@ConfigurationProperties 的原理

比如

```
@Bean
@ConfigurationProperties("readwrite.datasource")
MultiDataSourceProperties readWriteDataSourceProperties() {
	return new MultiDataSourceProperties();
}
```

比如 RedisAutoConfiguration.java 中（在 org\springframework\boot\autoconfigure\cache\CacheAutoConfiguration.java 中被引用）

```
@Configuration
@ConditionalOnClass(RedisOperations.class)
@EnableConfigurationProperties(RedisProperties.class)
@Import({ LettuceConnectionConfiguration.class, JedisConnectionConfiguration.class })
public class RedisAutoConfiguration {

	@Bean
	@ConditionalOnMissingBean(name = "redisTemplate")
	public RedisTemplate<Object, Object> redisTemplate(
			RedisConnectionFactory redisConnectionFactory) throws UnknownHostException {
		RedisTemplate<Object, Object> template = new RedisTemplate<>();
		template.setConnectionFactory(redisConnectionFactory);
		return template;
	}

	@Bean
	@ConditionalOnMissingBean
	public StringRedisTemplate stringRedisTemplate(
			RedisConnectionFactory redisConnectionFactory) throws UnknownHostException {
		StringRedisTemplate template = new StringRedisTemplate();
		template.setConnectionFactory(redisConnectionFactory);
```

```
            return template;
        }

}
```

@EnableConfigurationProperties 的定义如下

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Import(EnableConfigurationPropertiesImportSelector.class)
public @interface EnableConfigurationProperties {

        /**
          * Convenient way to quickly register {@link ConfigurationProperties} annotated beans
          * with Spring. Standard Spring Beans will also be scanned regardless of this value.
          * @return {@link ConfigurationProperties} annotated beans to register
          */
        Class<?>[] value() default {};

}
```

@Import 的原理参见《Spring 注解》第 8 节。

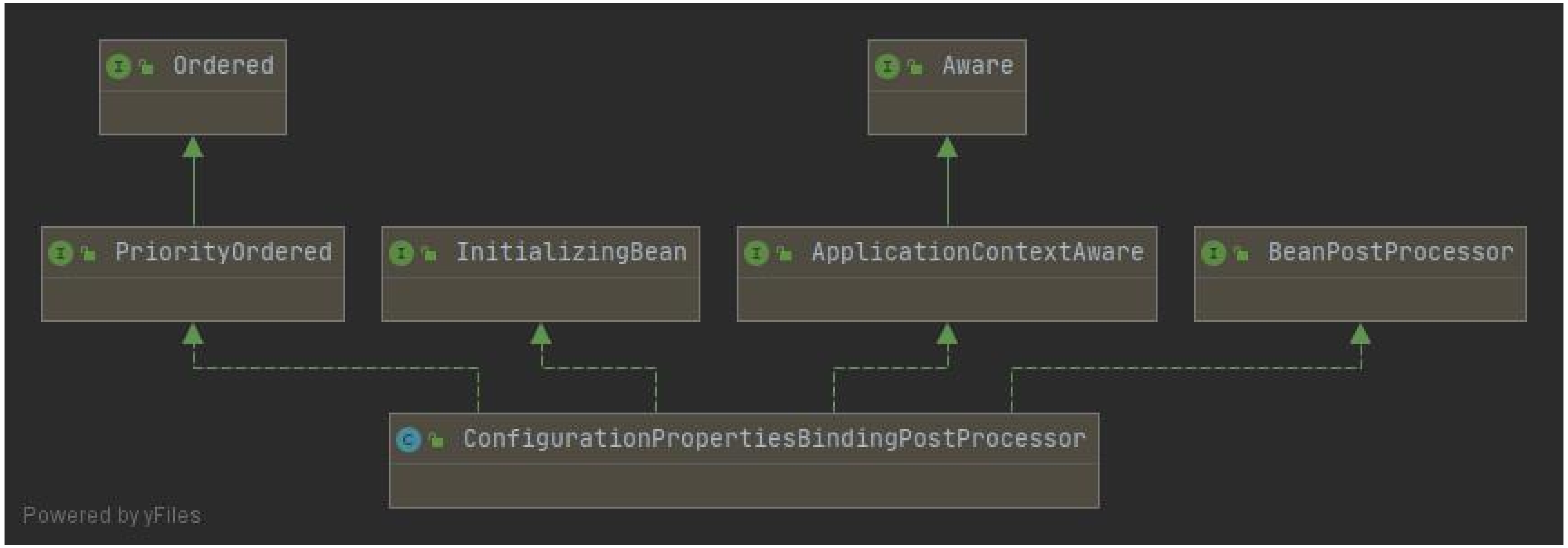
EnableConfigurationPropertiesImportSelector 的定义如下

```
class EnableConfigurationPropertiesImportSelector implements ImportSelector {

        private static final String[] IMPORTS = {
                        ConfigurationPropertiesBeanRegistrar.class.getName(),
                        ConfigurationPropertiesBindingPostProcessorRegistrar.class.getName() };

        @Override
        public String[] selectImports(AnnotationMetadata metadata) {
                return IMPORTS;
        }
        ........
}
```

ConfigurationPropertiesBindingPostProcessorRegistrar 是关键，ConfigurationPropertiesBindingPostProcessorRegistrar 向上下文重注入了一个 ConfigurationPropertiesBindingPostProcessor 类型的 BEAN，后者定义如下



可以看出，它实现了 BeanPostProcessor#postProcessBeforeInitialization,

```
@Override
public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {
        bind(ConfigurationPropertiesBean.get(this.applicationContext, bean, beanName));
        return bean;
}
```

上面的 ConfigurationPropertiesBindingPostProcessor#bind --> ConfigurationPropertiesBinder#bind --> Binder#bind --> Binder#bindObject --> Binder#findProperty。

Binder#findProperty 会在上下文中的各种 ConfigurationPropertySource 中寻找 ConfigurationProperty

```
private ConfigurationProperty findProperty(ConfigurationPropertyName name, Context context) {
    if (name.isEmpty()) {
        return null;
    }
    for (ConfigurationPropertySource source : context.getSources()) {
        ConfigurationProperty property = source.getConfigurationProperty(name);
        if (property != null) {
            return property;
        }
    }
    return null;
}
```

ConfigurationPropertySource 包括 OriginTrackedMapPropertySource {name='applicationConfig: [classpath:/application.properties]'} 、 PropertiesPropertySource {name='localProperties'}，此时会取出 @ConfigurationProperties("readwrite.datasource") 中的值 "readwrite.datasource" 当做键，但是上述循环并不会找到合适的 ConfigurationPropertySource ，接着调用 Binder#bindDataObject

上面绑定没有成功，是因为 application.properties 的定义如下：

```
readwrite.datasource.urls[0]=jdbc:mysql://127.0.01:3306/db?serverTimezone=UTC
readwrite.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
readwrite.datasource.readOnly=false
readwrite.datasource.username=root
readwrite.datasource.password=
readwrite.datasource.hikari.maximumPoolSize=200
readwrite.datasource.hikari.minimumIdle=5
readwrite.datasource.hikari.idleTimeout=600000
readwrite.datasource.hikari.connectionTimeout=30000
readwrite.datasource.hikari.maxLifetime=1800000
```

```
readonly.datasource.urls[0]=jdbc:mysql://127.0.01:3316/db?serverTimezone=UTC
readonly.datasource.readOnly=true
readonly.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
readonly.datasource.username=root
readonly.datasource.password=
readonly.datasource.hikari.maximumPoolSize=200
readonly.datasource.hikari.minimumIdle=5
readonly.datasource.hikari.idleTimeout=600000
readonly.datasource.hikari.connectionTimeout=30000
readonly.datasource.hikari.maxLifetime=1800000

上面并没有 readwrite.datasource 的属性，所以会失败。
```

此时继续递归，拼接处属性"readwrite.datasource.read-only"，此时就可以找到配置值了。虽然具体的形式不是 readwrite.datasource.readOnly，但是 Spring 做了处理，仍然可以读取到。此时就可以调用 Binder#bindProperty

```
private <T> Object bindProperty(Bindable<T> target, Context context, ConfigurationProperty property) {
        context.setConfigurationProperty(property);
        Object result = property.getValue();
        result = this.placeholdersResolver.resolvePlaceholders(result);
        result = context.getConverter().convert(result, target);
        return result;
}
```

此时的属性值是字符串，所以需要 getConverter 转换类型，然后将待绑定属性值返回到 Binder#bindObject、Binder#bind

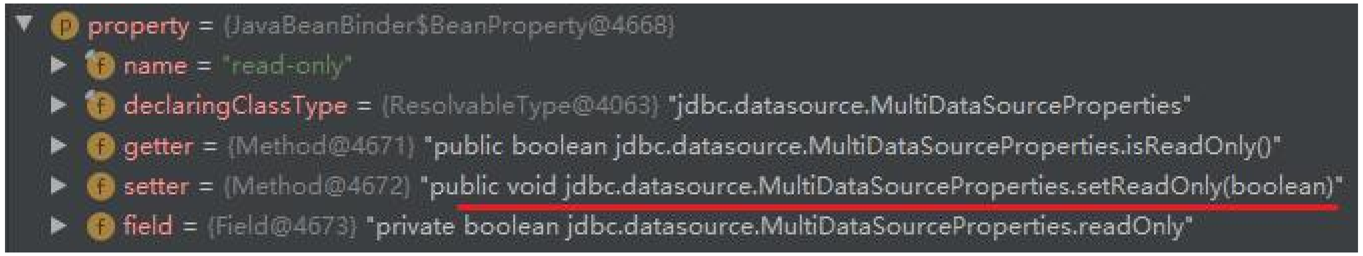
```
private <T> T bind(ConfigurationPropertyName name, Bindable<T> target, BindHandler handler, Context context,
                boolean allowRecursiveBinding, boolean create) {
        try {
                Bindable<T> replacementTarget = handler.onStart(name, target, context);
                if (replacementTarget == null) {
                        return handleBindResult(name, target, handler, context, null, create);
                }
```

```
            target = replacementTarget;
            Object bound = bindObject(name, target, handler, context, allowRecursiveBinding);
            return handleBindResult(name, target, handler, context, bound, create);
        }
        catch (Exception ex) {
            return handleBindError(name, target, handler, context, ex);
        }
}
```

最后调用 SETTER 方法，在 org\springframework\boot\context\properties\bind\JavaBeanBinder#bind 中将属性设置好

```
private <T> boolean bind(BeanSupplier<T> beanSupplier, DataObjectPropertyBinder propertyBinder,
            BeanProperty property) {
        String propertyName = property.getName();
        ResolvableType type = property.getType();
        Supplier<Object> value = property.getValue(beanSupplier);
        Annotation[] annotations = property.getAnnotations();
        Object bound = propertyBinder.bindProperty(propertyName,
                Bindable.of(type).withSuppliedValue(value).withAnnotations(annotations));
        if (bound == null) {
            return false;
        }
        if (property.isSettable()) {
            property.setValue(beanSupplier, bound);
        }
        else if (value == null || !bound.equals(value.get())) {
            throw new IllegalStateException("No setter found for property: " + property.getName());
        }
        return true;
}
```

此时 property 的状态如下



```
▼ property = {JavaBeanBinder$BeanProperty@4668}
  ▶ name = "read-only"
  ▶ declaringClassType = {ResolvableType@4063} "jdbc.datasource.MultiDataSourceProperties"
  ▶ getter = {Method@4671} "public boolean jdbc.datasource.MultiDataSourceProperties.isReadOnly()"
  ▶ setter = {Method@4672} "public void jdbc.datasource.MultiDataSourceProperties.setReadOnly(boolean)"
  ▶ field = {Field@4673} "private boolean jdbc.datasource.MultiDataSourceProperties.readOnly"
```

上述操作是在 org\springframework\boot\context\properties\bind\JavaBeanBinder.java 中完成的，

```
private <T> boolean bind(DataObjectPropertyBinder propertyBinder, Bean<T> bean, BeanSupplier<T> beanSupplier,
        Context context) {
    boolean bound = false;
    for (BeanProperty beanProperty : bean.getProperties().values()) {
        bound |= bind(beanSupplier, propertyBinder, beanProperty);
        context.clearConfigurationProperty();
    }
    return bound;
}
```

一次完成属性的注入。