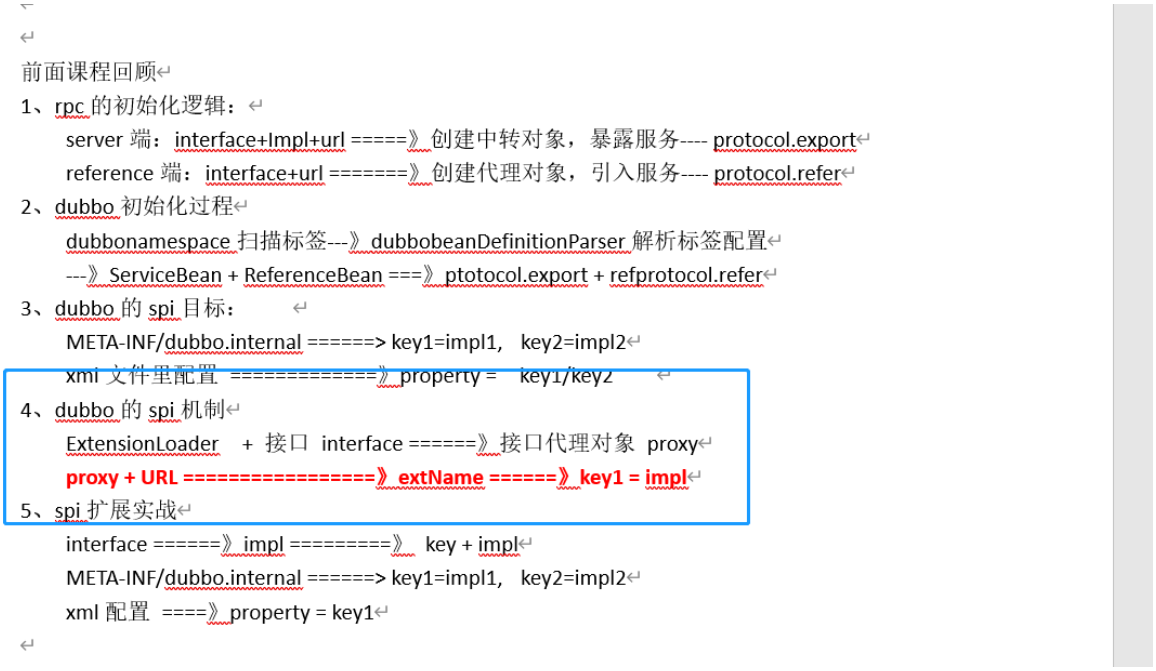
- **dubbo spi使得dubbo的扩展性很牛叉**

总结dubbo SPI

- 对Dubbo进行扩展，不需要改动Dubbo的源码
- 自定义的Dubbo的扩展点实现，是一个普通的Java类，Dubbo没有引入任何Dubbo特有的元素，对代码侵入性几乎为零。
- 将扩展注册到Dubbo中，只需要在ClassPath中添加配置文件。使用简单。而且不会对现有代码造成影响。符合开闭原则。
- Dubbo的扩展机制支持IoC,AoP等高级功能
- Dubbo的扩展机制能很好的支持第三方IOC容器，默认支持Spring Bean，可自己扩展来支持其他容器，比如Google的Guice。
- 切换扩展点的实现，只需要在配置文件中修改具体的实现，不需要改代码。使用方便。

22、dubbo中有@SPI注解的接口表示都是可扩展的。可以通过dubbo spi专门做一个对dubbo的扩展项目，让其它项目引用并使用。参考store_spi.

23、通过改变URL对象的内容获取不同的扩展名，然后找到相应的实现类（即通过URL内容动态加载不同实现类）：



23、dubbo spi相对于jdk spi来说，dubbo中是以key—value方式存储的，我们在标签中根据标签属性值（即key）去获取响应的实现类。但至于为什么要生成自编码代理对象（通过代理对象从url中获取key，然后在根据key获取实现类）是因为这样实现比较优雅，可以动态修改url的值来切换实现对象。

- **直接获取实现类方式**

5.3、对于 spi 的扩展加载器，可以直接使用它的 extName 取目标类的方式

```
Protocol dubboProtocol = (Protocol)loader.getExtension(extName);
```

- **自编码生成代理类获取实现类方式**

```
public Exporter export(Invoker arg0) throws RpcException {
    if (arg0 == null) throw new IllegalArgumentException("com.alibaba.dubbo.rpc.Invoker argument == null");

    if (arg0.getUrl() == null)
        throw new IllegalArgumentException("com.alibaba.dubbo.rpc.Invoker argument getUrl() == null");

    URL url = arg0.getUrl();

    //默认选择dubbo协议，否则根据url中带的协议属性来选择对应的协议处理对象，这样可以动态选择不同的协议
    String extName = (url.getProtocol() == null ? "dubbo" : url.getProtocol());

    if (extName == null)
        throw new IllegalStateException("Fail to get extension(com.alibaba.dubbo.rpc.Protocol) name from url(" + url.toString() + ") use k

    //根据拿到的协议key从缓存的map中取协议对象
    Protocol extension = (Protocol) ExtensionLoader.getExtensionLoader(Protocol.class).getExtension(extName);

    return extension.export(arg0);
}
```

24、dubbo中可以通过RpcContext来传递上下文信息。可参考Filter示例（在消费方将本机ip闯传进去，服务提供方就可以拿到这个ip）。

```
 * @see com.alibaba.dubbo.rpc.filter.ContextFilter
 */
public class RpcContext {

    private static final ThreadLocal<RpcContext> LOCAL = initialValue() -> { return new RpcContext(); };
    private final Map<String, String> attachments = new HashMap<~>();
    private final Map<String, Object> values = new HashMap<~>();
    private Future<?> future;

    private List<URL> urls;

    private URL url;
```

因为是static属性，所以可以做缓存，只要赋值了，就可以在任何地方拿到

我们工作中可以通过static属性做缓存来存取数据，hivegl中缓存ftp配置就是这样做的

25、dubbo负载是在消费端使用的，即标签的loadbalance属性。但是为什么也可以在服务端设置呢？因为标签属性继承，消费方会继承服务方标签属性，具体看继承关系。

26、dubbo统一数据模型URL。 dubbo扩展点的实现方法都包含URL对象，这样扩展点可以从URL拿到配置信息，所有的扩展点自己定好配置的Key后，配置信息从URL上从最外层传入。URL在配置传递上即是一条总线。

27、dubbo中zk客户端（zkCLient、Curator）的选择可以通过标签的client属性配置。

28、dubbo服务注册的时候会将url信息发布到zk，消费端本地会缓存一份zk中的服务注册信息并注册订阅事件，当zk服务提供者发生变化时（即zk中节点数据发生变化时）会回调监听从而更新消费端本地缓存。