51_精选面试题：Spring 同步和异步事件的使用场景

1、开篇

上节课带大家从打包、发布、启动服务几个步骤将登录功能演示了一遍。通过查看日志的方式，了解了代码从登陆、发送事件消息、接受事件消息，到最后更新登陆次数的执行过程。本节课我们分析 Spring 同步和异步事件的使用场景。今天课程的内容包括以下几个部分：

- 从一个例子切入 Spring 消息处理机制
- Spring 消息的同步处理
- Spring 消息的异步处理

2、从一个例子介绍 Spring 消息处理机制

Spring 事件的监听机制可以理解为是一种观察者模式，有数据发布者（事件源）和数据接受者（监听器）；在 Java 中，事件对象都是继承 java.util.EventObject 对象，事件监听器都是 java.util.EventListener 实例。

其中，java.util.EventObject 是事件状态对象的基类，它封装了事件源对象以及和事件相关的信息。所有 java 的事件类都需要继承该类。

而 java.util.EventListener 是一个接口，所有事件监听器都需要实现该接口。事件监听器注册在事件源上，当事件源的属性或状态改变的时候，调用相应监听器内的回调方法。

Source 作为事件源不需要实现或继承任何接口或类，它是事件最初发生的地方。因为事件源需要注册事件监听器，所以事件源内需要存放事件监听器的容器。

来看一个例子，我们需要将 User 信息保存到数据库中，然后发送一条 User 的消息，然后由对应的监听器来处理。

如图 1 所示，UserEvent 作为需要发送的消息继承与 ApplicationEvent，同时提供了 getData 和 setData 的方法来存放需要处理的数据。

```
public class UserEvent<T> extends ApplicationEvent {
    private T data;
    public UserEvent(T source) {
        super(source);
        this.data =source;
    }
    public T getData() {
        return this.data;
    }
    public void setData(final T data) {
        this.data = data;
    }
}
```

图 1 UserEvent

接下来就是发送 UserEvent 消息的 sendInsertUser 方法，如图 2 所示，该方法创建 User 的实例以后填写对应的属性，然后通过 ApplicationEventPublisher 类型的 publisher 变量执行 publishEvent 方法进行发送消息。

```
@Autowired
private ApplicationEventPublisher publisher;
private void sendInsertUser() {
    User user = new User();
    user.setId(1);
    user.setUsername("李四");
    UserEvent<User> userEvent = new UserEvent<>(user);
    // 发布事件
    publisher.publishEvent(userEvent);

}
```

图 2 sendInsertUser

如图 3 所示，有了发送消息的代码就有监听消息的代码，insertUserListener 方法在@EventListener 的修饰下就成为了消息监听器。它会接受 UserEvent 消息，并且对消息进行处理最后把处理的结果 message 记录到日志中。

```java
@EventListener
public void insertUserListener(UserEvent<User> userEvent) {
    User data = userEvent.getData();
    String message = String.format("get user message: %s", JSON.toJSONString(data));
    log.info(message);
}
```

图 3 insertUserListener

接下来就测试一下代码，如图 4 所示，在测试方法中先执行 insertUsers 方法用来往 User 表中插入数据，然后再执行 sendInsertUser 方法发送消息。

```java
public BackResult testEvent() {
    BackResult backResult = new BackResult();
    try {
        // 向user 表插入数据
        insertUsers();
        // 添加user插入对象操作
        sendInsertUser();
        backResult.setSuccess(true);
        backResult.setMessage("操作成功!");
    } catch (Exception e) {
        backResult.setMessage(e.getMessage());
        backResult.setSuccess(false);
    }
    return backResult;
}
```

图 4 testEvent

3、Spring 消息的同步处理

假设上面的例子中两个方法 insertUsers 和 sendInsertUser 需要同步执行，也就是 insertUsers 方法会先执行，完成数据库的提交以后再执行 sendInsertUser 方法发送消息。如果 insertUsers 数据库提交一直没有返回成功的结果，sendInsertUser 就需要一直等待。也就是说 insertUsers 会阻塞 sendInsertUser 方法，这就是我们常说的同步执行。为了做到同步执行就需要将监听器加入到主线程的事务中，让这两个方法顺序执行。

如图 5 所示，首先在 insertUsers 方法上面打上@Transcational 的注释，标注它是一个事物操作。

```
@Transactional(rollbackFor = Exception.class)
public void insertUsers() {
    //insert data into database
}
```

图 5 insertUsers

然后修改 insertUserListener 方法，在上面加上@TranscationalEventListener 的注释，并且通过 phase 属性标注 TransactionPhase.AFTER_COMMIT，意思是在事物提交以后在执行方法内容。也就是在 insert user 数据库事物提交以后，再处理监听到的 UserEvent 消息，从而保证事务的一致性。

```
@TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT,fallbackExecution = true)
public void insertUserListener(UserEvent<User> userEvent) {
    User data = userEvent.getData();
    String message = String.format("get user message: %s", JSON.toJSONString(data));
    log.info(message);
}
```

图 6 insertUserLinster

从这里例子可以发现如果执行的操作和发送消息以后对应的操作业务上面有高的关联性，可以作为一个事务的两个操作来处理，要么一起完成要么都不完成的话，那么就需要进行同步处理。常见的业务场景有，下单操作和通知扣减库存，下单以后商品被售出，库存随之扣减，如果下单失败扣减库存的操作也不能执行。

4、Spring 消息的异步处理

上面聊了同步处理的业务场景，还是这个例子我们接着聊。假设 insertUsers 和 sendInsertUser 这两个操作的业务关联性并不大，那么如何处理。例如：insertUsers 之后只是发一个通知给用户，说数据入库了。就算是入库不成功，消息发送也不用撤回，毕竟没有然后的业务损失。

这种场景业务就可以用异步处理来做，也就是说 sendInsertUser 不用等待 insertUsers 方法数据库条件完成就可以处理发送的消息了。换句话说这两个方法不是串行执行的而是并行执行的，两个操作的执行是互相不干扰的。

如图 7 所示，异步处理只需要在原方法上加上@Async 的注释，同时需要注意的是，需要使用@EnableAsync 开启 Spring 异步模式。

```
@Async
@TransactionalEventListener(phase = TransactionPhase.AFTER_COMMIT,fallbackExecution = true)
public void insertUserListener(UserEvent<User> userEvent) {
    User data = userEvent.getData();
    String message = String.format("get user message: %s", JSON.toJSONString(data));
    log.info(message);
    // 后续操作；继续处理
}
```

图 7 异步执行 insertUserListener

5、总结

本节课主要讲解 Spring 消息机制的同步和异步事件处理，通过一个例子介绍了 Spring 的 消息处理机制，并且在例子上进行 Spring 消息同步和异步处理的扩展。针对不同的业务场景描述了两种处理方式的不同。

下节课会梳理互联网教育系统的优惠券管理模块的业务流程。下期见，拜拜。