前面一篇文章我们分析了consumer代理的生成。在代理生成的过程中，会创建注册服务(com.alibaba.dubbo.registry.Registry)。通过注册服务提供url注册、订阅、查找的功能。

[java] view plain copy

```java
public interface RegistryService {

    /**
     * 注册数据，比如：提供者地址，消费者地址，路由规则，覆盖规则，等数据。
     *
     * 注册需处理契约：<br>
     * 1. 当URL设置了check=false时，注册失败后不报错，在后台定时重试，否则抛出异常。<br>
     * 2. 当URL设置了dynamic=false参数，则需持久存储，否则，当注册者出现断电等情况异常退出时，需自动删除。<br>
     * 3. 当URL设置了category=routers时，表示分类存储，缺省类别为providers，可按分类部分通知数据。<br>
     * 4. 当注册中心重启，网络抖动，不能丢失数据，包括断线自动删除数据。<br>
     * 5. 允许URI相同但参数不同的URL并存，不能覆盖。<br>
     *
     * @param url 注册信息，不允许为空，如：dubbo://10.20.153.10/com.alibaba.foo.BarService?version=1.0.0&application=kylin
     */
    void register(URL url);

    /**
     * 取消注册.
     *
     * 取消注册需处理契约：<br>
     * 1. 如果是dynamic=false的持久存储数据，找不到注册数据，则抛IllegalStateException，否则忽略。<br>
     * 2. 按全URL匹配取消注册。<br>
     *
     * @param url 注册信息，不允许为空，如：dubbo://10.20.153.10/com.alibaba.foo.BarService?version=1.0.0&application=kylin
     */
    void unregister(URL url);
```

```
27.
28.     /**
29.      * 订阅符合条件的已注册数据，当有注册数据变更时自动推送.
30.      *
31.      * 订阅需处理契约：<br>
32.      * 1. 当URL设置了check=false时，订阅失败后不报错，在后台定时重试。<br>
33.      * 2. 当URL设置了category=routers，只通知指定分类的数据，多个分类用逗号分隔，并允许星号通配，表示订阅所有分类数据。<br>
34.      * 3. 允许以interface,group,version,classifier作为条件查询，如：interface=com.alibaba.foo.BarService&version=1.0.0<br>
35.      * 4. 并且查询条件允许星号通配，订阅所有接口的所有分组的所有版本，或：interface=*&group=*&version=*&classifier=*<br>
36.      * 5. 当注册中心重启，网络抖动，需自动恢复订阅请求。<br>
37.      * 6. 允许URI相同但参数不同的URL并存，不能覆盖。<br>
38.      * 7. 必须阻塞订阅过程，等第一次通知完后再返回。<br>
39.      *
40.      * @param url 订阅条件，不允许为空，如：consumer://10.20.153.10/com.alibaba.foo.BarService?version=1.0.0&application=kylin
41.      * @param listener 变更事件监听器，不允许为空
42.      */
43.     void subscribe(URL url, NotifyListener listener);
44.
45.     /**
46.      * 取消订阅.
47.      *
48.      * 取消订阅需处理契约：<br>
49.      * 1. 如果没有订阅，直接忽略。<br>
50.      * 2. 按全URL匹配取消订阅。<br>
51.      *
52.      * @param url 订阅条件，不允许为空，如：consumer://10.20.153.10/com.alibaba.foo.BarService?version=1.0.0&application=kylin
53.      * @param listener 变更事件监听器，不允许为空
54.      */
55.     void unsubscribe(URL url, NotifyListener listener);
56.
57.     /**
58.      * 查询符合条件的已注册数据，与订阅的推模式相对应，这里为拉模式，只返回一次结果。
59.      *
60.      * @see com.alibaba.dubbo.registry.NotifyListener#notify(List)
61.      * @param url 查询条件，不允许为空，如：
```

```
consumer://10.20.153.10/com.alibaba.foo.BarService?
version=1.0.0&application=kylin
62.      * @return 已注册信息列表，可能为空，含义同{@link
com.alibaba.dubbo.registry.NotifyListener#notify(List<URL>)}的参
数。
63.      */
64.     List<URL> lookup(URL url);
65.
66. }
```

本机测试的时候我们可以选择Multicase注册中心，但这种方式受网络结构限制，只适合小规模应用或开发阶段使用，实际线上环境官方推荐Zookeeper中心。因此我们主要分析Zookeeper注册中心的实现。

上一篇文章我们讲到Registry的创建是通过“Registry registry = registryFactory.getRegistry(url);”这里的registryFactory根据不同的protocol而不同，我们来看看ZookeeperRegistryFactory的实现，ZookeeperRegistryFactory继承自AbstractRegistryFactory，其getRegistry方法代码如下：

[java] view plain copy

```
1.     public Registry getRegistry(URL url) {
2.         // 将path和interface都设置成
com.alibaba.dubbo.registry.RegistryService
3.         url = url.setPath(RegistryService.class.getName())
4.                 .addParameter(Constants.INTERFACE_KEY,
RegistryService.class.getName())
5.                 .removeParameters(Constants.EXPORT_KEY,
Constants.REFER_KEY);
6.         // 根据url生产serviceString,格式为
protocol://[username:password@]ip:port/[serviceKey或path]
7.         // 例如:
zookeeper://127.0.0.1:2181/com.alibaba.dubbo.registry.RegistrySer
vice
8.         String key = url.toServiceString();
9.         // 锁定注册中心获取过程，保证注册中心单一实例
10.         LOCK.lock();
11.         try {
12.             Registry registry = REGISTRIES.get(key);
13.             if (registry != null) {
14.                 return registry;
15.             }
```

```
16.             // 根据url创建registry
17.             registry = createRegistry(url);
18.             if (registry == null) {
19.                 throw new IllegalStateException("Can not
create registry " + url);
20.             }
21.             REGISTRIES.put(key, registry);
22.             return registry;
23.         } finally {
24.             // 释放锁
25.             LOCK.unlock();
26.         }
27.     }
28.
29.     // 这段在ZookeeperRegistryFactory中
30.     // 直接使用url创建ZookeeperRegistry, zookeeperTransporter对
zk的操作进行了封装，目前提供了zkclient和curator两种实现，默认为zkclient
31.     public Registry createRegistry(URL url) {
32.         return new ZookeeperRegistry(url,
zookeeperTransporter);
33.     }
```

目前zookeeperTransporter的修改方式有很多种：

spring配置：<dubbo:registry ... client="curator" />

系统参数：dubbo.registry.client=curator

注册链接：zookeeper://10.20.153.10:2181?client=curator

通过ZookeeperRegistryFactory创建了一个ZookeeperRegistry，来看看这个类的构造方法：

[java] view plain copy

```
1.     public ZookeeperRegistry(URL url, ZookeeperTransporter
zookeeperTransporter) {
2.         super(url);
3.         if (url.isAnyHost()) {
4.             throw new IllegalStateException("registry address
== null");
5.         }
6.         // 获取group，默认为dubbo，默认的root为/root
7.         String group = url.getParameter(Constants.GROUP_KEY,
DEFAULT_ROOT);
8.         if (! group.startsWith(Constants.PATH_SEPARATOR)) {
9.             group = Constants.PATH_SEPARATOR + group;
```

```
10.         }
11.         this.root = group;
12.         // 通过transporter创建连接
13.         zkClient = zookeeperTransporter.connect(url);
14.         // 添加状态变更的事件监听器。注意这里只是添加了监听器，但并没有
像zk注册。
15.         zkClient.addStateListener(new StateListener() {
16.             public void stateChanged(int state) {
17.                 if (state == RECONNECTED) {
18.                 try {// 重连后执行recover方法
19.             recover();
20.             } catch (Exception e) {
21.             logger.error(e.getMessage(), e);
22.             }
23.                 }
24.             }
25.         });
26.     }
27.     // ZookeeperRegistry的父类构造方法
28.     public FailbackRegistry(URL url) {
29.         super(url);
30.         int retryPeriod =
url.getParameter(Constants.REGISTRY_RETRY_PERIOD_KEY,
Constants.DEFAULT_REGISTRY_RETRY_PERIOD);
31.         // 添加一个任务，默认5秒后开始，每5秒钟进行一次连接检测
32.         this.retryFuture =
retryExecutor.scheduleWithFixedDelay(new Runnable() {
33.             public void run() {
34.                 // 检测并连接注册中心
35.                 try {
36.                     retry();
37.                 } catch (Throwable t) { // 防御性容错
38.                     logger.error("Unexpected error occur at
failed retry, cause: " + t.getMessage(), t);
39.                 }
40.             }
41.         }, retryPeriod, retryPeriod, TimeUnit.MILLISECONDS);
42.     }
43.
44.     // FailbackRegistry 的父类构造方法
45.     public AbstractRegistry(URL url) {
46.         // 设置registryUrl
47.         setUrl(url);
48.         syncSaveFile =
url.getParameter(Constants.REGISTRY_FILESAVE_SYNC_KEY, false);
```

```
49.         // 根据host生产注册信息的缓存文件地址
50.         String filename =
url.getParameter(Constants.FILE_KEY,
System.getProperty("user.home") + "/.dubbo/dubbo-registry-" +
url.getHost() + ".cache");
51.         File file = null;
52.         if (ConfigUtils.isNotEmpty(filename)) {
53.             file = new File(filename);
54.             if(! file.exists() && file.getParentFile() !=
null && ! file.getParentFile().exists()){
55.                 if(! file.getParentFile().mkdirs()){
56.                     throw new
IllegalArgumentException("Invalid registry store file " + file +
", cause: Failed to create directory " + file.getParentFile() +
"!");
57.                 }
58.             }
59.         }
60.         this.file = file;
61.         // 从file中加载配置到内存中
62.         loadProperties();
63.         // url变更通知，getBackupUrls通过url生成所有链接，如
zookeeper://10.20.153.10:2181?
backup=10.20.153.11:2181,10.20.153.12:2181，最终生成的链接：
64.         //
zookeeper://10.20.153.10:2181/com.alibaba.dubbo.registry.Registry
Service?xxx
65.         //
zookeeper://10.20.153.11:2181/com.alibaba.dubbo.registry.Registry
Service?
xxxzookeeper://10.20.153.12:2181/com.alibaba.dubbo.registry.Regis
tryService?xxx
66.         notify(url.getBackupUrls());
67.     }
```

可以看到，ZookeeperRegistry初始化主要有以下操作：

1、 获取缓存文件路径，并从该文件加载数据到内存，将注册地址拆分成多个地址（backup的情况有多个地址）；

2、 创建一个定时任务，定时对失败的操作进行重试；

3、通过transporter创建连接，并添加一个状态改变的监听器。

创建连接的过程（curator）：

connect（url）执行代码如下

[java] view plain copy

```
1. public class CuratorZookeeperTransporter implements
ZookeeperTransporter {
2.     public ZookeeperClient connect(URL url) {
3.         return new CuratorZookeeperClient(url);
4.     }
5. }
```

[java] view plain copy

```
1. public CuratorZookeeperClient(URL url) {
2.     super(url);
3.     try {
4.                         // 设置connectString, 这里的backupAddress
包括原地址和备用地址，最终得到字符串形式：ip0:port0,ip1:port1...
5.                         Builder builder =
CuratorFrameworkFactory.builder()
6.                 .connectString(url.getBackupAddress())
7.                 .retryPolicy(new
RetryNTimes(Integer.MAX_VALUE, 1000))
8.                 .connectionTimeoutMs(5000);
9.         String authority = url.getAuthority();
10.         if (authority != null && authority.length() > 0) {
11.             builder = builder.authorization("digest",
authority.getBytes());
12.         }
13.         client = builder.build();
14.                         // 注册连接状态改变事件的监听器，当状态变更时
调用stateChanged方法
15.         client.getConnectionStateListenable().addListener(new
ConnectionStateListener() {
16.             public void stateChanged(CuratorFramework client,
ConnectionState state) {
17.                 if (state == ConnectionState.LOST) {
18.
CuratorZookeeperClient.this.stateChanged(StateListener.DISCONNECT
ED);
19.                 } else if (state ==
ConnectionState.CONNECTED) {
20.
CuratorZookeeperClient.this.stateChanged(StateListener.CONNECTED)
;
21.                 } else if (state ==
```

```
ConnectionState.RECONNECTED) {
22.                     CuratorZookeeperClient.this.stateChanged(StateListener.RECONNECTED);
23.                 }
24.             }
25.         });
26.         client.start();
27.     } catch (IOException e) {
28.         throw new IllegalStateException(e.getMessage(), e);
29.     }
30. }
```

上面又出现一个addListener, 这个和前面的zkClient.addStateListener有啥区别呢？ addListener是注册了状态变更的监听器，也就是状态变更时会回匿名ConnectionStateListener中的stateChange方法，stateChange再调用client中的stateChange方法，而stateChange方法调用的正是addStateListener添加的listener。

到这里Registry的初始化工作完成，接下来在创建RegistryDirectory后会调用registry.register方法来进行注册：

[java] view plain copy

```
1. // 父类FailbackRegistry中
2. public void register(URL url) {
3.     super.register(url);
4.     failedRegistered.remove(url);
5.     failedUnregistered.remove(url);
6.     try {
7.         // 向服务器端发送注册请求
8.         doRegister(url);
9.     } catch (Exception e) {
10.         Throwable t = e;
11.
12.         // 如果开启了启动时检测，则直接抛出异常
13.         boolean check = getUrl().getParameter(Constants.CHECK_KEY, true)
14.                 && url.getParameter(Constants.CHECK_KEY, true)
15.                 && !Constants.CONSUMER_PROTOCOL.equals(url.getProtocol());
16.         boolean skipFailback = t instanceof
```

```
SkipFailbackWrapperException;
17.         if (check || skipFailback) {
18.             if(skipFailback) {
19.                 t = t.getCause();
20.             }
21.             throw new IllegalStateException("Failed to
register " + url + " to registry " + getUrl().getAddress() + ",
cause: " + t.getMessage(), t);
22.         } else {
23.             logger.error("Failed to register " + url + ",
waiting for retry, cause: " + t.getMessage(), t);
24.         }
25.
26.         // 将失败的注册请求记录到失败列表，定时重试
27.         failedRegistered.add(url);
28.     }
29. }
```

doRegister的实现如下：

[java] view plain copy

```
1.     protected void doRegister(URL url) {
2.         try {
3.             //
/dubbo/com.alibaba.dubbo.demo.DemoService/consumers/xxxxxx      ,
true
4.             zkClient.create(toUrlPath(url),
url.getParameter(Constants.DYNAMIC_KEY, true));
5.         } catch (Throwable e) {
6.             throw new RpcException("Failed to register " + url
+ " to zookeeper " + getUrl() + ", cause: " + e.getMessage(), e);
7.         }
8.     }
9.
10.     public void create(String path, boolean ephemeral) {
11.         // 从最顶层开始创建持久化节点，最后一层是非持久化节点
(ephemeral=true)
12.         int i = path.lastIndexOf('/');
13.         if (i > 0) {
14.             create(path.substring(0, i), false);
15.         }
16.         if (ephemeral) {
17.             createEphemeral(path);
18.         } else {
```

```
19.             createPersistent(path);
20.         }
21.     }
```

toUrlPath(url)得到的路径格式为：/group(默认为dubbo)/(interfaceName)/consumers/xxx，这也是在zookeeper中的层次结构。注册完成后在monitor中可以看到此consumer。

注册完自身后，还需要订阅provider的信息，调用方式为directory.subscribe，RegistryDirectory实现代码：

[java] view plain copy

```
1. public void subscribe(URL url) {
2.     setConsumerUrl(url);
3.     // RegistryDirectory实现了NotifyListener
4.     registry.subscribe(url, this);
5. }
```

registry.subscribe方法在抽象类FailbackRegistry中:

[java] view plain copy

```
1. public void subscribe(URL url, NotifyListener listener) {
2.     // 添加listener到url对应的集合中
3.     super.subscribe(url, listener);
4.     // 从失败的订阅集合中移除该listener
5.     removeFailedSubscribed(url, listener);
6.     try {
7.         // 向服务器端发送订阅请求
8.         doSubscribe(url, listener);
9.     } catch (Exception e) {
10.         Throwable t = e;
11.         // urls为文件缓存中的地址
12.         List<URL> urls = getCacheUrls(url);
13.         if (urls != null && urls.size() > 0) {
14.             // 订阅失败则使用缓存中的url
15.             notify(url, listener, urls);
16.             logger.error("Failed to subscribe " + url + ",
Using cached list: " + urls + " from cache file: " +
getUrl().getParameter(Constants.FILE_KEY,
System.getProperty("user.home") + "/dubbo-registry-" +
url.getHost() + ".cache") + ", cause: " + t.getMessage(), t);
17.         } else {
```

```
18.             // 如果开启了启动时检测，则直接抛出异常
19.             boolean check =
getUrl().getParameter(Constants.CHECK_KEY, true)
20.                     && url.getParameter(Constants.CHECK_KEY,
true);
21.             boolean skipFailback = t instanceof
SkipFailbackWrapperException;
22.             if (check || skipFailback) {
23.                 if(skipFailback) {
24.                     t = t.getCause();
25.                 }
26.                 throw new IllegalStateException("Failed to
subscribe " + url + ", cause: " + t.getMessage(), t);
27.             } else {
28.                 logger.error("Failed to subscribe " + url +
", waiting for retry, cause: " + t.getMessage(), t);
29.             }
30.         }
31.
32.         // 将失败的订阅请求记录到失败列表，定时重试
33.         addFailedSubscribed(url, listener);
34.     }
35. }
```

doSubscribe方法在ZookeeperRegistry中：

[java] view plain copy

```
1. protected void doSubscribe(final URL url, final NotifyListener
listener) {
2.     try {
3.         if
(Constants.ANY_VALUE.equals(url.getServiceInterface())) {
4.             // 这段先不讲
5.             。。。
6.         } else {
7.             List<URL> urls = new ArrayList<URL>();
8.             //
/dubbo/com.alibaba.dubbo.demo.DemoService/providers,
9.             //
/dubbo/com.alibaba.dubbo.demo.DemoService/configurators,
10.             //
/dubbo/com.alibaba.dubbo.demo.DemoService/routers
11.             for (String path : toCategoriesPath(url)) {
12.                 // 添加子节点变更事件处理
```

```
13.                 ConcurrentMap<NotifyListener, ChildListener>
listeners = zkListeners.get(url);
14.                 if (listeners == null) {
15.                     zkListeners.putIfAbsent(url, new
ConcurrentHashMap<NotifyListener, ChildListener>());
16.                     listeners = zkListeners.get(url);
17.                 }
18.                 ChildListener zkListener =
listeners.get(listener);
19.                 if (zkListener == null) {
20.                     listeners.putIfAbsent(listener, new
ChildListener() {
21.                         public void childChanged(String
parentPath, List<String> currentChilds) {
22.                             
ZookeeperRegistry.this.notify(url, listener, toUrlsWithEmpty(url,
parentPath, currentChilds));
23.                         }
24.                     });
25.                     zkListener = listeners.get(listener);
26.                 }
27.                 // 创建持久化的节点
28.                 zkClient.create(path, false);
29.                 // 创建监听，如果节点下有数据则会返回节点下数据；
30.                 // 如对于providers节点，会返回对应接口下已经注册的
provider url，相当于此处可以拿到服务端的连接信息
31.                 List<String> children =
zkClient.addChildListener(path, zkListener);
32.                 if (children != null) {
33.                     urls.addAll(toUrlsWithEmpty(url, path,
children));
34.                 }
35.             }
36.             // 通知变更
37.             notify(url, listener, urls);
38.         }
39.     } catch (Throwable e) {
40.         throw new RpcException("Failed to subscribe " + url +
" to zookeeper " + getUrl() + ", cause: " + e.getMessage(), e);
41.     }
42. }
```

notify方法主要有两个功能，一是将变更的url存入缓存文件中，二是调用listener.notify方法。步骤二的listener是RegistryDirectory，因此代码又执行到RegistryDirectory的notify方法：

[java] view plain copy

```java
    public synchronized void notify(List<URL> urls) {
        List<URL> invokerUrls = new ArrayList<URL>();
        List<URL> routerUrls = new ArrayList<URL>();
        List<URL> configuratorUrls = new ArrayList<URL>();
        for (URL url : urls) {
            String protocol = url.getProtocol();
            String category = url.getParameter(Constants.CATEGORY_KEY, Constants.DEFAULT_CATEGORY);
            if (Constants.ROUTERS_CATEGORY.equals(category)
                    || Constants.ROUTE_PROTOCOL.equals(protocol)) {
                routerUrls.add(url);
            } else if (Constants.CONFIGURATORS_CATEGORY.equals(category)
                    || Constants.OVERRIDE_PROTOCOL.equals(protocol)) {
                configuratorUrls.add(url);
            } else if (Constants.PROVIDERS_CATEGORY.equals(category)) {
                invokerUrls.add(url);
            } else {
                logger.warn("Unsupported category " + category + " in notified url: " + url + " from registry " + getUrl().getAddress() + " to consumer " + NetUtils.getLocalHost());
            }
        }
        // configurators
        if (configuratorUrls != null && configuratorUrls.size() >0 ){
            this.configurators = toConfigurators(configuratorUrls);
        }
        // routers
        if (routerUrls != null && routerUrls.size() >0 ){
            List<Router> routers = toRouters(routerUrls);
            if(routers != null){ // null - do nothing
                setRouters(routers);
            }
        }
```

```
31.         List<Configurator> localConfigurators =
this.configurators; // local reference
32.         // 合并override参数
33.         this.overrideDirectoryUrl = directoryUrl;
34.         if (localConfigurators != null &&
localConfigurators.size() > 0) {
35.             for (Configurator configurator :
localConfigurators) {
36.                 this.overrideDirectoryUrl =
configurator.configure(overrideDirectoryUrl);
37.             }
38.         }
39.         // providers
40.         refreshInvoker(invokerUrls);
41.     }
42.
43.     /**
44.      * 根据invokerURL列表转换为invoker列表。转换规则如下:
45.      * 1.如果url已经被转换为invoker，则不在重新引用，直接从缓存中获
取，注意如果url中任何一个参数变更也会重新引用
46.      * 2.如果传入的invoker列表不为空，则表示最新的invoker列表
47.      * 3.如果传入的invokerUrl列表是空，则表示只是下发的override规则
或route规则，需要重新交叉对比，决定是否需要重新引用。
48.      * @param invokerUrls 传入的参数不能为null
49.      */
50.     private void refreshInvoker(List<URL> invokerUrls){
51.         if (invokerUrls != null && invokerUrls.size() == 1 &&
invokerUrls.get(0) != null
52.                 &&
Constants.EMPTY_PROTOCOL.equals(invokerUrls.get(0).getProtocol())
) {
53.             this.forbidden = true; // 禁止访问
54.             this.methodInvokerMap = null; // 置空列表
55.             destroyAllInvokers(); // 关闭所有Invoker
56.         } else {
57.             this.forbidden = false; // 允许访问
58.             Map<String, Invoker<T>> oldUrlInvokerMap =
this.urlInvokerMap; // local reference
59.             if (invokerUrls.size() == 0 &&
this.cachedInvokerUrls != null){
60.                 invokerUrls.addAll(this.cachedInvokerUrls);
61.             } else {
62.                 this.cachedInvokerUrls = new HashSet<URL>();
63.                 this.cachedInvokerUrls.addAll(invokerUrls);//
缓存invokerUrls列表，便于交叉对比
```

```
64.             }
65.             if (invokerUrls.size() ==0 ){
66.                 return;
67.             }
68.             Map<String, Invoker<T>> newUrlInvokerMap =
toInvokers(invokerUrls) ;// 将URL列表转成Invoker列表
69.             Map<String, List<Invoker<T>>> newMethodInvokerMap
= toMethodInvokers(newUrlInvokerMap); // 换方法名映射Invoker列表
70.             // state change
71.             //如果计算错误，则不进行处理.
72.             if (newUrlInvokerMap == null ||
newUrlInvokerMap.size() == 0 ){
73.                 logger.error(new IllegalStateException("urls
to invokers error .invokerUrls.size :"+invokerUrls.size() + ",
invoker.size :0. urls :"+invokerUrls.toString()));
74.                 return ;
75.             }
76.             this.methodInvokerMap = multiGroup ?
toMergeMethodInvokerMap(newMethodInvokerMap) :
newMethodInvokerMap;
77.             this.urlInvokerMap = newUrlInvokerMap;
78.             try{
79.
destroyUnusedInvokers(oldUrlInvokerMap,newUrlInvokerMap); // 关闭
未使用的Invoker
80.             }catch (Exception e) {
81.                 logger.warn("destroyUnusedInvokers error. ",
e);
82.             }
83.         }
84.     }
```

此方法中包含了一个toInvokers方法，该方法通过invokerUrls创建对应的Invoker，并放入newUrlInvokerMap，而暴露的方法名对应invoker则放入newMethodInvokerMap中。此时客户端需要的信息都已经加载。因此toInvokers方法是比较关键的：

[java] view plain copy

```
1. /**
2.      * 将urls转成invokers,如果url已经被refer过，不再重新引用。
3.      *
4.      * @param urls
5.      * @param overrides
```

```
6.      * @param query
7.      * @return invokers
8.      */
9.     private Map<String, Invoker<T>> toInvokers(List<URL> urls)
{
10.         Map<String, Invoker<T>> newUrlInvokerMap = new
HashMap<String, Invoker<T>>();
11.         if(urls == null || urls.size() == 0){
12.             return newUrlInvokerMap;
13.         }
14.         Set<String> keys = new HashSet<String>();
15.         String queryProtocols =
this.queryMap.get(Constants.PROTOCOL_KEY);
16.         for (URL providerUrl : urls) {
17.             //如果reference端配置了protocol，则只选择匹配的
protocol
18.             if (queryProtocols != null &&
queryProtocols.length() >0) {
19.                 boolean accept = false;
20.                 String[] acceptProtocols =
queryProtocols.split(",");
21.                 for (String acceptProtocol : acceptProtocols)
{
22.                     if
(providerUrl.getProtocol().equals(acceptProtocol)) {
23.                         accept = true;
24.                         break;
25.                     }
26.                 }
27.                 if (!accept) {
28.                     continue;
29.                 }
30.             }
31.             if
(Constants.EMPTY_PROTOCOL.equals(providerUrl.getProtocol())) {
32.                 continue;
33.             }
34.             // 加载provider声明的protocol,如果加载不到则报错（默认
protocol=dubbo，实现为
com.alibaba.dubbo.rpc.protocol.dubbo.DubboProtocol）
35.             if (!
ExtensionLoader.getExtensionLoader(Protocol.class).hasExtension(p
roviderUrl.getProtocol())) {
36.                 logger.error(new
IllegalStateException("Unsupported protocol " +
```

```
providerUrl.getProtocol() + " in notified url: " + providerUrl +
" from registry " + getUrl().getAddress() + " to consumer " +
NetUtils.getLocalHost()
37.                             + ", supported protocol:
"+ExtensionLoader.getExtensionLoader(Protocol.class).getSupported
Extensions()));
38.                     continue;
39.                 }
40.                 // 合并url参数 顺序为override > -D >Consumer >
Provider
41.                 // 并且加入不检查连接是否成功的参数，总是创建Invoker!
42.                 URL url = mergeUrl(providerUrl);
43.
44.                 String key = url.toFullString(); // URL参数是排序的
45.                 if (keys.contains(key)) { // 重复URL
46.                     continue;
47.                 }
48.                 keys.add(key);
49.                 // 缓存key为没有合并消费端参数的URL，不管消费端如何合并参
数，如果服务端URL发生变化，则重新refer
50.                 Map<String, Invoker<T>> localUrlInvokerMap =
this.urlInvokerMap; // local reference
51.                 Invoker<T> invoker = localUrlInvokerMap == null ?
null : localUrlInvokerMap.get(key);
52.                 if (invoker == null) { // 缓存中没有，重新refer
53.                     try {
54.                         boolean enabled = true;
55.                         if
(url.hasParameter(Constants.DISABLED_KEY)) {
56.                             enabled = !
url.getParameter(Constants.DISABLED_KEY, false);
57.                         } else {
58.                             enabled =
url.getParameter(Constants.ENABLED_KEY, true);
59.                         }
60.                         if (enabled) {
61.                                         // 创建InvokerDelegete,主要用于
存储注册中心下发的url地址，用于重新重新refer时能够根据providerURL
queryMap overrideMap重新组装
62.                                     invoker = new
InvokerDelegete<T>(protocol.refer(serviceType, url), url,
providerUrl);
63.                         }
64.                     } catch (Throwable t) {
65.                         logger.error("Failed to refer invoker for
```

```
interface:"+serviceType+",url:("+url+")" + t.getMessage(), t);
66.                 }
67.                 if (invoker != null) { // 将新的引用放入缓存
68.                     newUrlInvokerMap.put(key, invoker);
69.                 }
70.             }else {
71.                 newUrlInvokerMap.put(key, invoker);
72.             }
73.         }
74.         keys.clear();
75.         return newUrlInvokerMap;
76.     }
```

回顾一下整个过程：

1、通过url创建Registry；

2、从本地的缓存文件加载之前订阅到的数据；

3、Registry通过url包含的地址连接到注册中心；

4、添加一个定时任务，该任务5s后执行，每5s执行一次，执行的内容为：对（注册/取消注册/订阅/取消订阅/通知）失败的列表进行重试；

5、添加一个状态变更的事件监听器，当连接断开后，加入到注册失败的url列表中；

6、将自身的连接注册到consumer节点下，供管理中心查询；

7、订阅接口下的其他节点（providers/configurators/routers）的变更，并获取其已有值；

8、将7中获取到的对应providers节点下的值保存到本地的缓存文件中，这些值就是服务端的连接信息；

9、通过服务端的连接信息创建Invoker；

Invoker的创建流程比较复杂，我们下一篇文章单独介绍。