crazyhzm 265

Dubbo源码解析（六）注册中心——redis

java dubbo 阅读约 61 分钟

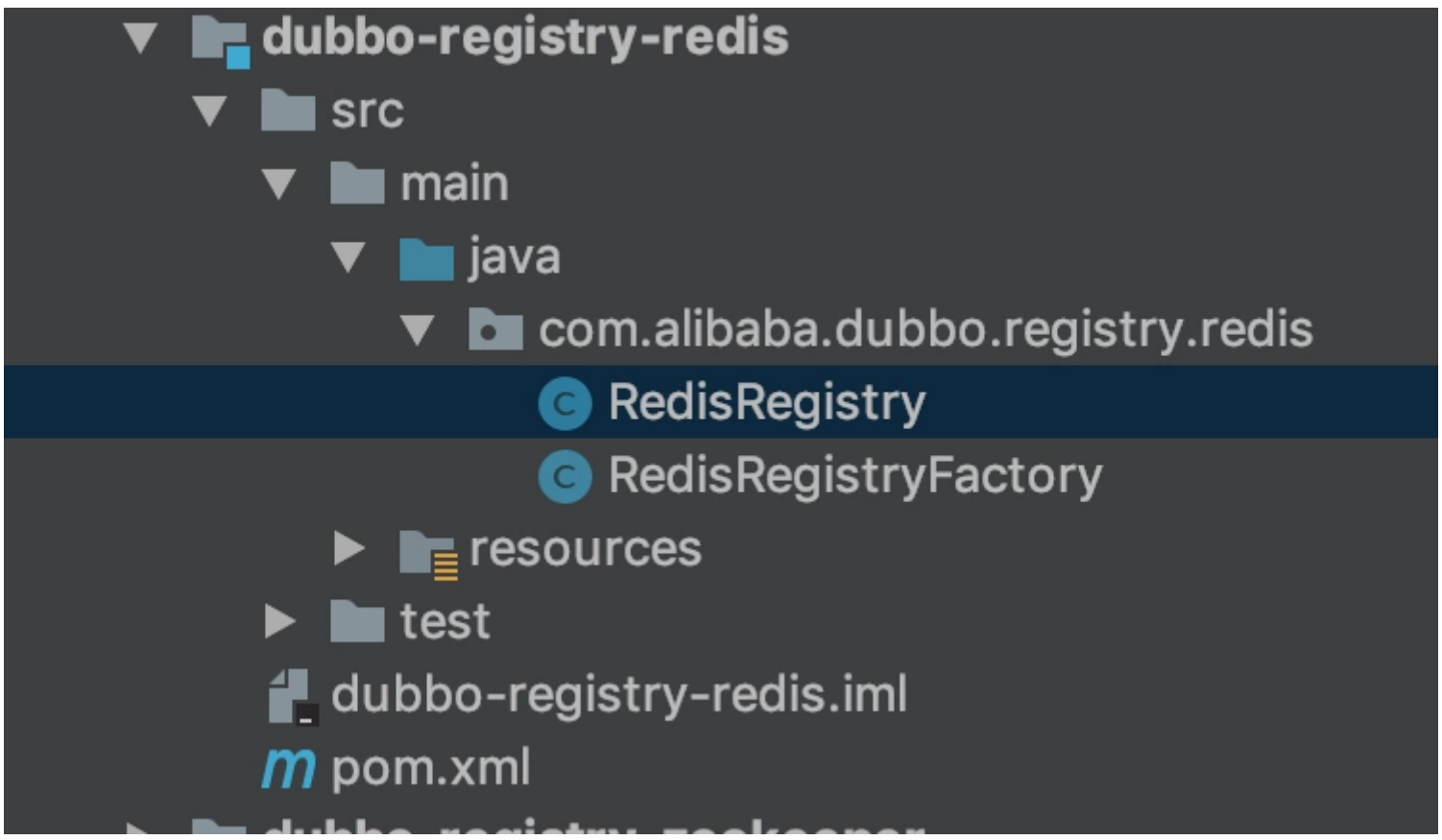
注册中心——redis

> 目标：解释以为redis实现的注册中心原理，解读duubo-registry-redis的源码

Redis是一个key-value存储系统，交换数据非常快，redis以内存作为数据存储的介质，所以读写数据的效率极高，远远超过数据库。redis支持丰富的数据类型，dubbo就利用了redis的value支持map的数据类型。redis的key为服务名称和服务的类型。map中的key为URL地址，map中的value为过期时间，用于判断脏数据，脏数据由监控中心删除。

dubbo利用JRedis来连接到Redis分布式哈希键-值数据库，因为Jedis实例不是线程安全的,所以不可以多个线程共用一个Jedis实例，但是创建太多的实现也不好因为这意味着会建立很多sokcet连接。 所以dubbo又用了JedisPool，JedisPool是一个线程安全的网络连接池。可以用JedisPool创建一些可靠Jedis实例，可以从池中获取Jedis实例，使用完后再把Jedis实例还回JedisPool。这种方式可以避免创建大量socket连接并且会实现高效的性能。

上述稍微介绍了dubbo用redis实现注册中心的依赖，接下来让我们来看看具体的实现逻辑。下图是包的结构：



包结构非常类似。接下来我们就来解读一下这两个类。

（一）RedisRegistry

该类继承了FailbackRegistry类，该类就是针对注册中心核心的功能注册、订阅、取消注册、取消订阅，查询注册列表进行展开，基于redis来实现。

1.属性

```
private final ScheduledExecutorService expireExecutor = Executors.newScheduledThreadPool(1, new
NamedThreadFactory("DubboRegistryExpireTimer", true));

//Redis Key 过期机制执行器
private final ScheduledFuture<?> expireFuture;

// Redis 根节点
private final String root;

// JedisPool集合，map 的key为 "ip:port"的形式
private final Map<String, JedisPool> jedisPools = new ConcurrentHashMap<String, JedisPool>();

// 通知器集合，key为 Root + Service的形式
// 例如 /dubbo/com.alibaba.dubbo.demo.DemoService
private final ConcurrentMap<String, Notifier> notifiers = new ConcurrentHashMap<String, Notifier>();

// 重连时间间隔，单位：ms
private final int reconnectPeriod;

// 过期周期，单位：ms
private final int expirePeriod;

// 是否通过监控中心，用于判断脏数据，脏数据由监控中心删除
private volatile boolean admin = false;

// 是否复制模式
private boolean replicate;
```

可以从属性中看到基于redis的注册中心可以被监控中心监控，并且对过期的节点有清理的机制。

2.构造方法

```
public RedisRegistry(URL url) {
    super(url);
    // 判断地址是否为空
    if (url.isAnyHost()) {
        throw new IllegalStateException("registry address == null");
    }
    // 实例化对象池
    GenericObjectPoolConfig config = new GenericObjectPoolConfig();
    // 如果 testOnBorrow 被设置，pool 会在 borrowObject 返回对象之前使用 PoolableObjectFactory的 validateObject 来验
证这个对象是否有效
    // 要是对象没通过验证，这个对象会被丢弃，然后重新选择一个新的对象。
    config.setTestOnBorrow(url.getParameter("test.on.borrow", true));
    // 如果 testOnReturn 被设置， pool 会在 returnObject 的时候通过 PoolableObjectFactory 的validateObject 方法验证对
象
    // 如果对象没通过验证，对象会被丢弃，不会被放到池中。
    config.setTestOnReturn(url.getParameter("test.on.return", false));
    // 指定空闲对象是否应该使用 PoolableObjectFactory 的 validateObject 校验，如果校验失败，这个对象会从对象池中被清除。
    // 这个设置仅在 timeBetweenEvictionRunsMillis 被设置成正值（ >0） 的时候才会生效。
    config.setTestWhileIdle(url.getParameter("test.while.idle", false));
    if (url.getParameter("max.idle", 0) > 0)
        // 控制一个pool最多有多少个状态为空闲的jedis实例。
        config.setMaxIdle(url.getParameter("max.idle", 0));
    if (url.getParameter("min.idle", 0) > 0)
        // 控制一个pool最少有多少个状态为空闲的jedis实例。
        config.setMinIdle(url.getParameter("min.idle", 0));
    if (url.getParameter("max.active", 0) > 0)
```

构造方法首先是调用了父类的构造函数，然后是对对象池的一些配置进行了初始化，具体的我已经在注释中写明。在构造方法中还做了连接池的创建、过期机制执行器的创建，其中过期会进行延长到期时间的操作具体是在deferExpired方法中实现。还有一个关注点事该执行器的时间是取周期的一半。

3.deferExpired

```
                    // 获得分类路径
                    String key = toCategoryPath(url);
                    // 以hash 散列表的形式存储
                    if (jedis.hset(key, url.toFullString(), String.valueOf(System.currentTimeMillis() +
expirePeriod)) == 1) {
                        // 发布 Redis 注册事件
                        jedis.publish(key, Constants.REGISTER);
                    }
                }
            }
            // 如果通过监控中心
            if (admin) {
                // 删除过时的脏数据
                clean(jedis);
            }
            // 如果服务器端已同步数据，只需写入单台机器
            if (!replicate) {
                break;//  If the server side has synchronized data, just write a single machine
            }
        } finally {
            jedis.close();
        }
    } catch (Throwable t) {
        logger.warn("Failed to write provider heartbeat to redis registry. registry: " + entry.getKey() + ",
cause: " + t.getMessage(), t);
    }
}
```

该方法实现了延长到期时间的逻辑，遍历了已经注册的服务url，这里会有一个是否为非动态管理模式的判断，也就是判断该节点是否为动态节点，只有动态节点是需要延长过期时间，因为动态节点需要人工删除节点。延长过期时间就是重新注册一次。而其他的节点则会被监控中心清除，也就是调用了clean方法。clean方法下面会讲到。

4.clean

```
        if (values != null && values.size() > 0) {
            boolean delete = false;
            long now = System.currentTimeMillis();
            for (Map.Entry<String, String> entry : values.entrySet()) {
                URL url = URL.valueOf(entry.getKey());
                // 是否为动态节点
                if (url.getParameter(Constants.DYNAMIC_KEY, true)) {
                    long expire = Long.parseLong(entry.getValue());
                    // 判断是否过期
                    if (expire < now) {
                        // 删除记录
                        jedis.hdel(key, entry.getKey());
                        delete = true;
                        if (logger.isWarnEnabled()) {
                            logger.warn("Delete expired key: " + key + " -> value: " + entry.getKey() + ",
expire: " + new Date(expire) + ", now: " + new Date(now));
                        }
                    }
                }
            }
            // 取消注册
            if (delete) {
                jedis.publish(key, Constants.UNREGISTER);
            }
        }
    }
}
```

该方法就是用来清理过期数据的，之前我提到过dubbo在redis存储数据的数据结构形式，就是redis的key为服务名称和服务的类型。map中的key为URL地址，map中的value为过期时间，用于判断脏数据，脏数据由监控中心删除，那么判断过期就是通过map中的value来判别。逻辑就是在redis中先把记录删除，然后在取消订阅。

5.isAvailable

```
@Override
public boolean isAvailable() {
    // 遍历连接池集合
    for (JedisPool jedisPool : jedisPools.values()) {
        try {
            // 从连接池中获得jedis实例
            Jedis jedis = jedisPool.getResource();
            try {
                // 判断是否有redis服务器被连接着
                // 只要有一台连接，则算注册中心可用
                if (jedis.isConnected()) {
                    return true; // At least one single machine is available.
                }
            } finally {
                jedis.close();
            }
        } catch (Throwable t) {
        }
    }
    return false;
}
```

该方法是判断注册中心是否可用，通过redis是否连接来判断，只要有一台redis可连接，就算注册中心可用。

6.destroy

```
@Override
public void destroy() {
    super.destroy();
    try {
        // 关闭过期执行器
        expireFuture.cancel(true);
    } catch (Throwable t) {
        logger.warn(t.getMessage(), t);
    }
    try {
        // 关闭通知器
        for (Notifier notifier : notifiers.values()) {
            notifier.shutdown();
        }
    } catch (Throwable t) {
        logger.warn(t.getMessage(), t);
    }
    for (Map.Entry<String, JedisPool> entry : jedisPools.entrySet()) {
        JedisPool jedisPool = entry.getValue();
        try {
            // 销毁连接池
            jedisPool.destroy();
        } catch (Throwable t) {
            logger.warn("Failed to destroy the redis registry client. registry: " + entry.getKey() + ", cause: "
+ t.getMessage(), t);
        }
```

这是销毁的方法，逻辑很清晰，gracefulShutdown方法在《dubbo源码解析（四）注册中心——dubbo》中已经讲到。

7.doRegister

```
            try {
                // 写入 Redis Map 键
                jedis.hset(key, value, expire);
                // 发布 Redis 注册事件
                // 这样订阅该 Key 的服务消费者和监控中心，就会实时从 Redis 读取该服务的最新数据。
                jedis.publish(key, Constants.REGISTER);
                success = true;
                // 如果服务器端已同步数据，只需写入单台机器
                if (!replicate) {
                    break; //  If the server side has synchronized data, just write a single machine
                }
            } finally {
                jedis.close();
            }
        } catch (Throwable t) {
            exception = new RpcException("Failed to register service to redis registry. registry: " +
entry.getKey() + ", service: " + url + ", cause: " + t.getMessage(), t);
        }
    }
    if (exception != null) {
        if (success) {
            logger.warn(exception.getMessage(), exception);
        } else {
            throw exception;
        }
    }
}
```

该方法是实现了父类FailbackRegistry的抽象方法，主要是实现了注册的功能，具体的逻辑是先将需要注册的服务信息保存到redis中，然后发布redis注册事件。

8.doUnregister

```
@Override
public void doUnregister(URL url) {
    // 获得分类路径
    String key = toCategoryPath(url);
    // 获得URL字符串作为 Value
    String value = url.toFullString();
    RpcException exception = null;
    boolean success = false;
    for (Map.Entry<String, JedisPool> entry : jedisPools.entrySet()) {
        JedisPool jedisPool = entry.getValue();
        try {
            Jedis jedis = jedisPool.getResource();
            try {
                // 删除redis中的记录
                jedis.hdel(key, value);
                // 发布redis取消注册事件
                jedis.publish(key, Constants.UNREGISTER);
                success = true;
                // 如果服务器端已同步数据，只需写入单台机器
                if (!replicate) {
                    break; //  If the server side has synchronized data, just write a single machine
                }
            } finally {
                jedis.close();
            }
        } catch (Throwable t) {
```

该方法也是实现了父类的抽象方法，当服务消费者或者提供者关闭时，会调用该方法来取消注册。逻辑就是跟注册方法方法，先从redis中删除服务相关记录，然后发布取消注册的事件，从而实时通知订阅者们。

9.doSubscribe

```
@Override
public void doSubscribe(final URL url, final NotifyListener listener) {
    // 返回服务地址
    String service = toServicePath(url);
    // 获得通知器
    Notifier notifier = notifiers.get(service);
    // 如果没有该服务的通知器，则创建一个
    if (notifier == null) {
        Notifier newNotifier = new Notifier(service);
        notifiers.putIfAbsent(service, newNotifier);
        notifier = notifiers.get(service);
        // 保证并发情况下，有且只有一个通知器启动
        if (notifier == newNotifier) {
            notifier.start();
        }
    }
    boolean success = false;
    RpcException exception = null;
    // 遍历连接池集合进行订阅，直到有一个订阅成功，仅仅向一个redis进行订阅
    for (Map.Entry<String, JedisPool> entry : jedisPools.entrySet()) {
        JedisPool jedisPool = entry.getValue();
        try {
            Jedis jedis = jedisPool.getResource();
            try {
                // 如果服务地址为*结尾，也就是处理所有的服务层发起的订阅
                if (service.endsWith(Constants.ANY_VALUE)) {
```

该方法是实现了订阅的功能。注意以下几个点：

1. 服务只会向一个redis进行订阅，只要有一个订阅成功就结束订阅。
2. 根据url携带的服务地址来调用doNotify的两个重载方法。其中一个只是遍历通知了所有服务的监听器，doNotify方法我会在后面讲到。

10.doUnsubscribe

```
@Override
public void doUnsubscribe(URL url, NotifyListener listener) {
}
```

该方法本来是取消订阅的实现，不过dubbo中并未实现该逻辑。

11.doNotify

```
private void doNotify(Jedis jedis, String key) {
    // 遍历所有的通知器，调用重载方法今天通知
    for (Map.Entry<URL, Set<NotifyListener>> entry : new HashMap<URL, Set<NotifyListener>>
(getSubscribed()).entrySet()) {
        doNotify(jedis, Arrays.asList(key), entry.getKey(), new HashSet<NotifyListener>(entry.getValue()));
    }
}

private void doNotify(Jedis jedis, Collection<String> keys, URL url, Collection<NotifyListener> listeners) {
    if (keys == null || keys.isEmpty()
            || listeners == null || listeners.isEmpty()) {
        return;
    }
    long now = System.currentTimeMillis();
    List<URL> result = new ArrayList<URL>();
    // 获得分类集合
    List<String> categories = Arrays.asList(url.getParameter(Constants.CATEGORY_KEY, new String[0]));
    // 通过url获得服务接口
    String consumerService = url.getServiceInterface();
    // 遍历分类路径，例如/dubbo/com.alibaba.dubbo.demo.DemoService/providers
    for (String key : keys) {
        // 判断服务是否匹配
        if (!Constants.ANY_VALUE.equals(consumerService)) {
            String prvoiderService = toServiceName(key);
            if (!prvoiderService.equals(consumerService)) {
                continue;
```

该方法实现了通知的逻辑，有两个重载方法，第二个比第一个多了几个参数，其实唯一的区别就是第一个重载方法是通知了所有的监听器，内部逻辑中调用了getSubscribed方法获取所有的监听器，该方法的解释可以查看《dubbo源码解析（三）注册中心——开篇》中关于subscribed属性的解释。而第二个重载方法就是对一个指定的监听器进行通知。

具体的逻辑在第二个重载的方法中，其中有以下几个需要注意的点：

1. 通知的事件要和监听器匹配。
2. 不同的角色会关注不同的分类，服务消费者会关注providers、configurations、routes这几个分类，而服务提供者会关注consumers分类，监控中心会关注所有分类。
3. 遍历分类路径，分类路径是Root + Service + Type。

12.toServiceName

```
private String toServiceName(String categoryPath) {
    String servicePath = toServicePath(categoryPath);
    return servicePath.startsWith(root) ? servicePath.substring(root.length()) : servicePath;
}
```

该方法很简单，就是从服务路径上获得服务名，这里就不多做解释了。

13.toCategoryName

```
private String toCategoryName(String categoryPath) {
    int i = categoryPath.lastIndexOf(Constants.PATH_SEPARATOR);
    return i > 0 ? categoryPath.substring(i + 1) : categoryPath;
}
```

该方法的作用是从分类路径上获得分类名。

14.toServicePath

```
private String toServicePath(String categoryPath) {
    int i;
    if (categoryPath.startsWith(root)) {
        i = categoryPath.indexOf(Constants.PATH_SEPARATOR, root.length());
    } else {
        i = categoryPath.indexOf(Constants.PATH_SEPARATOR);
    }
    return i > 0 ? categoryPath.substring(0, i) : categoryPath;
}

private String toServicePath(URL url) {
    return root + url.getServiceInterface();
}
```

这两个方法都是获得服务地址，第一个方法主要是截掉多余的部分，第二个方法主要是从url配置中获取关于服务地址的值跟根节点拼接。

15.toCategoryPath

```
private String toCategoryPath(URL url) {
    return toServicePath(url) + Constants.PATH_SEPARATOR + url.getParameter(Constants.CATEGORY_KEY,
Constants.DEFAULT_CATEGORY);
}
```

该方法是获得分类路径，格式是Root + Service + Type。

16.内部类NotifySub

```
private class NotifySub extends JedisPubSub {

    private final JedisPool jedisPool;

    public NotifySub(JedisPool jedisPool) {
        this.jedisPool = jedisPool;
    }

    @Override
    public void onMessage(String key, String msg) {
        if (logger.isInfoEnabled()) {
            logger.info("redis event: " + key + " = " + msg);
        }
        // 如果是注册事件或者取消注册事件
        if (msg.equals(Constants.REGISTER)
                || msg.equals(Constants.UNREGISTER)) {
            try {
                Jedis jedis = jedisPool.getResource();
                try {
                    // 通知监听器
                    doNotify(jedis, key);
                } finally {
                    jedis.close();
                }
            } catch (Throwable t) { // TODO Notification failure does not restore mechanism guarantee
                logger.error(t.getMessage(), t);
```

NotifySub是RedisRegistry的一个内部类，继承了JedisPubSub类，JedisPubSub类中定义了publish/subsribe的回调方法。通过继承JedisPubSub类并重新实现这些回调方法，当publish/subsribe事件发生时，我们可以定制自己的处理逻辑。这里实现了onMessage和onPMessage两个方法，当收到注册和取消注册的事件的时候通知相关的监听器数据变化，从而实现实时更新数据。

17.内部类Notifier

该类继承 Thread 类，负责向 Redis 发起订阅逻辑。

1.属性

```
// 服务名：Root + Service
private final String service;
// 需要忽略连接的次数
private final AtomicInteger connectSkip = new AtomicInteger();
// 已经忽略连接的次数
private final AtomicInteger connectSkiped = new AtomicInteger();
// 随机数
private final Random random = new Random();
// jedis实例
private volatile Jedis jedis;
// 是否是首次通知
private volatile boolean first = true;
// 是否运行中
private volatile boolean running = true;
// 连接次数随机数
private volatile int connectRandom;
```

上述属性中，部分属性都是为了redis的重连策略，用于在和redis断开链接时，忽略一定的次数和redis的连接，避免空跑。

2.resetSkip

```
private void resetSkip() {
    connectSkip.set(0);
    connectSkiped.set(0);
    connectRandom = 0;
}
```

该方法就是重置忽略连接的信息。

3.isSkip

```
private boolean isSkip() {
    // 获得忽略次数
    int skip = connectSkip.get(); // Growth of skipping times
    // 如果忽略次数超过10次，那么取随机数，加上一个10以内的随机数
    // 连接失败的次数越多，每一轮加大需要忽略的总次数，并且带有一定的随机性。
    if (skip >= 10) { // If the number of skipping times increases by more than 10, take the random number
        if (connectRandom == 0) {
            connectRandom = random.nextInt(10);
        }
        skip = 10 + connectRandom;
    }
    // 自增忽略次数。若忽略次数不够，则继续忽略。
    if (connectSkiped.getAndIncrement() < skip) { // Check the number of skipping times
        return true;
    }
    // 增加需要忽略的次数
    connectSkip.incrementAndGet();
    // 重置已忽略次数和随机数
    connectSkiped.set(0);
    connectRandom = 0;
    return false;
}
```

该方法是用来判断忽略本次对redis的连接。首先获得需要忽略的次数，如果忽略次数不小于10次，则加上一个10以内的随机数，然后判断自增的忽略次数，如果次数不够，则继续忽略，如果次数够了，增加需要忽略的次数，重置已经忽略的次数和随机数。主要的思想是连接失败的次数越多，每一轮加大需要忽略的总次数，并且带有一定的随机性。

4.run

```
@Override
public void run() {
    // 当通知器正在运行中时
    while (running) {
        try {
            // 如果不忽略连接
            if (!isSkip()) {
                try {
                    for (Map.Entry<String, JedisPool> entry : jedisPools.entrySet()) {
                        JedisPool jedisPool = entry.getValue();
                        try {
                            jedis = jedisPool.getResource();
                            try {
                                // 是否为监控中心
                                if (service.endsWith(Constants.ANY_VALUE)) {
                                    // 如果不是第一次通知
                                    if (!first) {
                                        first = false;
                                        Set<String> keys = jedis.keys(service);
                                        if (keys != null && !keys.isEmpty()) {
                                            for (String s : keys) {
                                                // 通知
                                                doNotify(jedis, s);
                                            }
                                        }
                                        // 重置
```

该方法是线程的run方法，应该很熟悉，其中做了相关订阅的逻辑，其中根据redis的重连策略做了一些忽略连接的策略，也就是调用了上述讲解的isSkip方法，订阅就是调用了jedis.psubscribe方法，它是订阅给定模式相匹配的所有频道。

4.shutdown

```
public void shutdown() {
    try {
        // 更改状态
        running = false;
        // jedis断开连接
        jedis.disconnect();
    } catch (Throwable t) {
        logger.warn(t.getMessage(), t);
    }
}
```

该方法是断开连接的方法。

（二）RedisRegistryFactory

该类继承了AbstractRegistryFactory类，实现了AbstractRegistryFactory抽象出来的createRegistry方法，看一下原代码：

```
public class RedisRegistryFactory extends AbstractRegistryFactory {

    @Override
    protected Registry createRegistry(URL url) {
        return new RedisRegistry(url);
    }

}
```

可以看到就是实例化了RedisRegistry而已，所有这里就不解释了。

后记

> 该部分相关的源码解析地址：https://github.com/CrazyHZM/i...