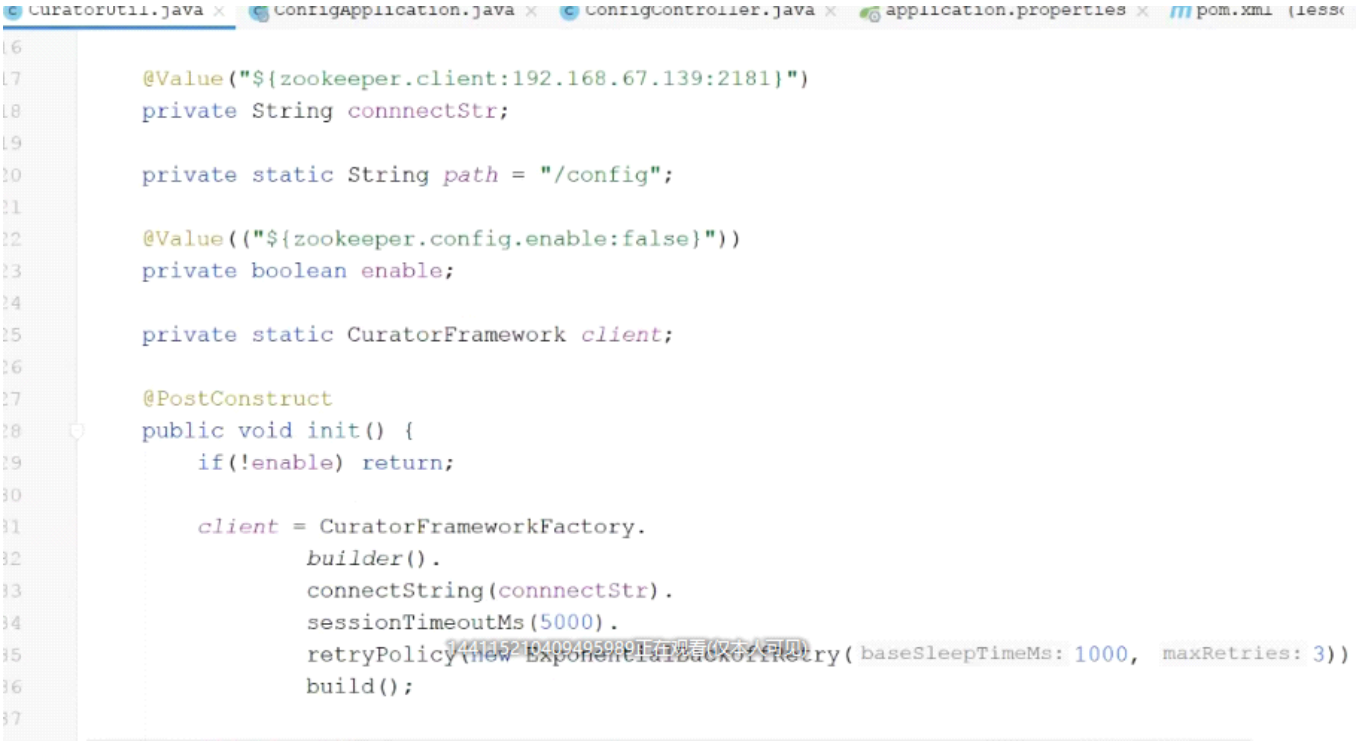
用zookeeper模拟一个分布式配置中心

2020年7月4日
20:27



```
    @Value("${zookeeper.client:192.168.67.139:2181}")
    private String connnectStr;

    private static String path = "/config";

    @Value(("${zookeeper.config.enable:false}"))
    private boolean enable;

    private static CuratorFramework client;

    @PostConstruct
    public void init() {
        if(!enable) return;

        client = CuratorFrameworkFactory.
                builder().
                connectString(connnectStr).
                sessionTimeoutMs(5000).
                retryPolicy(new ExponentialBackoffRetry( baseSleepTimeMs: 1000,  maxRetries: 3))
                build();
```

```
    try {
        Stat stat = stat = client.checkExists().forPath(path);
        if(stat == null) {
            client.create().creatingParentContainersIfNeeded().withMode(CreateMode.PERSISTENT)
                    .forPath(path,"zookeeper config".getBytes());
        } else {
            //如果/config这个节点存在，那么我就读取这个节点里面的配置信息，再把配置信息加载到spring容器中

        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

CuratorUtil.java | ConfigApplication.java | ConfigController.java | application.properties | pom.xml (less

```
            if(stat == null) {
                client.create().creatingParentContainersIfNeeded().withMode(CreateMode.PERSISTE
                        .forPath(path,"zookeeper config".getBytes());
                TimeUnit.SECONDS.sleep( timeout: 1);
            } else {
                //如果/config这个节点存在，那么我就读取这个节点里面的配置信息，再把配置信息加载到spring容器
                addChildToSpringProperty(client,path);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }


    }

    private void addChildToSpringProperty(CuratorFramework client,String path) {

    }
}
```

```
    private String connnectStr;

    private static String path = "/config";

    @Value(("${zookeeper.config.enable:false}"))
    private boolean enable;

    private static CuratorFramework client;

    @Autowired
    private ConfigurableApplicationContext applicationContext;

    private static String zkPropertyName = "zookeeperSource";

    @PostConstruct
    public void init() {
        if(!enable) return;

        client = CuratorFrameworkFactory.
```

```
    private boolean checkExistsSpringProperty() {
        MutablePropertySources propertySources = applicationContext.getEnvironment().getProper
        for (PropertySource<?> propertySource : propertySources) {
            if(zkPropertyName.equals(propertySource.getName())) {
                return true;
            }
        }
        return false;
    }
```

```
private void createZookeeperSpringProperty() {
    MutablePropertySources propertySources = applicationContext.getEnvironment().getPropert
    OriginTrackedMapPropertySource zkPropertySource = new OriginTrackedMapPropertySource(zk
    propertySources.addLast(zkPropertySource);
}
```

```
    if(!checkExistsSpringProperty()) {
        //创建永远zk加载属性的propertySource对象
        createZookeeperSpringProperty();
    }

    MutablePropertySources propertySources = applicationContext.getEnvironment().getPropert
    PropertySource<?> propertySource = propertySources.get(zkPropertyName);
    ConcurrentHashMap zkmap = (ConcurrentHashMap)propertySource.getSource();
}
```

```
enjoy.username=Jack
enjoy.password=123

zookeeper.config.enable=true
```



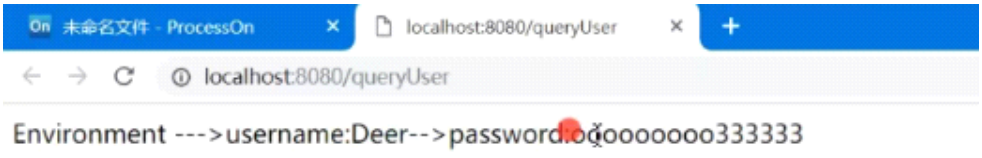
```
        }

        MutablePropertySources propertySources = applicationContext.getEnvironm
        PropertySource<?> propertySource = propertySources.get(zkPropertyName)
        ConcurrentHashMap zkmap = (ConcurrentHashMap)propertySource.getSource(

        try {
            List<String> strings = client.getChildren().forPath(path);
            //enjoy.username
            for (String string : strings) {
                //建立了映射关系
                zkmap.put(string,client.getData().forPath(path + "/" + string)
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

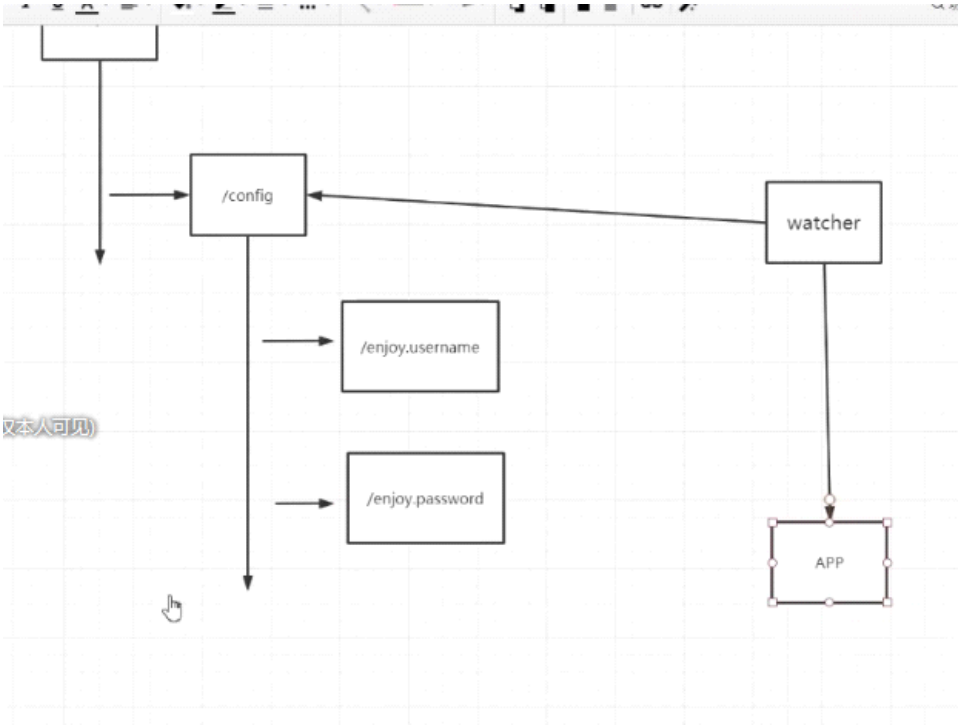
```
        createZookeeperSpringProperty();
    }

    MutablePropertySources propertySources = applicationContext.getEnvironment().getProper
    PropertySource<?> propertySource = propertySources.get(zkPropertyName);
    ConcurrentHashMap zkmap = (ConcurrentHashMap)propertySource.getSource();

    try {
        List<java.lang.String> strings = client.getChildren().forPath(path);
        //enjoy.username
        for (String string : strings) {
            //建立了映射关系
            zkmap.put(string, new String(client.getData().forPath(path + "/" + string)));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

}

private boolean checkExistsSpringProperty() {
```

Zk:

```
[zk: localhost:2181(CONNECTED) 2] get /config/enjoy.username
Deer
```

最终效果：





有节点新增的时候，进行监听

```
/*
* 有节点新增的时候要去通知应用程序
* */
private void addEnv(ChildData data, CuratorFramework client) {

}
```

```java
/*
* 有节点新增的时候要去通知应用程序
* */
private void addEnv(ChildData childData, CuratorFramework client) {
    ChildData next = childData;
    //  /config/enjoy.username
    String childpath = next.getPath();
    String data = null;

    try {
        data = new String(client.getData().forPath(childpath));
    } catch (Exception e) {
        e.printStackTrace();
    }

    MutablePropertySources propertySources = applicationContext.getEnvironment().getPrope
    PropertySource<?> propertySource = propertySources.get(zkPropertyName);
    ConcurrentHashMap chm = (ConcurrentHashMap)propertySource.getSource();
    chm.put(childpath.substring(path.length() + 1),data);
}
```

```java
pathChildrenCache.start(PathChildrenCache.StartMode.POST_INITIALIZED_EVENT);

pathChildrenCache.getListenable().addListener(new PathChildrenCacheListener() {
    @Override
    public void childEvent(CuratorFramework client, PathChildrenCacheEvent event)
        switch (event.getType()) {
            case CHILD_ADDED:
                System.out.println("增加了节点");
                addEnv(event.getData(), client);
                break;
            case CHILD_REMOVED:
                System.out.println("删除了节点");
                delEnv(event.getData());
                break;
            case CHILD_UPDATED:
                System.out.println("更新了节点");
                addEnv(event.getData(), client);
                break;
            default:
                break;
        }
    }
```

删除节点：

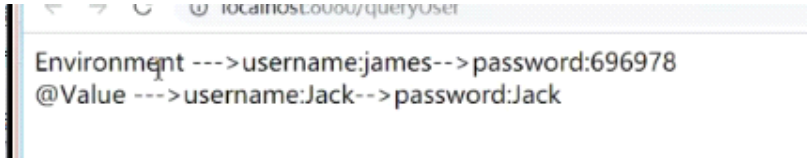
```java
}

private void delEnv(ChildData childData) {
    ChildData next = childData;
    String childpath = next.getPath();
    MutablePropertySources propertySources = applicationContext.getEnvironment().getProper
    PropertySource<?> propertySource = propertySources.get(zkPropertyName);
    ConcurrentHashMap chm = (ConcurrentHashMap) propertySource.getSource();
    chm.remove(childpath.substring(path.length() + 1));
```

ConfigApplication (1)

Controller

```java
@Slf4j
@RestController
public class ConfigController {

    @Autowired
    private Environment environment;

    @RequestMapping("/queryUser")
    public String queryUser() {
        String username = environment.getProperty("enjoy.username");
        String password = environment.getProperty("enjoy.password");
        log.info("Environment --->username:" + username + "-->password:" + password);
        return "Environment --->username:" + username + "-->password:" + password ;
    }
}
```

```java
    @Value("${enjoy.username:Jack}")
    private String username;

    @Value("${enjoy.password:Jack}")
    private String password;


    @RequestMapping("/queryUser")
    public String queryUser() {
        String username = environment.getProperty("enjoy.username");
        String password = environment.getProperty("enjoy.password");
        log.info("Environment --->username:" + username + "-->password:" + password);
        return "Environment --->username:" + username + "-->password:" + password ;
    }
}
```



Environment 的实现了动态刷新
@Value 的没办法动态刷新

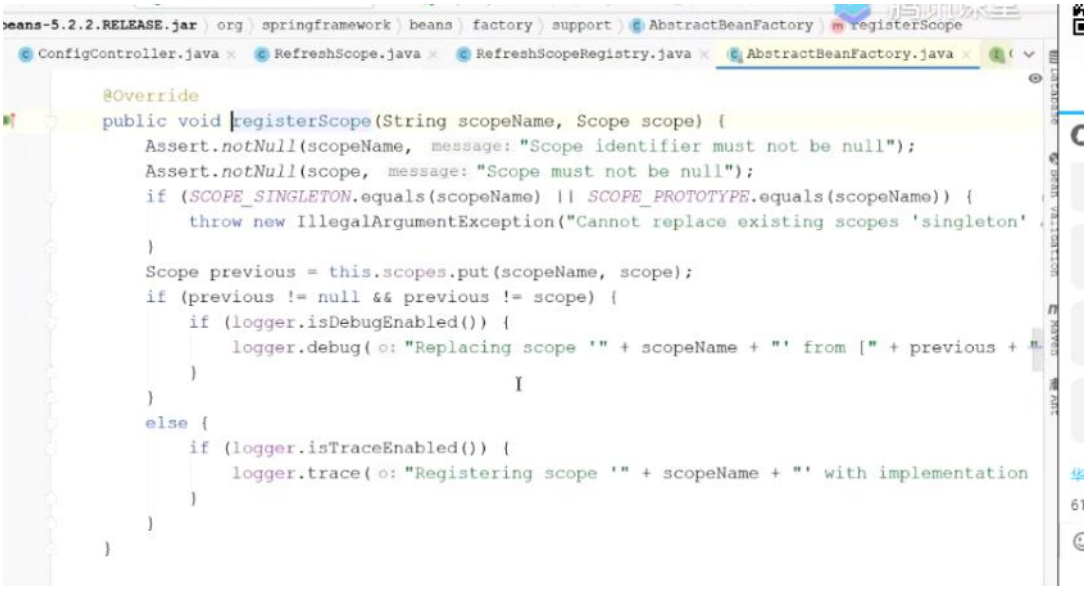
原因：@Value的原理，是从environment中的 propertySource拿取属性，进行依赖注入

让 ConfigController 类再一次触发 IOC，依赖注入操作

如果能让一个被 spring 管理的类再一次触发 IOC 依赖注入呢？？？

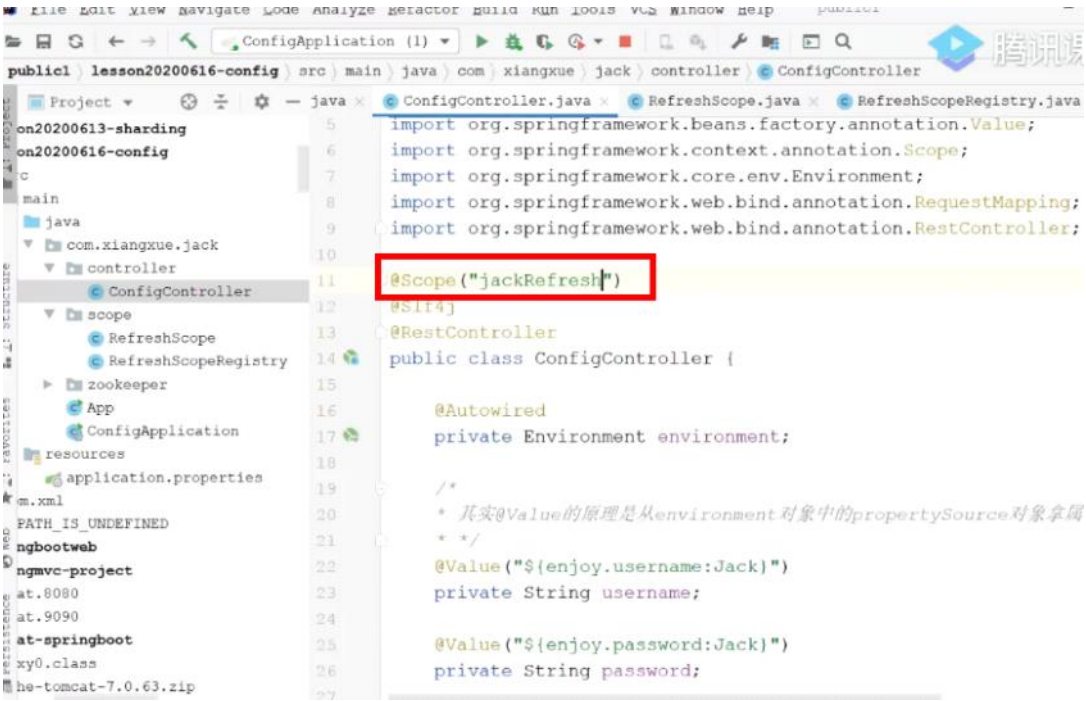
自定义bena的scope 作用域

```
@Getter
public class RefreshScopeRegistry implements BeanDefinitionRegistryPostProcessor {

    private BeanDefinitionRegistry beanDefinitionRegistry;

    @Override
    public void postProcessBeanDefinitionRegistry(BeanDefinitionRegistry registry) throws Bea
        beanDefinitionRegistry = registry;
    }

    @Override
    public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws Be
        beanFactory.registerScope( scopeName: "jackRefresh",new RefreshScope());
    }
}
```



```
@Override
public void registerScope(String scopeName, Scope scope) {
    Assert.notNull(scopeName,  message: "Scope identifier must not be null");
    Assert.notNull(scope,  message: "Scope must not be null");
    if (SCOPE_SINGLETON.equals(scopeName) || SCOPE_PROTOTYPE.equals(scopeName)) {
        throw new IllegalArgumentException("Cannot replace existing scopes 'singleton'
    }
    Scope previous = this.scopes.put(scopeName, scope);
    if (previous != null && previous != scope) {
        if (logger.isDebugEnabled()) {
            logger.debug( o: "Replacing scope '" + scopeName + "' from [" + previous + "
        }
    }
    else {
        if (logger.isTraceEnabled()) {
            logger.trace( o: "Registering scope '" + scopeName + "' with implementation
        }
    }
}
```

标识 这是一个需要动态刷新的类



```
@Scope("jackRefresh")
@Slf4j
@RestController
public class ConfigController {

    @Autowired
    private Environment environment;

    /*
     * 其实@Value的原理是从environment对象中的propertySource对象拿值
     * */
    @Value("${enjoy.username:Jack}")
    private String username;

    @Value("${enjoy.password:Jack}")
    private String password;
```

```
private void childNodeCache(CuratorFramework client, String path) {
    try {
        final PathChildrenCache pathChildrenCache = new PathChildrenCache(client, path,
        pathChildrenCache.start(PathChildrenCache.StartMode.POST_INITIALIZED_EVENT);

        pathChildrenCache.getListenable().addListener(new PathChildrenCacheListener() {
            @Override
            public void childEvent(CuratorFramework client, PathChildrenCacheEvent event)
                switch (event.getType()) {
                    case CHILD_ADDED:
                        System.out.println("增加了节点");
                        addEnv(event.getData(), client);
                        break;
                    case CHILD_REMOVED:
                        System.out.println("删除了节点");
                        delEnv(event.getData());
                        break;
                    case CHILD_UPDATED:
```

```
public void childEvent(CuratorFramework client, PathChildrenCacheEvent event)
    switch (event.getType()) {
        case CHILD_ADDED:
            System.out.println("增加了节点");
            addEnv(event.getData(), client);
            break;
        case CHILD_REMOVED:
            System.out.println("删除了节点");
            delEnv(event.getData());
            break;
        case CHILD_UPDATED:
            System.out.println("更新了节点");
            addEnv(event.getData(), client);
            break;
        default:
            break;
    }
}
});
```

CuratorUtil.java | ConfigApplication.java | ConfigController.java | RefreshScope.java | RefreshScop

```
                //在这里就需要刷新@Value修饰的属性了

            }
        });
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/*
* 就是要再一次触发有@Scope("jackRefresh")有这个标识的类，就需要再一次IOC
* */
private void refreshBean() {
    //从spring中要找到有这个标识的类

}
```

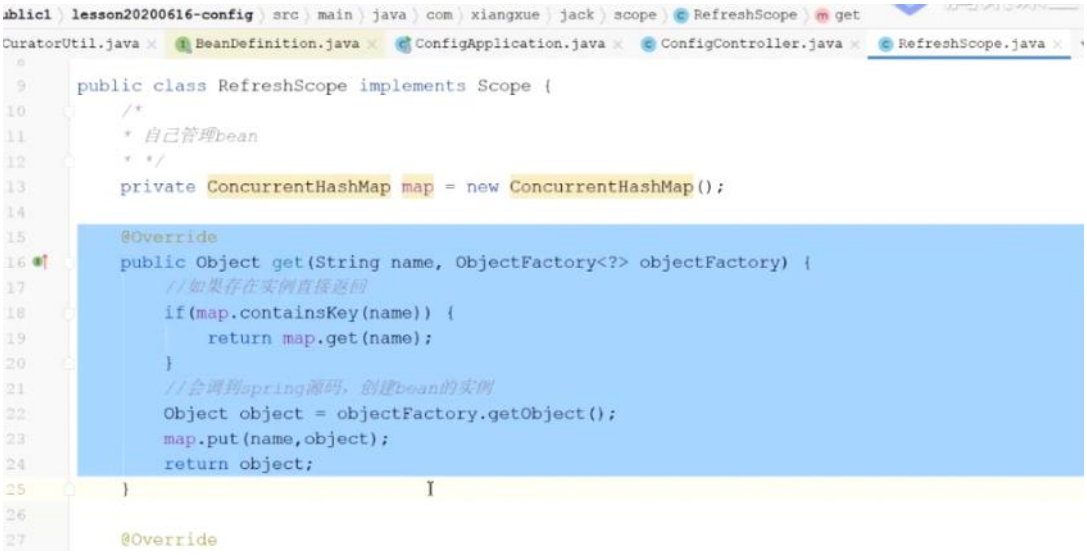
```
RefreshScopeRegistry refreshScopeRegistry = (RefreshScopeRegistry)applicationContext.g
BeanDefinitionRegistry beanDefinitionRegistry = refreshScopeRegistry.getBeanDefinitior
```

```
BeanDefinitionRegistry beanDefinitionRegistry;

@PostConstruct
public void init() {
    if (!enable) return;

    RefreshScopeRegistry refreshScopeRegistry = (RefreshScopeRegistry)applicationContext.g
    beanDefinitionRegistry = refreshScopeRegistry.getBeanDefinitionRegistry();

    client = CuratorFrameworkFactory.
* */
private void refreshBean() {
    //从spring中要找到有这个标识的类
    String[] beanDefinitionNames = applicationContext.getBeanDefinitionNames();
    for (String beanDefinitionName : beanDefinitionNames) {
        BeanDefinition beanDefinition = beanDefinitionRegistry.getBeanDefinition(beanDefir
        beanDefinition.getScope();
    }
}
```

```
/*
* 就是要再一次触发有@Scope("jackRefresh")有这个标识的类，就需要再一次IOC
* */
private void refreshBean() {
    //从spring中要找到有这个标识的类
    String[] beanDefinitionNames = applicationContext.getBeanDefinitionNames();
    for (String beanDefinitionName : beanDefinitionNames) {
        BeanDefinition beanDefinition = beanDefinitionRegistry.getBeanDefinition(beanDefir
        if(scopeName.equals(beanDefinition.getScope())) {
            //如果== 就说明这些类是需要动态刷新的
            applicationContext.getBeanFactory().destroyScopedBean(beanDefinitionName);
        }
    }
}
```

```
@Override
public void destroyScopedBean(String beanName) {
    RootBeanDefinition mbd = getMergedLocalBeanDefinition(beanName);
    if (mbd.isSingleton() || mbd.isPrototype()) {
        throw new IllegalArgumentException(
                "Bean name '" + beanName + "' does not correspond to
    }
    String scopeName = mbd.getScope();
    Scope scope = this.scopes.get(scopeName);
    if (scope == null) {
        throw new IllegalStateException("No Scope SPI registered for
    }
    Object bean = scope.remove(beanName);
    if (bean != null) {
        destroyBean(beanName, bean, mbd);
    }
}
```

getObject 会触发 这个bean的重新 IOC，


ublic1 lesson20200616-config src main java com xiangxue jack scope RefreshScope get
CuratorUtil.java BeanDefinition.java ConfigApplication.java ConfigController.java RefreshScope.java
public class RefreshScope implements Scope {
/*
* 自己管理bean
* */
private ConcurrentHashMap map = new ConcurrentHashMap();
@Override
public Object get(String name, ObjectFactory<?> objectFactory) {
//如果存在实例直接返回
if(map.containsKey(name)) {
return map.get(name);
}
//会调用spring源码，创建bean的实例
Object object = objectFactory.getObject();
map.put(name,object);
return object;
}
@Override