Spring 中的懒加载与事务 – 排坑记录

silvergoose 发布于2年前

案例描述

本文主要描述了开发中常见的几个与spring懒加载和事务相关的案例，主要描述常见的使用场景，以及如何规避他们，给出具体的代码。

1. 在新的线程中，访问某个持久化对象的懒加载属性。

2. 在quartz定时任务中，访问某个持久化对象的懒加载属性。

3. 在dubbo，motan一类rpc框架中，远程调用时服务端session关闭的问题。

上面三个案例，其实核心都是一个问题，就是牵扯到spring对事务的管理，而懒加载这个技术，只是比较容易体现出事务出错的一个实践，主要用它来引发问题，进而对问题进行思考。

前期准备

为了能直观的暴露出第一个案例的问题，我新建了一个项目，采用传统的mvc分层，一个student.Java实体类，一个studentDao.java持久层，一个studentService.java业务层，一个studentController控制层。

```
@Entity
@Table(name = "student")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;
    private String name;

    getter..setter..
}
```

持久层使用springdata，框架自动扩展出CURD方法

```
public interface StudentDao extends JpaRepository<Student, Integer>{
}
```

service层，先给出普通的调用方法。用于错误演示。

```
@Service
public class StudentService {

    @Autowired
    StudentDao studentDao;

    public void testNormalGetOne(){
        Student student = studentDao.getOne(1);
        System.out.println(student.getName());
    }
}
```

注意：getOne和findOne都是springdata提供的根据id查找单个实体的方法，区别是前者是懒加载，后者是立即加载。我们使用getOne来进行懒加载的实验，就不用大费周章去写懒加载属性，设置多个实体类了。

controller层，不是简简单单的调用，而是在新的线程中调用。使用controller层来代替单元测试（实际项目中，通常使用controller调用service，然后在浏览器或者http工具中调用触发，较为方便）

```
@RequestMapping("/testNormalGetOne")
    @ResponseBody
    public String testNormalGetOne() {
        new Thread(new Runnable() {
            @Override
            public void run() {
                studentService.testNormalGetOne();
            }
        }).start();
        return "testNormalGetOne";
    }
```

启动项目后，访问localhost:8080/testNormalGetOne报错如下：

```
Exception in thread "Thread-6" org.hibernate.LazyInitializationException: cou
    at org.hibernate.proxy.AbstractLazyInitializer.initialize(AbstractLazyIni
    at org.hibernate.proxy.AbstractLazyInitializer.getImplementation(Abstract
    at org.hibernate.proxy.pojo.javassist.JavassistLazyInitializer.invoke(Jav
    at com.example.transaction.entity.Student_$$_jvste17_0.getName(Student_$$_
    at com.example.transaction.service.StudentService.testNormalGetOne(Studen
    at com.example.transaction.service.StudentService$$FastClassBySpringCGLIB$
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:204
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedIntercept
    at com.example.transaction.service.StudentService$$EnhancerBySpringCGLIB$$
    at com.example.transaction.controller.StudentController$1.run(StudentCont
    at java.lang.Thread.run(Thread.java:745)
```

问题分析

no session说明了什么？

道理很简单，因为spring的session是和线程绑定的，在整个model->dao->service->controller的调用链中，这种事务和线程绑定的机制非常契合。而我们出现的问题正式由于新开启了一个线程，这个线程与调用链的线程不是同一个。

问题解决

我们先使用一种不太优雅的方式解决这个问题。在新的线程中，手动打开session。

```
lic void testNormalGetOne() {
      EntityManagerFactory entityManagerFactory = ApplicationContextProvider.get
      EntityManager entityManager = entityManagerFactory.createEntityManager();
      EntityManagerHolder entityManagerHolder = new EntityManagerHolder(entityMa
      TransactionSynchronizationManager.bindResource(entityManagerFactory, entit
      Student student = studentDao.getOne(1);
      System.out.println(student.getName());
      TransactionSynchronizationManager.unbindResource(entityManagerFactory);
      EntityManagerFactoryUtils.closeEntityManager(entityManager);
```

由于我们使用了JPA，所以事务是由EntityManagerFactory这个工厂类生成的EntityManager来管理的。TransactionSynchronizationManager.bindResource(entityManagerFactory, entityManagerHolder);这个方法使用事务管理器绑定session。

而ApplicationContextProvider这个工具类是用来获取spring容器中的EntityManagerFactory的，为什么不用注入的方式，下文讲解。它的代码如下：

```
public class ApplicationContextProvider implements ApplicationContextAware {

    private static ApplicationContext context = null;

    public static ApplicationContext getApplicationContext() {
        return context;
    }

    @Override
    public void setApplicationContext(ApplicationContext ac) throws BeansExce
        context = ac;
    }
}
```

问题暂时得到了解决。

问题再思考

我们一般情况下使用懒加载属性，为什么没有出现no session的问题呢？相信大家都知道@Transactional这个注解，他会帮我们进行事务包裹，当然也会绑定session；以及大家熟知的hiberbate中的OpenSessionInterceptor和OpenSessionInViewFilter以及jpa中的OpenEntityManagerInViewInterceptor都是在没有session的情况下，打开session的过滤器。这种方法开始前依赖事务开启，方法结束后回收资源的操作，非常适合用过滤器拦截器处理，后续的两个未讲解的案例，其实都是使用了特殊的过滤器。

看一下官方文档如何描述这个jpa中的过滤器的：

29.3.4 Open EntityManager in View

If you are running a web application, Spring Boot will by default register OpenEntityManagerInViewInterceptor to apply the “Open EntityManager in View” pattern, i.e. to allow for lazy loading in web views. If you don’t want this behavior you should set spring.jpa.open-in-view to false in your application.properties.

我们尝试着关闭这个过滤器：

配置application.properties/application.yml文件

```
spring.jpa.open-in-view=false
```

再使用正常的方式访问懒加载属性（而不是在一个新的线程中）：

```
@RequestMapping("/testNormalGetOne")
    @ResponseBody
    public String testNormalGetOne() {
//        new Thread(new Runnable() {
//            @Override
//            public void run() {
                studentService.testNormalGetOne();
//            }
//        }).start();
        return "testNormalGetOne";
    }
```

报错如下：

```
{"timestamp":1498194914012,"status":500,"error":"Internal Server Error","excep
```

是的，我们使用spring的controller作为单元测试时，以及我们平时在直接使用jpa的懒加载属性时没有太关注这个jpa的特性，因为springboot帮我们默认开启了这个过滤器。这也解释了，为什么在新的线程中，定时任务线程中，rpc远程调用时session没有打开的原因，因为这些流程没有经过springboot的web调用链。

另外两个实战案例

上文已经阐释了，为什么quartz定时任务中访问懒加载属性，rpc框架服务端访问懒加载属性（注意不是客户端，客户端访问懒加载属性那是一种作死的行为，因为是代理对象）为出现问题。我们仿照spring打开session的思路（这取决于你使用hibernate还是jpa，抑或是mybatis），来编写我们的过滤器。

quartz中打开session：

使用quartz提供的JobListenerSupport支持，编写一个任务过滤器，用于在每次任务执行时打开session

```
public class OpenEntityManagerJobListener extends JobListenerSupport implemen

    @Override
    public String getName() {
        return "OpenEntityManagerJobListener";
    }

    EntityManagerFactory entityManagerFactory;

    @Override
    public void jobToBeExecuted(JobExecutionContext context) {
        entityManagerFactory = applicationContext.getBean(EntityManagerFactor
        EntityManager entityManager = entityManagerFactory.createEntityManage
        EntityManagerHolder emHolder = new EntityManagerHolder(entityManager)
        TransactionSynchronizationManager.bindResource(entityManagerFactory,
    }

    @Override
    public void jobWasExecuted(JobExecutionContext context, JobExecutionExcep
        EntityManagerHolder emHolder = (EntityManagerHolder) TransactionSynch
        EntityManagerFactoryUtils.closeEntityManager(emHolder.getEntityManage
    }


    ApplicationContext applicationContext;

    @Override
    public void setApplicationContext(ApplicationContext applicationContext)
        this.applicationContext = applicationContext;
        if(this.applicationContext ==null) throw new RuntimeException("applic
    }
}
```

配置调度工厂：

```
//调度工厂
    @Bean
    public SchedulerFactoryBean schedulerFactoryBean() {
        SchedulerFactoryBean factoryBean = new SchedulerFactoryBean();
        factoryBean.setTriggers(triggerFactoryBeans().getObject());
        factoryBean.setGlobalJobListeners(openEntityManagerJobListener());
        return factoryBean;
    }
```

也可以参考我的另一篇描述更为细致的文章，那是我还是刚刚参加工作，可能有些许疏漏之处，不过参考是够了。传送门： 解决Quartz定时器中查询懒加载数据no session的问题

Motan（我现在使用的rpc框架）服务端打开session 利用了motan对spi扩展的支持，编写了一个Filter，主要参考了motan的spi过滤器写法和springdata打开session/entityManager的思路。

```java
@SpiMeta(name = "openjpasession")
@Activation(sequence = 100)
public class OpenEntityManagerInMotanFilter implements Filter {
    private Logger logger = LoggerFactory.getLogger(OpenEntityManagerInMotanF

    /**
     * Default EntityManagerFactory bean name: "entityManagerFactory".
     * Only applies when no "persistenceUnitName" param has been specified.
     *
     * @see #setEntityManagerFactoryBeanName
     * @see #setPersistenceUnitName
     */
    public static final String DEFAULT_ENTITY_MANAGER_FACTORY_BEAN_NAME = "er


    private String entityManagerFactoryBeanName;

    private String persistenceUnitName;

    private volatile EntityManagerFactory entityManagerFactory;


    /**
     * Set the bean name of the EntityManagerFactory to fetch from Spring's
     * root application context.
     * <p>Default is "entityManagerFactory". Note that this default only app
     * when no "persistenceUnitName" param has been specified.
     *
     * @see #setPersistenceUnitName
     * @see #DEFAULT_ENTITY_MANAGER_FACTORY_BEAN_NAME
     */
    public void setEntityManagerFactoryBeanName(String entityManagerFactoryBe
        this.entityManagerFactoryBeanName = entityManagerFactoryBeanName;
    }

    /**
     * Return the bean name of the EntityManagerFactory to fetch from Spring
     * root application context.
     */
    protected String getEntityManagerFactoryBeanName() {
        return this.entityManagerFactoryBeanName;
    }

    /**
     * Set the name of the persistence unit to access the EntityManagerFacto
     * <p>This is an alternative to specifying the EntityManagerFactory by b
     * resolving it by its persistence unit name instead. If no bean name an
     * unit name have been specified, we'll check whether a bean exists for
     * bean name "entityManagerFactory"; if not, a default EntityManagerFact
```

```
     * be retrieved through finding a single unique bean of type EntityManag
     *
     * @see #setEntityManagerFactoryBeanName
     * @see #DEFAULT_ENTITY_MANAGER_FACTORY_BEAN_NAME
     */
    public void setPersistenceUnitName(String persistenceUnitName) {
        this.persistenceUnitName = persistenceUnitName;
    }

    /**
     * Return the name of the persistence unit to access the EntityManagerFa
     */
    protected String getPersistenceUnitName() {
        return this.persistenceUnitName;
    }

    /**
     * Look up the EntityManagerFactory that this filter should use.
     * <p>The default implementation looks for a bean with the specified nam
     * in Spring's root application context.
     *
     * @return the EntityManagerFactory to use
     * @see #getEntityManagerFactoryBeanName
     */
    protected EntityManagerFactory lookupEntityManagerFactory() {

        String emfBeanName = getEntityManagerFactoryBeanName();
        String puName = getPersistenceUnitName();
        if (StringUtils.hasLength(emfBeanName)) {
            return ApplicationContextProvider.getApplicationContext().getBean
        } else if (!StringUtils.hasLength(puName) && ApplicationContextProvi
            return ApplicationContextProvider.getApplicationContext().getBean
        } else {
            // Includes fallback search for single EntityManagerFactory bean
            return EntityManagerFactoryUtils.findEntityManagerFactory(Applica
        }
    }

    /**
     * Create a JPA EntityManager to be bound to a request.
     * <p>Can be overridden in subclasses.
     *
     * @param emf the EntityManagerFactory to use
     * @see javax.persistence.EntityManagerFactory#createEntityManager()
     */
    protected EntityManager createEntityManager(EntityManagerFactory emf) {
        return emf.createEntityManager();
    }
```

```
@Override
public Response filter(Caller<?> caller, Request request) {
    if (!(caller instanceof Provider)) {
        return caller.call(request);
    }

    EntityManagerFactory emf = null;
    try {
        emf = lookupEntityManagerFactory();
    } catch (Exception e) {
        logger.error(e.getMessage(), e);
    }

    //可能没有启用openjpa
    if (emf == null) {
        return caller.call(request);
    }

    try {
        //如果没有绑定，绑定到当前线程
        if (TransactionSynchronizationManager.getResource(emf) == null)
            EntityManager em = createEntityManager(emf);
            EntityManagerHolder emHolder = new EntityManagerHolder(em);
            TransactionSynchronizationManager.bindResource(emf, emHolder
        }
    } catch (Exception e) {
        logger.error(e.getLocalizedMessage(), e);
    }
    try {
        return caller.call(request);
    } finally {
        //解除绑定
        closeManager(emf);
    }
}

/**
 * 关闭 emf
 *
 * @param emf
 */
private void closeManager(EntityManagerFactory emf) {
    if (emf == null || TransactionSynchronizationManager.getResource(emf
        return;
    }
    EntityManagerHolder emHolder = null;
    try {
        emHolder = (EntityManagerHolder) TransactionSynchronizationManag
    } catch (IllegalStateException e) {
```

```
                logger.error(e.getLocalizedMessage(), e);
            }
            try {
                if (emHolder != null) {
                    EntityManagerFactoryUtils.closeEntityManager(emHolder.getEnt
                }
            } catch (Exception e) {
                logger.error(e.getLocalizedMessage(), e);
            }
        }
    }
```

总结

springboot中的事务管理做的永远比我们想的多，事务管理器的使用场景，@Transactional究竟起了哪些作用，以及spring-data这个对DDD最佳的阐释，以及mybatis一类的非j2ee规范在微服务的地位中是否高于jpa，各个层次之间的实体传输，消息传递...都是值得思考的。

查看原文： Spring 中的懒加载与事务 – 排坑记录

上一篇 下一篇

基于SpringBoot2.x、SpringCloud和SpringCloudAlibaba并采用前后端分离的企业级微服务多租户、多系统的...

相关主题

MyBatis入门实例：整合Spring MVC与MyBatis开发问答网站

69道Spring面试题和答案

Spring知识点提炼

Spring知识点提炼

Spring框架详解

一线互联网企业Spring面试专题解析，你掌握了吗?

Spring Mybatis详解

JavaEE – JPA（6）：ORM的核心注解 – 基础类型以及嵌套类型

silvergoose

被 0人关注，获得了317个喜欢

✚关注

find . ! -type d -exec chmod -x {}\; 清除所有文件的执行权限 不影响目录