



当前位置: Java 技术驿站 (<http://cmsblogs.com>) > 死磕Java (<http://cmsblogs.com/?cat=189>) > 死磕 Spring (<http://cmsblogs.com/?cat=206>) > 正文

## 【死磕 Spring】—— IOC 之分析各 scope 的 bean 创建 (<http://cmsblogs.com/?p=2839>)

2018-10-24 分类: 死磕 Spring (<http://cmsblogs.com/?cat=206>) 阅读(6507) 评论(0)

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

在 Spring 中存在着不同的 scope, 默认是 singleton, 还有 prototype、request 等等其他的 scope, 他们的初始化步骤是怎样的呢? 这个答案在这篇博客中给出。 **singleton** Spring 的 scope 默认为 singleton, 其初始化的代码如下:

```
if (mbd.isSingleton()) {
    sharedInstance = getSingleton(beanName, () -> {
        try {
            return createBean(beanName, mbd, args);
        }
        catch (BeansException ex) {
            destroySingleton(beanName);
            throw ex;
        }
    });
    bean = getObjectForBeanInstance(sharedInstance, name, beanName, mbd);
}
```

第一部分分析了从缓存中获取单例模式的 bean, 但是如果缓存中不存在呢? 则需要从头开始加载 bean, 这个过程由 `getSingleton()` 实现。



```

public Object getSingleton(String beanName, ObjectFactory<?> singletonFactory) {
    Assert.notNull(beanName, "Bean name must not be null");

    // 全局加锁
    synchronized (this.singletonObjects) {
        // 从缓存中检查一遍
        // 因为 singleton 模式其实就是复用已经创建的 bean 所以这步骤必须检查
        Object singletonObject = this.singletonObjects.get(beanName);
        // 为空, 开始加载过程
        if (singletonObject == null) {
            // 省略 部分代码

            // 加载前置处理
            beforeSingletonCreation(beanName);
            boolean newSingleton = false;
            // 省略代码
            try {
                // 初始化 bean
                // 这个过程其实是调用 createBean() 方法
                singletonObject = singletonFactory.getObject();
                newSingleton = true;
            }
            // 省略 catch 部分
            finally {
                // 后置处理
                afterSingletonCreation(beanName);
            }
            // 加入缓存中
            if (newSingleton) {
                addSingleton(beanName, singletonObject);
            }
        }
        // 直接返回
        return singletonObject;
    }
}

```

其实这个过程并没有真正创建 bean, 仅仅只是做了一部分准备和预处理步骤, 真正获取单例 bean 的方法其实是由 `singletonFactory.getObject()` 这部分实现, 而 singletonFactory 由回调方法产生。那么这个方法做了哪些准备呢?

1. 再次检查缓存是否已经加载过, 如果已经加载了则直接返回, 否则开始加载过程。
2. 调用 `beforeSingletonCreation()` 记录加载单例 bean 之前的加载状态, 即前置处理。
3. 调用参数传递的 ObjectFactory 的 `getObject()` 实例化 bean。
4. 调用 `afterSingletonCreation()` 进行加载单例后的后置处理。
5. 将结果记录并加入值缓存中, 同时删除加载 bean 过程中所记录的一些辅助状态。

流程中涉及三个方法 `beforeSingletonCreation()` 与 `afterSingletonCreation()` 在博客【死磕 Spring】----- 加载 bean 之 缓存中获取单例 bean (<http://cmsblogs.com/?p=2808>) 中分析过了，所以这里不再阐述了，我们看另外一个方法 `addSingleton()`。

```
protected void addSingleton(String beanName, Object singletonObject) {
    synchronized (this.singletonObjects) {
        this.singletonObjects.put(beanName, singletonObject);
        this.singletonFactories.remove(beanName);
        this.earlySingletonObjects.remove(beanName);
        this.registeredSingletons.add(beanName);
    }
}
```




一个 put、一个 add、两个 remove。singletonObjects 单例 bean 的缓存，singletonFactories 单例 bean Factory 的缓存，earlySingletonObjects “早期” 创建的单例 bean 的缓存，registeredSingletons 已经注册的单例缓存。加载了单例 bean 后，调用 `getObjectForBeanInstance()` 从 bean 实例中获取对象。该方法已经在【死磕 Spring】----- 加载 bean 之 缓存中获取单例 bean (<http://cmsblogs.com/?p=2808>) 详细分析了。 **原型模式**

```
else if (mbd.isPrototype()) {
    Object prototypeInstance = null;
    try {
        beforePrototypeCreation(beanName);
        prototypeInstance = createBean(beanName, mbd, args);
    }
    finally {
        afterPrototypeCreation(beanName);
    }
    bean = getObjectForBeanInstance(prototypeInstance, name, beanName, mbd);
}
```

原型模式的初始化过程很简单：直接创建一个新的实例就可以了。过程如下：

1. 调用 `beforeSingletonCreation()` 记录加载原型模式 bean 之前的加载状态，即前置处理。
2. 调用 `createBean()` 创建一个 bean 实例对象。
3. 调用 `afterSingletonCreation()` 进行加载原型模式 bean 后的后置处理。
4. 调用 `getObjectForBeanInstance()` 从 bean 实例中获取对象。

## 其他作用域

```

String scopeName = mbd.getScope();
final Scope scope = this.scopes.get(scopeName);
if (scope == null) {
    throw new IllegalStateException("No Scope registered for scope name '" + scopeName
    e + "'");
}
try {
    Object scopedInstance = scope.get(beanName, () -> {
        beforePrototypeCreation(beanName);
        try {
            return createBean(beanName, mbd, args);
        }
        finally {
            afterPrototypeCreation(beanName);
        }
    });
    bean = getObjectForBeanInstance(scopedInstance, name, beanName, mbd);
}
catch (IllegalStateException ex) {
    throw new BeanCreationException(beanName,
        "Scope '" + scopeName + "' is not active for the current thread; consider
    " +
        "defining a scoped proxy for this bean if you intend to refer to it from
    a singleton",
        ex);
}

```

核心流程和原型模式一样，只不过获取 bean 实例是由 `scope.get()` 实现，如下：

```

public Object get(String name, ObjectFactory<?> objectFactory) {
    // 获取 scope 缓存
    Map<String, Object> scope = this.threadScope.get();
    Object scopedObject = scope.get(name);
    if (scopedObject == null) {
        scopedObject = objectFactory.getObject();
        // 加入缓存
        scope.put(name, scopedObject);
    }
    return scopedObject;
}

```

对于上面三个模块，其中最重要的有两个方法，一个是 `createBean()`、一个是 `getObjectForBeanInstance()`。这两个方法在上面三个模块都有调用，`createBean()` 后续详细说明，`getObjectForBeanInstance()` 在博客【死磕 Spring】----- 加载 bean 之 缓存中获取单例 bean (<http://cmsblogs.com/?p=2808>) 中有详细讲解，这里再次阐述下（此段内容来自《Spring 源码深度解析》）：这个方法主要是验证以下我们得到的 bean 的正确性，其实就是检测当前 bean 是否是 `FactoryBean` 类型的 bean，如果是，那么需要调用该 bean 对应的 `FactoryBean` 实例的 `getObject()` 作为返回值。无论是从缓存中获得到的 bean 还是通过不同的 scope 策略加载的 bean 都只是最原始的 bean 状态，并不一定就

是我们最终想要的 bean。举个例子，加入我们需要对工厂 bean 进行处理，那么这里得到的其实是工厂 bean 的初始状态，但是我们真正需要的是工厂 bean 中定义 factory-method 方法中返回的 bean，而 getObjectForBeanInstance() 就是完成这个工作的。至此，Spring 加载 bean 的三个部分（LZ自己划分的）已经分析完毕了。

**更多阅读** 【死磕 Spring】----- 加载 bean 之 开启 bean 的加载 (<http://cmsblogs.com/?p=2806>) 【死磕 Spring】----- IOC 之 IOC 初始化总结 (<http://cmsblogs.com/?p=2790>) 【死磕 Spring】----- 加载 bean 之 缓存中获取单例 bean (<http://cmsblogs.com/?p=2808>) 【死磕 Spring】----- 加载 bean 之 加载 BeanDefinition 与依赖处理 (<http://cmsblogs.com/?p=2810>)

👍 赞(6)

¥ 打赏

【公告】版权声明 ([http://cmsblogs.com/?page\\_id=1908](http://cmsblogs.com/?page_id=1908))

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

微服务实践（七）：从单体式架构迁移到微服务架构  
(<http://cmsblogs.com/?p=2834>)

下一篇

【死磕 Spring】—— IOC 之开启 bean 的实例化进程  
(<http://cmsblogs.com/?p=2846>)

- 【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>)
- 【死磕 Redis】—— 事务 (<http://cmsblogs.com/?p=18340>)
- 【死磕 Redis】—— Redis 的线程模型 (<http://cmsblogs.com/?p=18337>)
- 【死磕 Redis】—— Redis 通信协议 RESP (<http://cmsblogs.com/?p=18334>)
- 【死磕 Redis】—— 开篇 (<http://cmsblogs.com/?p=18332>)
- 【死磕 Spring】—— IOC 总结 (<http://cmsblogs.com/?p=4047>)
- 【死磕 Spring】—— 4 张图带你读懂 Spring IOC 的世界 (<http://cmsblogs.com/?p=4045>)
- 【死磕 Spring】—— 深入分析 ApplicationContext 的 refresh() (<http://cmsblogs.com/?p=4043>)
- 【死磕 Spring】—— ApplicationContext 相关接口架构分析 (<http://cmsblogs.com/?p=4036>)