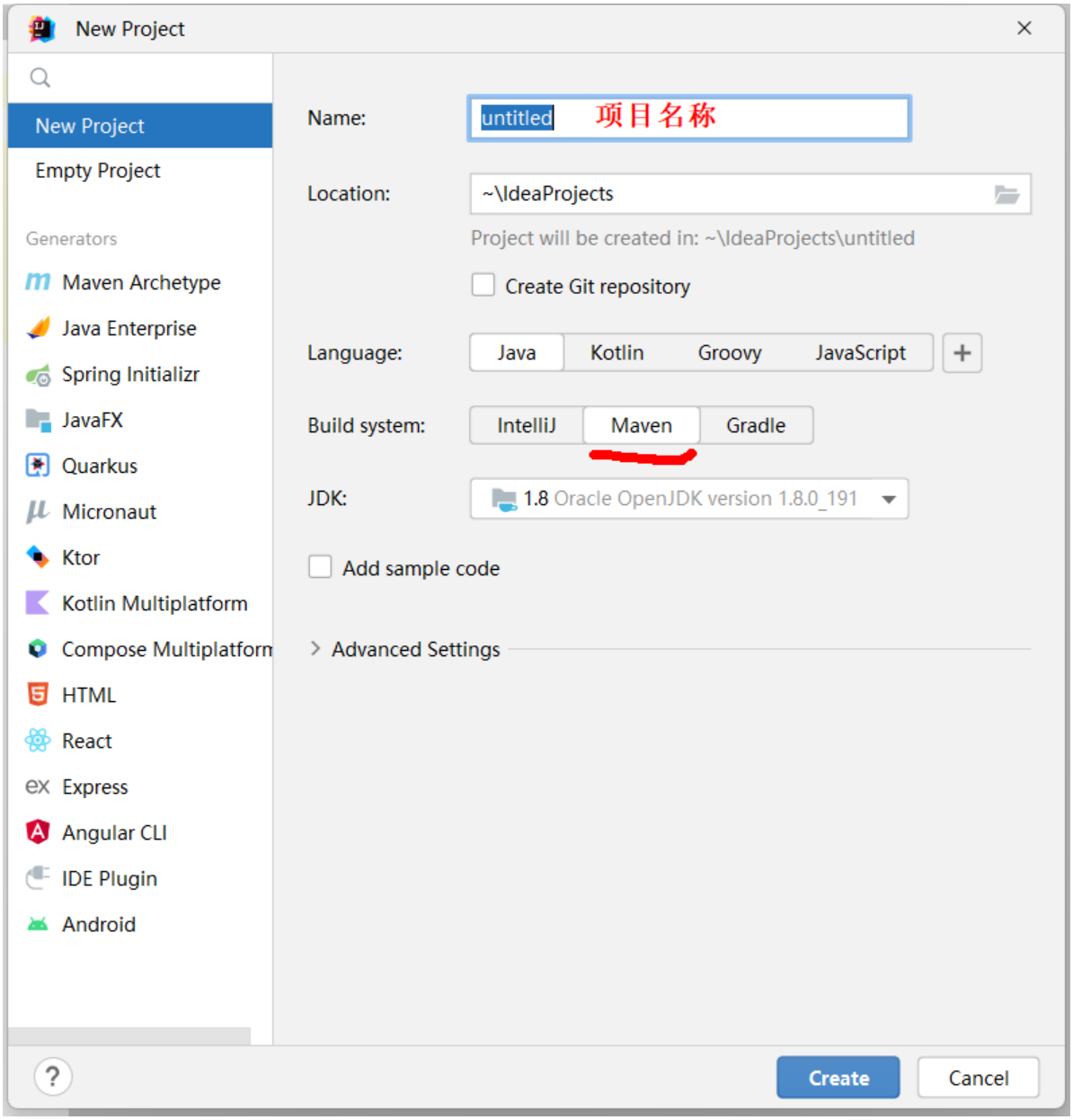
1.创建web应用服务器

1.1新建Maven项目



1.2修改pom.xml，加入我们项目所需要的组件

自properties节点一直到文件结束替换成如下内容

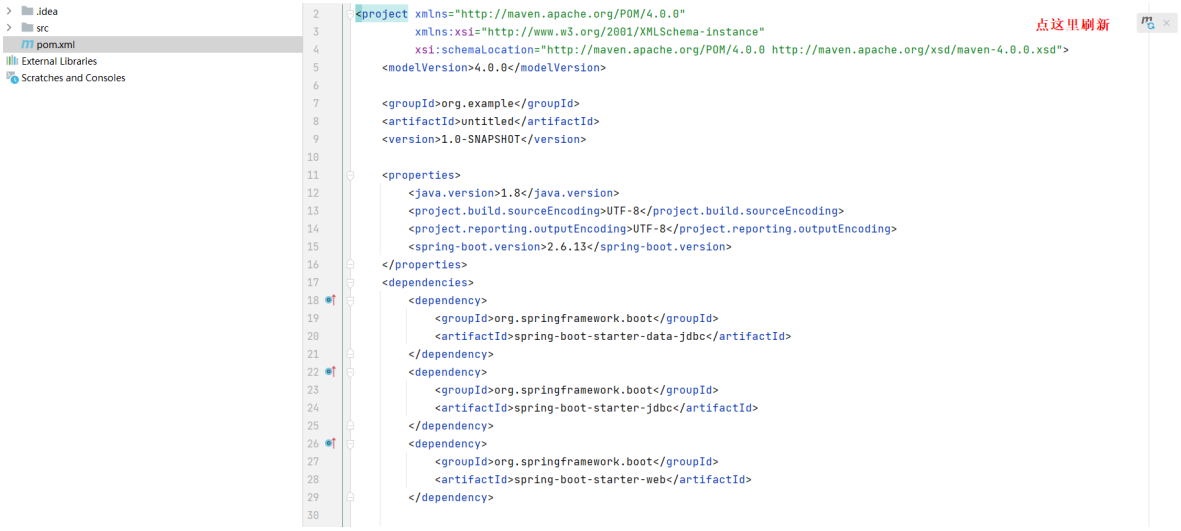
```
    <properties>
        <java.version>1.8</java.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
        <spring-boot.version>2.6.13</spring-boot.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-data-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>${spring-boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>1.8.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>${spring-boot.version}</version>
                <configuration>
                    <mainClass>com.project.BootOkApplication</mainClass>
                  <!--  <skip>true</skip>-->
```

```
                </configuration>
                <executions>
                    <execution>
                        <id>repackage</id>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

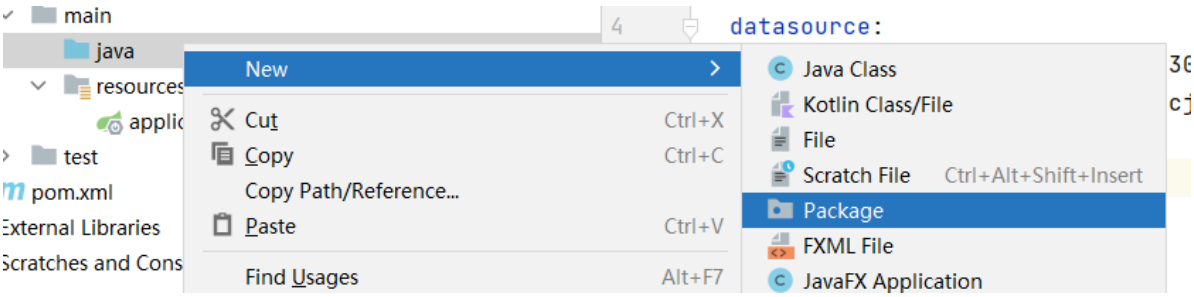


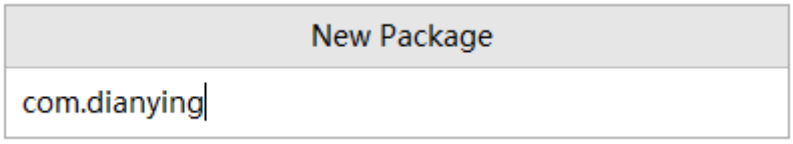
1.3编写application.yml,放在resources,配置web服务端口和数据库连接信息如下

server.port=8080

```
server:
  port: 8080
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/dianying?characterEncoding=UTF-
8&useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: mysql
# localhost 可以换成数据库服务器ip   username password 是数据库的用户名密码  作用连接数据库。yml文件格式百度
```

1.4java目录下，右键，新建包（包就是放java源代码的目录），com.dianying





1.5com.dianying下右键新建类，建议名称为StartApplication

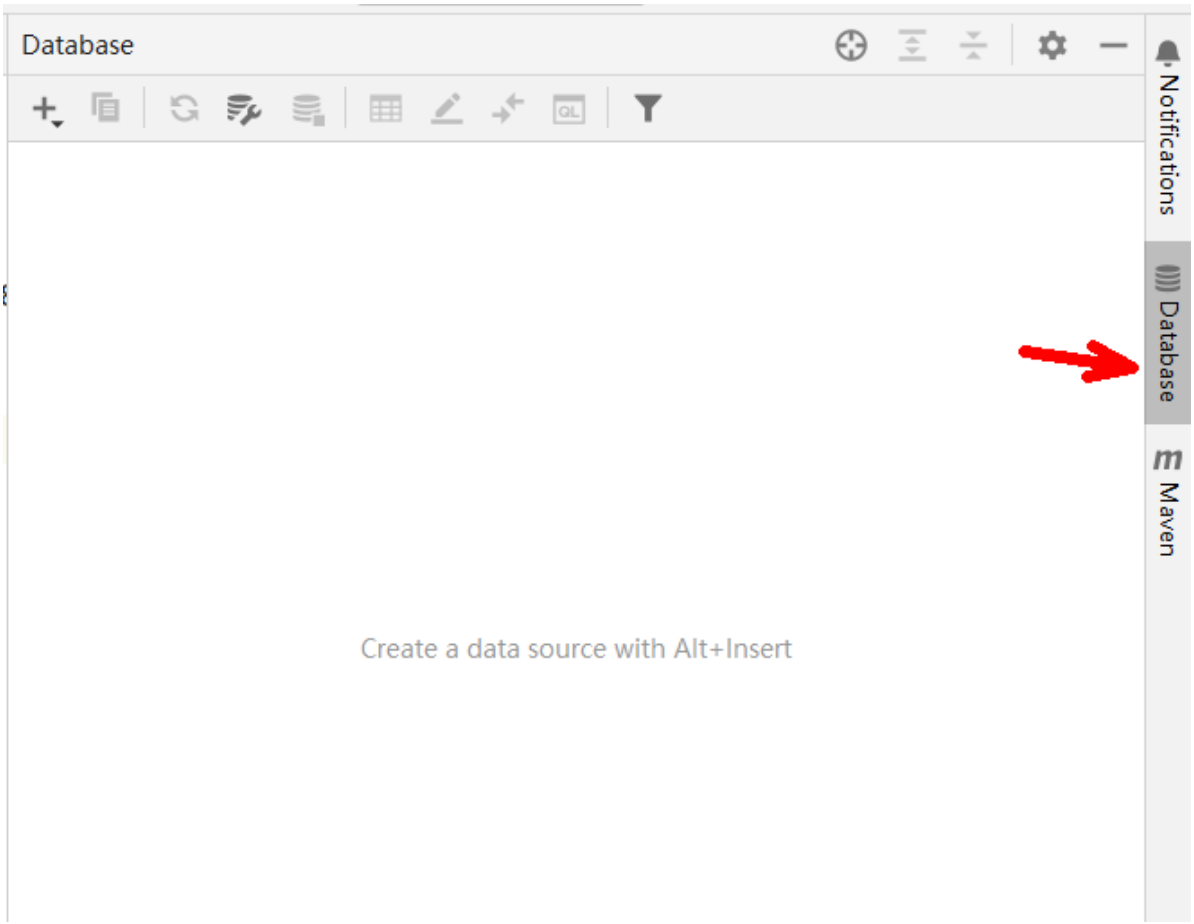
```
@SpringBootApplication
public class StartApplication {
    public static void main(String[] args) {
        SpringApplication.run(StartApplication.class, args);
    }
}
```

这个是程序的入口，就像c语言的main方法一样，当然如果需要让它正常运行，**之前pom.xml中需要将mainClass设置成当前这个类。**

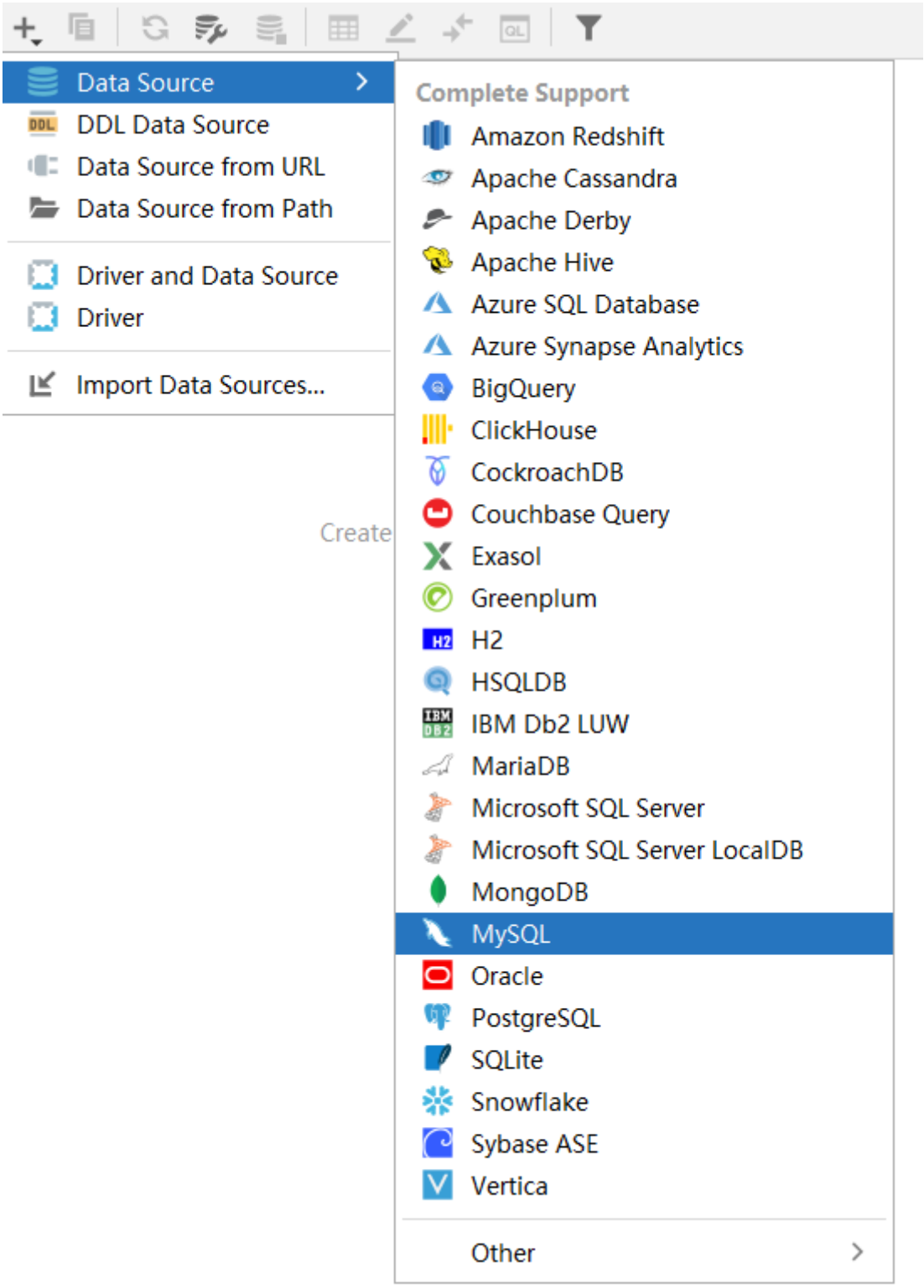
```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <version>${spring-boot.version}</version>
    <configuration>
        <mainClass>com.dianying.StartApplication</mainClass>
        <!--  <skip>true</skip>-->
    </configuration>
    <executions>
        <execution>
            <id>repackage</id>
            <goals>
                <goal>repackage</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

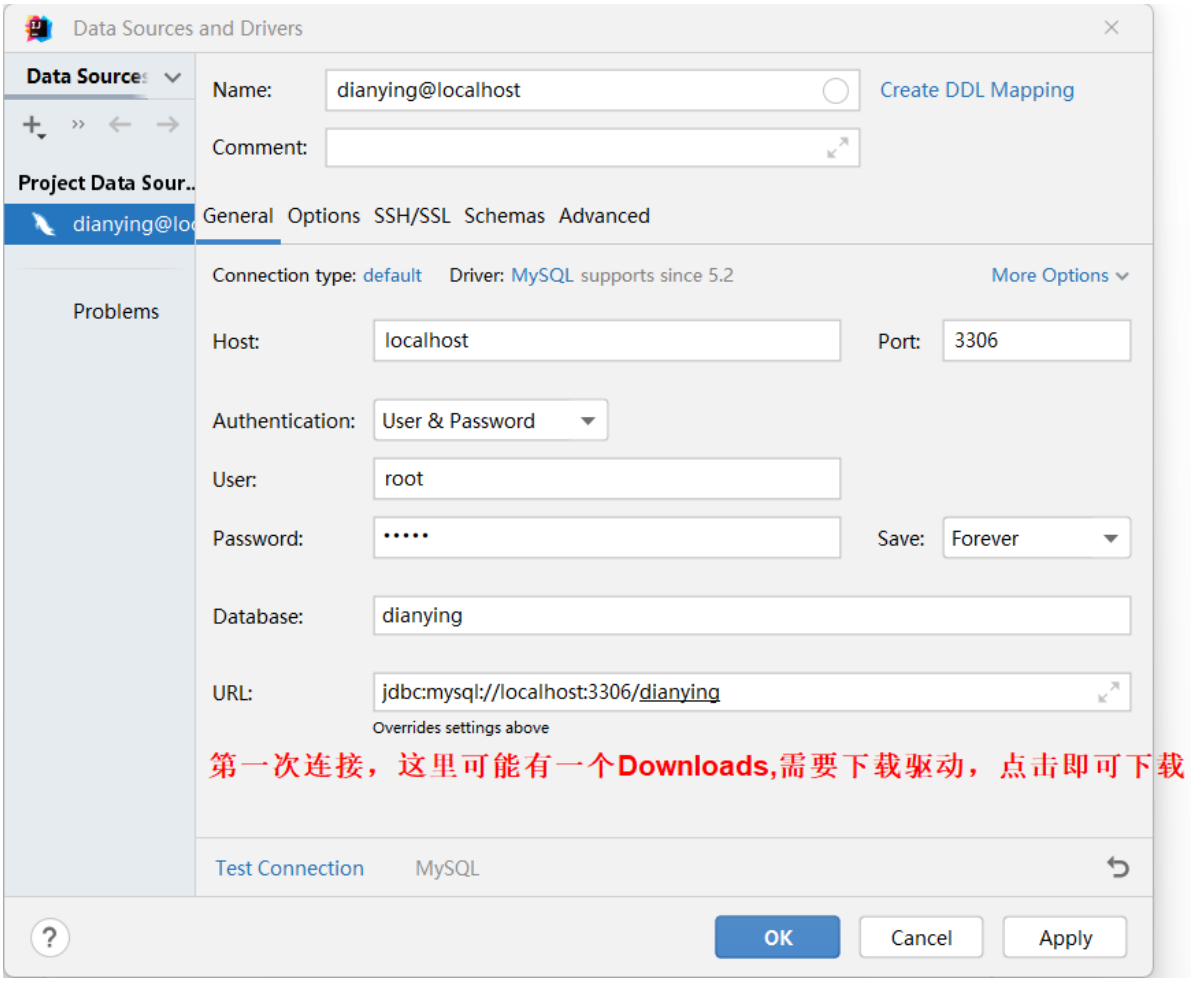
1.6在com.dianying上右键，新建包,com.dianying.entity,用来存放实体类（用来存数据库表数据的类）

1.6.1idea连接数据库

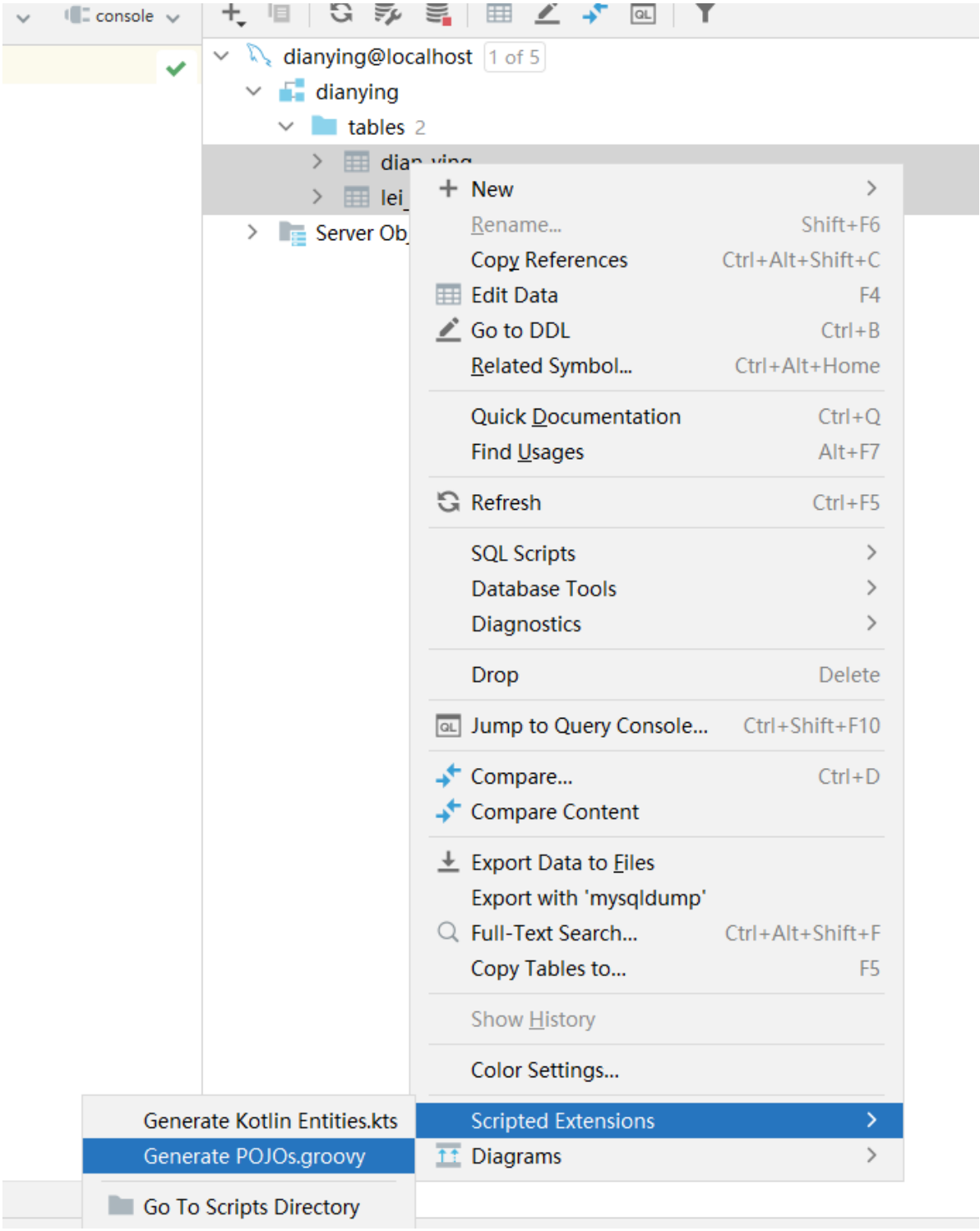


点+号

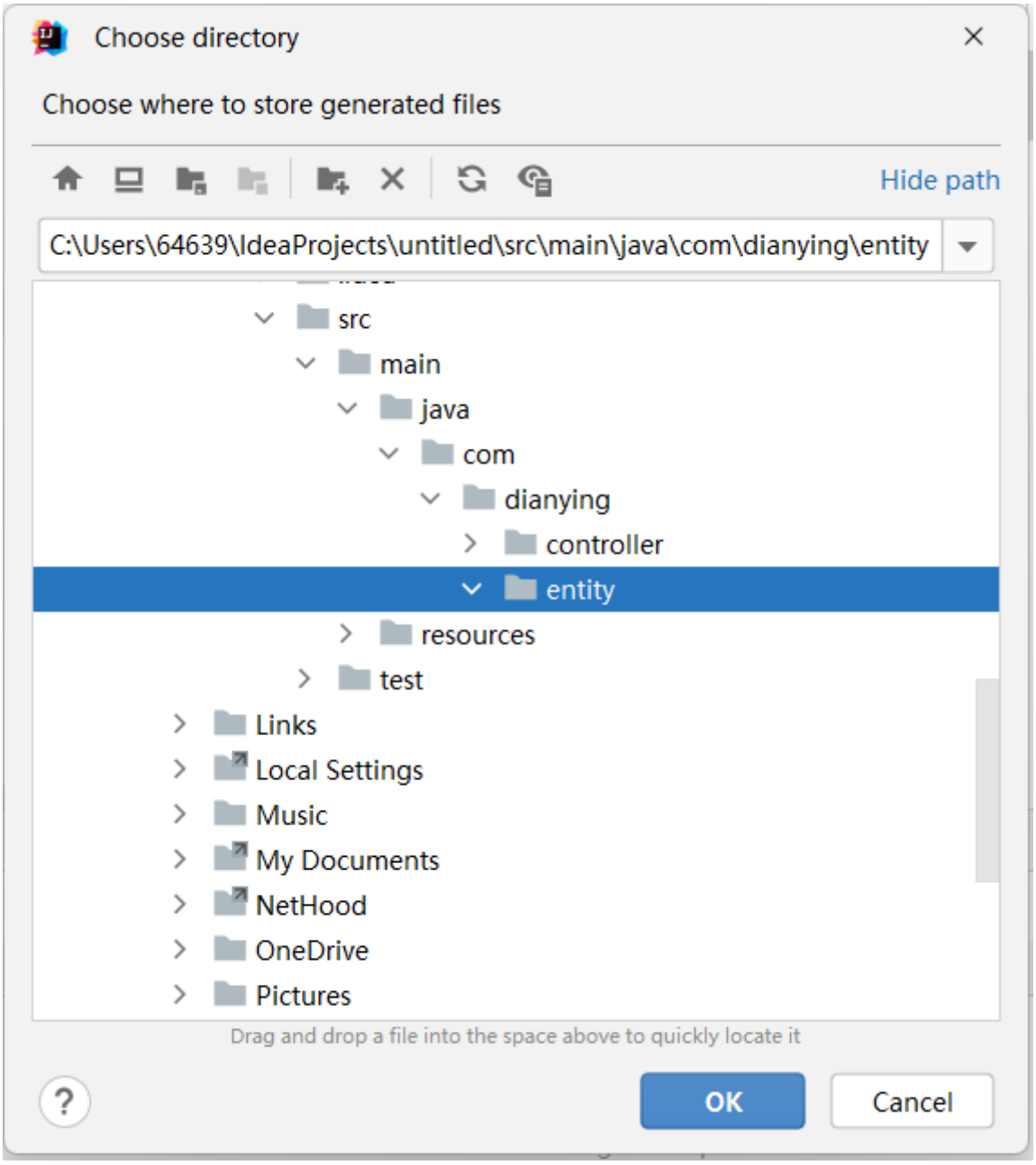

Data Source
DDL Data Source
Data Source from URL
Data Source from Path
Driver and Data Source
Driver
Import Data Sources...
Create
Complete Support
Amazon Redshift
Apache Cassandra
Apache Derby
Apache Hive
Azure SQL Database
Azure Synapse Analytics
BigQuery
ClickHouse
CockroachDB
Couchbase Query
Exasol
Greenplum
H2
HSQLDB
IBM Db2 LUW
MariaDB
Microsoft SQL Server
Microsoft SQL Server LocalDB
MongoDB
MySQL
Oracle
PostgreSQL
SQLite
Snowflake
Sybase ASE
Vertica
Other


Data Sources and Drivers
Data Source
Project Data Sour..
dianying@loc
Problems
Name:
dianying@localhost
Create DDL Mapping
Comment:
General
Options
SSH/SSL
Schemas
Advanced
Connection type: default
Driver: MySQL supports since 5.2
More Options
Host:
localhost
Port:
3306
Authentication:
User & Password
User:
root
Password:
Save:
Forever
Database:
dianying
URL:
jdbc:mysql://localhost:3306/dianying
Overrides settings above
第一次连接，这里可能有一个Downloads,需要下载驱动，点击即可下载
Test Connection
MySQL
OK
Cancel
Apply

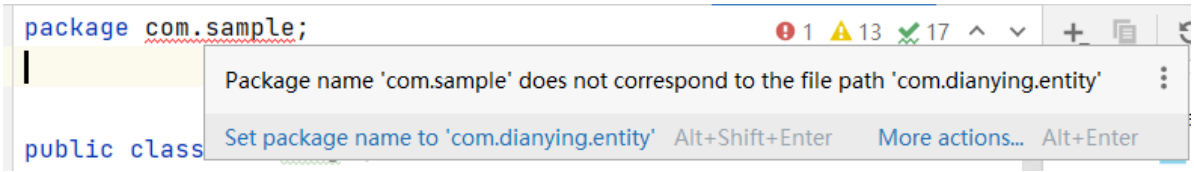
ok之后，选择表，点右键



选择生成实体类，目录选择entity



生成的类，打开报错，鼠标移动到错误位置



点击Set package name to ....这里就好了

类和C++中的类是一样的意思，和C语言的结构体也很像，只是多了方法。方法的访问要用类创建出来的变量(对象)进行调用。

1.7在com.dianying上右键，新建包，com.dianying.controller,用来存放服务程序

每个人可以自己创建自己的一个程序文件，Java中程序文件叫做类。

```
@RestController//说明是返回数据的httpApi接口(SpringMvc中叫做控制器)
@CrossOrigin(origins = "*")//允许跨域
public class DianYingController {


}
```

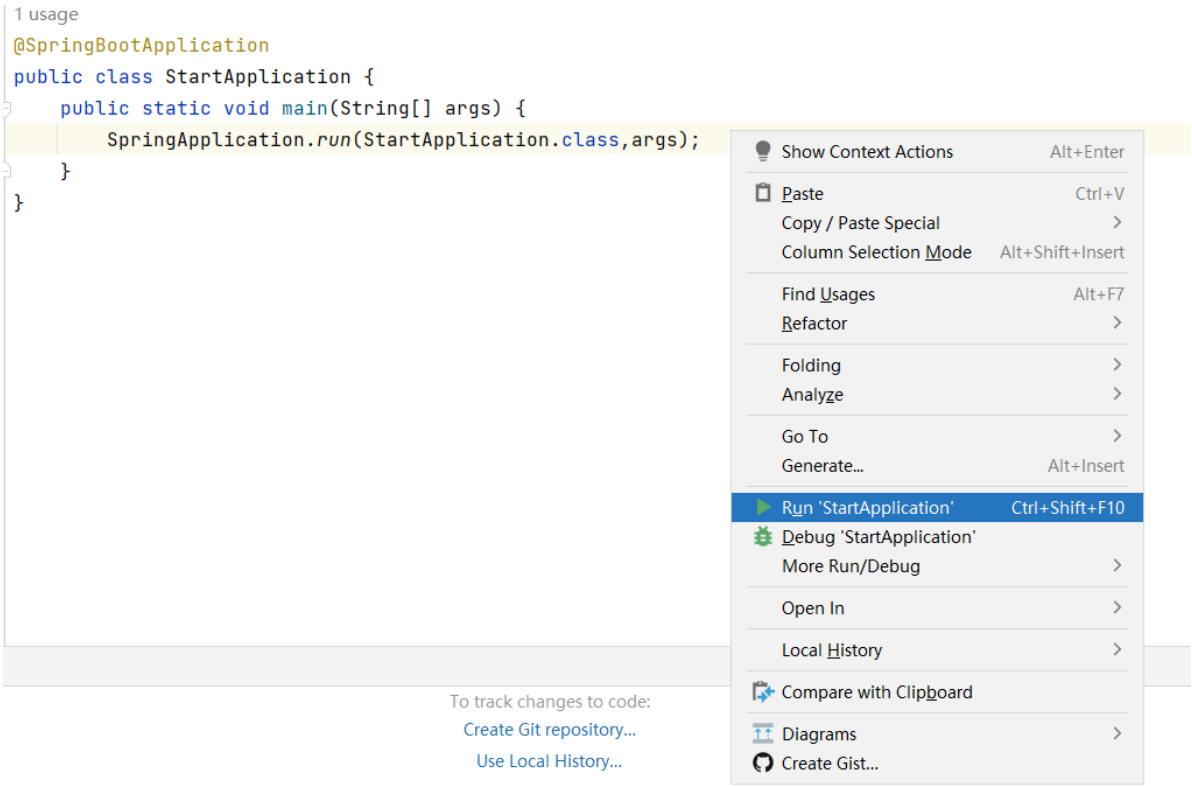
1.7.1Get请求传参

1.7.1.1简单的Get请求

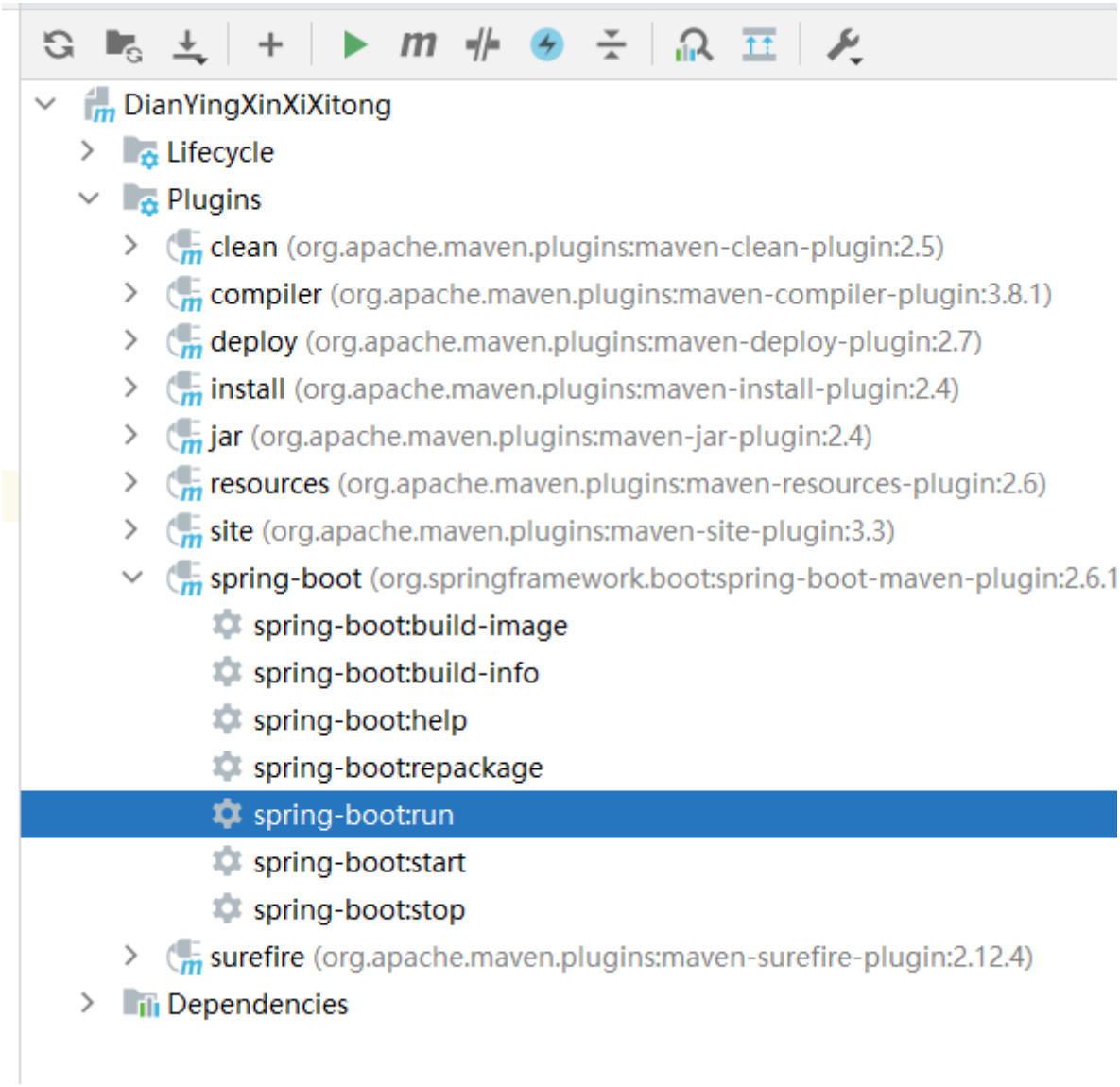
Get是获取的意思，一般用于查询，最直观的，通常浏览器通过超链接或者地址栏直接得到一个网址的内容（比如页面），就是一个Get请求。Js编写代码也可通过Get请求，获得一个网址的内容。（注：网址的内容可以是页面，也可以是其它格式）

```
@RestController//说明是返回数据的httpApi接口(SpringMvc中叫做控制器)
@CrossOrigin(origins = "*")//允许跨域
public class DianYingController {
    @GetMapping("/sayHello") //GetMapping 定义了一个服务端程序，映射网址 /sayHello，/代表网址中的域名（ip）加端口号（根地址）
    public String testSimpleGet(){
        return "你好，SpringBoot";
    }
}
```

程序入口类右键，运行main方法



或者SpringBoot Plugin 双击或右键run



地址栏访问：http://localhost:8080/sayHello

1.7.1.2Get请求通过QueryString（查询字符串）传参

Get请求可以通过QueryString向后台访问的网址对应程序传递数据，后台可以接收到

传参格式： url地址?变量名=值&变量名2=值2&变量名3=值3 （其中变量名也叫参数名）

例如：开发一个服务端程序

```
    @GetMapping("/testQueryStr")
    public String testQueryStringGet(Integer age,String name){//方法的变量名是 age
name
        String str = "name:"+name+";age:"+age;
        System.out.println(str);
        return str;
    }
```

浏览访问:http://localhost:8080/testQueryStr?name=zhangsan&age=18

如果传的数据变量需要很多的时候，这种方式方法的定义比较麻烦，可以将这些变量封装到一个类中，类中成员变量名和请求时QueryString中变量名保持一致也可以，例如：

```
//在entity目录下新建类
@Data   //这个自动生成getter setter方法 是实体类javaBean的规范
public class Persion {
    private Integer age;
    private String name;
}
```

```
@GetMapping("/testQueryStrEntity")
    public String testQueryStringGet2(Persion persion){//方法的变量名是 age  name
        String str = "name:"+persion.getName()+";age:"+persion.getAge();
        System.out.println(str);
        return str;
    }
```

浏览访问:http://localhost:8080/testQueryStrEntity?name=zhangsan&age=18

1.7.1.3Get请求通过路径变量传参

Get请求可以通过url网址对应程序传递数据，url地址都是以 “/”分割的，路径中的内容也可以作为变量值传给后端程序

例如:

```
 @GetMapping("/testPathVarible/{name}/{age}")
    public String testPathVarible(@PathVariable String name, @PathVariable
Integer age){//方法的变量名是 age  name
        String str = "name:"+name+";age:"+age;
        System.out.println(str);
        return str;
    }
```

浏览访问:http://localhost:8080/testPathVarible/zhangsan/18

1.7.2Post请求

Get请求在请求网址中携带数据，安全性差，并且传递数据很多的时候也不适合。页面的数据采集用传到后端保存的功能大多用post请求

例如:

```
@PostMapping("/testPostJson")
    public String postData(@RequestBody Persion persion){
        String str = "name:"+persion.getName()+";age:"+persion.getAge();
        System.out.println(str);
        return str;
    }
```

客户端提交数据以json格式提交，见本节 2.1 2.2内容

安装apipost, https://www.apipost.cn/client/?utm_source=10124&bd_vid=7477753421216474431

小提示：

严格Restful风格的api,删除用delete请求，还有修改用put请求，查询用get请求，新增用post请求,依靠请求的方式确定做增删改查中的什么操作，而不是靠网址，网址只代表资源本身。

2.Json

2.1什么是Json

JSON全称为“JavaScript Object Notation”，意为“JavaScript对象表示法”，是JavaScript的一个子集，**是一种轻量级的、基于文本的、开放的数据交换格式。**目前JSON已成为WEB数据交换的通用标准，其在WEB开发领域有着举足轻重的地位，是每一个WEB开发者必须掌握的知识。

2.2Json对象

一个JSON对象由大括号{}保存的对象是一个无序的名称(key)/值(value)对集合。以左括号{开始，右括号}结束。每个"键"后跟一个冒号(:)，多个名称/值对使用英文半角逗号(,)分隔。JSON对象的key为string类型，值可以为基本数据类型、对象或数组。

{"name":"zhangsan", "sex":"male"}

2.3Json数组

一个JSON数组以左中括号[开始，右中括号]结束，为一个值的集合，数组各个值之间使用英文半角逗号(,)分隔。数组各个值既可以是基本数据类型，也可以是对象或数组。
JSON数组的定义示意图如下：

["数学","英语","语文"]

2.4对象属性为数组，或者数组的元素内容为对象

```
{
  "students":[{"name":"zhangsan", "sex":"male"},
    {"name":"lisi", "sex":"fmale"},
    {"name":"wangwu", "sex":"male"}],
  "classname":"一班"
}
```

3.JdbcTemplate完成增删改查

3.1准备

3.1.1数据库增加用户表

本节以登录注册功能为例进行编写，在之前案例基础上，数据库增加用户表，yonghu

```
CREATE TABLE  yonghu  (
   id  int(11) NOT NULL AUTO_INCREMENT,
   xingming  varchar(30) DEFAULT NULL,
   zhanghao  varchar(50) DEFAULT NULL,
   mima  varchar(20) DEFAULT NULL,
   age int(11) DEFAULT NULL,
   leibie int(11) DEFAULT NULL,# 比如0表示普通用户 1表示超级用户等
   PRIMARY KEY ( id )
)
```

同样生成实体类。

3.1.2类中声明JdbcTempLate类型成员变量（其实Spring叫做ioc注入一个JdbcTempLate对象）

```
@RestController//说明是返回数据的httpApi接口(SpringMvc中叫做控制器)
@CrossOrigin(origins = "*")//允许跨域
public class DianYingController {
    @Autowired//这个不能省略，否则不能使用，java需要一个创建对象的过程，在Spring中就靠它了。
    private JdbcTemplate jdbc;//建议成员变量都放在成员方法的上边
```

详见：http://c.biancheng.net/spring/jdbc-template.html

3.2完成后端登录功能

```
最常用异常处理代码结构
try{
    正常逻辑
        a
        b
        c
}catch(Exception e){
    异常处理逻辑
}
```

```
@PostMapping("/doLogin")
    public Yonghu doLogin(@RequestBody Yonghu yonghu)
{//http://localhost:8080/doLogin
        System.out.println(yonghu.getZhanghao()+yonghu.getMima());
        //第一个参数是Sql,sql语句中用?占位符定义sql中的变量，
        //第二个参数相当与类型转换器，将返回的数据 转换成 YongHu类型的对象（C++里只叫变量）
        //第三个参数及之后的参数，是传给?占位符表示的sql中变量的值，顺序 类型  要一一对应
        try {
            Yonghu yh = jdbc.queryForObject("select * from yonghu where
zhanghao=? and mima=?",new BeanPropertyRowMapper<>
(Yonghu.class),yonghu.getZhanghao(),yonghu.getMima());
            return yh;//返回用户对象
        } catch (DataAccessException e) {
           e.printStackTrace();
           return null;
        }
    }

优化：
```

```
   // 新建类封装api接口的返回结果:
    @Data
public class Result {
    private Integer code;  //200 正常   201 异常
    private Object result; //存整整的返回的数据
}

    //返回值，永远返回 Result类的json对象
        @PostMapping("/doLogin")
    public Result doLogin(@RequestBody Yonghu yonghu){
       //Ctrl + Alt + T
        Result res = new Result();//多
        try {
            Yonghu yh = jdbc.queryForObject("select * from yonghu where
zhanghao=? and mima=?",new BeanPropertyRowMapper<>(Yonghu.class),
                    yonghu.getZhanghao(),yonghu.getMima());
            res.setCode(200);
            res.setResult(yh);//用户数据放入结果中
            return res;
        } catch (DataAccessException e) {
            e.printStackTrace();
            res.setCode(201);
            res.setResult("出现异常"+e.getMessage());//message 异常的信息
            return res;
        }
    }
```

3.3完成后端注册功能

```
    @PostMapping("/doRegster")
    public String doRegister(@RequestBody Yonghu yonghu){
        //http://localhost:8080/doRegster
        //同样第一个参数是增删改的sql
        //同样?占位符定义sql中的变量，从第二个参数开始 是为这些?变量进行值的绑定（相当于相应值
放在sql中），顺序 类型  要一一对应
        try {
            jdbc.update("INSERT INTO
yonghu(xingming,zhanghao,mima,age,leibie)VALUES(?,?,?,?,?)",yonghu.getXingming()
,yonghu.getZhanghao(),yonghu.getMima(),yonghu.getAge(),yonghu.getLeibie());
            return "ok";//返回用户对象
        } catch (DataAccessException e) {
            e.printStackTrace();
            return "false";
        }
    }

优化:
     @PostMapping("/doRegster")
    public Result doRegister(@RequestBody Yonghu yonghu){
        //http://localhost:8080/doRegster
        //同样第一个参数是增删改的sql
        //同样?占位符定义sql中的变量，从第二个参数开始 是为这些?变量进行值的绑定（相当于相应值
放在sql中），顺序 类型  要一一对应
        Result res = new Result();//示例化  分配内存
        try {
```

```
            jdbc.update("INSERT INTO
yonghu(xingming,zhanghao,mima,age,leibie)VALUES(?,?,?,?,?)",yonghu.getXingming()
,yonghu.getZhanghao(),yonghu.getMima(),yonghu.getAge(),yonghu.getLeibie());
            res.setCode(200);
           res.setResult("成功插入");
            return res;//result对象返回
        } catch (DataAccessException e) {
            e.printStackTrace();
            res.setCode(201);
            res.setResult("成功失败");
            return res;//result对象返回
        }
    }
```

3.4完成后端用户列表功能

传入参数类别，如果不传，返回所有的电影

```
 //List这种数据类型有一个最常用的子类型 ArrayList. 可以把它看成加强的数组。这个类型的变量(对
象)提供了很多操作集合的方法，而且长度可以不固定
    //List后面<Dianyingxinxi> 属于泛型，意思就是集合中每个元素的类型都是Dianyingxinxi
    @GetMapping("/dianyings")
    public List<Dianyingxinxi> queryDianYingByLeiXingId(Integer leiXingId){
        //http://localhost:8080/dianyings?leiXingId=3
        try {
            if(leiXingId!=null){
                //第一个参数是查询的Sql,
                //第二个参数相当于类型转换器，将返回的数据集合中的元素转换为相应类型，此处返
回的集合中的元素都是 Dianyingxinxi
                return  jdbc.query("SELECT * FROM dianyingxinxi where
leixingid=?",new BeanPropertyRowMapper<>(Dianyingxinxi.class),leiXingId);
            }else{
                return  jdbc.query("select * from dianyingxinxi",new
BeanPropertyRowMapper<>(Dianyingxinxi.class));
            }
        } catch (DataAccessException e) {
            e.printStackTrace();
            return null;
        }
    }


优化：
     //List这种数据类型有一个最常用的子类型 ArrayList. 可以把它看成加强的数组。这个类型的变
量(对象)提供了很多操作集合的方法，而且长度可以不固定
    //List后面<Dianyingxinxi> 属于泛型，意思就是集合中每个元素的类型都是Dianyingxinxi
    @GetMapping("/dianyings")
    public Result queryDianYingByLeiXingId(Integer leiXingId){
        //http://localhost:8080/dianyings?leiXingId=3
        Result res = new Result();
        try {
            List<Dianyingxinxi> dys=null;
            if(leiXingId!=null){
                //第一个参数是查询的Sql,
```

```
                //第二个参数相当于类型转换器，将返回的数据集合中的元素转换为相应类型，此处返回的集合中的元素都是 Dianyingxinxi
                dys =  jdbc.query("SELECT * FROM dianyingxinxi where leixingid=?",new BeanPropertyRowMapper<>(Dianyingxinxi.class),leiXingId);
            }else{
                dys =  jdbc.query("select * from dianyingxinxi",new BeanPropertyRowMapper<>(Dianyingxinxi.class));
            }
            res.setResult(dys);//数据
            res.setCode(200);//代码  成功
            return res;
        } catch (DataAccessException e) {
            res.setResult("出错了"+e.getMessage());//数据
            res.setCode(201);//代码  出错
            return res;
        }
    }
```