链滴

使用 Spring StateMachine 框架实现状态机

作者：yk

原文链接：https://ld246.com/article/1502953418691

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

Spring StateMachine框架可能对于大部分使用Spring的开发者来说还比较生僻，该框架目前差不多才刚满一岁多。它的主要功能是帮助开发者简化状态机的开发过程，让状态机结构更加层次化。前几刚刚发布了它的第三个Release版本1.2.0，其中增加了对Spring Boot的自动化配置，既然一直在写Sping Boot的教程，所以干脆就将该内容也纳入进来吧，希望对有需求的小伙伴有一定的帮助。

快速入门

依照之前的风格，我们通过一个简单的示例来对Spring StateMachine有一个初步的认识。假设我们要实现一个订单的相关流程，其中包括订单创建、订单支付、订单收货三个动作。

下面我们来详细的介绍整个实现过程：

● 创建一个Spring Boot的基础工程，并在 pom.xml中加入spring-statemachine-core的依赖，具如下：

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.7.RELEASE</version>
    <relativePath/>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.statemachine</groupId>
        <artifactId>spring-statemachine-core</artifactId>
        <version>1.2.0.RELEASE</version>
    </dependency>
</dependencies>
```

● 根据上面所述的订单需求场景定义状态和事件枚举，具体如下：

```
public enum States {
    UNPAID,              // 待支付
    WAITING_FOR_RECEIVE,    // 待收货
    DONE                 // 结束
}

public enum Events {
    PAY,       // 支付
    RECEIVE     // 收货
}
```

其中共有三个状态（待支付、待收货、结束）以及两个引起状态迁移的事件（支付、收货），其中支事件PAY会触发状态从待支付UNPAID状态到待收货WAITING_FOR_RECEIVE状态的迁移，而收货事件ECEIVE会触发状态从待收货WAITING_FOR_RECEIVE状态到结束DONE状态的迁移。

● 创建状态机配置类：

```
@Configuration
@EnableStateMachine
```

```
public class StateMachineConfig extends EnumStateMachineConfigurerAdapter<States, Event
> {

    private Logger logger = LoggerFactory.getLogger(getClass());

    @Override
    public void configure(StateMachineStateConfigurer<States, Events> states)
            throws Exception {
        states
            .withStates()
                .initial(States.UNPAID)
                .states(EnumSet.allOf(States.class));
    }

    @Override
    public void configure(StateMachineTransitionConfigurer<States, Events> transitions)
            throws Exception {
        transitions
            .withExternal()
                .source(States.UNPAID).target(States.WAITING_FOR_RECEIVE)
                .event(Events.PAY)
                .and()
            .withExternal()
                .source(States.WAITING_FOR_RECEIVE).target(States.DONE)
                .event(Events.RECEIVE);
    }

    @Override
    public void configure(StateMachineConfigurationConfigurer<States, Events> config)
            throws Exception {
        config
            .withConfiguration()
                .listener(listener());
    }

    @Bean
    public StateMachineListener<States, Events> listener() {
        return new StateMachineListenerAdapter<States, Events>() {

            @Override
            public void transition(Transition<States, Events> transition) {
                if(transition.getTarget().getId() == States.UNPAID) {
                    logger.info("订单创建，待支付");
                    return;
                }

                if(transition.getSource().getId() == States.UNPAID
                        && transition.getTarget().getId() == States.WAITING_FOR_RECEIVE) {
                    logger.info("用户完成支付，待收货");
                    return;
                }

                if(transition.getSource().getId() == States.WAITING_FOR_RECEIVE
                        && transition.getTarget().getId() == States.DONE) {
```

```
                logger.info("用户已收货，订单完成");
                return;
            }
        }

    };
}

}
```

在该类中定义了较多配置内容，下面对这些内容一一说明：

- @EnableStateMachine注解用来启用Spring StateMachine状态机功能
- configure(StateMachineStateConfigurer states)方法用来初始化当前状态机拥有哪些状态，中initial(States.UNPAID)定义了初始状态为UNPAID，states(EnumSet.allOf(States.class))则指定使用上一步中定义的所有状态作为该状态机的状态定义。

```
@Override
public void configure(StateMachineStateConfigurer<States, Events> states)
        throws Exception {
    // 定义状态机中的状态
    states
        .withStates()
            .initial(States.UNPAID)    // 初始状态
            .states(EnumSet.allOf(States.class));
}
```

- configure(StateMachineTransitionConfigurer transitions)方法用来初始化当前状态机有哪些态迁移动作，其中命名中我们很容易理解每一个迁移动作，都有来源状态source，目标状态target以触发事件event。

```
@Override
public void configure(StateMachineTransitionConfigurer<States, Events> transitions)
        throws Exception {
    transitions
        .withExternal()
            .source(States.UNPAID).target(States.WAITING_FOR_RECEIVE)// 指定状态来源和目标
            .event(Events.PAY)    // 指定触发事件
            .and()
        .withExternal()
            .source(States.WAITING_FOR_RECEIVE).target(States.DONE)
            .event(Events.RECEIVE);
}
```

- configure(StateMachineConfigurationConfigurer config)方法为当前的状态机指定了状态监器，其中listener()则是调用了下一个内容创建的监听器实例，用来处理各个各个发生的状态迁移事件。

```
@Override
public void configure(StateMachineConfigurationConfigurer<States, Events> config)
        throws Exception {
    config
        .withConfiguration()
            .listener(listener());    // 指定状态机的处理监听器
}
```

- StateMachineListener listener()方法用来创建StateMachineListener状态监听器的实例，在

实例中会定义具体的状态迁移处理逻辑，上面的实现中只是做了一些输出，实际业务场景会会有更负的逻辑，所以通常情况下，我们可以将该实例的定义放到独立的类定义中，并用注入的方式加载进来。

- 创建应用主类来完成整个流程：

```
@SpringBootApplication
public class Application implements CommandLineRunner {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Autowired
    private StateMachine<States, Events> stateMachine;

    @Override
    public void run(String... args) throws Exception {
        stateMachine.start();
        stateMachine.sendEvent(Events.PAY);
        stateMachine.sendEvent(Events.RECEIVE);
    }

}
```

在run函数中，我们定义了整个流程的处理过程，其中start()就是创建这个订单流程，根据之前的定，该订单会处于待支付状态，然后通过调用sendEvent(Events.PAY)执行支付操作，最后通过掉用senEvent(Events.RECEIVE)来完成收货操作。在运行了上述程序之后，我们可以在控制台中获得类似下的输出内容：

```
INFO 2312 --- [           main] eConfig$EnhancerBySpringCGLIB$a05acb3d : 订单创建，待支付
INFO 2312 --- [           main] o.s.s.support.LifecycleObjectSupport     : started org.springframework.statemachine.support.DefaultStateMachineExecutor@1d2290ce
INFO 2312 --- [           main] o.s.s.support.LifecycleObjectSupport     : started DONE UNPAID
AITING_FOR_RECEIVE  / UNPAID / uuid=c65ec0aa-59f9-4ffb-a1eb-88ec902369b2 / id=null
INFO 2312 --- [           main] eConfig$EnhancerBySpringCGLIB$a05acb3d : 用户完成支付，待货
INFO 2312 --- [           main] eConfig$EnhancerBySpringCGLIB$a05acb3d : 用户已收货，订单成
```

其中包括了状态监听器中对各个状态迁移做出的处理。

通过上面的例子，我们可以对如何使用Spring StateMachine做如下小结：

- 定义状态和事件枚举
- 为状态机定义使用的所有状态以及初始状态
- 为状态机定义状态的迁移动作
- 为状态机指定监听处理器

状态监听器

通过上面的入门示例以及最后的小结，我们可以看到使用Spring StateMachine来实现状态机的时候代码逻辑变得非常简单并且具有层次化。整个状态的调度逻辑主要依靠配置方式的定义，而所有的业逻辑操作都被定义在了状态监听器中，其实状态监听器可以实现的功能远不止上面我们所述的内容，

还有更多的事件捕获，我们可以通过查看StateMachineListener接口来了解它所有的事件定义：

```
public interface StateMachineListener<S,E> {

    void stateChanged(State<S,E> from, State<S,E> to);

    void stateEntered(State<S,E> state);

    void stateExited(State<S,E> state);

    void eventNotAccepted(Message<E> event);

    void transition(Transition<S, E> transition);

    void transitionStarted(Transition<S, E> transition);

    void transitionEnded(Transition<S, E> transition);

    void stateMachineStarted(StateMachine<S, E> stateMachine);

    void stateMachineStopped(StateMachine<S, E> stateMachine);

    void stateMachineError(StateMachine<S, E> stateMachine, Exception exception);

    void extendedStateChanged(Object key, Object value);

    void stateContext(StateContext<S, E> stateContext);

}
```

注解监听器

对于状态监听器，Spring StateMachine还提供了优雅的注解配置实现方式，所有StateMachineListener接口中定义的事件都能通过注解的方式来进行配置实现。比如，我们可以将之前实现的状态监听器注解配置来做进一步的简化：

```
@WithStateMachine
public class EventConfig {

    private Logger logger = LoggerFactory.getLogger(getClass());

    @OnTransition(target = "UNPAID")
    public void create() {
        logger.info("订单创建，待支付");
    }

    @OnTransition(source = "UNPAID", target = "WAITING_FOR_RECEIVE")
    public void pay() {
        logger.info("用户完成支付，待收货");
    }

    @OnTransition(source = "WAITING_FOR_RECEIVE", target = "DONE")
    public void receive() {
        logger.info("用户已收货，订单完成");
```

```
    }
}
```

上述代码实现了与快速入门中定义的listener()方法创建的监听器相同的功能，但是由于通过注解的方配置，省去了原来事件监听器中各种if的判断，使得代码显得更为简洁，拥有了更好的可读性。

本文完整示例：

- 开源中国： http://git.oschina.net/didispace/SpringBoot-Learning/tree/master/Chapter6-1-1
- GitHub： https://github.com/dyc87112/SpringCloud-Learning/tree/master/Chapter6-1-1

作者：程序猿DD

链接：http://www.jianshu.com/p/326bd3ac2bf2

來源：简书

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。