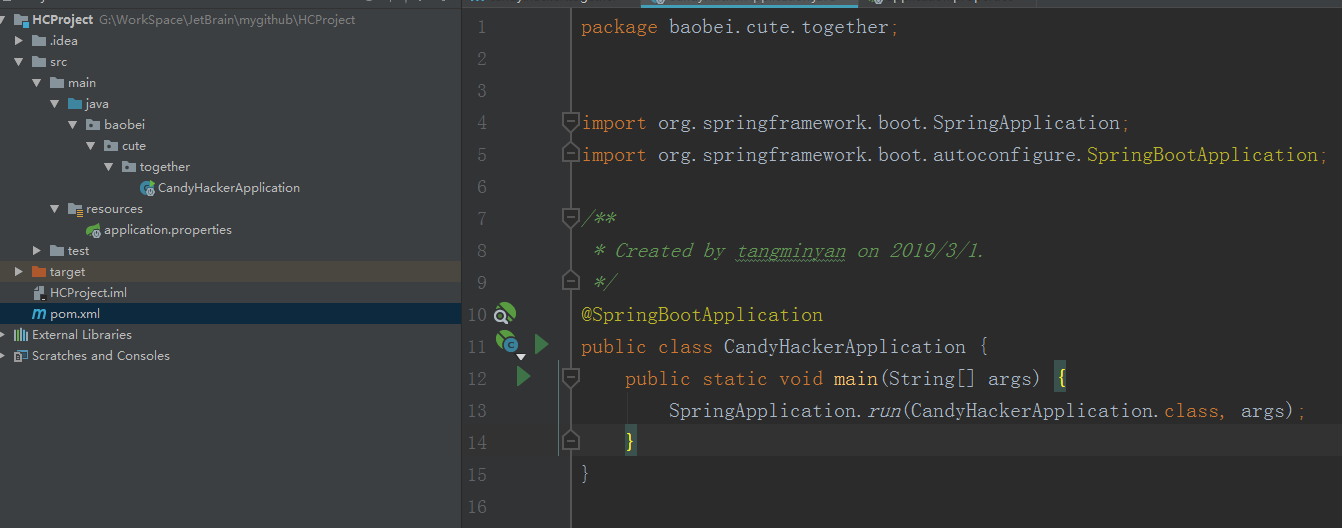
1. 基础搭建
2. 新建Maven项目，导入springboot启动依赖和web依赖



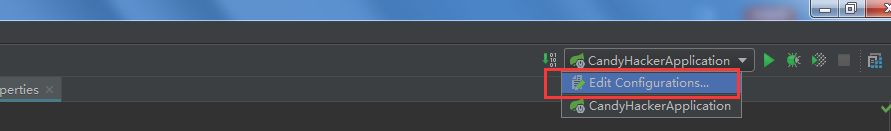
1. 新建启动函数

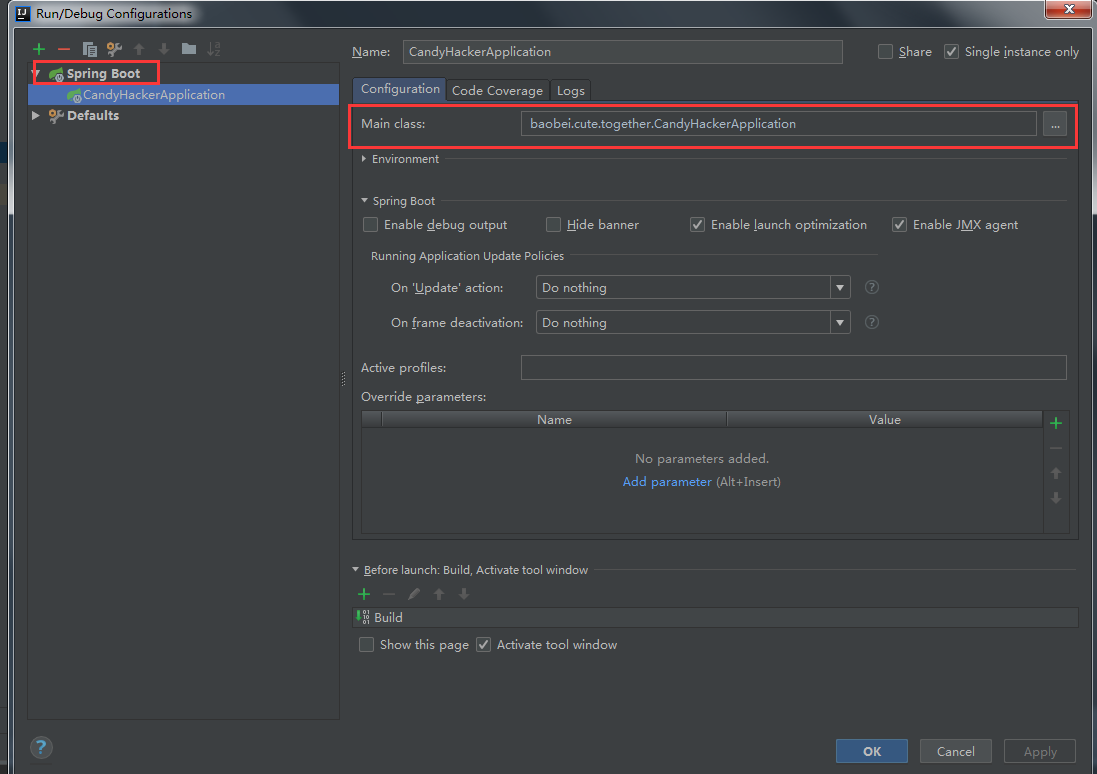


1. resources下新建application.properties配置文件，更改端口号(可不改)



1. 配置启动项，添加springboot





1. 启动
2. 连数据库

1）导入依赖：

<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
</dependency>

1. 设置配置文件

spring.datasource.url=jdbc:mysql://127.0.0.1:3306/better-us  
spring.datasource.username=root  
spring.datasource.password=123

注：

设置hibernate自动建表规则

spring.jpa.hibernate.ddl-auto=update

1. 测试是否成功自动建表，创建测试PO类，为此先引入部分注解的依赖
2. 导入JPA操作数据库的依赖

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>

1. 导入lombok依赖

<dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>1.18.2</version>  
 <scope>provided</scope>  
</dependency>

1. 状态机基础（statemachinedemo包下）
2. pom文件导入依赖

<dependency>  
 <groupId>org.springframework.statemachine</groupId>  
 <artifactId>spring-statemachine-core</artifactId>  
 <version>2.0.2.RELEASE</version>  
</dependency>

1. 新建基本PO类，DAO类
2. 新建enum， 订单状态类 和 操作类

public enum OrderStatus {  
 // 待支付，待发货，待收货，订单结束  
 *WAIT\_PAYMENT*, *WAIT\_DELIVER*, *WAIT\_RECEIVE*, *FINISH*;  
}

public enum OrderStetusChangeEvent {  
 // 支付，发货，确认收货  
 *PAYED*, *DELIVER*, *RECEIVED*}

4）注入状态机的状态，事件的配置。起主要涉及到以下两个类：

1> StateMachineStateConfigurer < S, E> 配置状态集合以及初始状态，泛型参数S代表状态，E代表事件。

2> StateMachineTransitionConfigurer 配置状态流的转移，可以定义状态转换接受的事件。

@Configuration  
@EnableStateMachineFactory  
public class OrderStateMachineConfig extends StateMachineConfigurerAdapter<OrderStatus, OrderStetusChangeEvent> {  
  
 @Override  
 public void configure(StateMachineStateConfigurer<OrderStatus, OrderStetusChangeEvent> states) throws Exception {  
 states  
 .withStates()  
 .initial(OrderStatus.*WAIT\_PAYMENT*)  
 .states(EnumSet.*allOf*(OrderStatus.class));  
 super.configure(states);  
 }  
  
 @Override  
 public void configure(StateMachineTransitionConfigurer<OrderStatus, OrderStetusChangeEvent> transitions) throws Exception {  
 transitions  
 .withExternal()  
 .source(OrderStatus.*WAIT\_PAYMENT*).target(OrderStatus.*WAIT\_DELIVER*)  
 .event(OrderStetusChangeEvent.*PAYED*)  
 .and()  
 .withExternal()  
 .source(OrderStatus.*WAIT\_DELIVER*).target(OrderStatus.*WAIT\_RECEIVE*)  
 .event(OrderStetusChangeEvent.*DELIVER*)  
 .and()  
 .withExternal()  
 .source(OrderStatus.*WAIT\_PAYMENT*).target(OrderStatus.*FINISH*)  
 .event(OrderStetusChangeEvent.*RECEIVED*);  
 }  
}

5）设置监听

@WithStateMachine  
@Slf4j  
public class OrderEventConfig {  
 @OnTransition(target = "UNPAYED")  
 public void create() {  
 *log*.info("待支付");  
 }  
 @OnTransition(source = "UNPAYED", target = "WAITING\_FOR\_RECEIVE")  
 public void pay() {  
 *log*.info("支付完成，待收货");  
 }  
 @OnTransition(source = "WAITING\_FOR\_RECEIVE", target = "DONE")  
 public void receive() {  
 *log*.info("用户已收货，订单完成");  
 }  
}

1. 测试

