https://blog.csdn.net/weixiaomao0801/article/details/56287194?utm\_source=blogxgwz0

**使用ThreadLocal却出现了有并发问题？**

2017年02月21日 16:20:11 [tech\_zhangwei](https://me.csdn.net/weixiaomao0801) 阅读数：704更多

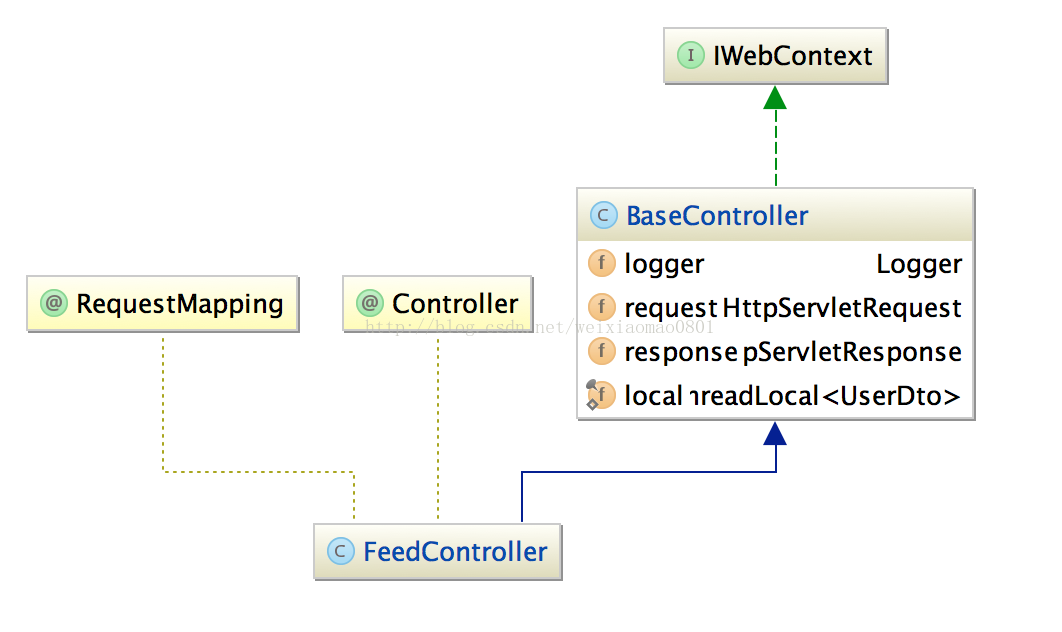
个人分类： [技术](https://blog.csdn.net/weixiaomao0801/article/category/6541612)[内存管理](https://blog.csdn.net/weixiaomao0801/article/category/6547044)[spring](https://blog.csdn.net/weixiaomao0801/article/category/6541616)[并发](https://blog.csdn.net/weixiaomao0801/article/category/6738595)[ThreadLocal](https://blog.csdn.net/weixiaomao0801/article/category/6738594)

最近同事发现所做的app在未登陆的情况刷feed的时候，相同参数请求，返回的结果确不稳定。于是找到我帮忙排查问题。

听到问题第一反应应该是并发场景下数据安全的问题，于是clone了他们的项目，对着代码开始排查。

使用的是springmvc，每个Controller(单例)都继承于一个BaseController，BaseController实现了IWebContext接口，注入了request、response对象

类图：



首先发现问题：Controller默认为单例， 故request作为成员变量肯定会在并发情况下出现参数相互覆盖的问题，解决方式：

1. @Override
2. public void setWebContext(HttpServletRequest request, HttpServletResponse response) {错误错误是使用
3. *//错误使用*
4. this.request = request;
5. *//直接使用参数request*
6. request.getPrameter("key");
8. *//dosomething*
9. }
11. *//该类其他方法使用request方式，request其实也是放在ThreadLocal中*
12. HttpServletRequest request = ((ServletRequestAttributes) RequestContextHolder.currentRequestAttributes()).getRequest();


16. empty

去掉request、response成员变量，使用线程私有变量，这个问题到此应该解决了？  fix之后再测试发现这个问题依然存在。

仔细回顾下代码，现在成员变量只有一个ThreadLocal 里边放的是userInfo信息。

jdk Thread 实现每个线程都维护了变量为threadLocals的ThreadLocal.ThreadLocalMap。ThreadLocal set、get方法都是在当前线程的ThreadMap中进行读写操作，

ThreadMap是以：ThreadLocal对象为key(set操作传入的key都是同一个对象，ThreadLocal解决key重复的算法，基于一个0x61cc88647递增获取一个新的hash值&整个table的长度-1)，在这个实现下保证了每个线程操作的就是当前线程的副本。这里是不会有线程安全问题的。

那为什么还会出现数据相互覆盖问题呢？

BaseController现在表面上看是一个线程安全的类了，那问题出在哪呢？ 多线程访问的确是做到了线程安全， 通过测试根据日志分析，在单线程访问下，也可能出现返回结果不是预期的值。 这时候想到了线程池线程复用的情况，在线程复用的情况下，先后访问可能，拿到的是同一个线程对象，这时候ThreadLocal中的数据如果没有被清理掉，同一个线程在下次访问依然可以拿到，然后重新扫描了下代码，确实没有在线程处理完之后清除ThreadLocal的代码逻辑，进一步验证了对问题的定位，然后在interceptor中加入Controller处理完后清除ThreadLocal里边的缓存信息，再测试观察，问题解决了。

总结：使用一项技术的时候，很多时候，不知道原理，拿来主义直接使用，很可能隐藏着一些风险，可能在某个时段集中爆发出来，这对于线上的项目影响是不容小觑的，我们应该在对技术的使用之后，多去看看使用的技术背后的原理，没问题学习了知识，发现问题能让问题尽早暴露