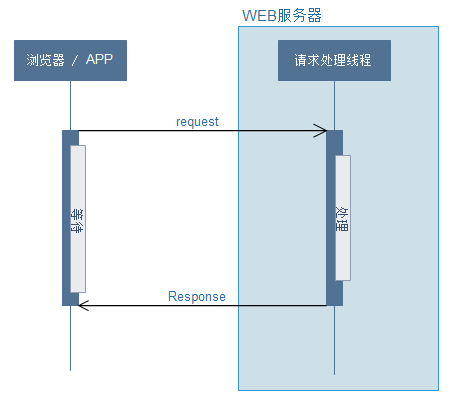
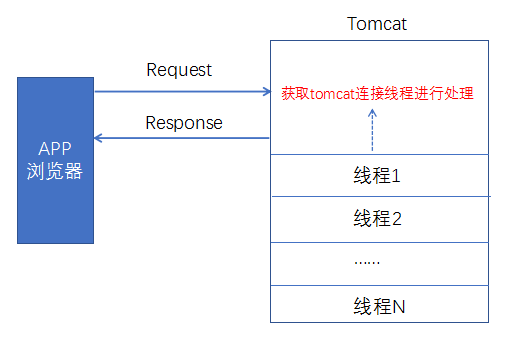
1. servlet3.0异步请求分析

8.1，什么是同步处理，请求发出后，等待服务端响应



8.2同步请求原理，从tomcat中获取连接线程进行处理，但tomcat的线程数有限，会造成线程资源的紧张。

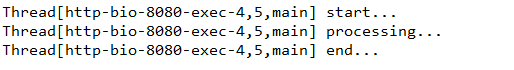


8.3同步机制操作步骤：

a,在servlet工程目录下，修改JamesServlet类，把当前处理的线程也打印出来



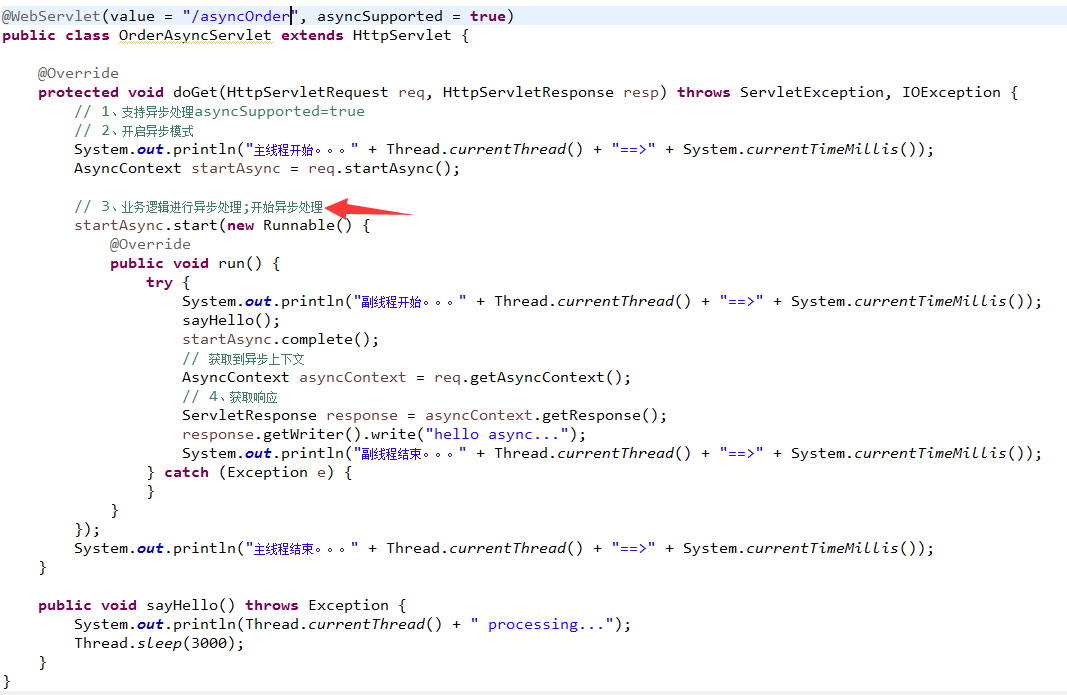
重启tomcat，得到如下结果



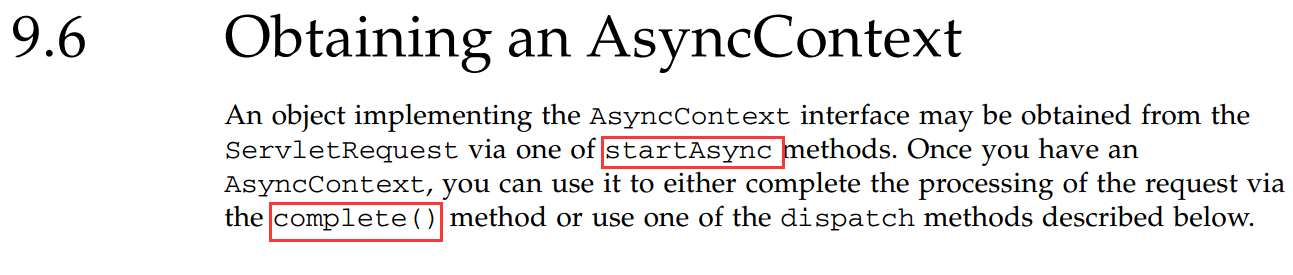
从头到尾都是由同一个线程4进行处理的，同一线程处理

很明显，线程从头执行到尾，会造成资源占用不能释放

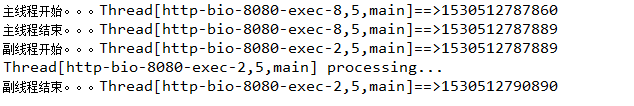
b,异步请求操作步骤：



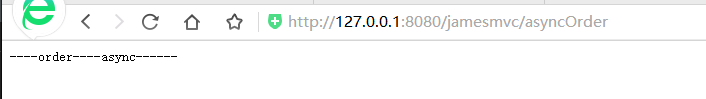
从servlet3.0文档的9.6章节也可以看到，要声明的内容



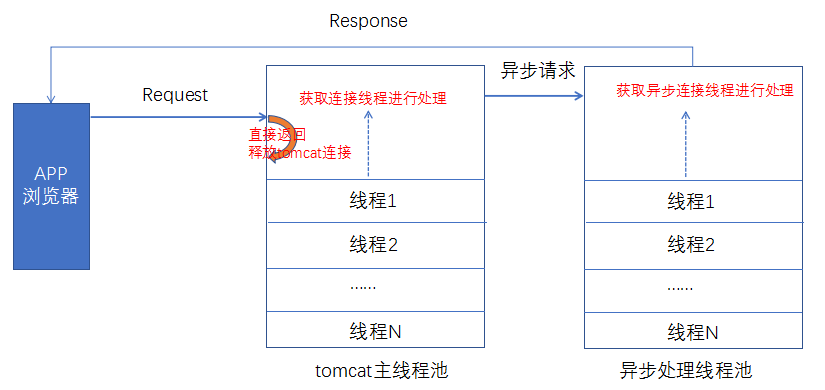
重启tomcat,查看运行结果如下，主线程与副线程分别为不同的线程，主线程从开始到结束不等待副线程就返回了：



页面返回结果如下：等待3S后才返回，但线程资源其实早已释放



c,异步请求原理



当然怎么样定义servlet的异步处理线程池，不多讲，springmvc已集成了异步处理线程池

9，springmvc的异步请求

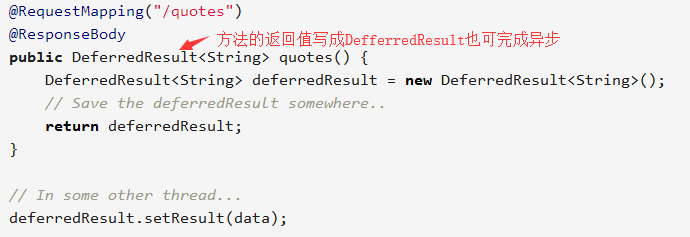
我们可以打官网

<https://docs.spring.io/spring/docs/5.0.2.RELEASE/spring-framework-reference/web.html#mvc-servlet-context-hierarchy>

第1.7.1章节，讲述得很清晰

springmvc异步机制是基于servlet3来做的封装处理，通过这两种返回值都可以完成异步

官网例子如下：

或

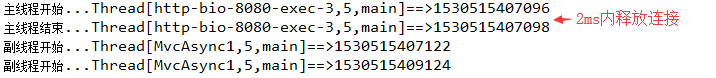
我们接下来实现一下：

操作步骤：

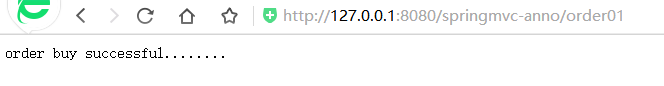
a,新建AsyncOrderController测试类



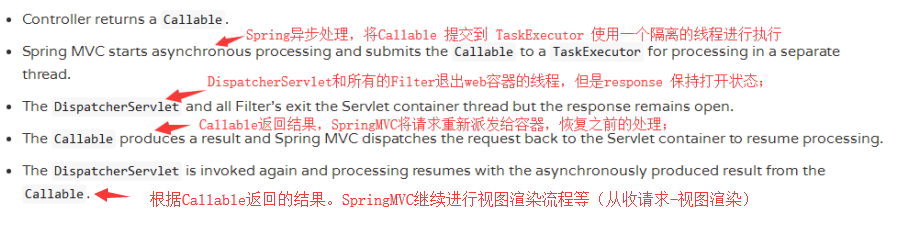
控制台打印结果如下：



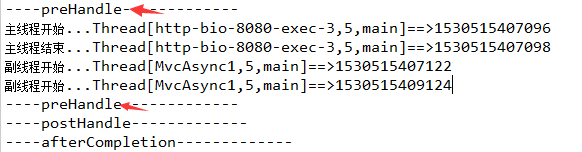
页面运行结果如下：



Callable的原理是什么呢？还是一样，请打开官网哈…………第1.7.1章节

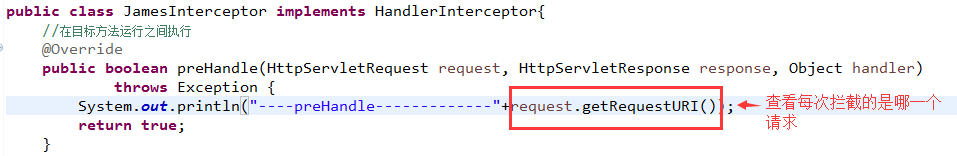


思考：控制台打印结果为什么会两次进入拦截器preHandle呢？

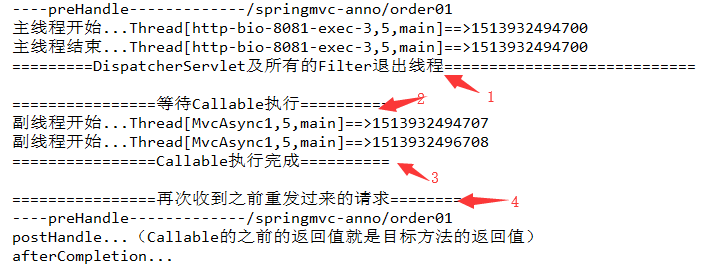


很明显可知：请求进入时拦截了一次，将Callable返回结果时，将请求重新派发给容器时又拦截了一次，所以进了两次拦截；

如何验证？只要在拦截器打印的地方加上getRequestURI()便知晓。



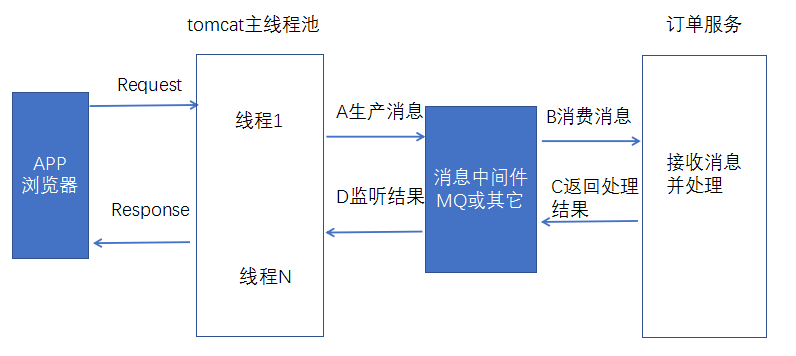
重启tomcat，并测试结果如下：分析见箭头的中文描述：



10，springmvc异步请求及返回实战

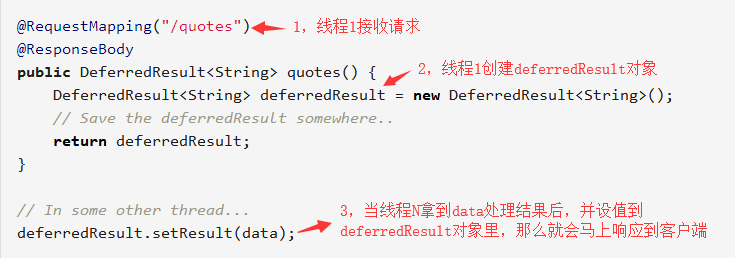
以上只是原理，但在开发的过程并不是像Callable简单

比如现在我们有以下需求：



需求描述：以创建订单为例，tomcat启动线程1来完成一个请求，但实际上是订单服务才能创建订单，那么tomcat线程应该把请求转发给订单服务，使用消息中间件来处理，订单服务把处理结果也放到消息中间件，由tomcat的线程N拿到结果后，响应给客户端。

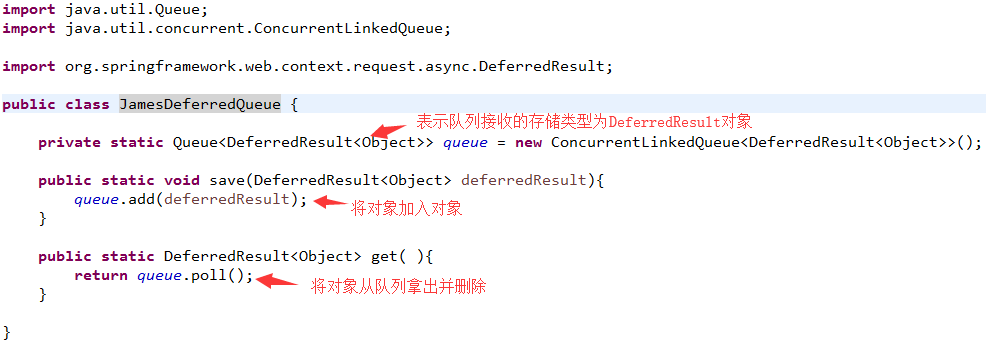
我们打开官网第1.7.1章节



操作步骤：

很明显我们不会用到MQ等消息中间件，写一个队列为模拟消息中间件

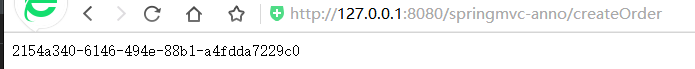
a,新建JamesDefferdQueue消息队列类

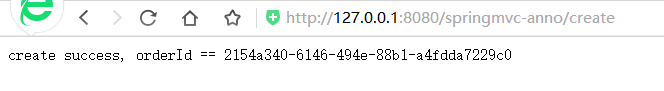


b,在AsyncOrderController新增两方法（其实就是两个线程，线程1和线程N）



测试结果为：





通过create (tomcat线程N处理的订单结果)，异步返回给createOrder(tomcat线程1),两结果一致，异步返回了结果。