**后端开发与软件配置**

1. 开发相关工具安装

IntelliJ IDEA或Eclipse，官网直接下载安装 https://www.jetbrains.com/idea/

数据库：MySQL下载安装地址和教程 <https://www.cnblogs.com/baimt/p/5688517.html>

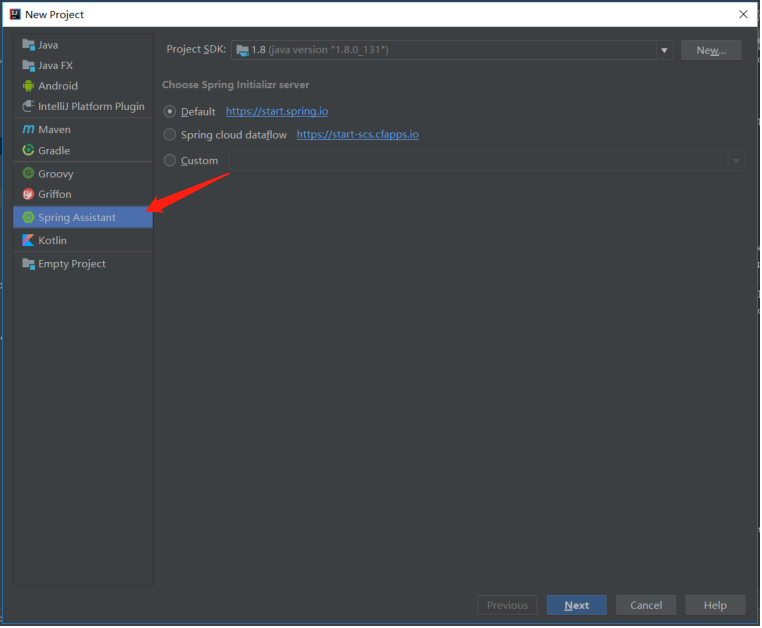
JDK1.8下载安装教程 https://blog.csdn.net/qq\_26631651/article/details/82666336

1. 后端框架

Spring + SpringMVC + JPA，基于Spring-boot。

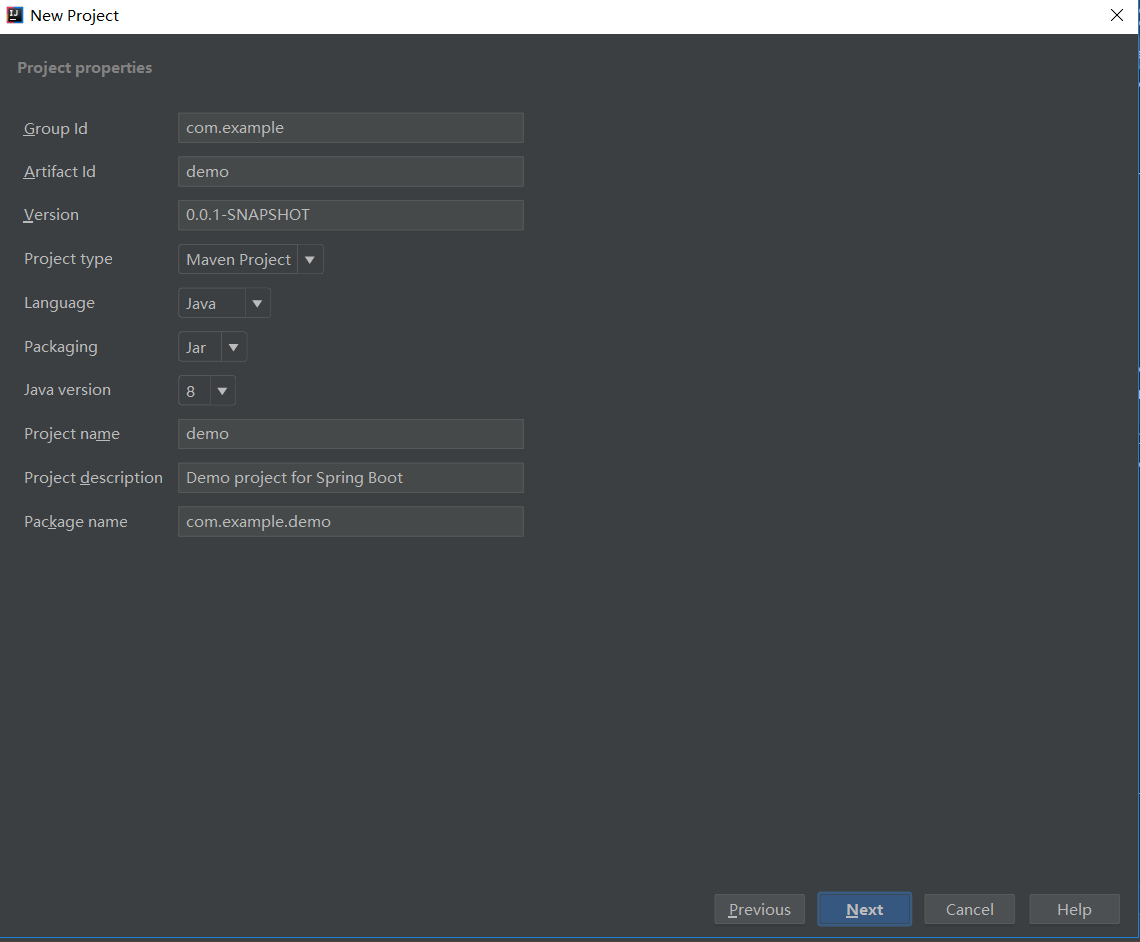
1. 在IDEA中搭建一个Spring-boot项目的基本流程。

①: New Project 并选择箭头所示行，随后点击Next。

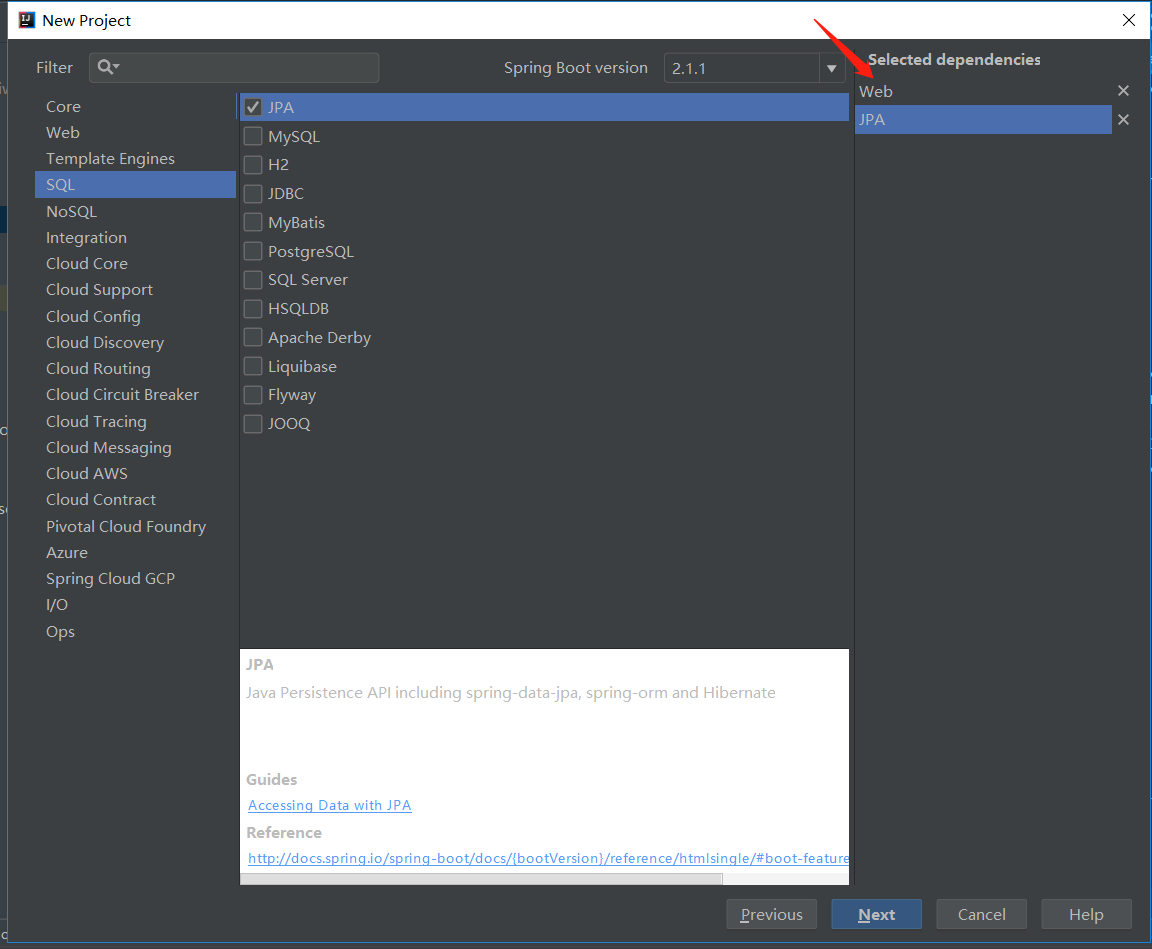


注意：社区版的IDEA需要自行安装Spring Assistant。

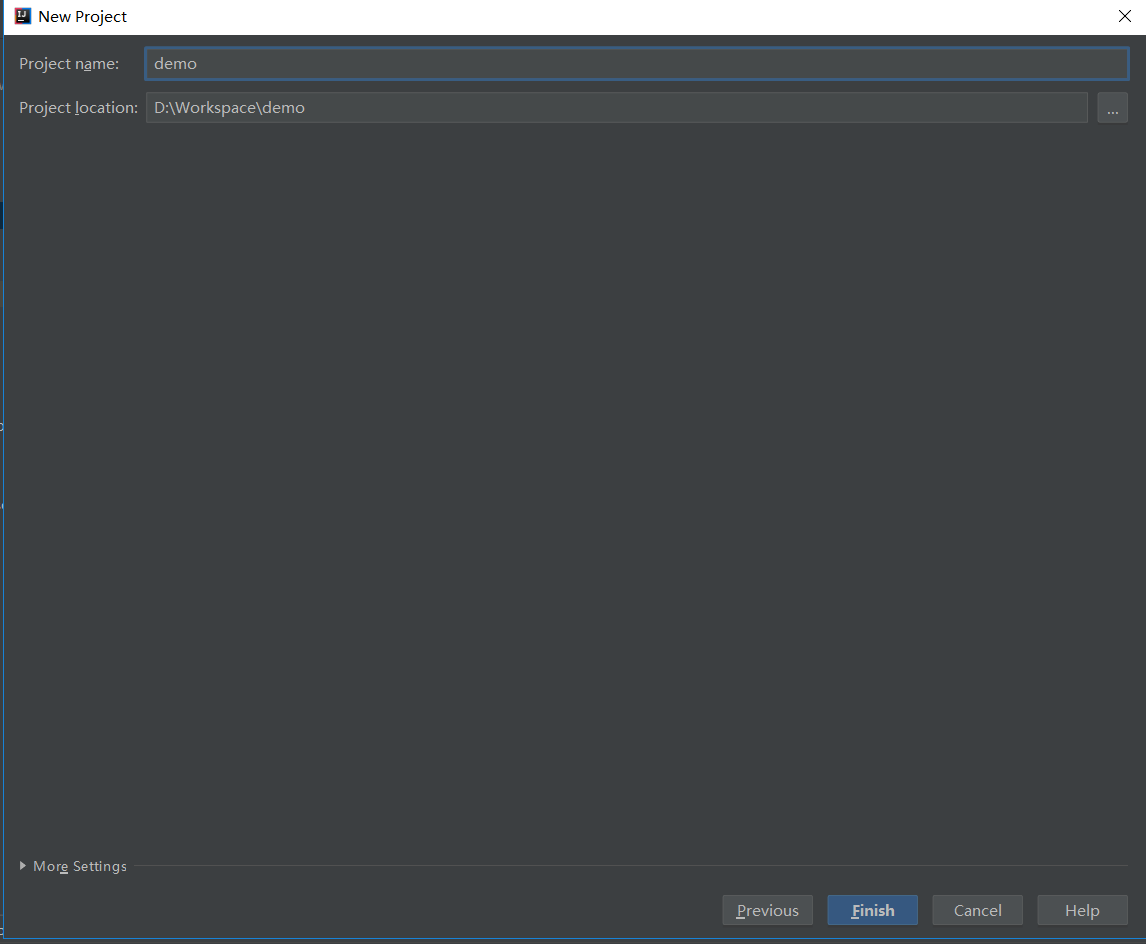
②: 默认下图所示配置，点击Next，也可根据个人喜好更改。



③: 选取项目依赖的Jar包，在本次项目中我们选择Web中的Web项打钩，SQL中的JPA和MySQL项打钩。

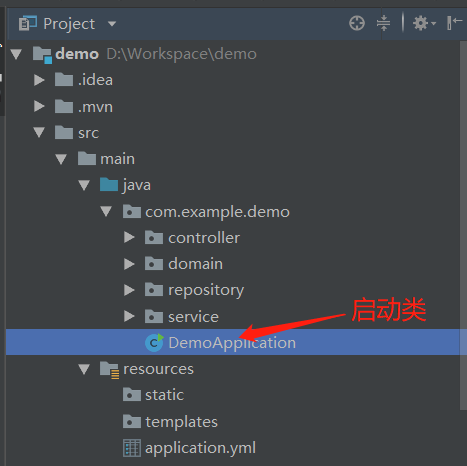


④: 可自行选择更改项目名称或项目地址，点击Finish完成框架搭建。



1. 后端开发流程

①在项目包下分别建立controller、domain、repository、service四个包。将application的后缀名更改为yml，右击图中启动类选择Run便可以运行项目。



② 修改配置文件application.yml。

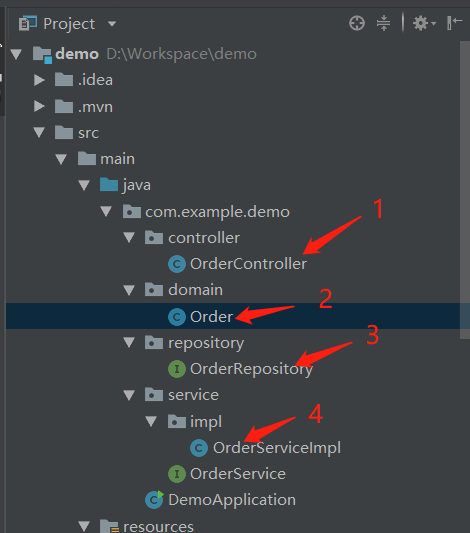
**spring:  
 datasource:  
 driver-class-name:** com.mysql.jdbc.Driver  
**username:** root #数据库账号  
**password:** 199412 #数据库密码  
**url:** jdbc:mysql://localhost:3306/demo?characterEncoding=utf-8&useSSL=false #demo为数据库名称  
**jpa:  
 show-sql:** true  
**hibernate:  
 ddl-auto:** update  
**devtools:  
 restart:  
 enabled:** true  
**jackson:  
 date-format:** yyyy-MM-dd  
**time-zone:** GMT+8  
  
**server:  
 tomcat:  
 uri-encoding:** UTF-8  
**port:** 8080

③ pom.xml文件内容

<?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">  
<modelVersion>4.0.0</modelVersion>  
<parent>  
<groupId>org.springframework.boot</groupId>  
<artifactId>spring-boot-starter-parent</artifactId>  
<version>1.5.3.RELEASE</version>  
<relativePath/><!-- lookup parent from repository -->  
</parent>  
<groupId>com.example</groupId>  
<artifactId>demo</artifactId>  
<version>0.0.1-SNAPSHOT</version>  
<name>demo</name>  
<description>Demo project for Spring Boot</description>  
<properties>  
<java.version>1.8</java.version>  
</properties>  
<dependencies>  
<dependency>  
<groupId>org.springframework.boot</groupId>  
<artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
<dependency>  
<groupId>org.springframework.boot</groupId>  
<artifactId>spring-boot-starter-web</artifactId>  
</dependency>  
<dependency>  
<groupId>org.springframework.boot</groupId>  
<artifactId>spring-boot-starter-test</artifactId>  
<scope>test</scope>  
</dependency>  
<dependency>  
<groupId>mysql</groupId>  
<artifactId>mysql-connector-java</artifactId>  
<scope>runtime</scope>  
</dependency>  
<dependency>  
<groupId>org.projectlombok</groupId>  
<artifactId>lombok</artifactId>  
<scope>compile</scope>  
</dependency>  
</dependencies>  
<build>  
<plugins>  
<plugin>  
<groupId>org.springframework.boot</groupId>  
<artifactId>spring-boot-maven-plugin</artifactId>  
</plugin>  
</plugins>  
</build>  
</project>

④．本次项目主要是实现一个订单的增删改查功能，下面以增加一个订单为例，描述开发过程。

下图中1为前端访问后端的入口，即Controller；2即为订单实体类；3为数据接口层，负责通过SQL与数据库交互；4为业务逻辑层，具体的业务流程在impl中实现。



1.首先创建一个订单实体类Order：

package com.example.demo.domain;  
import lombok.Data;  
import javax.persistence.\*;@Entity  
@Data  
@Table(name = "Order\_info")  
public class Order {  
  
//主键  
@javax.persistence.Id  
 @GeneratedValue  
private Integer Id;  
  
//订单编号  
@Column(length = 20, nullable = false)  
private String orderId;  
  
 private String orderSource;  
  
 private Double totalPrice;  
  
 public Integer getId() {  
return Id;  
}  
public void setId(Integer id) {  
Id = id;  
}  
public String getOrderId() {  
return orderId;  
}  
public void setOrderId(String orderId) {  
this.orderId = orderId;  
}  
  
public String getOrderSource() {  
return orderSource;  
}  
public void setOrderSource(String orderSource) {  
this.orderSource = orderSource;  
}  
public Double getTotalPrice() {  
return totalPrice;  
}  
public void setTotalPrice(Double totalPrice) {  
this.totalPrice = totalPrice;  
}  
}

1. 创建Order对应的数据接口层接口OrderRepository:

package com.example.demo.repository;  
  
import com.example.demo.domain.Order;  
import org.springframework.data.jpa.repository.JpaRepository;  
public interface OrderRepository extends JpaRepository<Order, Integer> {  
 Order findOneByOrderId(String orderId);  
}

1. 创建业务逻辑层接口OrderService：

package com.example.demo.service;  
  
import com.example.demo.domain.Order;  
import org.springframework.stereotype.Service;public interface OrderService {  
  
//增  
String saveOne(Order order);  
  
//删  
String deleteOne(String orderId);  
  
//改  
String updateOrder(Order order);  
  
//查  
Order findOne(String orderId);  
}

1. 实现接口OrderService：

package com.example.demo.service.impl;  
import com.example.demo.domain.Order;  
import com.example.demo.repository.OrderRepository;  
import com.example.demo.service.OrderService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;

@Service  
public class OrderServiceImpl implements OrderService {  
  
@Autowired  
private OrderRepository orderRepository;  
  
//增加一条订单  
@Override  
public String saveOne(Order order) {  
boolean flag = false;  
 try{  
orderRepository.save(order);  
flag = true;  
}catch (Exception e){  
 System.*out*.println("保存失败");  
}  
return flag == true ? "success": "failed";  
}  
  
//删除一条订单  
@Override  
public String deleteOne(String orderId) {  
 Order order = this.findOne(orderId);  
 if (order == null){  
return "failed";  
}  
boolean flag = false;  
 try{  
orderRepository.delete(order);  
flag = true;  
}catch (Exception e){  
 System.*out*.println("删除失败");  
}  
return flag == true ? "success": "failed";  
}  
  
//更新订单  
@Override  
public String updateOrder(Order order) {  
boolean flag = false;  
 try{  
orderRepository.save(order);  
flag = true;  
}catch (Exception e){  
 System.*out*.println("更新失败");  
}  
return flag == true ? "success": "failed";  
}  
  
//查找某条订单  
@Override  
public Order findOne(String orderId) {  
 Order order = orderRepository.findOneByOrderId(orderId);  
 return order == null ? null : order;  
}  
}

1. 创建controller调用业务逻辑层接口：

package com.example.demo.controller;  
  
import com.example.demo.domain.Order;  
import com.example.demo.service.OrderService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
@RestController  
@RequestMapping("/Order")  
public class OrderController {  
  
@Autowired  
private OrderService orderService;  
  
*/\*\*  
 \* 保存订单  
 \** ***@param****\** ***@return****\*/*@PostMapping("/saveOne")  
public String saveOne(@RequestBody Order order){  
return orderService.saveOne(order);  
}  
*/\*\*  
 \* 删除订单  
 \** ***@param****\** ***@return****\*/*@GetMapping("/deleteOne/{orderId}")  
public String deleteOne(@PathVariable("orderId") String orderId){  
return orderService.deleteOne(orderId);  
}  
  
*/\*\*  
 \* 更新订单  
 \** ***@param****\** ***@return****\*/*@PostMapping("/updateOne")  
public String updateOne(@RequestBody Order order){  
return orderService.updateOrder(order);  
}  
  
*/\*\*  
 \* 查找某条订单  
 \** ***@param****\** ***@return****\*/*@GetMapping("/findOne/{orderId}")  
public Order findOne(@PathVariable("orderId") String orderId){  
return orderService.findOne(orderId);  
}  
}

前端访问的入口即为Controller中的方法，同时通过调用Service中的接口完成与数据库的交互，并取得对应信息返回给前端。

解决前后端跨域问题

package com.example.demo;  
  
import org.springframework.context.annotation.Configuration;  
import org.springframework.web.servlet.config.annotation.CorsRegistry;  
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;  
  
*/\*\*  
 \* 解决前后端跨域问题  
 \** ***@author*** *lhy  
 \** ***@create*** *2018-05-27-18:06  
 \* 什么是同一个域？  
 \* 同一协议，同一ip，同一端口，三同中有一不同就产生了跨域。  
 \*\*/*@Configuration  
public class CrosConfig extends WebMvcConfigurerAdapter {  
@Override  
public void addCorsMappings(CorsRegistry registry) {  
 registry.addMapping("/\*\*")  
 .allowedOrigins("\*")  
 .allowCredentials(true)  
 .allowedMethods("GET", "POST", "DELETE", "PUT")  
 .maxAge(3600);  
}  
  
}