dubbo（官网地址）是一个分布式服务框架，致力于提供高性能和透明化的RPC远程服务调用方案，是阿里巴巴SOA服务化治理方案的核心框架。目前，阿里巴巴内部已经不再使用dubbo，但对很对未到一定量级的公司来说，dubbo依然是一个很好的选择。

之前在使用duubo的时候，对dubbo有了一些初步的了解，但没有深入，有些问题还是不清楚。所以准备静下心来看下dubbo源码。这里假设你对dubbo有一定的了解，不再详细的讲解dubbo的架构。如果没接触过dubbo，可以先从其官网了解。

dubbo号称通过spring的方式可以透明化接入应用，对应用没有任何api侵入。下面看看官方的consumer demo，其配置如下：

[html] view plain copy

```
1. <beans xmlns="http://www.springframework.org/schema/beans"
2.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3.     xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
4.     xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
5.     http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd">
6.
7.         <dubbo:reference id="demoService"
interface="com.alibaba.dubbo.demo.DemoService" />
8.
9.     <bean class="com.alibaba.dubbo.demo.consumer.DemoAction"
init-method="start">
10.         <property name="demoService" ref="demoService" />
11.     </bean>
12.
13. </beans>
```

另外还提供了一份properties文件：

[plain] view plain copy

```
1. dubbo.container=log4j,spring
2. dubbo.application.name=demo-consumer
```

```
3. dubbo.application.owner=
4. dubbo.registry.address=multicast://224.5.6.7:1234
```

应用中的调用：

[java] view plain copy

```
1. package com.alibaba.dubbo.demo.consumer;
2.
3. import com.alibaba.dubbo.demo.DemoService;
4.
5. public class DemoAction {
6.
7.     private DemoService demoService;
8.
9.     public void setDemoService(DemoService demoService) {
10.         this.demoService = demoService;
11.     }
12.
13. <pre name="code" class="java">    public void start() throws
Exception {
14.         String hello = demoService.sayHello("who are you");
15.         System.out.println(hello);
16.     }
17. }
```

可以看到，代码方面确实是零侵入，而在配置方面，则是增加了一些服务的声明，环境配置之类的（不可缺少）。对于开发者来说非常友好。那么dubbo是如何做到这点的呢。 这个demoService在consumer端明明没有具体的实现，为何能够正常的调用并获取到结果？ 要了解原因，必须先了解spring开发的一个接口：FactoryBean。

spring中有两种bean，一种是普通的bean，一种是工厂bean，即FactoryBean。普通bean通过class字符串代表的类直接实例化对象，而工厂bean则是通过class字符串代表的工厂类的getObject()方法来实例化对象。如以下FactoryBean在spring中返回的是MyObject对象，而不是MyFactoryBean对象。 这里不过多介绍FactoryBean，有兴趣的同学可以自行google。

[java] view plain copy

```
1. class MyFactoryBean implements FactoryBean<Object> {
2.     public Object getObject() throws Exception {
3.         return new MyObject();
4.     }
5.
6.     ........
7. }
```

了解了FactoryBean后，仍然有困惑。在上面的配置里，并没有出现FactoryBean的实现类。这里需要了解另外一个spring的知识点：schema扩展。在dubbo中，所有namespace=dubbo的标签将被dubbo自己解析，具体见com.alibaba.dubbo.config.spring.schema.DubboNamespaceHandler。其中reference对应的类为com.alibaba.dubbo.config.spring.ReferenceBean，ReferenceBean是一个FactoryBean，通过getObject()来产生代理类，我们的故事也是从此类开始。

ReferenceBean继承自ReferenceConfig，并实现了FactoryBean, ApplicationContextAware, InitializingBean, DisposableBean。

由于实现了InitializingBean，在初始化各个属性后会调用afterPropertiesSet， 实现比较简单，主要是对没有初始化的几个属性尝试用公共的默认配置进行初始化，这里就不再细讲：

[java] view plain copy

```
1. public void afterPropertiesSet() throws Exception {
2.     if (getConsumer() == null) {
3.         // 如果没有配置consumer属性，则使用默认的consumer属性
4.     }
5.     if (getApplication() == null
6.             && (getConsumer() == null ||
getConsumer().getApplication() == null)) {
7.         // 如果没有配置application则使用默认的application
8.     }
9.     if (getModule() == null
10.             && (getConsumer() == null ||
getConsumer().getModule() == null)) {
11.         // 如果没有配置module则使用默认的module
12.     }
```

```
13.     if ((getRegistries() == null || getRegistries().size() ==
0)
14.             && (getConsumer() == null ||
getConsumer().getRegistries() == null ||
getConsumer().getRegistries().size() == 0)
15.             && (getApplication() == null ||
getApplication().getRegistries() == null ||
getApplication().getRegistries().size() == 0)) {
16.         // 如果没有配置registries则使用默认的registries
17.     }
18.     if (getMonitor() == null
19.             && (getConsumer() == null ||
getConsumer().getMonitor() == null)
20.             && (getApplication() == null ||
getApplication().getMonitor() == null)) {
21.         // 如果没有配置monitor则使用默认monitor
22.     }
23.     // 如果设置了init属性且为true，则初始化对象，默认是不初始化的
24.     Boolean b = isInit();
25.     if (b == null && getConsumer() != null) {
26.         b = getConsumer().isInit();
27.     }
28.     if (b != null && b.booleanValue()) {
29.         getObject();
30.     }
31. }
```

由于实现了FactoryBean，当需要初始化或者应用中需要用到时，会调用getObject()方法获取实际的对象：

[java] view plain copy

```
1. public Object getObject() throws Exception {
2.     return get();
3. }
```

get方法在父类ReferenceConfig中， 其实现为：

[java] view plain copy

```
1. public synchronized T get() {
2.     // 已经销毁则不能再获取
3.     if (destroyed){
4.         throw new IllegalStateException("Already destroyed!");
```

```
5.      }
6.      // 如果ref为空则初始化后再返回
7.      if (ref == null) {
8.          init();
9.      }
10.     return ref;
11. }
```

init方法比较大：

[java] view plain copy

```
1.    private void init() {
2.        // 先判断initialized标记，如果为true，表示已经初始化过，初始化过则不再初始化。 从这里看出如果第一次初始化失败了，则后续该consumer无法再使用
3. if (initialized) {
4.     return;
5. }
6. initialized = true;
7.      if (interfaceName == null || interfaceName.length() == 0) {
8.          throw new IllegalStateException("<dubbo:reference interface=\"\" /> interface not allow null!");
9.      }
10.     // 再次检查consumer配置，同时通过appendProperties修改代码/xml中的配置。
11.         // appendProperties从System.getProperty中获取配置，如果有相应值则替换配置文件中的值。
12.         // 比如对于consumer来说，如果配置了timeout=5000，可以通过：
13.         // 1、在启动参数中设置dubbo.consumer.timeout=3000来修改这个值；
14.         // 2、在启动参数中设置 dubbo.consumer.com.alibaba.dubbo.config.ConsumerConfig.timeout=3000来修改这个值。
15.         // 其中第二种方式优先级高于第一种，修改其他属性只需要替换"timeout"。
16.         // 如果系统设置中没有，还可以在启动参数中设置 dubbo.properties.file（如果没设置则默认为dubbo.properties），加载文件中的配置
17.         // 其中系统设置的优先级高于文件设置
18.         checkDefault();
19.         // 与上面修改consumer一样，通过启动参数/系统设置/文件配置的方式修改代码/xml中的配置。
20.         appendProperties(this);
21.         // 泛化调用设置，如果是泛化调用则接口类为GeneicService，否则为
```

```
配置的interfaceName
22.         if (getGeneric() == null && getConsumer() != null) {
23.             setGeneric(getConsumer().getGeneric());
24.         }
25.         if (ProtocolUtils.isGeneric(getGeneric())) {
26.             interfaceClass = GenericService.class;
27.         } else {
28.             try {
29.                 interfaceClass = Class.forName(interfaceName,
true, Thread.currentThread()
30.                         .getContextClassLoader());
31.             } catch (ClassNotFoundException e) {
32.                 throw new IllegalStateException(e.getMessage(),
e);
33.             }
34.             // 1、检查是否是接口，2、如果有methods配置，检查methods中
声明的方法在接口中是否存在
35.             checkInterfaceAndMethods(interfaceClass, methods);
36.         }
37.         // 用户可以通过系统属性的方式来指定interfaceName对应的url
38.         String resolve = System.getProperty(interfaceName);
39.         String resolveFile = null;
40.         if (resolve == null || resolve.length() == 0) {
41.             resolveFile =
System.getProperty("dubbo.resolve.file");
42.             if (resolveFile == null || resolveFile.length() == 0)
{
43.                 File userResolveFile = new File(new
File(System.getProperty("user.home")), "dubbo-
resolve.properties");
44.                 if (userResolveFile.exists()) {
45.                     resolveFile =
userResolveFile.getAbsolutePath();
46.                 }
47.             }
48.             if (resolveFile != null && resolveFile.length() > 0)
{
49.                 Properties properties = new Properties();
50.                 FileInputStream fis = null;
51.                 try {
52.                     fis = new FileInputStream(new
File(resolveFile));
53.                     properties.load(fis);
54.                 } catch (IOException e) {
55.                     throw new IllegalStateException("Unload " +
```

```
resolveFile + ", cause: " + e.getMessage(), e);
            } finally {
                try {
                    if(null != fis) fis.close();
                } catch (IOException e) {
                    logger.warn(e.getMessage(), e);
                }
            }
            resolve = properties.getProperty(interfaceName);
        }
    }
    if (resolve != null && resolve.length() > 0) {
        url = resolve;
        if (logger.isWarnEnabled()) {
            if (resolveFile != null && resolveFile.length() > 0) {
                logger.warn("Using default dubbo resolve file " + resolveFile + " replace " + interfaceName + "" + resolve + " to p2p invoke remote service.");
            } else {
                logger.warn("Using -D" + interfaceName + "=" + resolve + " to p2p invoke remote service.");
            }
        }
    }

    // 这里忽略掉使用各个默认值来初始化null值的代码
    ......
    // 检测application配置，同样也会调用appendProperties进行参数的修改
    checkApplication();
    // 检查local/stub/mock三个配置是否正确
    checkStubAndMock(interfaceClass);
    // 将所有配置加入到map中
    Map<String, String> map = new HashMap<String, String>();
    Map<Object, Object> attributes = new HashMap<Object, Object>();
    map.put(Constants.SIDE_KEY, Constants.CONSUMER_SIDE);
    map.put(Constants.DUBBO_VERSION_KEY, Version.getVersion());
    map.put(Constants.TIMESTAMP_KEY, String.valueOf(System.currentTimeMillis()));
    if (ConfigUtils.getPid() > 0) {
        map.put(Constants.PID_KEY,
```

```
String.valueOf(ConfigUtils.getPid()));
91.         }
92.         if (! isGeneric()) {
93.             String revision =
Version.getVersion(interfaceClass, version);
94.             if (revision != null && revision.length() > 0) {
95.                 map.put("revision", revision);
96.             }
97.
98.             String[] methods =
Wrapper.getWrapper(interfaceClass).getMethodNames();
99.             if(methods.length == 0) {
100.                 logger.warn("NO method found in service
interface " + interfaceClass.getName());
101.                 map.put("methods", Constants.ANY_VALUE);
102.             }
103.             else {
104.                 map.put("methods", StringUtils.join(new
HashSet<String>(Arrays.asList(methods)), ","));
105.             }
106.         }
107.         map.put(Constants.INTERFACE_KEY, interfaceName);
108.         // 依次将application/module/consumer/reference的配置放入
map
109.         appendParameters(map, application);
110.         appendParameters(map, module);
111.         appendParameters(map, consumer,
Constants.DEFAULT_KEY);
112.         appendParameters(map, this);
113.         String prifix = StringUtils.getServiceKey(map);
114.         if (methods != null && methods.size() > 0) {
115.             for (MethodConfig method : methods) {
116.                 appendParameters(map, method,
method.getName());
117.                 String retryKey = method.getName() +
".retry";
118.                 if (map.containsKey(retryKey)) {
119.                     String retryValue = map.remove(retryKey);
120.                     if ("false".equals(retryValue)) {
121.                         map.put(method.getName() +
".retries", "0");
122.                     }
123.                 }
124.                 appendAttributes(attributes, method, prifix +
"." + method.getName());
```

```
125.                 checkAndConvertImplicitConfig(method, map,
attributes);
126.             }
127.         }
128.         //attributes通过系统context进行存储.
129.         StaticContext.getSystemContext().putAll(attributes);
130.         // 使用map中的参数通过createProxy方法创建代理对象
131.         ref = createProxy(map);
132.     }
```

可以看到init方法会先做大量的配置初始化和检查工作，并将生成的配置放入map中，通过map创建代理，下面看看创建代理的方法：

[java] view plain copy

```
1. private T createProxy(Map<String, String> map) {
2.             // 使用map创建一个URL对象，注意该URL是dubbo重新实现的URL
3.             URL tmpUrl = new URL("temp", "localhost", 0, map);
4.     final boolean isJvmRefer; // 是否是本地服务
5.         if (isInjvm() == null) {
6.             if (url != null && url.length() > 0) { //指定URL的情
况下，不做本地引用
7.                 isJvmRefer = false;
8.             } else if
(InjvmProtocol.getInjvmProtocol().isInjvmRefer(tmpUrl)) {
9.                 //默认情况下如果本地有服务暴露，则引用本地服务.
10.                 isJvmRefer = true;
11.             } else {
12.                 isJvmRefer = false;
13.             }
14.         } else {
15.             isJvmRefer = isInjvm().booleanValue();
16.         }
17.     // 如果是本地服务，则url使用本地服务的协议形式
18.     if (isJvmRefer) {
19.         URL url = new URL(Constants.LOCAL_PROTOCOL,
NetUtils.LOCALHOST, 0,
interfaceClass.getName()).addParameters(map);
20.         invoker = refprotocol.refer(interfaceClass, url);
21.             if (logger.isInfoEnabled()) {
22.                 logger.info("Using injvm service " +
interfaceClass.getName());
23.             }
24.     } else {
```

```
25.             if (url != null && url.length() > 0) { // 用户指定URL，指定的URL可能是对点对直连地址，也可能是注册中心URL
26.                 String[] us = Constants.SEMICOLON_SPLIT_PATTERN.split(url);
27.                 if (us != null && us.length > 0) {
28.                     for (String u : us) {
29.                         URL url = URL.valueOf(u);
30.                         if (url.getPath() == null || url.getPath().length() == 0) {
31.                             url = url.setPath(interfaceName);
32.                         }
33.                         if (Constants.REGISTRY_PROTOCOL.equals(url.getProtocol())) {
34.                             urls.add(url.addParameterAndEncoded(Constants.REFER_KEY, StringUtils.toQueryString(map)));
35.                         } else {
36.                             urls.add(ClusterUtils.mergeUrl(url, map));
37.                         }
38.                     }
39.                 }
40.             } else { // 通过注册中心配置拼装URL，如果没有注册信息会报错
41.                 List<URL> us = loadRegistries(false);
42.                 if (us != null && us.size() > 0) {
43.                     for (URL u : us) {
44.                         // 加载monitor配置，如果加载到配置则返回非空的monitorUrl
45.                         URL monitorUrl = loadMonitor(u);
46.                         if (monitorUrl != null) {
47.                             map.put(Constants.MONITOR_KEY, URL.encode(monitorUrl.toFullString()));
48.                         }
49.                         urls.add(u.addParameterAndEncoded(Constants.REFER_KEY, StringUtils.toQueryString(map)));
50.                     }
51.                 }
52.                 if (urls == null || urls.size() == 0) {
53.                     throw new IllegalStateException("No such any registry to reference " + interfaceName  + " on the consumer " + NetUtils.getLocalHost() + " use dubbo version " + Version.getVersion() + ", please config <dubbo:registry address=\"...\" /> to your spring config.");
```

```
54.                 }
55.             }
56.             // 此处refprotocol为RegistryProtocol，它的refer方法会
生成一个Registry（如ZookeeperRegisty、MulticastRegistry），
57.             // 生成的Registry通过url中的注册中心地址与注册中心建立连
接，并订阅相关信息，最终封装成一个invoker
58.             if (urls.size() == 1) {
59.                 invoker = refprotocol.refer(interfaceClass,
urls.get(0));
60.             } else {
61.                 List<Invoker<?>> invokers = new
ArrayList<Invoker<?>>();
62.                 URL registryURL = null;
63.                 for (URL url : urls) {
64.
invokers.add(refprotocol.refer(interfaceClass, url));
65.                     if
(Constants.REGISTRY_PROTOCOL.equals(url.getProtocol())) {
66.                         registryURL = url; // 用了最后一个
registry url
67.                     }
68.                 }
69.                 if (registryURL != null) { // 有 注册中心协议的
URL
70.                     // 对有注册中心的Cluster 只用
AvailableCluster
71.                     URL u =
registryURL.addParameter(Constants.CLUSTER_KEY,
AvailableCluster.NAME);
72.                     invoker = cluster.join(new
StaticDirectory(u, invokers));
73.                 }  else { // 不是 注册中心的URL
74.                     invoker = cluster.join(new
StaticDirectory(invokers));
75.                 }
76.             }
77.         }
78.
79.         // 检查check属性，如果check为空或未true，检测与服务提供方是否
连接成功，连接失败则报错
80.         Boolean c = check;
81.         if (c == null && consumer != null) {
82.             c = consumer.isCheck();
83.         }
84.         if (c == null) {
```

```
85.             c = true; // default true
86.         }
87.         if (c && ! invoker.isAvailable()) {
88.             throw new IllegalStateException("Failed to check
the status of the service " + interfaceName + ". No provider
available for the service " + (group == null ? "" : group + "/")
+ interfaceName + (version == null ? "" : ":" + version) + " from
the url " + invoker.getUrl() + " to the consumer " +
NetUtils.getLocalHost() + " use dubbo version " +
Version.getVersion());
89.         }
90.         if (logger.isInfoEnabled()) {
91.             logger.info("Refer dubbo service " +
interfaceClass.getName() + " from url " + invoker.getUrl());
92.         }
93.         // 调用代理生成工厂创建服务代理
94.         return (T) proxyFactory.getProxy(invoker);
95.     }
```

可以看到这个创建代理的方法是通过调用proxyFactory.getProxy来创建代理。而创建代理之前需要先加载注册中心的配置，并生成monitor对应的key。接下来就是根据上面加载到的信息来创建invoker，invoker是dubbo中非常重要的一个概念，它代表一个可执行体，通过调用它的invoke方法来进行本地/远程调用，并获取结果。它的生成有两个分支，第一个分支是urls.size() == 1，从前面的代码可以知道当注册中心只有一个（单个或集群）时进入此分支，此处的refprotocal是由ExtensionLoader生成的代理类，其关键代码如下：

[java] view plain copy

```
1. public class Protocol$Adpative implements Protocol {
2.     public Exporter export(Invoker invoker) throws
com.alibaba.dubbo.rpc.RpcException {
3.         if (invoker == null || invoker.getUrl() == null) {
4.             throw new IllegalArgumentException("xxx");
5.         }
6.
7.         URL url = invoker.getUrl();
8.         String extName = url.getProtocol() == null ? "dubbo" :
url.getProtocol();
9.         if(extName == null) {
10.             throw new IllegalStateException("xxx");
```

```
11.         }
12.         
13.         Protocol extension =
(Protocol)ExtensionLoader.getExtensionLoader(Protocol.class).getE
xtension(extName);
14.         return extension.export(invoker);
15.     }
16.     
17.     public Invoker refer(Class cls, URL u) throws
RpcException {
18.         if (u == null) {
19.             throw new IllegalArgumentException("url ==
null");
20.         }
21.         URL url = u;
22.         String extName = url.getProtocol() == null ? "dubbo"
: url.getProtocol();
23.         if(extName == null) {
24.             throw new IllegalStateException("xxx");
25.         }
26.         Protocol extension =
(Protocol)ExtensionLoader.getExtensionLoader(Protocol.class).getE
xtension(extName);
27.         return extension.refer(cls, u);
28.     }
29. }
```

这个代理类的实现比较简单，就是根据URL中的protocol加载实际的实现，并找到对应的wrapper class(即构造方法中含对应接口的类，如类A有构造方法A(B b)且在META-INF\dubbo目录下有配置，则A是B的wrapper class），通过wrapper class包装真正的Protocol实现返回，如果没有wrapper则直接返回其实现类。注意，ExtensionLoader.getExtensionLoader(XX.class).getAdaptiveExtension()生成的代理类思路都是一样的，后续出现这样的代码就不再给出了。前面的代码“refprotocol.refer(interfaceClass, urls.get(0));”中urls.get(0)的protocol为registry, 因此最终调用的是com.alibaba.dubbo.registry.integration.RegistryProtocol的refer方法：

[java] view plain copy

```
1.     public <T> Invoker<T> refer(Class<T> type, URL url) throws
RpcException {
2.         // 将url的protocal改为registry对应的协议，如multicast或者
zookeeper
3.         url =
url.setProtocol(url.getParameter(Constants.REGISTRY_KEY,
Constants.DEFAULT_REGISTRY)).removeParameter(Constants.REGISTRY_K
EY);
4.         // 此处的registryFactory也是个代理类，根据protocal可能是
ZookeeperRegistryFactory、MulticastRegistryFactory等，
5.         // getRegistry方法根据url返回ZookeeperRegistry、
MulticastRegistry对象等，各Registry对象在初始化的时候会根据url和注册中心
建立连接
6.         Registry registry = registryFactory.getRegistry(url);
7.         if (RegistryService.class.equals(type)) {
8.             return proxyFactory.getInvoker((T) registry, type,
url);
9.         }
10.
11.          // group="a,b" or group="*"
12.         Map<String, String> qs =
StringUtils.parseQueryString(url.getParameterAndDecoded(Constants
.REFER_KEY));
13.         String group = qs.get(Constants.GROUP_KEY);
14.         if (group != null && group.length() > 0 ) {
15.             if ( ( Constants.COMMA_SPLIT_PATTERN.split( group
) ).length > 1
16.                     || "*".equals( group ) ) {
17.                 return doRefer( getMergeableCluster(),
registry, type, url );
18.             }
19.         }
20.
21.         return doRefer(cluster, registry, type, url);
22.     }
23.
24.     private <T> Invoker<T> doRefer(Cluster cluster, Registry
registry, Class<T> type, URL url) {
25.         // 创建Directory服务，通过它可以进行消息的注册、订阅，同时负责
注册中心变更时的消息接收及处理
26.         RegistryDirectory<T> directory = new
RegistryDirectory<T>(type, url);
27.         directory.setRegistry(registry);
28.         directory.setProtocol(protocol);
29.         URL subscribeUrl = new
```

```
URL(Constants.CONSUMER_PROTOCOL, NetUtils.getLocalHost(), 0,
type.getName(), directory.getUrl().getParameters());
30.         if (!
Constants.ANY_VALUE.equals(url.getServiceInterface())
31.                 && url.getParameter(Constants.REGISTER_KEY,
true)) {
32.
registry.register(subscribeUrl.addParameters(Constants.CATEGORY_K
EY, Constants.CONSUMERS_CATEGORY,
33.                         Constants.CHECK_KEY,
String.valueOf(false)));
34.         }
35.
directory.subscribe(subscribeUrl.addParameter(Constants.CATEGORY_
KEY,
36.                 Constants.PROVIDERS_CATEGORY
37.                 + "," + Constants.CONFIGURATORS_CATEGORY
38.                 + "," + Constants.ROUTERS_CATEGORY));
39.         // directory中url的cluster为null，则取默认值failover（默
认值见接口Cluster的SPI注解），
40.         // 对应类为
com.alibaba.dubbo.rpc.cluster.support.FailoverCluster，该类的join方
法返回new FailoverClusterInvoker<T>(directory)
41.         return cluster.join(directory);
42.     }
```

可以看到第一个分支在非group模式时返回的invoker为FailoverClusterInvoker，顾名思义，failover就是故障转移，即当对某个服务器调用失败时，选用其他的服务器重试。而在group模式时返回的invoker为MergeableClusterInvoker，作用就是将多个group的返回数据进行组合。不管是FailoverClusterInvoker还是MergeableClusterInvoker，其实现都比较复杂，所以细节先不展开，后面再单独讲。

再来看另一个分支urls.size() > 1, 此时针对每个url都会生成一个ClusterInvoker，然后将其放入StaticDirectory管理。对于StaticDirectory来说，其下的任意一个注册中心可用则其可用。

再回头看ReferenceConfig生成代理的最后一步：return (T) proxyFactory.getProxy(invoker)；这里默认的实现为com.alibaba.dubbo.rpc.proxy.javassist.JavassistProxyFactory, 它的getProxy方法：

[java] view plain copy

```
1.      // 继承自抽象父类AbstractProxyFactory
2.      public <T> T getProxy(Invoker<T> invoker) throws
RpcException {
3.          Class<?>[] interfaces = null;
4.          String config =
invoker.getUrl().getParameter("interfaces");
5.          if (config != null && config.length() > 0) {
6.              // 通过逗号拆分interfaces参数来获取所有的interface
7.              String[] types =
Constants.COMMA_SPLIT_PATTERN.split(config);
8.              if (types != null && types.length > 0) {
9.                  interfaces = new Class<?>[types.length + 2];
10.                 interfaces[0] = invoker.getInterface();
11.                 interfaces[1] = EchoService.class;
12.                 for (int i = 0; i < types.length; i ++) {
13.                     interfaces[i + 1] =
ReflectUtils.forName(types[i]);
14.                 }
15.             }
16.         }
17.         // 未指定interfaces则直接取invoker中的interface
18.         if (interfaces == null) {
19.             interfaces = new Class<?>[]
{invoker.getInterface(), EchoService.class};
20.         }
21.         return getProxy(invoker, interfaces);
22.     }
23.
24.     // JavassistProxyFactory
25.     public <T> T getProxy(Invoker<T> invoker, Class<?>[]
interfaces) {
26.         // com.alibaba.dubbo.common.bytecode.Proxy
27.         return (T) Proxy.getProxy(interfaces).newInstance(new
InvokerInvocationHandler(invoker));
28.     }
```

这里需要注意的是，生成的consumer代理除了实现指定接口外，还需要实现EchoService(支持回声测深，用于检查服务是否可用）。接下来就是真正的代理类生成了，对于DemoService来说，其生成的类代码如下：

[java] view plain copy

```java
package com.alibaba.dubbo.common.bytecode;

import com.alibaba.dubbo.demo.DemoService;
import com.alibaba.dubbo.rpc.service.EchoService;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;

// 见com.alibaba.dubbo.common.bytecode.Proxy.getProxy
public class proxy0 implements ClassGenerator.DC, EchoService,
DemoService {
  // methods包含proxy0实现的所有接口方法（去重）
  public static Method[] methods;
  private InvocationHandler handler;

  public String sayHello(String paramString) {
    Object[] arrayOfObject = new Object[1];
    arrayOfObject[0] = paramString;
    Object localObject = this.handler.invoke(this,
methods[0], arrayOfObject);
    return (String)localObject;
  }

  public Object $echo(Object paramObject) {
    Object[] arrayOfObject = new Object[1];
    arrayOfObject[0] = paramObject;
    Object localObject = this.handler.invoke(this,
methods[1], arrayOfObject);
    return (Object)localObject;
  }

  public proxy0() {
  }

  public proxy0(InvocationHandler paramInvocationHandler) {
    this.handler = paramInvocationHandler;
  }
}
```

最终生成的类传入InvokerInvocationHandler，此handler包装了invoker，handler拦截了toString/hashCode/equals方法，其他的则是直接调用invoker.invoke(new RpcInvocation(method, args)).recreate()；例如在代码中调用demoService.sayHello("hello world")， 实际的调用为invoker.invoke(new RpcInvocation(sayHelloMethod, new Object[] {"hello world"})).recreate();

到此客户端的代理生成完成，总结如下：

1、spring加载时拦截namespace=dubbo的标签进行解析，生成dubbo中的config；

2、consumer对应的直接配置为ReferenceConfig， reference的config加载完成后，分别使用ConsumerConfig、ApplicationConfig、ModuleConfig、RegistryConfig、MonitorConfig等的默认值来初始化ReferenceConfig;

3、在创建bean的对象时，如果已经创建过则不重复创建，否则进入创建流程，并将是否创建的标识设为true；

4、使用系统参数、配置文件覆盖api/xml中设置的配置，将所有配置项存入map;

5、获取注册中心配置，根据配置连接注册中心，并注册和订阅url的变动提醒；

6、生成cluster invoker；

7、根据接口生成代理类，并创建对象，创建时传入InvokerInvocationHandler，该handler封装了上面生成的invoker，最终的接口调用都是通过invoker.invoke（。。。）。