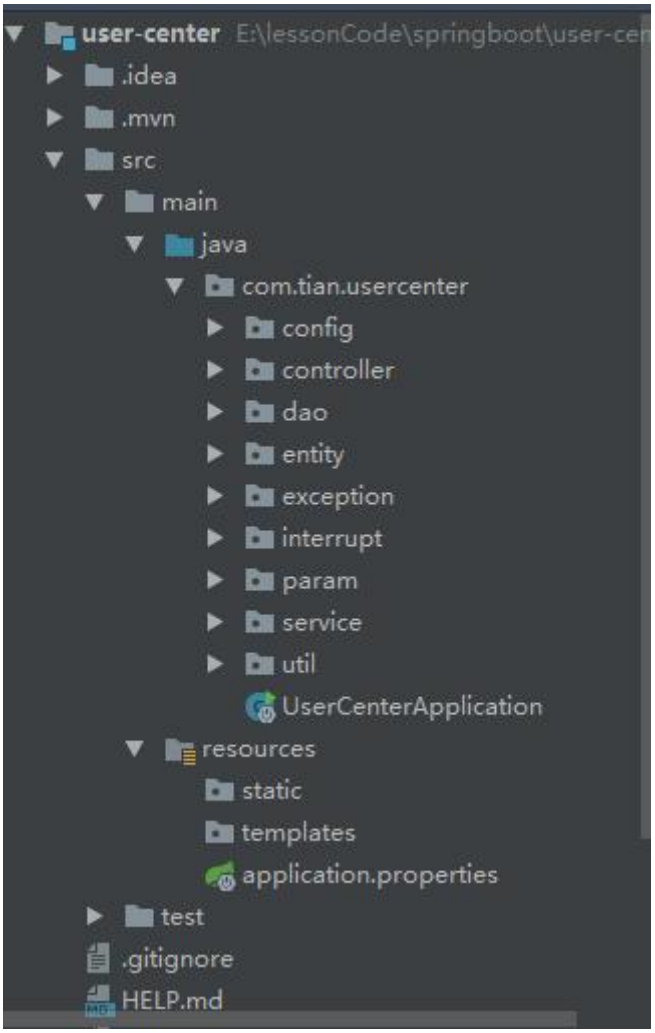
写一个简单的后端项目

首先要清楚后端项目的结构



这些包的作用：

- **controller：此目录主要是存放Controllerde ,比如：UserController.java，也有的项目是把action放在controller目录下，有的是把UserController.java放在action目录下。**
- **service：这里分接口和实现类，接口在service目录下，接口实现类在service/impl目录下。**
- **dao：持久层，目前比较流行的Mybatis或者jpa之类的。**
- **entity：就是数据库表的实体对象。**
- param：放的是请求参数和相应参数UserQueryRequest、BaseResponse等
- util：通常是一些工具类，比如说：DateUtil.java、自定义的StringUtil.java
- interrupt：项目统一拦截处理，比如：登录信息，统一异常处理
- exception：自定义异常，异常错误码
- config：配置读取相关，比如RedisConfig.java

高亮部分比较重要！

一、编译环境

不赘述，之前学习文档有

需要注意的是：环境不仅是pom和yml文件环境，还有数据库环境

二、编写实体类

实体类可以使用pojo的名称或者entity

实体类指代的是数据库中表的映射，所有尽量使用表的名称定义，见名只意方便维护

```
1
2  import lombok.AllArgsConstructor;
3  import lombok.Data;
4  import lombok.NoArgsConstructor;
5  import lombok.ToString;
6
7  @Data
8  @AllArgsConstructor
9  @NoArgsConstructor
10 @ToString
11 public class Employee {
12     private int id;
13     private String name;
14     private int age;
15     private String position;
16 }
```

Java代码	意思
Employee	表的名称
name	表的字段
age	表的字段
position	表的字段
int	表的字段类型为int
String	表的字段类型为varchar或者

三、编译Mapper

Mapper接口可以使用mapper的包名称或者dao

书写所要实现的接口,**注意需要编写Mapper注释注入**

```
1  import org.apache.ibatis.annotations.Mapper;
2
3  import java.util.List;
4
5  @Mapper
6  public interface EmployeeMapper {
7      //根据id查询员工信息
8      Employee selectById(int id);
```

```
    //新增员工信息
    int insert(Employee employee);
    //根据id修改员工信息
    int update(Employee employee);
    //根据id删除员工信息
    int delete(int id);
    List<Employee> selectAll();
}

```

四、编写对应的sql语句

利用上学期所学的知识编写sql语句

需要注意的是参数的占位#{name},利用#或者$加{}的形式，{}内编写对应的变量名称进行占位

```
<?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.employee.mapper.EmployeeMapper">
    <insert id="insert">
        insert into test.employee(id,name,age,position) values(null,#
{name},#{age},#{position})
    </insert>
    <update id="update">
        update test.employee set name = #{name},age = #{age},position = #
{position} where id = #{id}
    </update>
    <select id="selectById" resultType="com.employee.pojo.Employee">
    select * from test.employee where id = #{id}
</select>
    <select id="selectAll" resultType="com.employee.pojo.Employee">
        select * from test.employee
    </select>
    <delete id="delete">
            delete from test.employee where id = #{id}
    </delete>
</mapper>
```

五、测试

不赘述，测试学习文档也有，**测试成功后编写后续步骤**

六、编写Service

复制Mapper中的方法或对Mapper中的方法进行编改，此处无要求则进行复制

```java

public interface EmployeeService {
    //根据id查询员工信息
    Employee selectById(int id);
    //新增员工信息
    int insert(Employee employee);
    //根据id修改员工信息
    int update(Employee employee);
    //根据id删除员工信息
    int delete(int id);
    List<Employee> selectAll();
}

```

七、编译Service的实现

在Service中创建一个Impl包，编译其实现ServiceImpl类

实现类中对需求做具体的分析，如用户登入的接口实现，不应该直接调用接口，要先对数据进行验证，验证通过再调用

需要注意加入Service注释

```java
@Service
public class EmployeeServiceImpl implements EmployeeService {
    @Autowired
    private EmployeeMapper employeeMapper;
    @Override
    public Employee selectById(int id) {
        return employeeMapper.selectById(id);
    }

    @Override
    public int insert(Employee employee) {
        return employeeMapper.insert(employee);
    }

    @Override
    public int update(Employee employee) {
        return employeeMapper.update(employee);
    }


    @Override
    public int delete(int id) {
        return employeeMapper.delete(id);
    }

    @Override
    public List<Employee> selectAll() {
        return employeeMapper.selectAll();
    }
}
```

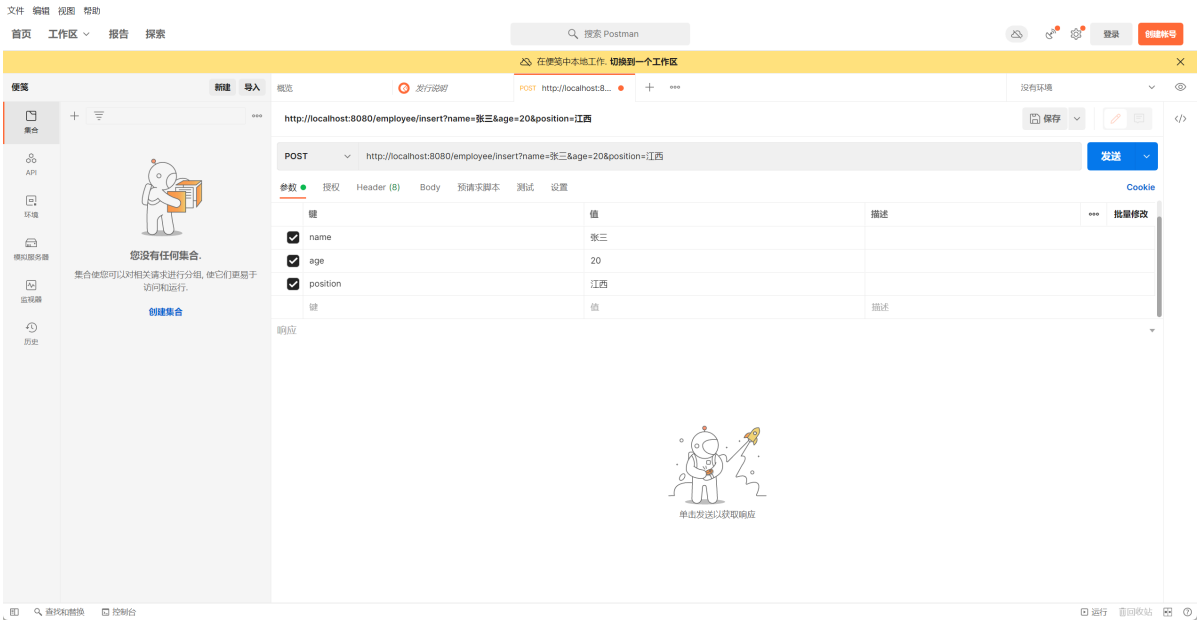
```
31
```

八、编译控制类

Controller用于编写对应的请求映射，**需要注意加入对应的映射注释**

```java
@RestController
@RequestMapping(value = "/employee")
public class EmployeeController {
    @Autowired
    private EmployeeService employeeService;
    //根据id查询员工信息
    @RequestMapping(value = "/selectById/{id}", method = RequestMethod.GET)
    public Employee selectById(@PathVariable("id") int id) {
        return employeeService.selectById(id);
    }
    //新增员工信息
    @RequestMapping(value = "/insert", method = RequestMethod.POST)
    public String  insert(Employee employee) {
        if (employeeService.insert(employee) == 1){
            return "新增员工信息成功";
        }else {
            return "新增员工信息失败";
        }
    }
    //修改员工信息
    @RequestMapping(value = "/update", method = RequestMethod.POST)
    public String update(Employee employee) {
        if (employeeService.update(employee) == 1){
            return "修改员工信息成功";
        }else {
            return "修改员工信息失败";
        }
    }
    //删除员工信息
    @RequestMapping(value = "/delete/{id}", method = RequestMethod.DELETE)
    public String  delete(@PathVariable("id") int id) {
        return employeeService.delete(id) == 1 ? "删除员工信息成功" : "删除员工信息失败";
    }
    //查询所有员工信息
    @RequestMapping(value = "/selectAll", method = RequestMethod.GET)
    public List<Employee> selectAll() {
        return employeeService.selectAll();
    }
}
```

九、再Postman进行测试



十、对接口文档进行编写

可以引入knife4j注解形式，或者自己编写接口文档

```
<dependency>
      <groupId>com.github.xiaoymin</groupId>
      <artifactId>knife4j-spring-boot-starter</artifactId>
      <version>2.0.8</version>
</dependency>
```

```
@Configuration
@EnableSwagger2WebMvc
public class WebMvcConfiguration {
    @Bean
    public Docket api() {
        Docket docket = new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(new ApiInfoBuilder()
                        .title("项目接口文档")
                        .description("这是项目接口文档")
                        .termsOfServiceUrl(".....")
                        .contact(new Contact("张三",
"http:localhost:8080/demo", "xxxx@qq.com"))
                        .license("...")
                        .licenseUrl("...")
                        .version("1.0")
                        .build())
                .groupName("项目接口文档")
                .select()
                /*
                 * RequestHandlerSelectors
                 *      .any() // 扫描全部的接口，默认
                 *      .none() // 全部不扫描
                 *      .basePackage("com.mike.server") // 扫描指定包下的接口，
最为常用
```

```
                 *      .withClassAnnotation(RestController.class) // 扫描带有指定注解的类下所有接口
                 *      .withMethodAnnotation(PostMapping.class) // 扫描带有只当注解的方法接口
                 */
                .apis(RequestHandlerSelectors.basePackage("com.example"))
                .paths(PathSelectors.any())
                .build();
        return docket;
    }
}
```

按Ctrl点击后可跳转

Knide4j使用教程：Knife4j-的使用(详细教程)_knife4j使用-CSDN博客

Knide4j整合视频：SpringBoot 整合 Knife4j教学视频*哔哩哔哩*bilibili