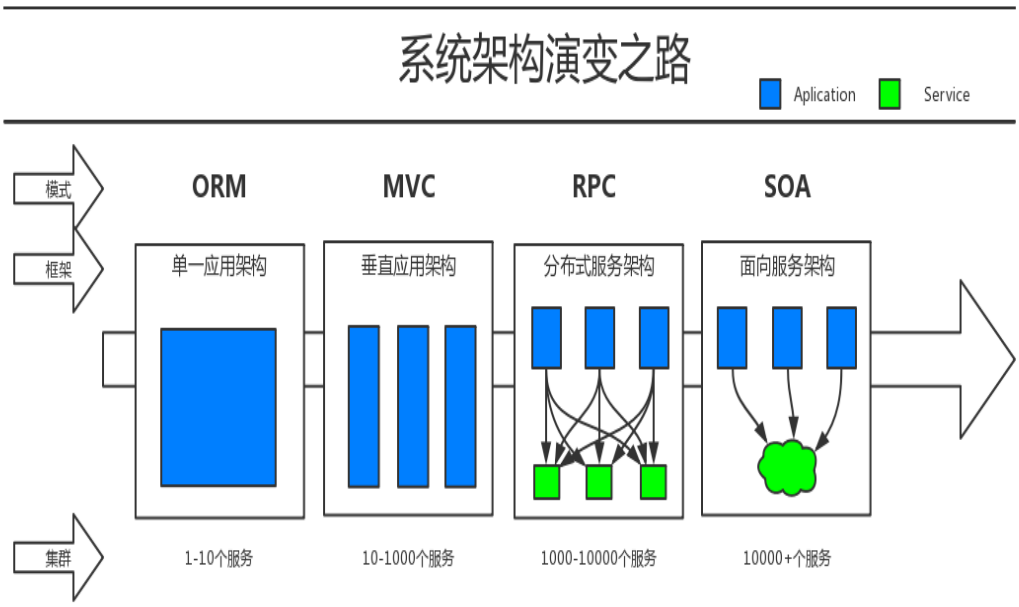
一、系统架构演变之路(回顾)



1.1 单一应用架构

当网站流量很小时，只需要一个应用，所有功能部署在一起，减少部署节点成本的框架称之为集中式框架。此时，用于 **简化增删改查工作量的数据访问框架(ORM)** 是影响项目开发的关键。

1.2 垂直应用架构

当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。此时，用于 **加速前端页面开发的Web框架(MVC)** 是关键。

1.3 分布式服务架构

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于 **提高业务复用及整合的分布式服务框架(RPC)** 是关键。

1.4 面向服务(SOA)架构

典型代表有两个：流动计算架构和微服务架构；

流动计算架构：

当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需**增加一个调度中心基于访问压力实时管理集群容量，提高集群利用率**。此时，**用于提高机器利用率的资源调度和治理中心(SOA)** 是关键。流动计算架构的最佳实践阿里的Dubbo。

微服务架构

与流动计算架构很相似，除了具备流动计算架构优势外，微服务架构中的微服务可以独立部署，独立发展。且微服务的开发不会限制于任何技术栈。微服务架构的最佳实践是SpringCloud。

二、模拟微服务业务场景

模拟开发过程中的服务间关系。抽象出来，开发中的微服务之间的关系是 **生产者和消费者关系**。

目标：模拟一个最简单的服务调用场景，场景中保护微服务提供者(Producer)和微服务调用者(Consumer)，方便后面学习微服务架构

注意：实际开发中，每个微服务为一个独立的SpringBoot工程。

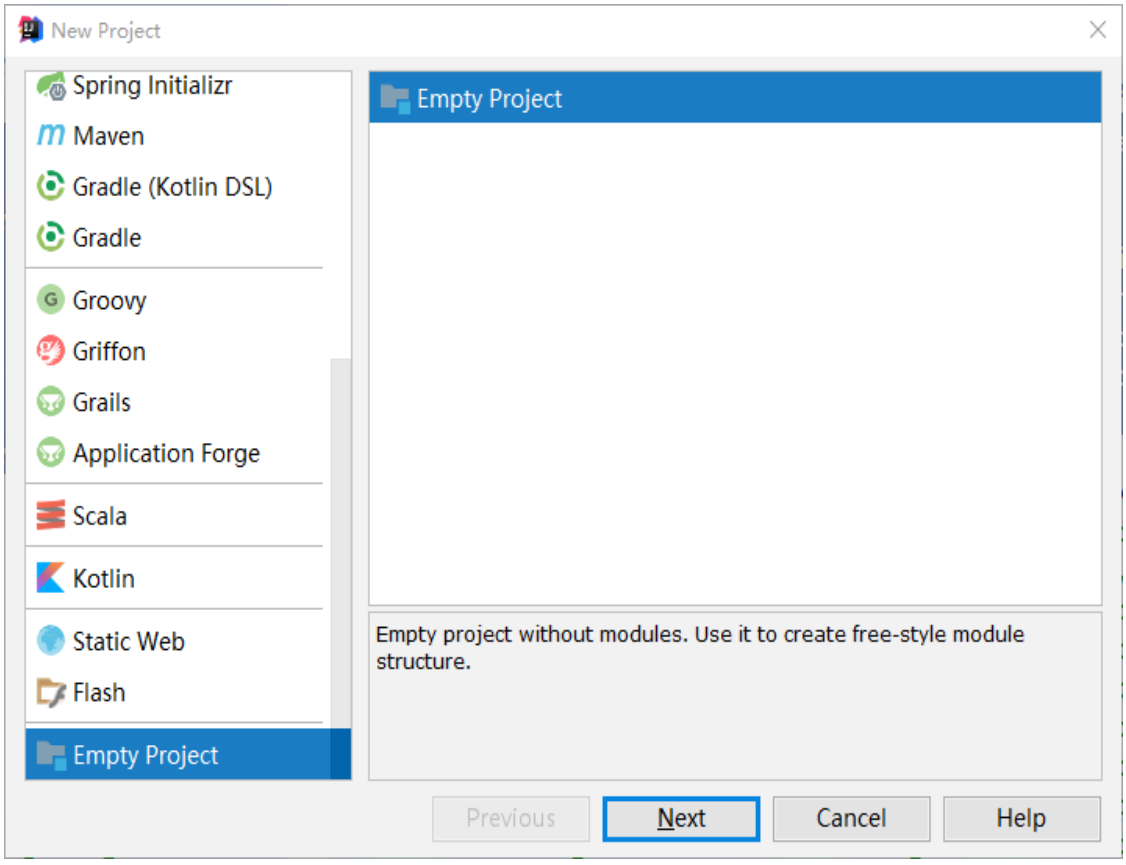
2.1 创建服务的工程

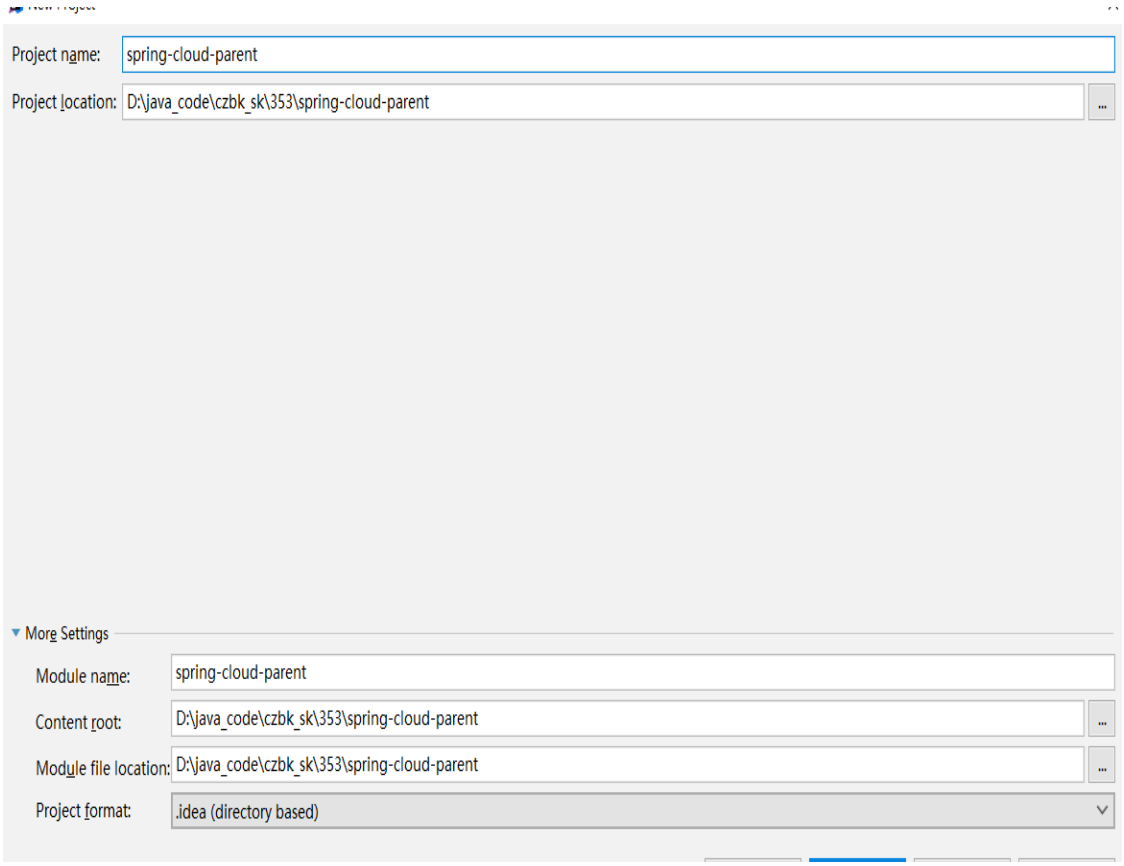
目标： 新建一个父项目spring-cloud-parent

实现步骤：

1. 创建Empty project 的工程

实现过程：





2.2 创建服务提供者(provider)工程

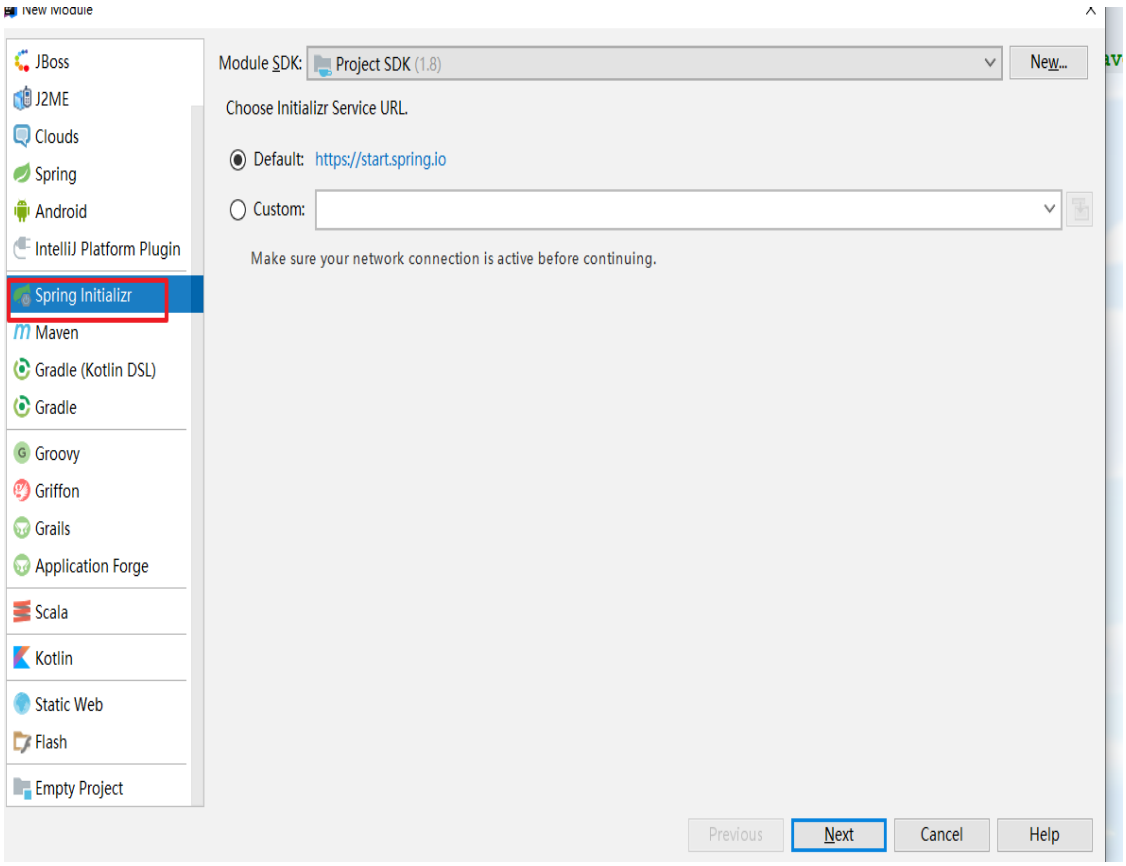
目标： 新建一个项目provider-service，对外提供根据id查询用户的服务

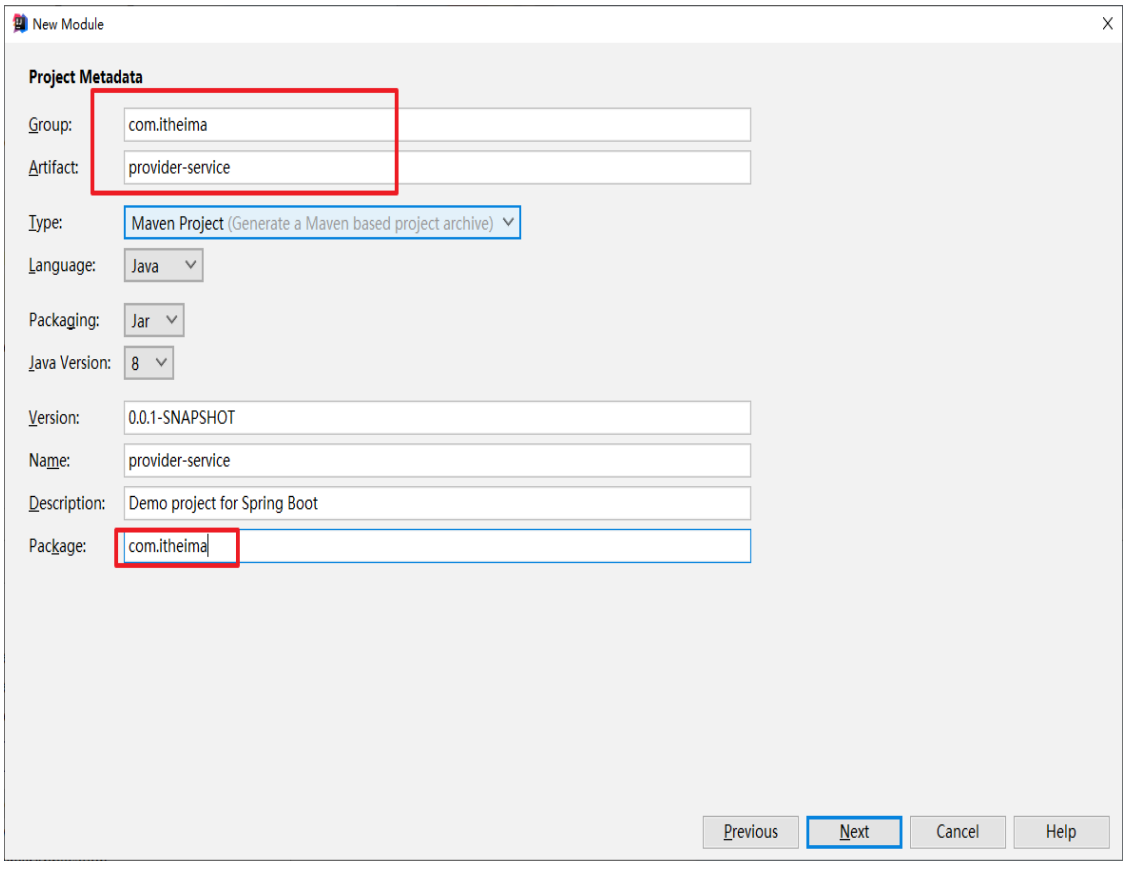
实现步骤：

1. 创建SpringBoot工程
2. 勾选依赖坐标
3. 配置数据库连接信息
4. 创建User表、创建实体User
5. 编写三层架构：dao、service、controller，编写根据id查询用户的方法
6. 配置Mapper映射文件
7. 在application.yml中添加MyBatis配置，扫描mapper.xml和mapper
8. 访问测试地址

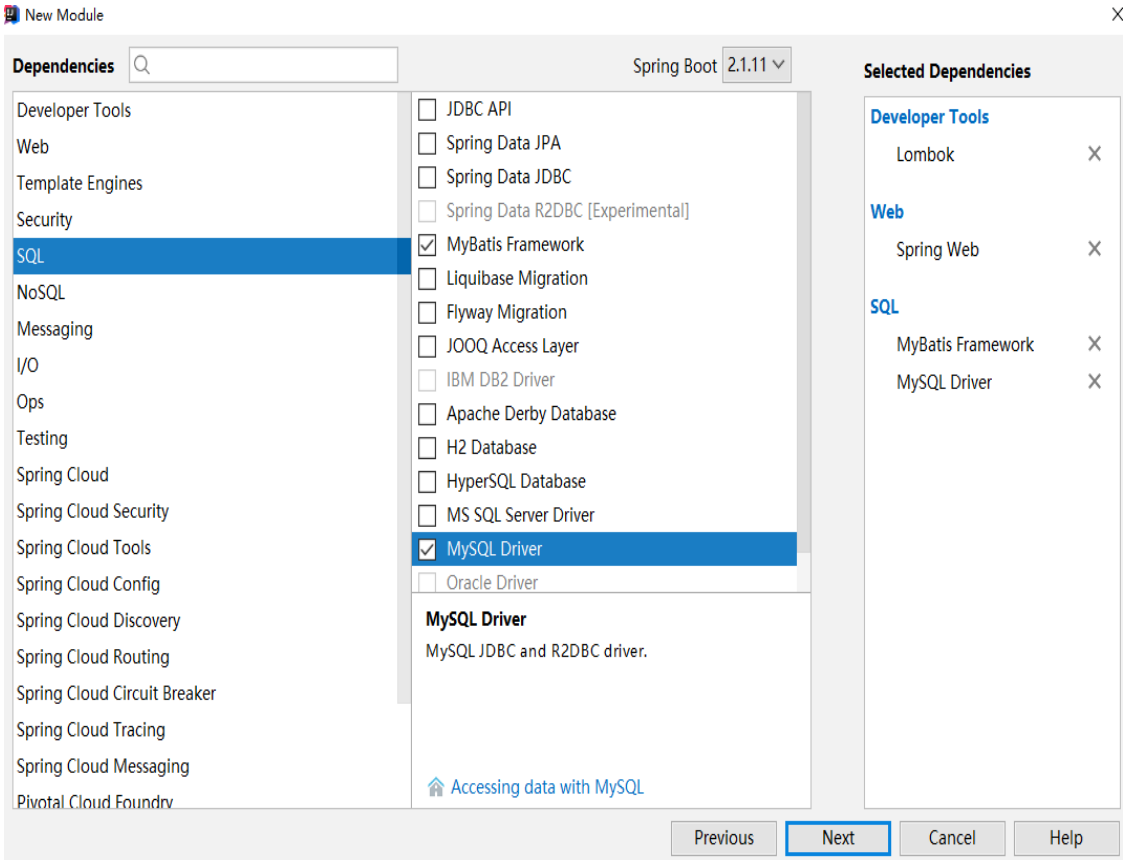
实现过程：

1. new model 创建SpringBoot工程


New Module
JBoss
J2ME
Clouds
Spring
Android
IntelliJ Platform Plugin
Spring Initializr
Maven
Gradle (Kotlin DSL)
Gradle
Groovy
Griffon
Grails
Application Forge
Scala
Kotlin
Static Web
Flash
Empty Project
Module SDK: Project SDK (1.8)
New...
Choose Initializr Service URL.
Default: https://start.spring.io
Custom:
Make sure your network connection is active before continuing.
Previous
Next
Cancel
Help


New Module
Project Metadata
Group: com.itheima
Artifact: provider-service
Type: Maven Project (Generate a Maven based project archive)
Language: Java
Packaging: Jar
Java Version: 8
Version: 0.0.1-SNAPSHOT
Name: provider-service
Description: Demo project for Spring Boot
Package: com.itheima
Previous
Next
Cancel
Help

2．勾选依赖坐标



pom.xml中添加cglib 的BeanCopier工具

```
<!-- cglib的BeanCopier依赖 -->
<dependency>
    <groupId>asm</groupId>
    <artifactId>asm</artifactId>
    <version>3.3.1</version>
</dependency>
<dependency>
    <groupId>asm</groupId>
    <artifactId>asm-commons</artifactId>
    <version>3.3.1</version>
</dependency>
<dependency>
    <groupId>asm</groupId>
    <artifactId>asm-util</artifactId>
    <version>3.3.1</version>
</dependency>
<dependency>
    <groupId>cglib</groupId>
    <artifactId>cglib-nodep</artifactId>
    <version>2.2.2</version>
</dependency>
```

3. 数据库连接信息

```
#db配置
spring:
 datasource:
   url: jdbc:mysql://127.0.0.1:3306/itheima?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
   username: root
   password: root
   driver-class-name: com.mysql.cj.jdbc.Driver
```

4. 创建User表、创建实体User

```sql
DROP TABLE  if EXISTS tb_user;
CREATE TABLE `tb_user` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `username` varchar(100) DEFAULT NULL COMMENT '用户名',
 `password` varchar(100) DEFAULT NULL COMMENT '密码',
 `name` varchar(100) DEFAULT NULL COMMENT '姓名',
 `age` int(11) DEFAULT NULL COMMENT '年龄',
 `sex` int(11) DEFAULT NULL COMMENT '性别，1男，2女',
 `birthday` date DEFAULT NULL COMMENT '出生日期',
 `created` date DEFAULT NULL COMMENT '创建时间',
 `updated` date DEFAULT NULL COMMENT '更新时间',
 `note` varchar(1000) DEFAULT NULL COMMENT '备注',
 PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='用户
信息表';
-- ----------------------------
-- Records of tb_user
-- ----------------------------
INSERT INTO `tb_user` VALUES ('1', 'zhangsan', '123456', '张三',
'13', '1', '2006-08-01', '2019-05-16', '2019-05-16', '张三');
INSERT INTO `tb_user` VALUES ('2', 'lisi', '123456', '李四', '13',
'1', '2006-08-01', '2019-05-16', '2019-05-16', '李四');
-- ----------------------------
-- select tb_user
-- ----------------------------
SELECT * FROM tb_user;
```

实体bean：

```java
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {

   private Integer id;//主键id
   private String username;//用户名
   private String password;//密码
   private String name;//姓名
   private Integer age;//年龄
   private Integer sex;//性别 1男性，2女性
   private Date birthday; //出生日期
   private Date created; //创建时间
   private Date updated; //更新时间
   private String note;//备注

}
```

5. 编写三层架构：dao、service、controller，编写根据id查询用户的方法

```java
/**
 * 用户管理模块的mapper组件接口
 * @author by SangJiacun
 * @Date 2020/8/13 14:28
 */
@Mapper
public interface UserMapper {
    /**
     * 根据用户ID查找用户信息
     * @param id 用户id
     * @return 用户信息
     */
    UserDO findById(Integer id);
}
```

dao类

```java
/**
 * 用户管理模块的dao组件的实现类
 * @author by SangJiacun
 * @Date 2020/8/13 14:43
 */
 @Repository
public class UserDAOImpl implements UserDAO {

    /**
     * 用户管理模块的mapper组件
     */
    @Autowired
    private UserMapper userMapper;

    /**
     * 根据用户ID查找用户信息
     *
     * @param id 用户ID
     * @return 用户信息
     */
    @Override
    public UserDO findById(Integer id) {
        return userMapper.findById(id);
    }
}
```

service类：

```java
/**
 * 用户管理模块的service组件实现类
 * @author by SangJiacun
 * @Date 2020/8/13 14:45
 */
@Service
public class UserServiceImpl implements UserService {
    private final BeanCopier beanCopier =
BeanCopier.create(UserDO.class, UserDTO.class, false);

    /**
     * 用户管理模块的dao组件
     */
    @Autowired
    private UserDAO userDAO;

    /**
     * 根据用户ID查找用户信息
     * @param id 用户ID
     * @return 用户信息
     */
    @Override
    public UserDTO findById(Integer id) {
        UserDTO userDTO = new UserDTO();
        beanCopier.copy(userDAO.findById(id),userDTO,null);
        return userDTO;
    }
}
```

controller：

```java
/**
 * 用户管理模块的controller组件
 * @author by SangJiacun
 * @Date 2020/8/13 14:56
 */
@RestController
@RequestMapping("/user")
public class UserController {
    private final BeanCopier beanCopier =
BeanCopier.create(UserDTO.class, UserVO.class, false);

    /**
     * 用户管理模块的service组件
     */
    @Autowired
    private UserService userService;

    /**
     * 根据id查询用户信息
     * @param id 用户id
     * @return
     */
    @GetMapping("/{id}")
    public UserVO findUserById(@PathVariable Integer id){
        UserVO user = new UserVO();
        beanCopier.copy(userService.findById(id),user,null);
        return  user;
    }
}
```

6. 配置Mapper映射文件

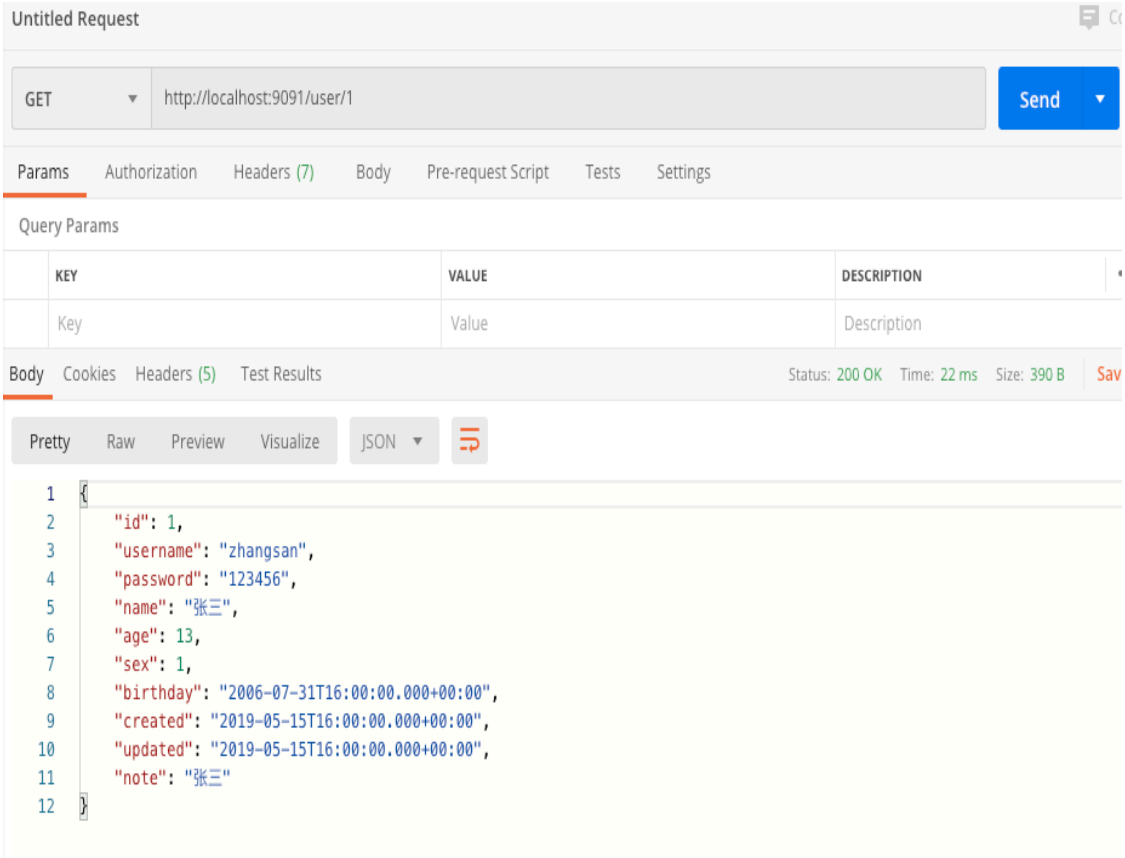
```xml
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.sjc.mapper.UserMapper">

    <select id="findById" resultType="UserDO"
parameterType="integer">
        SELECT * FROM tb_user WHERE  id = #{id}
    </select>

</mapper>
```

7. 在application.yml中添加MyBatis配置，扫描mapper.xml和mapper

```yaml
server:
  port: 9091
#db配置
spring:
  datasource:
    url: jdbc:mysql://127.0.0.1:3306/itheima?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
    username: root
    password: sang,1230
    driver-class-name: com.mysql.cj.jdbc.Driver
#mybatis配置
mybatis:
  #实体位置
  type-aliases-package: com.sjc.domain
  #mapper.xml位置
  mapper-locations: classpath:mapper/*Mapper.xml
```

8. 访问测试地址

http://localhost:9091/user/1



2.3 创建服务消费者(consumer)工程

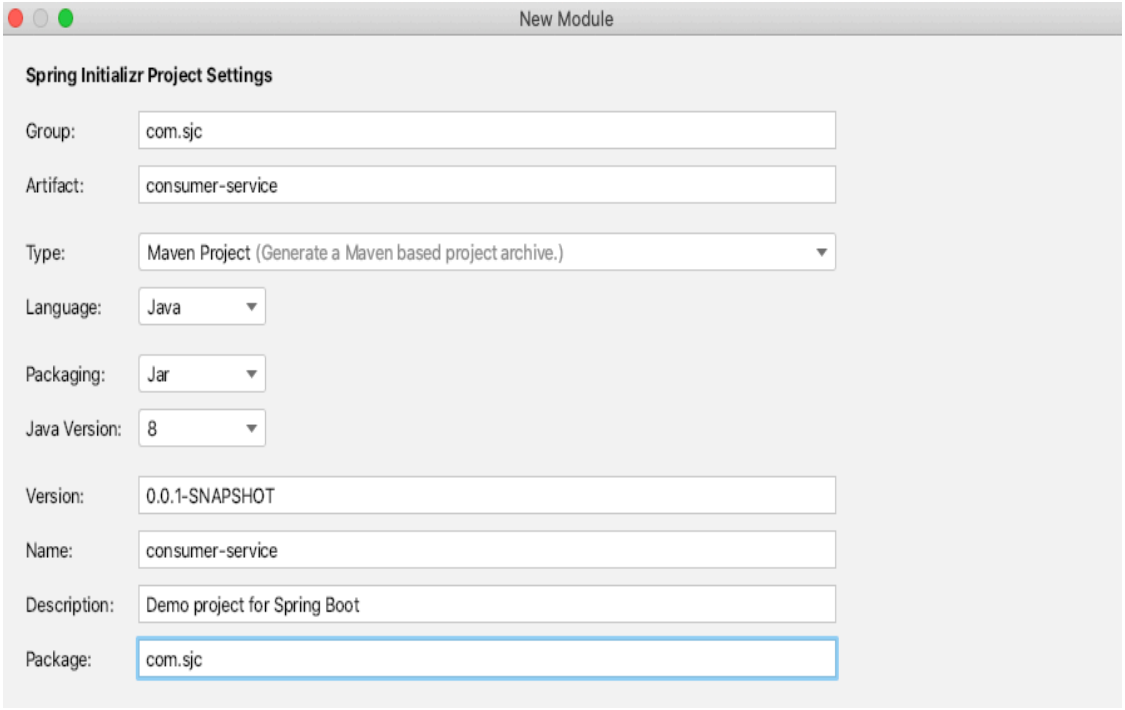
目标： 新建一个项目consumer-service，使用RestTemplate调用生产者的根据id查询用户的服务

实现步骤：

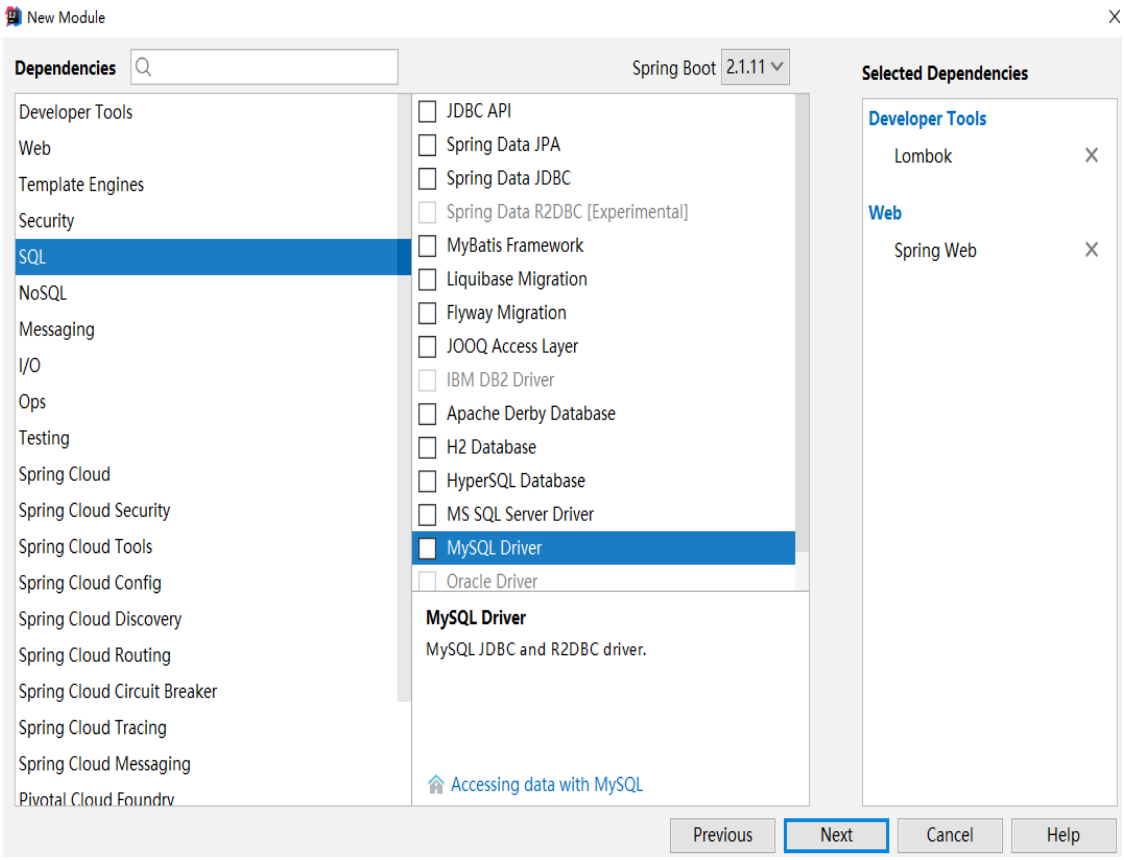
1. 创建消费者SpringBoot工程consumer-service
2. 勾选starter：开发者工具devtools、web
3. 注册http请求客户端对象RestTemplate
4. 编写Controller，用RestTemplate访问服务提供者
5. 启动服务并测试

实现过程：

1. 创建consumer-service的SpringBoot工程


New Module
Spring Initializr Project Settings
Group: com.sjc
Artifact: consumer-service
Type: Maven Project (Generate a Maven based project archive.)
Language: Java
Packaging: Jar
Java Version: 8
Version: 0.0.1-SNAPSHOT
Name: consumer-service
Description: Demo project for Spring Boot
Package: com.sjc

2．勾选需要的相关依赖


New Module
Dependencies
Spring Boot 2.1.11
Selected Dependencies
Developer Tools
Web
Template Engines
Security
SQL
NoSQL
Messaging
I/O
Ops
Testing
Spring Cloud
Spring Cloud Security
Spring Cloud Tools
Spring Cloud Config
Spring Cloud Discovery
Spring Cloud Routing
Spring Cloud Circuit Breaker
Spring Cloud Tracing
Spring Cloud Messaging
Pivotal Cloud Foundry
JDBC API
Spring Data JPA
Spring Data JDBC
Spring Data R2DBC [Experimental]
MyBatis Framework
Liquibase Migration
Flyway Migration
JOOQ Access Layer
IBM DB2 Driver
Apache Derby Database
H2 Database
HyperSQL Database
MS SQL Server Driver
MySQL Driver
Oracle Driver
MySQL Driver
MySQL JDBC and R2DBC driver.
Accessing data with MySQL
Lombok
Spring Web
Previous
Next
Cancel
Help

3. 编写代码

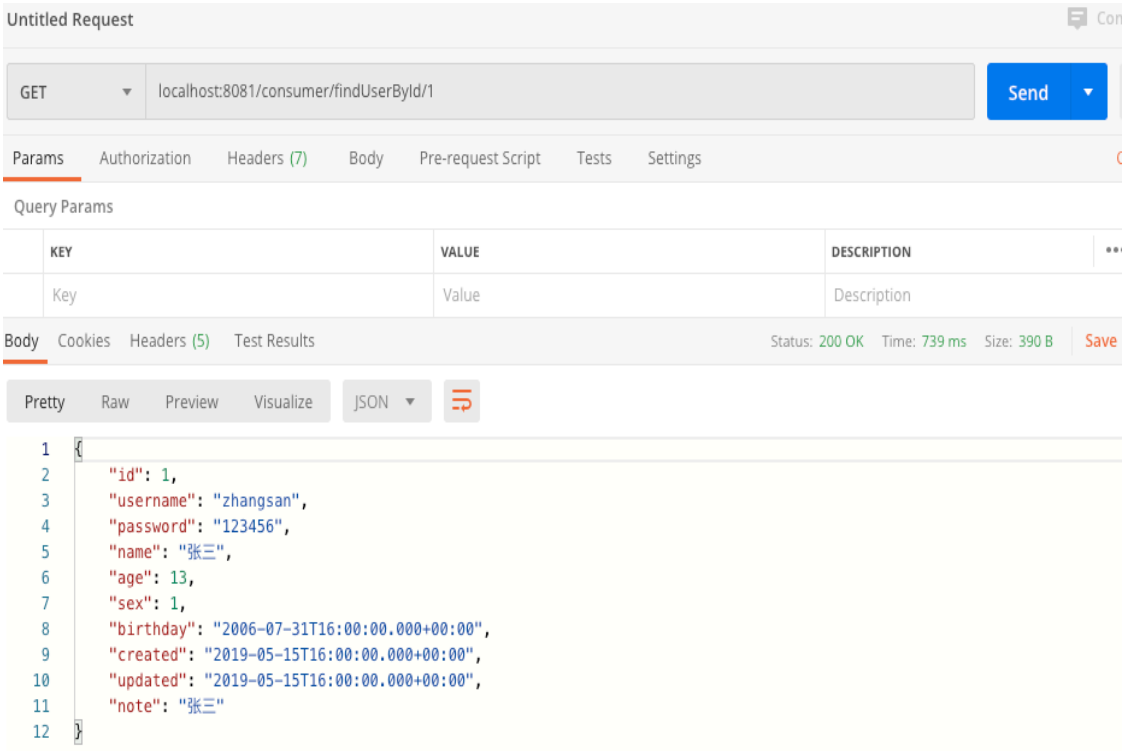
   i. 在启动类中注册RestTemplate

```
@SpringBootApplication
public class ConsumerServiceApplication {

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

    /**
     * 注册RestTemplate
     * @return
     */
    @Bean
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }

}
```

2. 编写ConsumerController，使用RestTemplate远程访问provider-service服务

```java
/**
 * 消费服务的controller组件
 * @author by SangJiacun
 * @Date 2020/8/13 15:15
 */
@RestController
@RequestMapping("/consumer")
public class ConsumerController {
    @Autowired
    private RestTemplate restTemplate;

    /**
     * 发送http请求调用provider服务根据id查找用户的接口
     */
    @RequestMapping("/findUserById/{id}")
    public UserVO findUserById(@PathVariable Integer id){
        String url = "http://127.0.0.1:9091/user/" + id;
        UserVO user = restTemplate.getForObject(url, UserVO.class);
        return user;
    }
}
```

4. 启动并测试，访问：http://localhost:8081/consumer/findUserById/1



2.4 思考问题

- provider-service：对外提供用户查询接口
- consumer-service：通过RestTemplate访问接口查询用户数据

存在的问题：

1. 在服务消费者中，我们把**url地址硬编码**到代码中，不方便后期维护。
2. 在服务消费者中，**不清楚服务提供者的状态**。
3. 服务提供者只有一个服务，即便服务提供者形成集群，服务消费者还需要自己实现**负载均衡**。
4. 服务提供者如果出现故障，能否及时发现，不向用户抛出异常页面?
5. **RestTemplate这种请求调用方式是否还有优化空间?**
6. **多服务权限拦截如何实现？怎么保证服务的可用性?**
7. **配置文件每次都修改好多个是不是很麻烦！？**1千的服务的配置文件
8. ....

其实上面说的部分问题，概括一下就是微服务架构必然面临的一些问题。

- 服务管理：自动注册与发现、状态监管
- 服务负载均衡
- 熔断
- **远程过程调用**
- **网关拦截、路由转发**
- **统一配置修改，及配置实时修改**