微服务构建

2020年7月20日　16:45

构建步骤

1、**微服务提供**支付module模块8001

2、热部署devtools

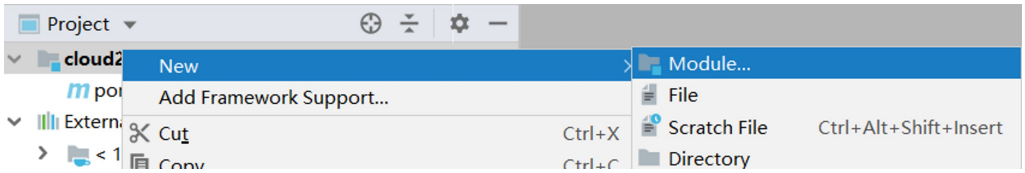
3、**微服务消费者**订单module模块80

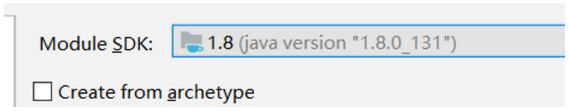
4、工程重构

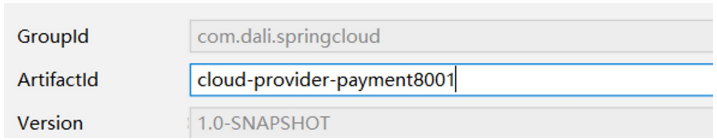
微服务模块

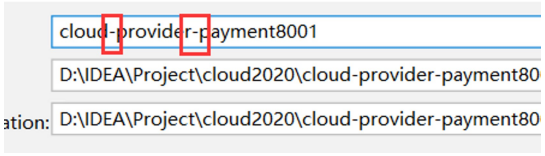
1、建module

cloud-provider-payment8001







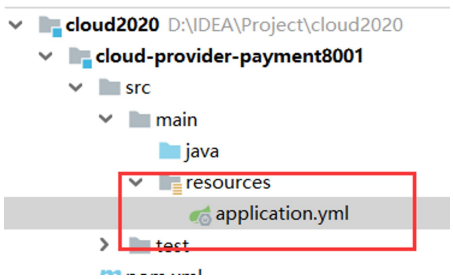


2、改pom

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>cloud2020</artifactId>
        <groupId>com.dali.springcloud</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>cloud-provider-payment8001</artifactId>

    <dependencies>
        <!-- spring boot 2.2.2 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!--监控-->
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>

        <!-- mybatis -->
        <dependency>
            <groupId>com.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
        </dependency>

        <!-- alibaba -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
        </dependency>

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

        <!--jdbc-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>

        <!--热部署-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>

        <!-- lombok -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>

        <!-- test -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

    </dependencies>
</project>
```

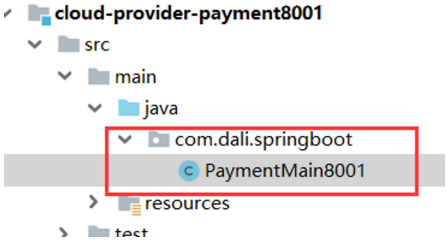
3、写YML




```
server:
  port: 8001
```

```
spring:
  application:
    name: cloud-payment-service
  datasource:
    # 当前数据源操作类型
    type: com.alibaba.druid.pool.DruidDataSource
    # mysql驱动类,userSSL与服务器进行通信时使用SSL（真 / 假），默认值为“假”
    driver-class-name: org.jdbc.cj.mysql.Driver
    url: jdbc:mysql://localhost:3306/db2020?useUnicode=true&characterEncoding=UTF-8&useSSL=false
    username: root
    password: root

  mybatis:
    mapper-locations: classpath:mapper/*.xml
    type-aliases-package: com.dali.springcloud.entities        #放实体类的包
```

4、主启动




```
@SpringBootApplication
public class PaymentMain8001 {
    public static void main(String[] args) {
        SpringApplication.run(PaymentMain8001.class,args);
    }
}
```

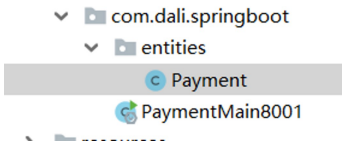
5、业务类

建表-实体-dao-service-controller

建表

```
CREATE TABLE `payment` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `serial` varchar(200) DEFAULT '',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

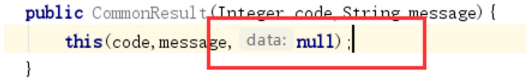
实体

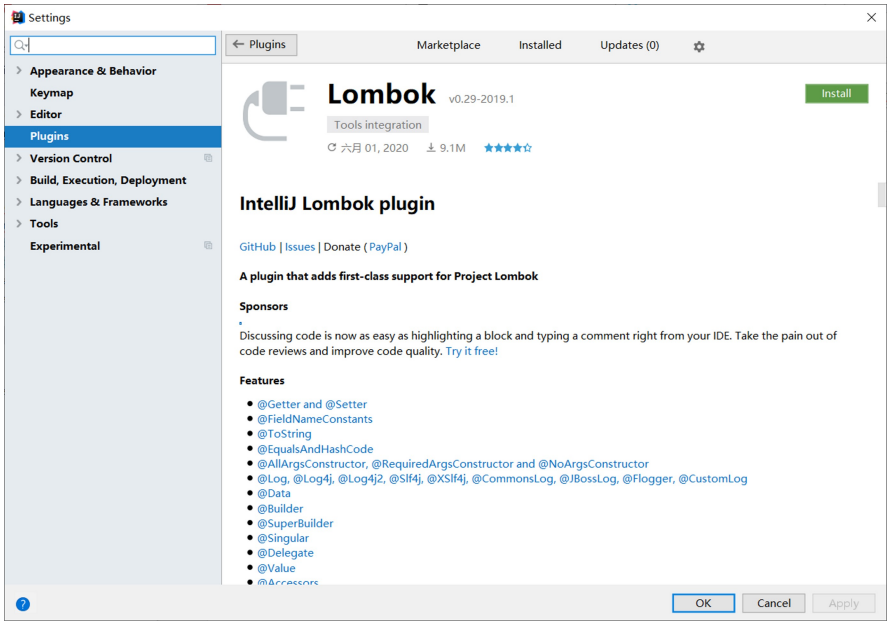


安装lombok插件

作用：使用对象,必须还要写一些getter和setter方法,可能还要写一个构造器、equals方法、或者hash方法.这些方法很冗长而且没有技术含量,lombok帮你做

不导插件，null会报错





@Data注解的作用相当于 @Getter @Setter @RequiredArgsConstructor @ToString @EqualsAndHashCode的合集

@AllArgsConstructor 为该类产生所有参数的构造方法

@NoArgsConstructor 为该类产生无参的构造方法

dao

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.dali.springcloud.dao.PaymentDao">

    <!--    主要是在主键是自增的情况下，添加成功后可以直接使用主键值，其中keyProperty的值是对象的属
    <insert id="create" parameterType="Payment" useGeneratedKeys="true" keyProperty="id">
        insert into payment(serial) values (#{serial});
    </insert>

    <resultMap id="BaseResultMap" type="com.dali.springcloud.entities.Payment">
        <id column="id" property="id" jdbcType="BIGINT" />
        <id column="serial" property="serial" jdbcType="VARCHAR"/>
    </resultMap>

    <select id="getPaymentById" parameterType="Long" resultMap="BaseResultMap">
        select * from payment where id=#{id};
    </select>
</mapper>
```

param注解含义

@Param("orId") String orId

这里orId是前端传过来的参数名，不加@Param("orId") 默认就是找orId，

也可以@Param("orId") String nb，前端传入的orId参数的值就赋值到nb中

service

```java
public interface PaymentService {
    int create(Payment payment);
    Payment getPaymentById(@Param("id") Long id);
}
```

```java
@Service
public class PaymentServiceImpl implements PaymentService{
    @Resource
    private PaymentDao paymentDao;

    public int create(Payment payment) { return paymentDao.create(payment); }

    public Payment getPaymentById(Long id) { return paymentDao.getPaymentById(id); }
}
```

private final Logger log = LoggerFactory.getLogger(getClass());

log.info("backlog={}", new Object[]{backlog});//backlog为Java对象,可重写toString()方法来实现输出具体属性

访问http://localhost:8001/payment/get/1出现以下报错

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Wed Jul 22 10:47:14 CST 2020
There was an unexpected error (type=Internal Server Error, status=500).
Invalid bound statement (not found): com.dali.springcloud.dao.PaymentDao.getPaymentById
org.apache.ibatis.binding.BindingException: Invalid bound statement (not found): com.dali.springcloud.dao.PaymentDao.getPaymentById
at org.apache.ibatis.binding.MapperMethod$SqlCommand.<init>(MapperMethod.java:235)
at org.apache.ibatis.binding.MapperMethod.<init>(MapperMethod.java:53)
at org.apache.ibatis.binding.MapperProxy.lambda$cachedMapperMethod$0(MapperProxy.java:98)

最后发现在mapper.xml里多加了一个空格，yml文件对空格要求比较大

```yaml
 mybatis:
    mapper-locations: classpath:mapper/*.xml
    type-aliases-package: com.dali.springcloud.entities        #放实体类的包
```

空格取消后就可以了

```yaml
mybatis:
  mapper-locations: classpath:mapper/*.xml
  type-aliases-package: com.dali.springcloud.entities        #放实体类的包
```

3、微服务消费者

只有一个controller去调用，使用RestTemplate去建立联系

RestTemplate提供了多种便捷访问HTTP服务的方法

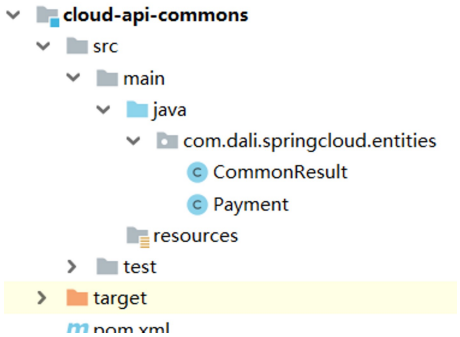
是一种简单便捷的访问restful服务模板类，是spring提供用于访问Rest服务的**客户端模板工具集**

调用端口之间的通信

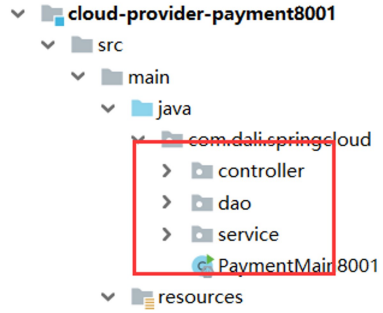
强大的Java辅助类工具箱Hutool

工程重构

目录结构图



删除其他文件的entities包

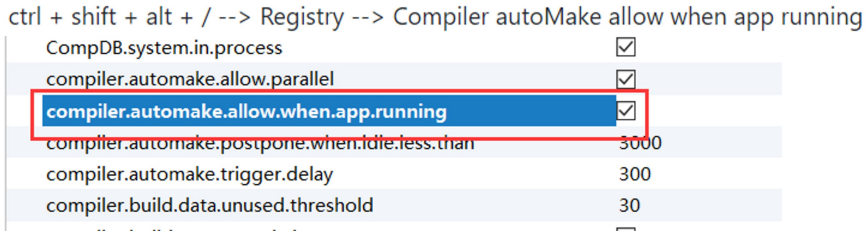


然后在删掉的项目pom文件中添加依赖

```
<!-- 引入自己的API通用包，可以使用payment支付的Entity -->
<dependency>
    <groupId>com.dali.springcloud</groupId>
    <artifactId>cloud-api-commons</artifactId>
    <version>${project.version}</version>
</dependency>
```

最好将maven-clean-install一遍。报错检查配置文件路径

热部署



在pom文件中引入该依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
```