jetcache 再一次踩坑

作者：sirwsl
原文链接：https://ld246.com/article/1646376319126
来源网站：链滴

先给我吐槽一下，简直就是玩我个der，不过打铁还得自身硬，自己还是太菜了，玩个毛球哦。

故事

今天突然发现，哎，服务缓存失效了，具体原因就是因为redis数据库设置密码时候设置了个@符号好家伙，就是因为这个东西，加上jetcache采用lettuce进行连接，而在lettuce连接时采用解析uri形式，但是uri解析时候@会当作分割符号进行处理，这样就导致redis连接不上，之后看了源码后进更改，更改后又给我报了一个redis不支持集群，哦豁，最后没得办法改用redis的jedis连接方式，果放弃了lettuce，问题得以解决。

问题一： lettuce解析uri中的@分割符

描述

在配置jetcache文件时候，由于redis数据库密码存在@，由于是线上环境，不能更改，然后jetcach采用lettuce连接，解析uri时出错，导致连接失败，缓存失效。具体配置如下：

```
jetcache:
  statIntervalMinutes: 1 #统计间隔
  areaInCacheName: false
  local:
    default: #默认area
      type: caffeine
      keyConvertor: fastjson
      limit: 10000              #本地缓存最大个数
      defaultExpireInMillis: 10000   #缓存的时间全局 默认值
  remote:
    default:
      type: redis.lettuce #使用lettuce
      keyConvertor: fastjson
      valueEncoder: java
      valueDecoder: java
      poolConfig:
        minIdle: 1
        maxIdle: 50
        maxTotal: 1000
        maxWait: 1000
      uri: redis://xxx@xxx@ip:6379/0  #redis://密码@IP:端口/库
```

分析源码

先看源码（只拿出来关键的看）：

```
package io.lettuce.core.cluster;
public class RedisClusterClient extends AbstractRedisClient {

    private final Iterable<RedisURI> initialUris;

    //传入uri,采用URI.create(uri)进行解析，获取到URIs的list集合
    public static RedisClusterClient create(String uri) {
```

```
        LettuceAssert.notEmpty(uri, "URI must not be empty");
        return create((Iterable)RedisClusterURIUtil.toRedisURIs(URI.create(uri)));
    }
    //之后开始调用create(Iterable<RedisURI> redisURIs)进行创建
    public static RedisClusterClient create(Iterable<RedisURI> redisURIs) {
        assertNotEmpty(redisURIs);
        assertSameOptions(redisURIs);
        return new RedisClusterClient((ClientResources)null, redisURIs);
    }
    //创建RedisClusterClient
    protected RedisClusterClient(ClientResources clientResources, Iterable<RedisURI> redisURI
) {
        super(clientResources);
        assertNotEmpty(redisURIs);
        assertSameOptions(redisURIs);
        this.initialUris = Collections.unmodifiableList(LettuceLists.newList(redisURIs));
        this.setDefaultTimeout(this.getFirstUri().getTimeout());
        this.setOptions(ClusterClientOptions.builder().build());
    }

}
```

解析uri源码：

```
package java.net;
public final class URI implements Comparable<URI>, Serializable{

    //解析字符串创建URI
    public static URI create(String str) {
        try {
            return new URI(str);
        } catch (URISyntaxException x) {
            throw new IllegalArgumentException(x.getMessage(), x);
        }
    }
    //构建URI
    public URI(String str) throws URISyntaxException {
        new Parser(str).parse(false);
    }


     private class Parser {
        //就是这里可以看出它采用@进行分割，
          // [<userinfo>@]<host>[:<port>]
        //
        private int parseServer(int start, int n)
            throws URISyntaxException
        {
            // userinfo
            q = scan(p, n, "/?#", "@");
            if ((q >= p) && at(q, n, '@')) {
                checkChars(p, q, L_USERINFO, H_USERINFO, "user info");
                userInfo = substring(p, q);
                p = q + 1;              // Skip '@'
```

```
            }

            // port
            if (at(p, n, ':')) {
              ...
            }
            if (p < n)
                failExpecting("port number", p);

            return p;
        }
    }

}
```

解决方式

既然是因为解析URI出现的问题，那么我们是不是重新实现一下就好了，于是我的解决方式是写一个redisConfig配置文件，通过直接调用RedisClusterClient.create()来解决问题。

关键代码如下：

```
  @Bean
    public RedisClusterClient redisClient(){
        ArrayList<RedisURI> list = new ArrayList<>();
        RedisURI redisURI = new RedisURI();
        redisURI.setHost(host);
        redisURI.setPassword(password);
        redisURI.setPort(Integer.parseInt(port));
        redisURI.setDatabase(Integer.parseInt(database));
        list.add(redisURI);
        return RedisClusterClient.create(list);
    }
```

或者：

```
      @Bean
    public RedisClusterClient redisClient(){
        ArrayList<RedisURI> list = new ArrayList<>();
        RedisURI redisURI = new RedisURI();
        redisURI.setHost("127.0.0.1");
        redisURI.setPassword("XXXX@XXXX");
        redisURI.setPort(1234);
        list.add(redisURI);
        RedisClusterClient client = RedisClusterClient.create(list);
        client.setOptions(ClusterClientOptions.builder().
                disconnectedBehavior(ClientOptions.DisconnectedBehavior.REJECT_COMMANDS)
                .build());
        return client;
    }
```

完整实现：

```
@Configuration
public class JetCacheConfig{
```

```
    @Value(value = "${spring.redis.host}")
    private String host;

    @Value(value = "${spring.redis.port}")
    private String port;

    @Value(value = "${spring.redis.database}")
    private String database;

    @Value(value = "${spring.redis.password}")
    private String password;

    @Bean
    public RedisClusterClient redisClient(){
        ArrayList<RedisURI> list = new ArrayList<>();
        RedisURI redisURI = new RedisURI();
        redisURI.setHost(host);
        redisURI.setPassword(password);
        redisURI.setPort(Integer.parseInt(port));
        redisURI.setDatabase(Integer.parseInt(database));
        list.add(redisURI);
        return RedisClusterClient.create(list);
    }

    @Bean
    public SpringConfigProvider springConfigProvider() {
        return new SpringConfigProvider();
    }

    @Bean
    public GlobalCacheConfig config(SpringConfigProvider configProvider, RedisClusterClient r
disClient){
        Map localBuilders = new HashMap();
        EmbeddedCacheBuilder localBuilder = LinkedHashMapCacheBuilder
                .createLinkedHashMapCacheBuilder()
                .keyConvertor(FastjsonKeyConvertor.INSTANCE);
        localBuilders.put(CacheConsts.DEFAULT_AREA, localBuilder);

        Map remoteBuilders = new HashMap();
        RedisLettuceCacheBuilder remoteCacheBuilder = RedisLettuceCacheBuilder.createRedisL
ttuceCacheBuilder()
                .keyConvertor(FastjsonKeyConvertor.INSTANCE)
                .valueEncoder(JavaValueEncoder.INSTANCE)
                .valueDecoder(JavaValueDecoder.INSTANCE)
                .redisClient(redisClient);
        remoteBuilders.put(CacheConsts.DEFAULT_AREA, remoteCacheBuilder);

        GlobalCacheConfig globalCacheConfig = new GlobalCacheConfig();
        //globalCacheConfig.setConfigProvider(configProvider);//for jetcache <=2.5
        globalCacheConfig.setLocalCacheBuilders(localBuilders);
        globalCacheConfig.setRemoteCacheBuilders(remoteBuilders);
        globalCacheConfig.setStatIntervalMinutes(15);
        globalCacheConfig.setAreaInCacheName(false);
```

```
        return globalCacheConfig;
    }

}
```

就在这个时候我以为万事俱备只欠东风的时候，我一启动，哦豁，凉凉。

问题二、采用lettuce出现集群禁用问题

问题

redis设计时候只想到了单机，并没考虑集群，导致lettuce进行RedisClusterClient连接时候出现reds不支持集群操作

```
io.lettuce.core.RedisCommandExecutionException: ERR This instance has cluster support disab
ed
    at io.lettuce.core.ExceptionFactory.createExecutionException(ExceptionFactory.java:135) ~[l
ttuce-core-5.1.7.RELEASE.jar:na]
```

看到网上的解决办法是修改redis配置文件，可是这是线上环境，不敢乱动，所以果断放弃，修改方如下（未实验是否可行）：

修改redis配置文件redis.conf

```
去掉前面的#号
# cluster-enabled yes
```

解决方式

果断放弃lettuce，采用jedis进行连接

```
jetcache:
  statIntervalMinutes: 15 #统计间隔
  areaInCacheName: false
  local:
    default: #默认area
      type: caffeine
      keyConvertor: fastjson
      limit: 1000             #本地缓存最大个数
      defaultExpireInMillis: 100000   #缓存的时间全局 默认值
  remote:
    default:
      type: redis
      keyConvertor: fastjson
      valueEncoder: java
      valueDecoder: java
      poolConfig:
        minIdle: 1
        maxIdle: 50
        maxTotal: 1000
        maxWait: 1000
      host: ${spring.redis.host}
```

```
    port: ${spring.redis.port}
    database: ${spring.redis.database}
    password: ${spring.redis.password}
```

区别在原来是remote.default.type是redis.lettuce现在直接用redis

ps： 可能会出现

Caused by: java.lang.NoClassDefFoundError: redis/clients/util/Pool

这是由于jedis不存在或者是版本不匹配，自己引入就可以

```
<dependency>
        <groupId>redis.clients</groupId>
        <artifactId>jedis</artifactId>
        <version>3.0.0</version>
    </dependency>
```