服务器端搭建

SSM框架：Spring+SpringMVC+MyBatis SpringBoot工具

创建所有项目都是基于maven创建。 maven项目管理(清理-编译-测试-打包-部署)和依赖(jar)管理的工具。

> 解压maven

```
D:\tools\apache-maven-3.8.8
```

> 配置maven

打开D:\tools\apache-maven-3.8.8\conf\settings.xml

```
-- 本地仓库    首先创建一个本地仓库文件夹 D:\tools\repository2
<!-- 配置maven的本地仓库的地址 -->
<localRepository>D:\tools\repository2</localRepository>

-- 下载jar服务器镜像改为国内
<mirror>
    <id>aliyunmaven</id>
    <mirrorOf>*</mirrorOf>
    <name>阿里云公共仓库</name>
    <url>https://maven.aliyun.com/repository/public</url>
</mirror>
```

> maven集成Idea

pom.xml

```
<?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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <!-- 当前项目坐标 -->
    <groupId>com.haiyang</groupId>
    <artifactId>mall-server-021</artifactId>
    <version>1.0.0</version>

    <packaging>jar</packaging>

    <!-- 运行参数 -->
    <properties>
        <java.version>1.8</java.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <!-- springboot版本 -->
        <spring-boot.version>2.3.7.RELEASE</spring-boot.version>
    </properties>

    <!-- 依赖 -->
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!-- MySQL数据库驱动-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <!-- lombok插件-->
        <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>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </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>3.8.1</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>2.3.7.RELEASE</version>
            </plugin>
        </plugins>
    </build>
</project>
```

application.yaml

```
server:
  port: 10001
  servlet:
    context-path: /
```

```
注意：
    yaml文件配置属性节点层次必须对其。
    属性值前面必须有一个空格

项目启动，浏览器访问：
http://localhost:10001/index

如果设置 context-path: /api
http://localhost:10001/api/index
```

> 启动类

包：包名全部小写。包是分层级。

```
例如：com.haiyang
对应物理磁盘上就是文件夹。

包名--域名倒置   cctv.com   baidu.com
```

分层开发：

```
Vue界面
    |
控制层：  处理前端发来的请求. Vue请求( 登录 )。    com.haiyang.controller
    |
业务逻辑层   项目具体功能和流程   登录方法    com.haiyang.service
    |
数据访问层：  对数据库CRUD操作        com.haiyang.mapper
    |
MySQL
```

```
package com.haiyang;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MallServer021Application {
    public static void main(String[] args) {
        SpringApplication.run(MallServer021Application.class, args);
    }
}
```

> 创建IndexController控制器

```
package com.haiyang.controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController   //当前类是一个控制器
@RequestMapping("/index")
public class IndexController {
```

```
    //处理请求方法
    //方法返回数据  就是响应给前端的数据
    @RequestMapping("/test")
    public String test(){
        return "Hello SpringBoot";
    }
}
```

实体类

封装数据:

```
MySQL sys_account表 -----查询account_id是19988999988----->实体类对象
      sys_account表-------------所有用户--------------->List<实体类>    泛型

Account       sys_account
实体类<--映射-->表
变量  --------- 字段
```

```
---实体类代码结构---
私有变量 -- 映射表中字段
无参构造方法
变量set和get方法
```

```
package com.haiyang.entity;
import lombok.Data;

@Data
public class Account {
    private String accountId;
    private String password;
    private String accountName;
}
```

代码逆向生成

```
<dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
   <version>5.1.49</version>
</dependency>
```

编写控制器的父类

```
package com.haiyang.common;
//控制器的父类
public class BaseController {

}
```

编写实体类父类

```
package com.haiyang.common;
import com.baomidou.mybatisplus.annotation.TableField;
import java.time.LocalDateTime;
@Data
public class BaseEntity {
    /**
     * 创建时间
     */
    @TableField("created")
    private LocalDateTime created;

    /**
     * 修改时间
     */
    @TableField("updated")
    private LocalDateTime updated;

    @TableField("statu")
    private Integer statu;
}
```

> 配置mybatis-plus框架

导入mybatisPlus逆向代码生成的相关jar包，修改pom.xml

```
<!--  mybatis-plus -->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.4.2</version>
</dependency>
<!-- 代码生成器 -->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-generator</artifactId>
    <version>3.4.1</version>
</dependency>
<!-- 代码自动生成器模板依赖-->
<dependency>
    <groupId>org.apache.velocity</groupId>
    <artifactId>velocity-engine-core</artifactId>
    <version>2.2</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-freemarker</artifactId>
</dependency>
```

配置springboot项目，整合myBatisPlus，修改application.yaml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/system?useUnicode=true&characterEncoding=utf-
8&useSSL=false&serverTimezone=GMT%2B8
    username: root
    password: root
```

修改CategroyMapper接口

```
加上@Mapper
package com.haiyang.mapper;
import com.haiyang.entity.Category;
import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import org.apache.ibatis.annotations.Mapper;

@Mapper
public interface CategoryMapper extends BaseMapper<Category> {

}
```

编写CategoryController中测试方法

```
package com.haiyang.controller;
import com.haiyang.entity.Category;
import com.haiyang.service.CategoryService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;
import com.haiyang.common.BaseController;

import java.util.List;

@RestController
@RequestMapping("/category")
public class CategoryController extends BaseController {
    //需要创建CategoryService对象
    //@Autowired Spring框架自动创建好该对象
    @Autowired
    private CategoryService cService;

    //处理 前端请求所有商家分类的请求
    @RequestMapping("/list")
    public List<Category> list(){
        //获得所有的商家分类数据  sys_category表中数据

        //返回 List<Category>
        return cService.list();
    }
}
```

```
访问地址:http://localhost:10001/category/list
```

定义接口通用返回类型

所有的Controller控制器中的方法都是返回Result类

```
返回类代码结构：
Result.java
1、code 前端请求服务器端状态码    20000成功   30001          30002
2、message    请求操作提示信息      执行成功     用户名不存在     密码错误
3、Object     返回数据，Object存储就是返回数据对象，实体类对象，List集合对象。
    动态原理：  父类是可以引用子类对象。
```

Result.java

```
package com.haiyang.common;
import lombok.Data;

@Data
public class Result {
    private Integer code;  //请求操作是否成功状态码 20000成功
    private String message;   //请求操作提示信息。
    private Object resultdata;  //如果请求进行是查询操作，存储就查询返回数据对象。List集合,实体
类。

    //成功：   重载
    public static Result success(int code,String message,Object data){
        Result r = new Result();
        r.setCode(code);
        r.setMessage(message);
        r.setResultdata(data);
        return r;
    }

    public static Result success(Object data){
        return success(20000,"操作成功",data);
    }

    //失败：
    public static Result fail(int code,String message,Object data){
        Result r = new Result();
        r.setCode(code);
        r.setMessage(message);
        r.setResultdata(data);
        return r;
    }
    public static Result fail(String message){
        return fail(400,message,null);
    }
}
```

SpringBoot项目热部署

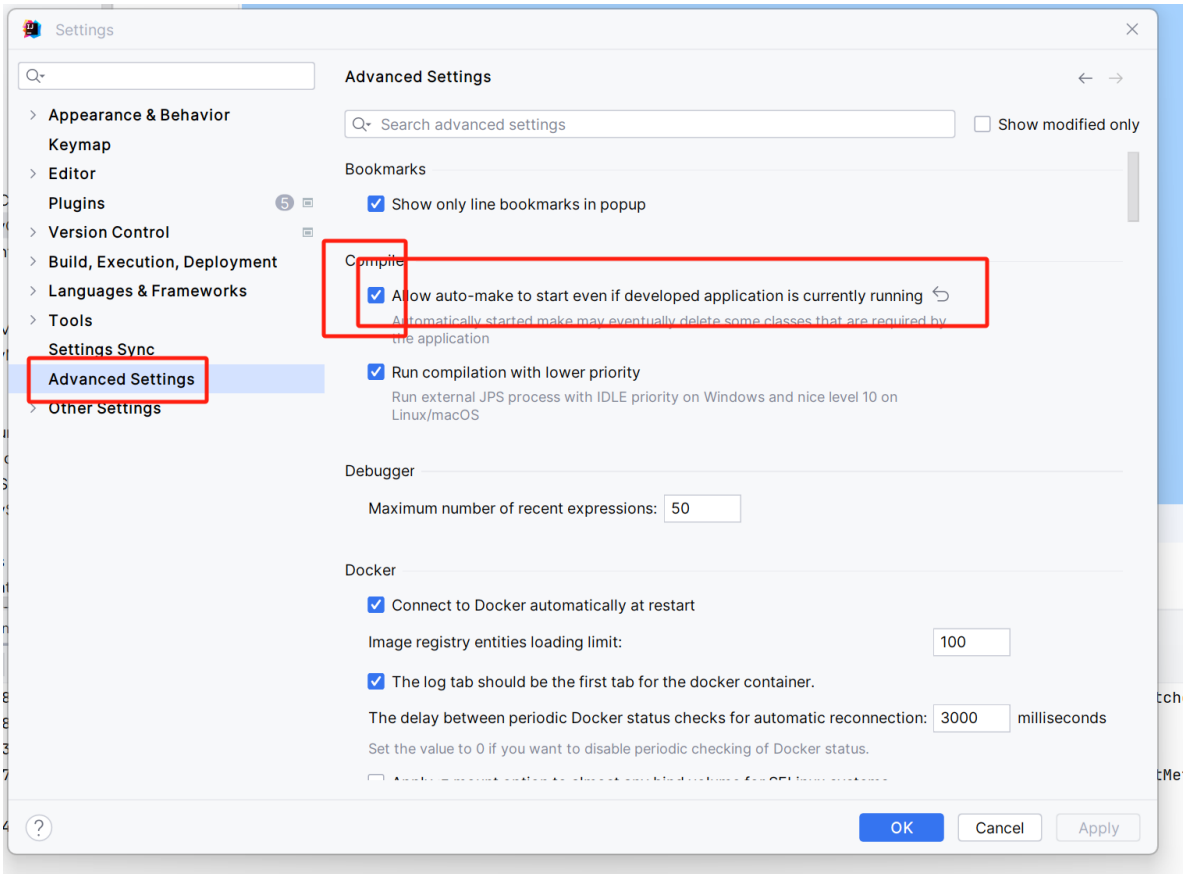
修改代码不需要手动重启服务器（自动）。

首先需要修改pom.xml文件，添加依赖

```
 <!-- 热部署 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

配置Idea自动编译

路径：File-> Settings -> Build,Execution,Deployment -> Compiler -> Build project automatically



注意： 配置完毕，将服务器先手动重启一下，以后修改代码变为热部署（自动重启）。

MyBatis配置类

```
Spring框架配置：
1、XML配置，所有的配置都是以.xml文件为主。
2、基于注解配置，所有的配置使用注解完成。
    @Autowired 注解，自动装配
3、Java配置，所有配置以一个类为主（配置类 -- 方法[配置对象信息]）。
    类 @Configuration
            方法 @Bean  -->创建以类对象

Spring核心概念：
    依赖注入，通过Ioc容器。解耦
    面向切面，通过AOP实现。
```

```
目前项目：
    @RestController
    CategoryController
        |                          @Service
    CategoryService接口 ----实现---- CategoryServiceImpl实现类（生成）
        |   @Mapper
    CatgoryMapper接口（生成）

springIoc可以将三层中 控制层、业务逻辑层、数据访问层对象全部自动创建好，自动创建好的对象存到Ioc容器（内存）。
```

添加MyBatisPlus配置类，创建一个分页拦截器对象到 Ioc容器

```
package com.haiyang.config;
import com.baomidou.mybatisplus.annotation.DbType;
import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import com.baomidou.mybatisplus.extension.plugins.inner.BlockAttackInnerInterceptor;
import com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
@MapperScan("com.haiyang.mapper")  //所有Mapper接口就不需要手动添加@Mapper
public class MyBatisPlusConfig {

    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor(){
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();
        //分页  方言
        interceptor.addInnerInterceptor( new PaginationInnerInterceptor( DbType.MYSQL
));
        //防止全表更新和删除
        interceptor.addInnerInterceptor( new BlockAttackInnerInterceptor());
        return interceptor;
    }
}
```

加入密码加密工具类

首先配置pom.xml，导入依赖

```
<!-- MD5加密 -->
<dependency>
    <groupId>commons-codec</groupId>
    <artifactId>commons-codec</artifactId>
    <version>1.15</version>
</dependency>
```

创建一个MD5加密的工具类

```
package com.haiyang.utils;
import org.apache.commons.codec.digest.DigestUtils;
public class MD5Utils {
    //调用MD5加密
    public static String md5(String str){
        return DigestUtils.md5Hex(str);
    }
    //加密字符 基数
    private static final String privateKey = "1xc2d34f3p";

    //123123 ，密码加工
    public static String inputPassToNewPass(String pass){
        String newPass =
privateKey.charAt(0)+privateKey.charAt(1)+pass+privateKey.charAt(5)+"";
        return newPass; //1x1231233
    }

    public static void main(String[] args) {
        System.out.println(MD5Utils.md5("123123"));
    }
}
```

登录功能的实现

编写服务器端 AccountController中的login方法

```
package com.haiyang.controller;
import com.haiyang.common.Result;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;
import com.haiyang.common.BaseController;

@RestController
@RequestMapping("/account")
@Slf4j
public class AccountController extends BaseController {

    @PostMapping("/login")
    public Result login(String accountId,String password){
        log.info("手机号为 {} 用户正在登录APP端--",accountId);

        return Result.success(null);
    }
}
```

密码加密之后如何对比

```
注册:
    原始123123 ---->录入数据库---->加密---->4297f44b13955235245b2497399d7a93
登录:
    页面输入原文123123---->先加密(4297f44b13955235245b2497399d7a93)---->用密文对比密文<----
--4297f44b13955235245b2497399d7a93
```

```
@PostMapping("/login")
    public Result login(String accountId,String password){
        log.info("手机号为 {} 用户正在登录APP端--",accountId);

        //手机号在sys_account表中是主键，是不重复的，查询返回就是单一对象
        Account account = aService.getOne(new QueryWrapper<Account>
().eq("account_id",accountId));

        if(account==null){
           return Result.fail("账户手机号码不存在");
        }else{
            //account不等于null，说明查询到该手机号码，继续比较密码是否一致
            String newPwd = MD5Utils.md5(password);  //将登录原文密码 先加密，变为 密文
            if(newPwd.equals(account.getPassword())){
                if(account.getStatu() == 0){
                    return Result.fail("该账户被禁用或被注销，暂不可用");
                }else{
                    //登录成功，直接反馈登录账户对象信息
                    return Result.success(account);
                }
            }else{
                return Result.fail("登录密码不正确");
```

```
            }
        }
    }
```

MyBatisPlus

MyBatisPlus常规查询是不需要输入SQL，定义查询条件(QueryWrapper)，根据条件自动生成执行SQL。

service层次：

`getOne()`	查询单一对象，返回是一个Java对象。 根据主键作为查询条件，进行查询返回的都是单一对象。