【死磕 Spring】—— IOC 之 bean 的实例化策略：InstantiationStrategy (http://cmsblogs.com/?p=4022)

2019-01-28 分类：死磕 Spring (http://cmsblogs.com/?cat=206) 阅读(6293) 评论(0)

> 原文出自：http://cmsblogs.com (http://cmsblogs.com)

在开始分析 InstantiationStrategy 之前，我们先来简单回顾下 bean 的实例化过程：

1. bean 的创建，主要是 `AbstractAutowireCapableBeanFactory.doCreateBean()` ，在这个方法中有 bean 的实例化、属性注入和初始化过程，对于 bean 的实例化过程这是根据 bean 的类型来判断的，如果是单例模式，则直接从 factoryBeanInstanceCache 缓存中获取，否则调用 `createBeanInstance()` 创建。
2. 在 `createBeanInstance()` 中，如果 Supplier 不为空，则调用 `obtainFromSupplier()` 实例化 bean。如果 factory 不为空，则调用 `instantiateUsingFactoryMethod()` 实例化 bean ，如果都不是则调用 `instantiateBean()` 实例化 bean 。但是无论是 `instantiateUsingFactoryMethod()` 还是 `instantiateBean()` 最后都一定会调用到 InstantiationStrategy 接口的 `instantiate()` 。

InstantiationStrategy

InstantiationStrategy 接口定义了 Spring Bean 实例化的策略，根据创建对象情况的不同，提供了三种策略：无参构造方法、有参构造方法、工厂方法。如下：

```
public interface InstantiationStrategy {

    /**
     * 默认构造方法
     */
    Object instantiate(RootBeanDefinition bd, @Nullable String beanName, BeanFactory owner)
            throws BeansException;

    /**
     * 指定构造方法
     */
    Object instantiate(RootBeanDefinition bd, @Nullable String beanName, BeanFactory owner,
            Constructor<?> ctor, @Nullable Object... args) throws BeansException;

    /**
     * 工厂方法
     */
    Object instantiate(RootBeanDefinition bd, @Nullable String beanName, BeanFactory owner,
            @Nullable Object factoryBean, Method factoryMethod, @Nullable Object... args)
            throws BeansException;

}
```

SimpleInstantiationStrategy

InstantiationStrategy 接口有两个实现类：SimpleInstantiationStrategy 和 CglibSubclassingInstantiationStrategy。SimpleInstantiationStrategy 对以上三个方法都做了简单的实现。

如果是工厂方法实例化，则直接使用反射创建对象，如下：

```
public Object instantiate(RootBeanDefinition bd, @Nullable String beanName, BeanFactory owner,
  @Nullable Object factoryBean, final Method factoryMethod, @Nullable Object... args) {

 try {
  if (System.getSecurityManager() != null) {
   AccessController.doPrivileged((PrivilegedAction<Object>) () -> {
    ReflectionUtils.makeAccessible(factoryMethod);
    return null;
   });
  }
  else {
   ReflectionUtils.makeAccessible(factoryMethod);
  }

  Method priorInvokedFactoryMethod = currentlyInvokedFactoryMethod.get();
  try {
   currentlyInvokedFactoryMethod.set(factoryMethod);
   Object result = factoryMethod.invoke(factoryBean, args);
   if (result == null) {
    result = new NullBean();
   }
   return result;
  }
  finally {
   if (priorInvokedFactoryMethod != null) {
    currentlyInvokedFactoryMethod.set(priorInvokedFactoryMethod);
   }
   else {
    currentlyInvokedFactoryMethod.remove();
   }
  }
 }
 // 省略 catch
}
```

如果是构造方法实例化，则是先判断是否有 MethodOverrides，如果没有则是直接使用反射，如果有则就需要 CGLIB 实例化对象。如下：

```
public Object instantiate(RootBeanDefinition bd, @Nullable String beanName, BeanFactory owner) {
 // Don't override the class with CGLIB if no overrides.
 if (!bd.hasMethodOverrides()) {
  Constructor<?> constructorToUse;
  synchronized (bd.constructorArgumentLock) {
   constructorToUse = (Constructor<?>) bd.resolvedConstructorOrFactoryMethod;
   if (constructorToUse == null) {
    final Class<?> clazz = bd.getBeanClass();
    if (clazz.isInterface()) {
     throw new BeanInstantiationException(clazz, "Specified class is an interface");
    }
    try {
     if (System.getSecurityManager() != null) {
      constructorToUse = AccessController.doPrivileged(
        (PrivilegedExceptionAction<Constructor<?>>) clazz::getDeclaredConstructor);
     }
     else {
      constructorToUse =   clazz.getDeclaredConstructor();
     }
     bd.resolvedConstructorOrFactoryMethod = constructorToUse;
    }
    catch (Throwable ex) {
     throw new BeanInstantiationException(clazz, "No default constructor found", ex);
    }
   }
  }
  return BeanUtils.instantiateClass(constructorToUse);
 }
 else {
  // Must generate CGLIB subclass.
  return instantiateWithMethodInjection(bd, beanName, owner);
 }
}

public Object instantiate(RootBeanDefinition bd, @Nullable String beanName, BeanFactory owner,
  final Constructor<?> ctor, @Nullable Object... args) {

 if (!bd.hasMethodOverrides()) {
  if (System.getSecurityManager() != null) {
   // use own privileged to change accessibility (when security is on)
   AccessController.doPrivileged((PrivilegedAction<Object>) () -> {
    ReflectionUtils.makeAccessible(ctor);
    return null;
   });
  }
  return (args != null ? BeanUtils.instantiateClass(ctor, args) : BeanUtils.instantiateClass(ctor));
 }
 else {
  return instantiateWithMethodInjection(bd, beanName, owner, ctor, args);
 }
}
```

beanDefintion无lookup-method，replace-method直接使用构造器反射实例化

beanDefintion有lookup-method，replace-method直接使用cglib实例化

SimpleInstantiationStrategy 对 `instantiateWithMethodInjection()` 的实现任务交给了子类 CglibSubclassingInstantiationStrategy。

MethodOverrides

对于 MethodOverrides，如果读者是跟着小编文章一路跟过来的话一定不会陌生，在 BeanDefinitionParserDelegate 类解析 `<bean/>` 的时候是否还记得这两个方法：`parseLookupOverrideSubElements()` 和 `parseReplacedMethodSubElements()` 这两个方法分别用于解析 lookup-method 和 replaced-method。`parseLookupOverrideSubElements()` 源码如下：

```
/**
 * Parse lookup-override sub-elements of the given bean element.
 */
public void parseLookupOverrideSubElements(Element beanEle, MethodOverrides overrides) {
    NodeList nl = beanEle.getChildNodes();
    for (int i = 0; i < nl.getLength(); i++) {
        Node node = nl.item(i);
        if (isCandidateElement(node) && nodeNameEquals(node, LOOKUP_METHOD_ELEMENT)) {
            Element ele = (Element) node;
            String methodName = ele.getAttribute(NAME_ATTRIBUTE);
            String beanRef = ele.getAttribute(BEAN_ELEMENT);
            LookupOverride override = new LookupOverride(methodName, beanRef);
            override.setSource(extractSource(ele));
            overrides.addOverride(override);
        }
    }
}
```

(https://gitee.com/chenssy/blog-home/raw/master/image/201811/15395929815414.jpg)

更多关于 lookup-method 和 replaced-method 请看：【死磕 Spring】----- IOC 之解析 bean 标签：meta、lookup-method、replace-method ()

CGLIB 实例化策略

类 CglibSubclassingInstantiationStrategy 为 Spring 实例化 bean 的默认实例化策略，其主要功能还是对父类功能进行补充：其父类将 CGLIB 的实例化策略委托其实现。

```
--- SimpleInstantiationStrategy
 protected Object instantiateWithMethodInjection(RootBeanDefinition bd, @Nullable String beanName, BeanFa
ctory owner) {
  throw new UnsupportedOperationException("Method Injection not supported in SimpleInstantiationStrategy"
);
 }

--- CglibSubclassingInstantiationStrategy
 @Override
 protected Object instantiateWithMethodInjection(RootBeanDefinition bd, @Nullable String beanName, BeanFa
ctory owner) {
  return instantiateWithMethodInjection(bd, beanName, owner, null);
 }
```

CglibSubclassingInstantiationStrategy 实例化 bean 策略是通过其内部类 CglibSubclassCreator 来实现的。

```
protected Object instantiateWithMethodInjection(RootBeanDefinition bd, @Nullable String beanName, BeanFactory owner,
    @Nullable Constructor<?> ctor, @Nullable Object... args) {
   return new CglibSubclassCreator(bd, owner).instantiate(ctor, args);
  }
```

创建 CglibSubclassCreator 实例然后调用其 `instantiate()` ，该方法用于动态创建子类实例，同时实现所需要的 lookups（lookup-method、replace-method）。

```
public Object instantiate(@Nullable Constructor<?> ctor, @Nullable Object... args) {
 Class<?> subclass = createEnhancedSubclass(this.beanDefinition);
 Object instance;
 if (ctor == null) {
  instance = BeanUtils.instantiateClass(subclass);
 }
 else {
  try {
   Constructor<?> enhancedSubclassConstructor = subclass.getConstructor(ctor.getParameterTypes());
   instance = enhancedSubclassConstructor.newInstance(args);
  }
  catch (Exception ex) {
   throw new BeanInstantiationException(this.beanDefinition.getBeanClass(),
     "Failed to invoke constructor for CGLIB enhanced subclass [" + subclass.getName() + "]", ex);
  }
 }

 //这个地方解决一个bug，bug提交报告https://jira.spring.io/browse/SPR-10785
 // SPR-10785: set callbacks directly on the instance instead of in the
 // enhanced class (via the Enhancer) in order to avoid memory leaks.
 Factory factory = (Factory) instance;
 factory.setCallbacks(new Callback[] {NoOp.INSTANCE,
   new LookupOverrideMethodInterceptor(this.beanDefinition, this.owner),
   new ReplaceOverrideMethodInterceptor(this.beanDefinition, this.owner)});
 return instance;
}
```

调用 `createEnhancedSubclass()` 为提供的 BeanDefinition 创建 bean 类的增强子类。

```
private Class<?> createEnhancedSubclass(RootBeanDefinition beanDefinition) {
 // cglib里面的用法，对原始class进行增强，并设置callback
 Enhancer enhancer = new Enhancer();
 enhancer.setSuperclass(beanDefinition.getBeanClass());
 enhancer.setNamingPolicy(SpringNamingPolicy.INSTANCE);
 if (this.owner instanceof ConfigurableBeanFactory) {
  ClassLoader cl = ((ConfigurableBeanFactory) this.owner).getBeanClassLoader();
  enhancer.setStrategy(new ClassLoaderAwareGeneratorStrategy(cl));
 }
 // 过滤，自定义逻辑来指定调用的callback下标
 enhancer.setCallbackFilter(new MethodOverrideCallbackFilter(beanDefinition));
 enhancer.setCallbackTypes(CALLBACK_TYPES);
 return enhancer.createClass();
}
```

获取子类增强 class 后，如果 Constructor 实例 ctr 为空，则调用默认构造函数（BeanUtils.instantiateClass()）来实例化类，否则则根据构造函数类型获取具体的构造器，调用 newInstance() 实例化类。在 createEnhancedSubclass() 我们注意两行代码：

```
enhancer.setCallbackFilter(new MethodOverrideCallbackFilter(beanDefinition));
enhancer.setCallbackTypes(CALLBACK_TYPES);
```

通过 MethodOverrideCallbackFilter 来定义调用 callback 类型，MethodOverrideCallbackFilter 是用来定义 CGLIB 回调过滤方法的拦截器行为，它继承 CglibIdentitySupport 实现 CallbackFilter 接口，CallbackFilter 是 CGLIB 的一个回调过滤器，CglibIdentitySupport 则为 CGLIB 提供 hashCode() 和 equals() 方法，以确保 CGLIB 不会为每个 bean 生成不同的类。MethodOverrideCallbackFilter 实现 CallbackFilter accept()：

```
public int accept(Method method) {
 MethodOverride methodOverride = getBeanDefinition().getMethodOverrides().getOverride(method);
 if (logger.isTraceEnabled()) {
  logger.trace("Override for '" + method.getName() + "' is [" + methodOverride + "]");
 }
 if (methodOverride == null) {
  return PASSTHROUGH;
 }
 else if (methodOverride instanceof LookupOverride) {
  return LOOKUP_OVERRIDE;
 }
 else if (methodOverride instanceof ReplaceOverride) {
  return METHOD_REPLACER;
 }
 throw new UnsupportedOperationException("Unexpected MethodOverride subclass: " +
   methodOverride.getClass().getName());
}
```

根据 BeanDefinition 中定义的 MethodOverride 不同，返回不同的值， 这里返回的 PASSTHROUGH 、 LOOKUP_OVERRIDE 、 METHOD_REPLACER 都是 Callbak 数组的下标，这里对应的数组为 CALLBACK_TYPES 数组，如下：

```
private static final Class<?>[] CALLBACK_TYPES = new Class<?>[]
  {NoOp.class, LookupOverrideMethodInterceptor.class, ReplaceOverrideMethodInterceptor.class};
```

这里又定义了两个熟悉的拦截器 ： LookupOverrideMethodInterceptor 和 ReplaceOverrideMethodInterceptor，两个拦截器分别对应两个不同的 callback 业务：

LookupOverrideMethodInterceptor

```
private static class LookupOverrideMethodInterceptor extends CglibIdentitySupport implements MethodInter
ceptor {

 private final BeanFactory owner;

 public LookupOverrideMethodInterceptor(RootBeanDefinition beanDefinition, BeanFactory owner) {
  super(beanDefinition);
  this.owner = owner;
 }

 @Override
 public Object intercept(Object obj, Method method, Object[] args, MethodProxy mp) throws Throwable {
  // Cast is safe, as CallbackFilter filters are used selectively.
  LookupOverride lo = (LookupOverride) getBeanDefinition().getMethodOverrides().getOverride(method);
  Assert.state(lo != null, "LookupOverride not found");
  Object[] argsToUse = (args.length > 0 ? args : null); // if no-arg, don't insist on args at all
  if (StringUtils.hasText(lo.getBeanName())) {
   return (argsToUse != null ? this.owner.getBean(lo.getBeanName(), argsToUse) :
     this.owner.getBean(lo.getBeanName()));
  }
  else {
   return (argsToUse != null ? this.owner.getBean(method.getReturnType(), argsToUse) :
     this.owner.getBean(method.getReturnType()));
  }
 }
}
```

ReplaceOverrideMethodInterceptor

```
private static class ReplaceOverrideMethodInterceptor extends CglibIdentitySupport implements MethodInterceptor {

    private final BeanFactory owner;

    public ReplaceOverrideMethodInterceptor(RootBeanDefinition beanDefinition, BeanFactory owner) {
     super(beanDefinition);
     this.owner = owner;
    }

    @Override
    public Object intercept(Object obj, Method method, Object[] args, MethodProxy mp) throws Throwable {
     ReplaceOverride ro = (ReplaceOverride) getBeanDefinition().getMethodOverrides().getOverride(method);
     Assert.state(ro != null, "ReplaceOverride not found");
     // TODO could cache if a singleton for minor performance optimization
     MethodReplacer mr = this.owner.getBean(ro.getMethodReplacerBeanName(), MethodReplacer.class);
     return mr.reimplement(obj, method, args);
    }
   }
```

通过这两个拦截器，再加上这篇博客：【死磕 Spring】----- IOC 之解析 bean 标签：meta、lookup-method、replace-method ()，是不是一道绝佳的美食。

标签： Spring 源码解析 (http://cmsblogs.com/?tag=spring-%e6%ba%90%e7%a0%81%e8%a7%a3%e6%9e%90)

死磕Java (http://cmsblogs.com/?tag=%e6%ad%bb%e7%a3%95java)

死磕Spring (http://cmsblogs.com/?tag=%e6%ad%bb%e7%a3%95spring)

chenssy (http://cmsblogs.com/?author=1)
不想当厨师的程序员不是好的架构师....

上一篇
【死磕 Spring】—— IOC 之分析 BeanWrapper (http://cmsblogs.com/?p=4020)

下一篇
【死磕 Spring】—— IOC 之 BeanDefinition 注册机：BeanDefinitionRegistry (http://cmsblogs.com/?p=4026)

- 【死磕 Redis】—— 如何排查 Redis 中的慢查询 (http://cmsblogs.com/?p=18352)
- 【死磕 Redis】—— 发布与订阅 (http://cmsblogs.com/?p=18348)
- 【死磕 Redis】—— 布隆过滤器 (http://cmsblogs.com/?p=18346)
- 【死磕 Redis】—— 理解 pipeline 管道 (http://cmsblogs.com/?p=18344)