美团点评技术团队

Spring MVC注解故障追踪记

占康 ·2016-09-30 18:46

Spring MVC是美团点评很多团队使用的Web框架。在基于Spring MVC的项目里，注解的使用几乎遍布在项目中的各个模块，有Java提供的注解，如：@Override、@Deprecated等；也有Spring提供的注解，如：@Controller、@Service、@Autowired等；同时还可能有自定义注解等。注解一方面可以作为标记说明使用；另一方面也能帮助我们省去一些配置工作，加快开发速度。注解就像语法糖一样，我有时候会“随心所欲”的把它带入到代码里，一直乐 (hú)此(lǐ)不(hú)疲(tú)。直到笔者遇到了一个由@Service注解引发的空指针问题时，才真正意识到乱用注解的危害，同时也有了下文的深入探讨！

事件起因

接到业务方需求需要封装上游的一个HTTP接口来提供系统内的服务支持，我封装这个接口并通过本地单元测试后就部署到测试环境中开始测试了。没想到一测试就报NullPointerException异常，异常栈信息如下：

```
ERROR [qtp384587033-86] 2015-12-21 16:29:00.905 com.meituan.trip.mobile.hermes.common.utils.HttpClientUtils.c
  org.apache.http.client.ClientProtocolException
    at org.apache.http.impl.client.InternalHttpClient.doExecute(InternalHttpClient.java:186) ~[httpclient-4.3
  …
  Caused by: org.apache.http.ProtocolException: Target host is not specified
  ...
```

从异常栈上可以清楚的看出错误原因，是由于请求地址不标准（以 http:// 开头）导致的。这个错误其实很诡异，因为我已经在配置文件中通过XML的方式注入URL属性值了，而且在本地写单元测试都能通过，为什么还会属性注入失败呢？经过反复的检查和尝试，发现只要在class的定义上加@Service注解，问题就会重现，去掉则正常运行。

问题定位

在保留@Service注解的情况下，重新在本地部署并启动工程，从启动日志上发现此实现Bean被替换过：

```
  INFO [main] 2015-12-21 16:28:47.078 org.springframework.beans.factory.support.DefaultListableBeanFactory.re
```

Spring Bean发生替换是因为在同一个WebApplicationContext下，重复注入同一名称的Bean实例。从上面的日志中我们可以看出，queryPartnerImpl对象最终保留的是通过[sal/service-outer.xml]配置文件注入的Bean，在这个配置文件里详细的设置了相关属性。从替换结果来看，即使发生过替换也不会影响程序到正确运行。那问题会出在哪里呢？

经过反复调试发现，只要在QueryPartnerImpl类的定义前面加上@Service注解，问题就会重现。

问题排查及解决

遇到如此诡异的问题，且又不能确定此问题是否是系统其他环境配置导致的时候，不妨可以从这个类在系统中的实例对象身上着手分析，最简单的办法是通过Jmap查询系统中的对象实例个数。

使用Jmap查询QueryPartnerImpl类在系统中的实例个数及结果：（Jmap是JDK自带的堆分析工具Java Memory Map，可以通过此工具打印出某个Java进程内存内的所有对象大小和数量；建议在测试环境中使用jmap -histo:live命令查询，执行此命令会触发一次Full GC）

```
$ jmap -histo:live 20881 | grep QueryPartnerImpl
1354:             2             80  com.meituan.trip.mobile.hermes.sal.meilv.impl.QueryPartnerImpl
```

查看发现系统中居然有2个实例！这和我们对“Spring创建Bean默认是单例的”认知不符，那就把进程Dump出来详细解刨下这2个对象吧！通过Jmap的dump参数把进程镜像dump出来：

```
$ jmap -dump:format=b,file=/tmp/heap.bin 20881
Dumping heap to /private/tmp/dump.data ...
Heap dump file created
```

此时可以使用MAT（内存分析工具，Memory Analysis Tool）并配合Jhat快速定位到此类的实例对象上，通过对象间的引用关系来查找定位原因。

首先通过Jhat工具来查看QueryPartnerImpl对象及对象间的引用关系：

```
$ jhat  /tmp/heap.bin
.....................................................................
Snapshot resolved.
Started HTTP server on port 7000
Server is ready.
```

（Jhat是JDK自带的堆分析工具Java Heap Analyse Tool，可以将堆中的对象以HTML的形式显示出来，包括对象的数量、大小等，默认端口7000。）

通过Jhat加载dump文件成功后，访问localhost:7000进入对象列表页，此时通过关键字“QueryPartnerImpl”搜索定位到具体的类上，再点击进去查看详情：

```
Class 0x6c36938b0
class com.meituan.trip.mobile.hermes.sal.meilv.impl.QueryPartnerImpl
Instances （类的实例）
Exclude subclasses
Include subclasses
References summary by Type（对象的引用关系）
References summary by type
```

点击链接Instances -> Exclude subclasses查看类的实例对象：

```
com.meituan.trip.mobile.hermes.sal.meilv.impl.QueryPartnerImpl@0x6c41b6f80 (64 bytes)
com.meituan.trip.mobile.hermes.sal.meilv.impl.QueryPartnerImpl@0x7aeafac20 (64 bytes)
```

这2个就是QueryPartnerImpl在系统中创建的2个实例对象，点击查看每个对象属性注入情况：

```
QueryPartnerImpl@0x6c41b6f80 (64 bytes)
属性：
clientId (L) : trip_trade (28 bytes)
clientSecret (L) : 6ee952489a93b51b1ffcadd040ca562e (28 bytes)
connectTimeout (I) : 15000
encode (L) : UTF-8 (28 bytes)
log (L) : org.apache.logging.slf4j.Log4jLogger@0x6c3f26240 (41 bytes)
readTimeout (I) : 15000
url (L) :  http://test.url.meituan.com/  (28 bytes)

引用关系：
com.meituan.trip.mobile.hermes.biz.cs.GroupTravelCsOrderDetailBiz@0x6c41b6f60 (48 bytes) : field queryPartner
java.util.concurrent.ConcurrentHashMap$Node@0x6c4420fe8 (44 bytes) : field val
org.springframework.beans.factory.support.DisposableBeanAdapter@0x6c41b79f0 (66 bytes) : field bean
com.meituan.trip.mobile.hermes.biz.driven.listener.snapshot.GroupTravelOrderSnapshotEventListener@0x7ae57c490
com.meituan.trip.mobile.hermes.web.controller.api.ApiAliveController@0x6c3619fe8 (24 bytes) : field queryPart

QueryPartnerImpl@0x7aeafac20 (64 bytes)
属性：
clientId (L) : <null>
clientSecret (L) : <null>
connectTimeout (I) : 0
encode (L) : <null>
log (L) : org.apache.logging.slf4j.Log4jLogger@0x6c3f26240 (41 bytes)
readTimeout (I) : 0
url (L) : <null>
引用关系：
org.springframework.beans.factory.support.DisposableBeanAdapter@0x7aeccfd40 (66 bytes) : field bean
java.util.concurrent.ConcurrentHashMap$Node@0x7aeb05b18 (44 bytes) : field val
com.meituan.trip.mobile.hermes.biz.cs.GroupTravelCsOrderDetailBiz@0x7aeafab88 (48 bytes) : field queryPartner
com.meituan.trip.mobile.hermes.web.controller.api.ApiAliveController@0x7aeb80228 (24 bytes) : field queryPart
com.meituan.trip.mobile.hermes.biz.driven.listener.snapshot.GroupTravelOrderSnapshotEventListener@0x7aeb03908
```

结果发现QueryPartnerImpl@0x6c41b6f80对象的属性是注入成功的，而QueryPartnerImpl@0x7aeafac20对象的属性却注入失败。从这里可以初步判断：导致错误的原因是我们使用的对象是属性注入失败的QueryPartnerImpl@0x7aeafac20。

问题排除到这里，我们不禁有2个疑问：

1）**为什么会出现2个对象？**

从Spring启动日志看到queryPartnerImpl有被替换的情况，其实替换的结果是把通过@Service注入的Bean替换成了用XML定义并注入的Bean，这也只能有1个对象，另一个对象怎么出现的？

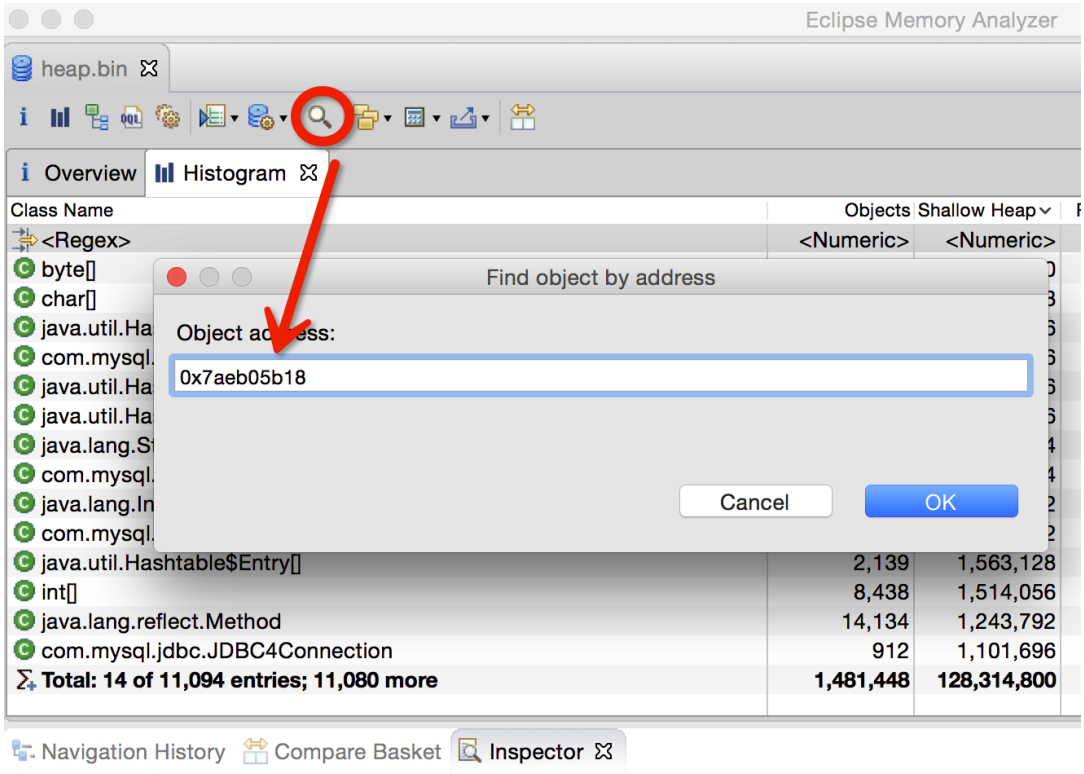
2）**谁在使用这2个对象？**

既然错误已成事实，那是谁在使用这个属性注入失败的QueryPartnerImpl@0x7aeafac20呢？而且我们每次都是使用它，而不是属性注入成功的QueryPartnerImpl@0x6c41b6f80。

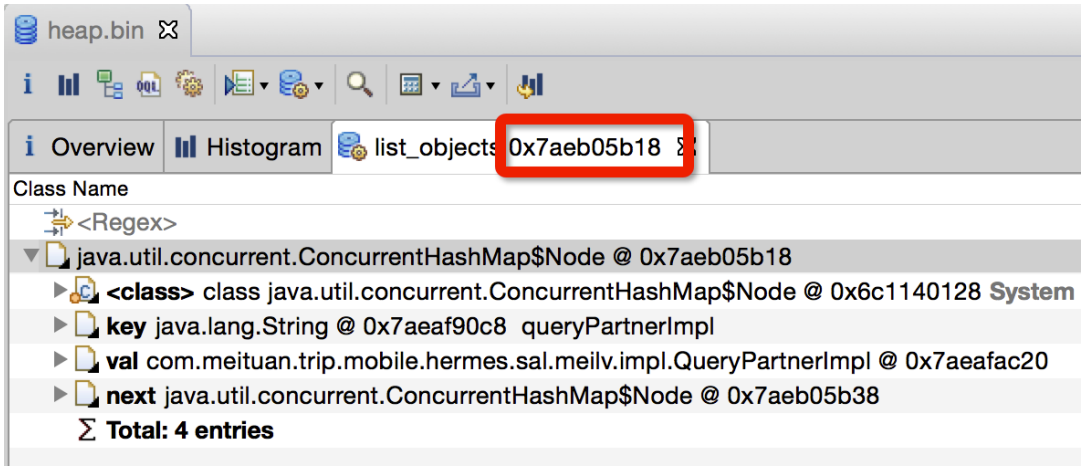
通过Jhat展示的对象引用关系看，只有org.springframework.beans.factory.support.DisposableBeanAdapter和

java.util.concurrent.ConcurrentHashMap$Node 比较可疑。但DisposableBeanAdapter是用来管理Spring Bean的销毁，所以和本事故无关，重点就落在java.util.concurrent.ConcurrentHashMap$Node 上了。

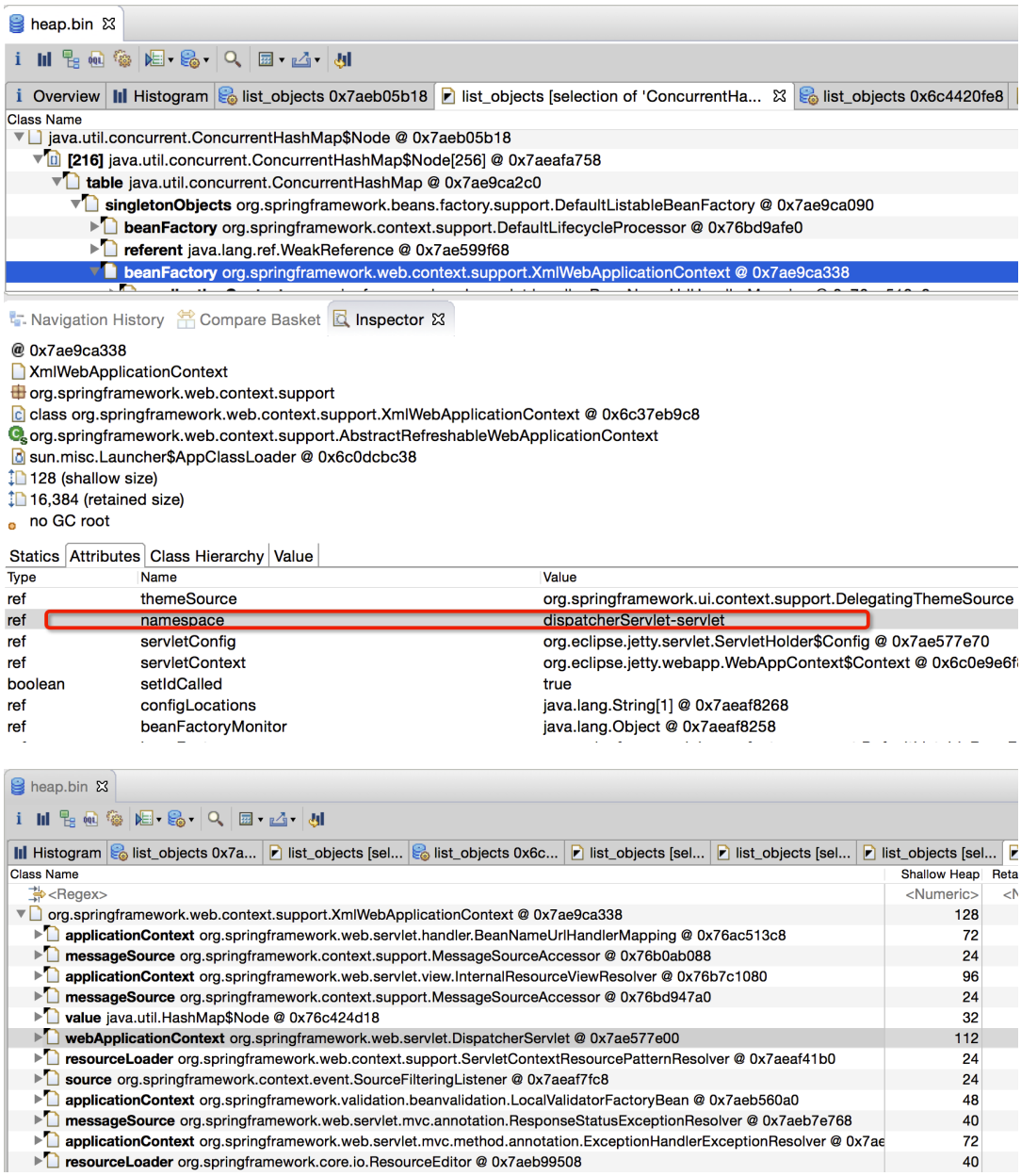
通过MAT工具来分析java.util.concurrent.ConcurrentHashMap$Node@0x7aeb05b18的引用关系，通过对象查找工具并输入对象的内存地址定位：



可直接查看此对象：

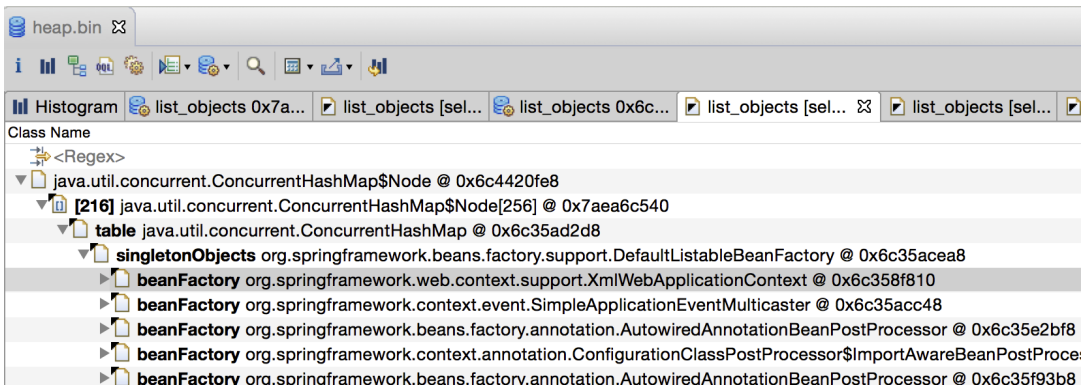


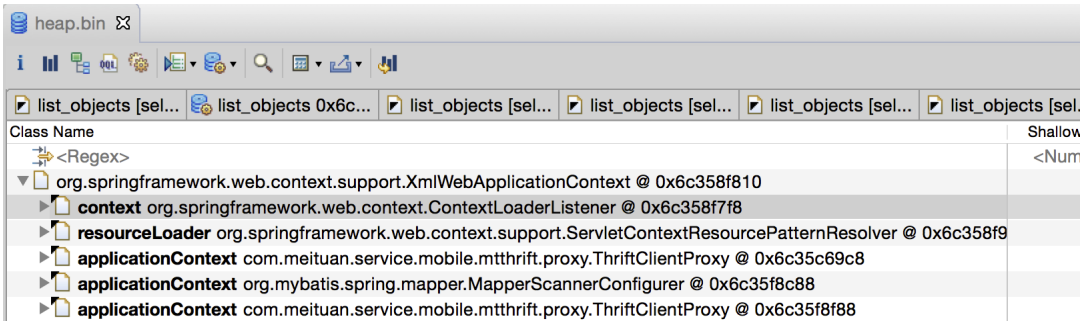
选中这个对象，右键打开菜单选项，选择：Lists objects -> with incoming references查看都有哪些对象持有此对象（with outgoing references表示此对象拥有哪些对象）：



通过上面对象引用追踪路径可以看到，queryPartnerImpl@0x7aeafac20最终被DispatcherServlet@0x7ae577e00对象引用。

采用同样的方式来分析queryPartnerImpl@0x6c41b6f80的对象引用关系：





queryPartnerImpl@0x6c41b6f80最终被ContextLoaderListener@0x6c358f7f8引用。
通过对比发现:

```
queryPartnerImpl@0x6c41b6f80 被 XmlWebApplicationContext@0x6c358f810 引用，而 XmlWebApplicationContext@0x6c35
queryPartnerImpl@0x7aeafac20 被 XmlWebApplicationContext@0x7ae9ca338 引用，而 XmlWebApplicationContext@0x7ae9
```

ContextLoaderListener和DispatcherServlet对我们来说非常熟悉，这是在Spring MVC项目中的web.xml中配置的，ContextLoaderListener用来初始化root WebApplicationContext；DispatcherServlet是请求分发控制器，启动时也会初始化一个自己的WebApplicationContext，并设置parent为root WebApplicationContext，从而形成常说的“父子关系”。DispatcherServlet如果在自己的WebApplicationContext能找到需要用的对象就直接使用，只有在找不到对象的情况下才会去查找父容器里的。

到这里我们找到了引起事故发生的根本原因，但是我们还需要找出引发事故的罪魁祸首！通过前面的分析我们知道这和ContextLoaderListener、DispatcherServlet有关系，那就定位到web.xml的配置文件中来：

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring/spring-servlet.xml</param-value>
    </init-param>
    <init-param>
        <param-name>dispatchOptionsRequest</param-name>
        <param-value>true</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/*</url-pattern>
</servlet-mapping>
```

在spring/spring-servlet.xml配置文件中我们开启了注解扫描功能，并且从项目路径“com.meituan.trip.mobile.hermes”开始扫描：

```
<context:annotation-config/>
<aop:aspectj-autoproxy/>
<context:component-scan base-package="com.meituan.trip.mobile.hermes"/>
<bean id="jacksonObjectMapper" class="org.codehaus.jackson.map.ObjectMapper">
</bean>
<bean class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
    <property name="targetObject" ref="jacksonObjectMapper"/>
    <property name="targetMethod" value="configure"/>
    <property name="arguments">
        <list>
            <value type="org.codehaus.jackson.map.DeserializationConfig$Feature">FAIL_ON_UNKN(
            <value>false</value>
        </list>
    </property>
</bean>
```

我们知道Spring会通过@Service注解去实例化一个Bean，属性如果没有通过注解注入进来的话，就用默认值。在此配置文件后面就再没有对queryPartnerImpl的定义，也就不会发生替换的情况。DispatcherServlet只能获得由注解加载的半成品Bean。

再来看看ContextLoaderListener的配置文件applicationContext.xml：

```
<context:annotation-config/>
<context:component-scan base-package="com.meituan.trip.mobile.hermes,com.sankuai.meituan"/>
<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>
<bean id="propertyConfigurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>classpath:config/config.properties</value>
            <value>classpath:config/thrift.properties</value>
            <value>classpath:dal/db.properties</value>
            <value>classpath:sal/outer/mail.properties</value>
            <value>classpath:sal/outer/mtservice.properties</value>
        </list>
    </property>
</bean>
<import resource="dal/mysql-datasource.xml"/>
<import resource="sal/outer/apibase.xml"/>
<import resource="spring/spring-mail.xml"/>
<import resource="sal/service-outer.xml"/>
<import resource="sal/service-thrift.xml"/>
<import resource="classpath:es-base.xml"/>
```

在此文件中对queryPartnerImpl使用了XML定义

我们在applicationContext.xml中也同样开启了注解扫描功能，也是从项目路径“com.meituan.trip.mobile.hermes”开始扫描，但是在下文的sal/service-out.xml配置文件中，又重新对queryPartnerImpl通过XML定义，所以会发生替换现象。

到这里我们才最终搞清楚发生这次事故的最根本原因，解决办法是要让整个系统中只有一个属性注入成功的queryPartnerImpl对象，途径有如下几种：

1）删除@Service注解：这个方法治标不治本，因为配置、 注解扫描功能后会开启包括@Service在内的超过6种注解，而这些注解部分在用；

2）扫描隔离：通过配置的属性use-default-filters并配合include-filter/exclude-filter实现扫描过滤，只扫描指定注解。

修改后的spring-servlet.xml配置（applicationContext.xml配置也需要做调整）：

```
<context:annotation-config/>
<aop:aspectj-autoproxy/>
<context:component-scan base-package="com.meituan.trip.mobile.hermes" use-default-filters="false">
    <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"></context:include-filter>
    <context:include-filter type="annotation" expression="org.springframework.web.bind.annotation.ControllerAdvice"></context:include-filter>
</context:component-scan>
```

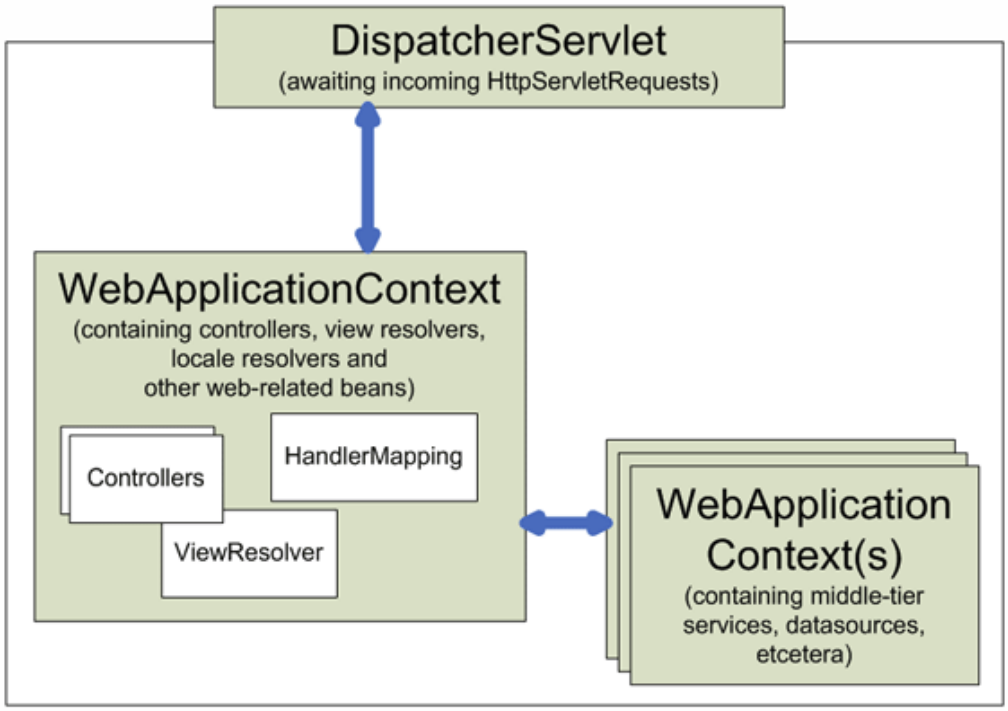
use-default-filters = true，表示Spring将会创建那些被@Component, @Repository, @Service 或 @Controller等注解标注的Bean，默认值为true。如果use-default-filters = true，同时使用并指定注解类，表示不扫描指定base-package路径下的此注解；如果use-default-filters = false，同时使用并指定注解类，表示扫描指定base-package路径下面的此注解。

问题总结

1. 使用注解并不一定会引起错误，但是注解要使用规范，不能乱用。如果通过注解注入，属性值最好也要通过注解方式注入；
2. 注解扫描功能虽然很强大、很方便，但是要注意区分扫描范围及过滤特定注解；
3. 单元测试能通过的原因：我们一般只指定加载一个配置文件作为测试环境，类实例只会出现一个，故能测试通过；
4. 最好最重要的一点就是在使用任何框架时，最好按"Best Practice"规范，避免出现一些莫名其妙的问题。

进一步探讨

通过阅读Spring源码中涉及ContextLoaderListener和DispatcherServlet的部分学习到，ContextLoaderListener在Context初始化的时候会创建一个root WebApplicationContext，并将此对象存储在ServletContext中，Key为：WebApplicationContext.class.getName() + ".ROOT” ；
DispatcherServlet在初始化过程也实例化了一个自己的WebApplicationContext，设置在ServletContext中的key为：
FrameworkServlet.class.getName() + ".CONTEXT.” + getServletName()，同时设置此对象的parent为ContextLoaderListener定义的 root WebApplicationContext。DispatcherServlet所创建的WebApplicationContext被称为子容器，子容器可以访问父容器中的内容，但父容器不能访问子容器中的内容。
Spring官方在介绍Spring MVC的同时，也给我们介绍了WebApplicationContext的继承关系：



Context hierarchy in Spring Web MVC

从图中可以看出，每个DispatcherServlet都会去实例化一个自己的WebApplicationContext，而这个WebApplicationContext可以获得root WebApplicationContext中已经实例化好的Bean。

参考文献

Spring Web MVC框架文档 (http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#d5e14359)

Java Spring MVC @Service注解 WebApplicationContext 酒店旅游 后端

关注我们

微信搜索 "美团技术团队"