在分析IntrospectorCleanupListener之前，先了解一下Introspector。Introspector是JDK中java.beans包下的类，它为目标JavaBean提供了一种了解原类方法、属性和事件的标准方法。通俗的说，就是可以通过Introspector构建一个BeanInfo对象，而这个BeanInfo对象中包含了目标类中的属性、方法和事件的描述信息，然后可以使用这个BeanInfo对象对目标对象进行相关操作。

下面看一个简单的示例会很容易明白。为了简单，Student类中只有一个name属性。

```
public class Student {

    private String name ;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Student{" + "name='" + name + '\'' + '}';
    }
}
```

```
public class IntrospectorTest {

    public void test() throws Exception {
        Class clazz = Class.forName("com.wind.demo.beans.Student");
        BeanInfo beanInfo = Introspector.getBeanInfo(clazz);

        MethodDescriptor[] methodDescriptors = beanInfo.getMethodDescriptors();
        Student student = new Student();
        for (MethodDescriptor methodDescriptor : methodDescriptors) {
            if (methodDescriptor.getName().equals("setName")) {
                methodDescriptor.getMethod().invoke(student, "张三");
            }
        }

        System.out.println(student);
    }
}
```

```
结果输出：Student{name='张三'}
```

通过查看Introspector.getBeanInfo方法的源码会发现，Introspector在构建一个BeanInfo对象的时候，会将构建的BeanInfo对象和原类缓存到一个Map中，源码如下。

```
    //Introspector类的getBeanInfo方法
    public static BeanInfo getBeanInfo(Class<?> beanClass)throws IntrospectionException{
        if (!ReflectUtil.isPackageAccessible(beanClass)) {
            return (new Introspector(beanClass, null, USE_ALL_BEANINFO)).getBeanInfo();
        }
        ThreadGroupContext context = ThreadGroupContext.getContext();
        BeanInfo beanInfo;
        synchronized (declaredMethodCache) {
            beanInfo = context.getBeanInfo(beanClass);
        }
        if (beanInfo == null) {
            beanInfo = new Introspector(beanClass, null, USE_ALL_BEANINFO).getBeanInfo();
            synchronized (declaredMethodCache) {
                //放入一个Map中
                context.putBeanInfo(beanClass, beanInfo);
            }
        }
        return beanInfo;
    }

    //ThreadGroupContext类的putBeanInfo方法
    BeanInfo putBeanInfo(Class<?> type, BeanInfo info) {
        if (this.beanInfoCache == null) {
            this.beanInfoCache = new WeakHashMap<>();
        }
        return this.beanInfoCache.put(type, info);
    }

    //ThreadGroupContext类的beanInfoCache
    private Map<Class<?>, BeanInfo> beanInfoCache;
```

通过上的代码可以得出，Introspector间接持有了BeanInfo的强引用。如果使用Introspector操作了很多类，那么Introspector将间接持有这些BeanInfo的强引用。在发生垃圾收集的时候，检测到这些BeanInfo存在引用链，则这些类和对应的类加载器将不会被垃圾收集器回收，进而导致内存泄漏。所以，为了解决这个问题，在使用Introspector操作完成后，调用Introspector类的flushCaches方法清除缓存。

```
    //Introspector类的flushCaches方法
    public static void flushCaches() {
        synchronized (declaredMethodCache) {
            ThreadGroupContext.getContext().clearBeanInfoCache();
            declaredMethodCache.clear();
        }
    }

    //ThreadGroupContext类的clearBeanInfoCache方法
    void clearBeanInfoCache() {
        if (this.beanInfoCache != null) {
            this.beanInfoCache.clear();
        }
    }
```

通过上面的代码会发现，清除的时候是清空了整个缓存，因为没有很好的办法来确定每个缓存是属于哪个应用的，所以清除的时候会清除所有应用的缓存。

IntrospectorCleanupListener解析

上面分析了Introspector的作用和影响，那IntrospectorCleanupListener和Introspector有什么关系呢？
IntrospectorCleanupListener是spring-web jar中的类，源码如下。

```
public class IntrospectorCleanupListener implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent event) {
        CachedIntrospectionResults.acceptClassLoader(Thread.currentThread().getContextClassLoader());
    }

    @Override
    public void contextDestroyed(ServletContextEvent event) {
        CachedIntrospectionResults.clearClassLoader(Thread.currentThread().getContextClassLoader());
        Introspector.flushCaches();
    }

}
```

IntrospectorCleanupListener实现了ServletContextListener接口，也就是说，在web容器初始化(准确的说是在filters或servlets初始化之前)的时候会执行contextInitialized方法，在ServletContext销毁(准确的说是在filters和servlets销毁之后)的时候会执行contextDestroyed方法。从图中contextDestroyed方法，可以看到在销毁ServletContext的时候调用了Introspector.flushCaches方法，清空了对应缓存。IntrospectorCleanupListener中为什么要这么做？难道是Spring使用Introspector操作后没有清空对应缓存？查看IntrospectorCleanupListener类的源码，会发现有这样一段标注。

```
<p>Note that this listener is <i>not</i> necessary when using Spring's beans
infrastructure within the application, as Spring's own introspection results
cache will immediately flush an analyzed class from the JavaBeans Introspector
cache and only hold a cache within the application's own ClassLoader.
```

大意是说，在使用Spring本身的时候并不需要使用此监听器，因为Spring自己的内部机制会立即清空对应的缓存。虽然，Spring本身不存在这样的问题，但是如果和其它框架结合使用，而其它框架有这个问题，如Struts、Quartz等，那就需要配置这个监听器，在销毁ServletContext的时候清空对应缓存。

有一点需要注意的是，像这样一个简单的Introspector内存泄漏将会导致整个应用的类加载器不会被垃圾收集器回收，如果有内存泄漏的问题，可以考虑此因素。

配置IntrospectorCleanupListener

在以往的工作经历中，多次看到在web.xml中将IntrospectorCleanupListener配置成非第一个listener。

```
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <listener>
        <listener-class>org.springframework.web.util.IntrospectorCleanupListener</listener-class>
    </listener>
```

其实，看过源码的都知道，官方的表述是必须将此监听器配置成web.xml中的第一个listener，才能在合适的时间发挥最有效的作用。

原因其实很简单，在Servlet3.0规范之前，监听器的调用是随机的，而从Servlet3.0开始，监听器的调用顺序是根据其在web.xml中配置的顺序，并且实现ServletContextListener的监听器，contextInitialized方法调用顺序是按照在web.xml中配置的顺序正序依次执行，而contextDestroyed方法的调用顺序是按照在web.xml中配置的顺序逆序依次执行。所以，如果IntrospectorCleanupListener被配置成了第一个listener，那么它的contextDestroyed方法将最后一个执行，将发挥最有效的清除作用；而如果不是，那么可能会残留未被清除的缓存。