规范化的用户中心开发

> 此系统旨在完成一套规范化的，易于扩展的用户中心项目，此项目可以作为各个项目的后台系统

技术选型

前端：Ant Design Pro

后端：SpringBoot + SSM + MyBatisPlus

初始化项目

前端初始化

```
E:\项目文件\用户中心>pro create myapp
? 🐑 使用 umi@4 还是 umi@3 ? umi@3
? 🚀 要全量的还是一个简单的脚手架? simple
> clone repo url: https://gitee.com/ant-design/ant-design-pro
Cloning into 'myapp'...
remote: Enumerating objects: 208, done.
remote: Counting objects: 100% (208/208), done.
remote: Compressing objects: 100% (180/180), done.
remote: Total 208 (delta 33), reused 116 (delta 23), pack-reused 0Receiving
objects:  88% (184/208)
Receiving objects: 100% (208/208), 118.44 KiB | 310.00 KiB/s, done.
Resolving deltas: 100% (33/33), done.
> 🚚 clone success
> Clean up...

No change to package.json was detected. No package manager install will be
executed.
```

> 进入项目执行yarn命令或者npm install命令安装对应的依赖

> 接下来使用执行start启动脚本，运行前端项目

image-20230423104013887-1682217621921-1

image-20230423103936467

> 开启umi ui，自动生成界面
>
> yarn add @umijs/preset-ui -D

image-20230423104401202-1682217844640-3

> 注意这一步，需要开启梯子进行添加页面，因为其本质就是从github上拉取代码

image-20230423111537718

后端初始化

> 使用idea创建SpringBoot项目

image-20230423115251014-1682221973732-5

```
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>2.3.0</version>
        </dependency>
        <!--热部署-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <!--数据库连接-->
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <!--数据层框架-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.5.2</version>
        </dependency>
        <!--配置文件注释-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-processor</artifactId>
            <optional>true</optional>
        </dependency>
        <!--工具类-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <!--测试-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

> 编写yml配置文件

```
spring:
  application:
    name: usercenter-backend
  # 配置数据库连接信息
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/user_center_backend
    username: root
    password: root
server:
  port: 8080
mybatis-plus:
  configuration:
    # 配置日志信息
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
    # 关闭驼峰映射
    map-underscore-to-camel-case: false
```

用户数据库表设计

字段	属性	备注
id	bigint	主键
userName	varchar	用户名
userAccount	varchar	用户账号
gender	tinyint	性别
userPassword	varchar	密码
phone	varchar	电话
email	varchar	邮箱
userStatus	tinyint	状态（1 有效 0 禁用）
avatarUrl	varchar	头像
createTime	datetime	创建时间
updateTime	datetime	更新时间
isDelete	tinyint	是否删除（1 未删除 0 删除）

> 最后三个字段属于通用字段，适用于任何一张表

> 这里userName和userPassword和userStatus之所以这样命名，是避免和MySQL的关键词重复

```
create table user
(
    id bigint auto_increment comment '主键',
    userName varchar(255) not null comment '用户名',
    userPassword varchar(255) not null,
    userAccount varchar(255) null comment '用户账号',
    gender tinyint default 1 not null comment '性别 1-男 0-女',
    phone varchar(255) null comment '电话',
    email varchar(255) null comment '邮箱',
    userStatus tinyint default 1 null comment '状态 1-启用 0-禁用',
    avatarUrl varchar(1024) null comment '头像',
    updateTime datetime default CURRENT_TIMESTAMP not null,
    createTime datetime default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP not
null,
    isDelete tinyint default 1 not null comment '逻辑删除 1-未删除 0-删除',
    constraint user_pk
        primary key (id)
)
    comment '用户表';
```

一个基本的项目目录

image-20230424201551843-1682338555067-1

使用MyBatisX插件自动生成相关文件

image-20230424201651384

image-20230424201708767

> 使用MyBatisX插件可以快速生成对应的model、service、mapper文件，提高开发效率
>
> 接下来测试生成的文件是否可用

```
@SpringBootTest
public class UserServiceTest {
    @Resource
    private UserService userService;
    @Test
    public void testAddUser(){
        User user = new User();
        user.setUserName("object");
        user.setUserPassword("object");
        user.setUserAccount("object");

        boolean save = userService.save(user);
        assertTrue(save);
    }
}
```

业务逻辑编写

注册逻辑

1. 用户在前端输入账户、密码、确认密码、校验码（code）
2. 校验账户、密码是否符合要求
   1. 账户不能小于6位
   2. 密码不能小于6位，不能有特殊字符
   3. 密码和确认密码是否一致
   4. 账户不能重复
   5. 账户不含特殊字符
3. 对密码进行加密存储（**密码不能明文存储在数据库中，如果数据库泄露等事件发生，会很危险**）
4. 向数据库中插入数据

> 编写请求体，封装请求数据

```
/**
 * 注册请求体
 * @author ObjectY
 */
@Data
public class RegisterRequest {
    String userAccount;
    String userPassword;
    String checkPassword;
}
```

> 导入commons-lang3工具包，辅助我们做账户的校验

```
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
        </dependency>
```

```
/**
* @author ObjectY
* @description 针对表【user(用户表)】的数据库操作Service实现
* @createDate 2023-04-24 20:16:55
*/
@Service
public class UserServiceImpl extends ServiceImpl<UserMapper, User>
    implements UserService{
```

```
    private static final String PASSWORD_REGEX = "^[a-zA-Z0-9]{6,10}$";

    private static final String SALT = "Object";

    @Override
    public Long userRegister(RegisterRequest registerRequest) {
        String userAccount = registerRequest.getUserAccount();
        String userPassword = registerRequest.getUserPassword();
        String checkPassword = registerRequest.getCheckPassword();
        //非空判断
        if(StringUtils.isAnyBlank(userAccount,userPassword,checkPassword)){
            return -1L;
        }
        //账号和密码不能包含特殊字符(只能使用大小写字母和数字)
        Pattern pattern = Pattern.compile(PASSWORD_REGEX);
        Matcher matcher = pattern.matcher(userPassword);
        Matcher matcher2 = pattern.matcher(userAccount);
        if(!matcher.matches() || !matcher2.matches()){
            return -1L;
        }
        //账户不能重复
        LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
        wrapper.eq(User::getUserAccount, userAccount);
        User user = this.getOne(wrapper);
        if(!Objects.isNull(user)){
            return -1L;
        }

        //给密码进行加密
        String newPassword = DigestUtils.md5DigestAsHex((SALT +
userPassword).getBytes(StandardCharsets.UTF_8));

        //存储用户
        User currentUser = new User();
        currentUser.setUserName(userAccount);
        currentUser.setUserAccount(userAccount);
        currentUser.setUserPassword(newPassword);
        boolean save = this.save(currentUser);
        if(!save){
            return -1L;
        }
        return 1L;
    }
}
```

> 下面是对此代码的测试

```

    @Test
    public void testRegister(){
        //密码为空
        RegisterRequest registerRequest = new RegisterRequest();
        registerRequest.setUserAccount("object");
        registerRequest.setUserPassword("");
        registerRequest.setCheckPassword("1234567");
        assertEquals(-1L, userService.userRegister(registerRequest));
        //账户长度小于4位
        registerRequest.setUserAccount("obj");
        registerRequest.setUserPassword("");
        registerRequest.setCheckPassword("1234567");
        assertEquals(-1L, userService.userRegister(registerRequest));
        //密码长度小于6位
        registerRequest.setUserAccount("object");
        registerRequest.setUserPassword("12345");
        registerRequest.setCheckPassword("12345");
        assertEquals(-1L, userService.userRegister(registerRequest));
        //两次密码不一致
        registerRequest.setUserAccount("object");
        registerRequest.setUserPassword("12345678");
        registerRequest.setCheckPassword("1234567");
        assertEquals(-1L, userService.userRegister(registerRequest));
        //账户名不能重复
        registerRequest.setUserAccount("object");
        registerRequest.setUserPassword("1234567");
        registerRequest.setCheckPassword("1234567");
        assertEquals(-1L, userService.userRegister(registerRequest));
        //密码包含特殊字符
        registerRequest.setUserAccount("object");
        registerRequest.setUserPassword("1234567(");
        registerRequest.setCheckPassword("1234567(");
        assertEquals(-1L, userService.userRegister(registerRequest));
        //账户包含特殊字符
        registerRequest.setUserAccount("object**");
        registerRequest.setUserPassword("1234567");
        registerRequest.setCheckPassword("1234567");
        assertEquals(-1L, userService.userRegister(registerRequest));
        //注册成功
        registerRequest.setUserAccount("object123");
        registerRequest.setUserPassword("1234567");
        registerRequest.setCheckPassword("1234567");
        assertEquals(1L, userService.userRegister(registerRequest));
    }
```

登录逻辑编写（目前是单机登录 后续改成分布式登录）

1. 用户在前端输入账号和密码
2. 后端对账号和密码进行校验
3. 根据账号查询出用户信息
4. 对用户信息进行脱敏
5. 存储用户的登录态（session)
6. 返回脱敏后的用户信息

> 这里需要注意的是我们存在一个逻辑删除字段，所以需要如果删除的用户是不需要去查询出来的，并且使用了session去记录登录态，所以要配置一下session的过期时间

```
      # 配置逻辑删除
      logic-delete-value: 0
      logic-not-delete-value: 1

        session:
         timeout: 86400
```

```
    /**
     * 逻辑删除 1-未删除 0-删除
     */
    @TableLogic
    private Integer isDelete;
```

> 接下来是代码编写

```
    @Override
    public User userLogin(LoginRequest loginRequest, HttpServletRequest request) {
        String userAccount = loginRequest.getUserAccount();
        String userPassword = loginRequest.getUserPassword();

        // 基本的校验
        if(StringUtils.isAnyBlank(userAccount,userPassword)){
            return null;
        }
        Pattern pattern = Pattern.compile(PASSWORD_REGEX);
        Matcher matcher = pattern.matcher(userPassword);
        Matcher matcher2 = pattern.matcher(userAccount);
        if(!matcher.matches() || !matcher2.matches()){
            return null;
        }
        // 查询用户信息
        LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
        wrapper.eq(User::getUserAccount, userAccount);
        User user = this.getOne(wrapper);
        if(Objects.isNull(user)){
            return null;
        }
        //判断密码是否相等
        if(!DigestUtils.md5DigestAsHex((SALT +
userPassword).getBytes(StandardCharsets.UTF_8)).equals(user.getUserPassword())){
            return null;
        }
        //用户信息脱敏
        User safetyUser = getSafetyUser(user);

        //存储用户登录态
        request.getSession().setAttribute(USER_LOGIN_STATE, safetyUser);
        return safetyUser;
    }

    /**
     * 用户信息脱敏方法
     * @param user 原用户
     * @return 安全用户
     */
    private User getSafetyUser(User user) {
        User safetyUser = new User();
        safetyUser.setId(user.getId());
        safetyUser.setUserName(user.getUserName());
        safetyUser.setUserPassword(null);
        safetyUser.setUserAccount(user.getUserAccount());
        safetyUser.setGender(user.getGender());
        safetyUser.setPhone(user.getPhone());
        safetyUser.setEmail(user.getEmail());
        safetyUser.setUserStatus(user.getUserStatus());
        safetyUser.setAvatarUrl(user.getAvatarUrl());
        safetyUser.setUpdateTime(user.getCreateTime());

        return safetyUser;
    }
```

Controller层编写

```
/**
 * 用户接口层
 * @author ObjectY
 */
@RestController
@RequestMapping("/user")
public class UserController {
    @Resource
    private UserService userService;

    @PostMapping("/register")
    public Long register(@RequestBody RegisterRequest registerRequest){
        //这里重新进行校验是保证service的可复用性，service方法不止一个controller会进行
调用
        if(Objects.isNull(registerRequest)){
            return -1L;
        }
        String userAccount = registerRequest.getUserAccount();
        String userPassword = registerRequest.getUserPassword();
        String checkPassword = registerRequest.getCheckPassword();
        if(StringUtils.isAnyBlank(userAccount, userPassword,checkPassword)){
            return -1L;
        }

        return userService.userRegister(registerRequest);
    }

    @PostMapping("/login")
    public User login(@RequestBody LoginRequest loginRequest, HttpServletRequest
request){

        if(Objects.isNull(loginRequest)){
            return null;
        }
        String userAccount = loginRequest.getUserAccount();
        String userPassword = loginRequest.getUserPassword();
        if(StringUtils.isAnyBlank(userAccount, userPassword)){
            return null;
        }

        return userService.userLogin(loginRequest, request);
    }
}

    @PostMapping("/login")
    public User login(@RequestBody LoginRequest loginRequest, HttpServletRequest
request){

        if(Objects.isNull(loginRequest)){
            return null;
        }
        String userAccount = loginRequest.getUserAccount();
        String userPassword = loginRequest.getUserPassword();
        if(StringUtils.isAnyBlank(userAccount, userPassword)){
            return null;
        }

        return userService.userLogin(loginRequest, request);
    }

    /**
     * 查询用户
     * @param userName 用户名
     * @return 用户列表
     */
    @GetMapping("/search")
    public List<User> searchUsers(String userName){
        LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
        wrapper.like(User::getUserName, userName);

        return userService.list(wrapper);
    }
```

测试接口

> idea中自带接口测试的工具，可以右键 -> 新建HTTP请求文件

```
POST http://localhost:8080/user/login
Content-Type: application/json
{
  "userAccount": "object123",
  "userPassword": "1234567"
}
```

```
POST http://localhost:8080/user/register
Content-Type: application/json
{
  "userAccount": "object1234",
  "userPassword": "1234567",
  "checkPassword": "1234567"
}
```

补充字段

> 补充一个userRole字段，用来做用户身份校验，判断用户是否是管理员

```
    private Integer userRole;
```

> 接下来修改用户脱敏方法，返回用户角色信息

```
    /**
     * 用户信息脱敏方法
     * @param user 原用户
     * @return 安全用户
     */
    private User getSafetyUser(User user) {
        User safetyUser = new User();
        safetyUser.setId(user.getId());
        safetyUser.setUserName(user.getUserName());
        safetyUser.setUserPassword(null);
        safetyUser.setUserAccount(user.getUserAccount());
        safetyUser.setGender(user.getGender());
        safetyUser.setPhone(user.getPhone());
        safetyUser.setEmail(user.getEmail());
        safetyUser.setUserStatus(user.getUserStatus());
        safetyUser.setAvatarUrl(user.getAvatarUrl());
        safetyUser.setUpdateTime(user.getCreateTime());
        safetyUser.setUserRole(user.getUserRole());

        return safetyUser;
    }
```

编写管理员可访问接口

> 补充常量类，存放用到的所用常量

```
/**
 * 用户常量类
 * @author ObjectY
 */
public interface UserConstant {

    String PASSWORD_REGEX = "^[a-zA-Z0-9]{6,10}$";

    String SALT = "Object";

    String USER_LOGIN_STATE = "userLoginState";
}
```

> 编写通用方法，判断当前用户是否是管理员

```
/**
 * 通用方法类
 * @author ObjectY
 */
public class CommonUtils {

    public static boolean checkAdmin(HttpServletRequest request){
        //获取登录用户信息
        User loginUser = (User)
request.getSession(false).getAttribute(USER_LOGIN_STATE);
        //判断用户是否有权限访问此接口
        if(Objects.isNull(loginUser)){
            return false;
        }
        if(!loginUser.getUserRole().equals(ADMIN_ROLE)){
            return false;
        }
        return true;
    }
}
```

> 最后编写接口即可

```
    /**
     * 查询用户
     * @param userName 用户名
     * @param request request
     * @return 用户列表
     */
    @GetMapping("/search")
    public List<User> searchUsers(String userName,HttpServletRequest request){
        if(!CommonUtils.checkAdmin(request)){
            return new ArrayList<>();
        }
        LambdaQueryWrapper<User> wrapper = new LambdaQueryWrapper<>();
        wrapper.like(User::getUserName, userName);

        return userService.list(wrapper).stream().map(user ->
userService.getSafetyUser(user)).collect(Collectors.toList());
    }

    /**
     * 删除用户
     * @param id 用户id
     * @param request request
     * @return bool
     */
    @PostMapping("/delete")
    public Boolean delete(@RequestBody Long id,HttpServletRequest request){
        if(!CommonUtils.checkAdmin(request)){
            return false;
        }
        if(id <= 0){
            return false;
        }

        return userService.removeById(id);
    }
```

> 至此，完成用户的增删改查方法

后端优化部分

> 补充获取当前用户信息接口

```
    @GetMapping("/currentUser")
    public User getCurrentUser(HttpServletRequest request){
        User loginUser = (User)
request.getSession().getAttribute(USER_LOGIN_STATE);
        if(Objects.isNull(loginUser)){
            return null;
        }
        Long userId = loginUser.getId();
        User user = userService.getById(userId);
        User currentUser = userService.getSafetyUser(user);

        return currentUser;
    }
```

> 这里为了防止用户信息更新后，session中的信息并没有更新，所以在这里选择查库返回当前用户信息
>
> 当然具体情况具体分析，如果对于用户信息更新并不频繁的系统来说，当更新数据库的时候更新一下session，然后从session中去信息即可

补充邀请码机制

> 用户注册的时候需要填写邀请码，对用户进行校验

1. 给用户补充一个邀请码字段
2. 用户在前端填写邀请码
3. 后端校验此邀请码是否存在
4. 当用户注册的时候自动生成一个随机的邀请码

```
        //判断当前邀请码是否存在
        LambdaQueryWrapper<User> queryWrapper = new LambdaQueryWrapper<>();
        queryWrapper.eq(User::getInvitationCode, invitationCode);
        User invitationUser = this.getOne(queryWrapper);
        if(Objects.isNull(invitationUser)){
            return -1L;
        }

        //随机生成邀请码并判断是否存在
        String userInvitationCode = RandomInvitationCode.generateCode();
        queryWrapper.clear();
        queryWrapper.eq(User::getInvitationCode, userInvitationCode);
        User one = this.getOne(queryWrapper);
        while(one != null){
            userInvitationCode = RandomInvitationCode.generateCode();
            queryWrapper.clear();
            queryWrapper.eq(User::getInvitationCode, userInvitationCode);
            one = this.getOne(queryWrapper);
        }

        currentUser.setInvitationCode(userInvitationCode);
```

封装通用返回对象

> 目的：规范返回给前端的信息，让前端可以更好的将处理结果展现给用户或者展现到界面上，同时也方便后端人员调试找到错误

```
/**
 * 通用返回体
 * @author ObjectY
 */
@Data
public class BaseResponse<T> implements Serializable {
    private static final long serialVersionUID = 947757056421018877L;

    private Integer code;
    private String message;

    private T data;

    private String description;

    public BaseResponse(int code ,T data, String message,String description){
        this.code = code;
        this.data = data;
        this.message = message;
        this.description = description;
    }

    public BaseResponse(int code,T data ,String message){
        this(code,data,message,"");
    }

    public BaseResponse(int code ,String message){
        this(code,null,message);
    }

    public BaseResponse(ErrorCode errorCode){

this(errorCode.getCode(),null,errorCode.getMessage(),errorCode.getDescription());
    }
}
```

```
/**
 * 封装通用返回类
 * @author ObjectY
 */
public class ResultUtils {
    public static <T> BaseResponse<T> success(T data){
        return new BaseResponse<>(ErrorCode.SUCCESS.getCode(),data,"success");
    }

    public static <T> BaseResponse<T> error(ErrorCode errorCode){
        return new BaseResponse<>
(errorCode.getCode(),null,errorCode.getMessage(),errorCode.getDescription());
    }
}
```

```
/**
 * 自定义错误码，封装错误信息
 * @author ObjectY
 */
public enum ErrorCode {
    PARAMS_ERROR(40000,"参数错误",""),
    NOT_LOGIN(40001,"未登录",""),
    NO_AUTH(40100,"无权限",""),
    SYSTEM_ERROR(50000,"系统错误",""),
    SUCCESS(200,"success","")
    ;
    private final int code;
    private final String message;
    private final String description;

    ErrorCode(int code, String message, String description) {
        this.code = code;
        this.message = message;
        this.description = description;
    }
}
```

封装全局异常处理

> 目的：编写自定义全局异常，封装程序中的异常信息，将其统一处理，然后编写全局异常捕获，捕获程序中的自定义异常，将自定义异常集中在一起进行返回

```
/**
 * 自定义异常
 * @author ObjectY
 */
public class BusinessException extends RuntimeException{

    private int code;
    private String description;

    public BusinessException(String message,int code,String description) {
        super(message);
        this.code = code;
        this.description = description;
    }

    public BusinessException(ErrorCode errorCode){
        super(errorCode.getMessage());
        this.code = errorCode.getCode();
        this.description = errorCode.getDescription();
    }

    public int getCode() {
        return code;
    }

    public String getDescription() {
        return description;
    }
}
```

```
/**
 * 全局异常捕获
 * @author ObjectY
 */
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(BusinessException.class)
    public BaseResponse businessExceptionHandler(BusinessException ex){
        return
ResultUtils.error(ex.getCode(),ex.getMessage(),ex.getDescription());
    }
}
```

项目部署

多环境

> 什么是多环境？
>
> (22条消息) 多环境设计_程序员鱼皮的博客-CSDN博客

> 为什么需要多环境？
>
> 1. 每个环境中的代码互不影响（例如需要改动代码，不需要动上线环境，只需要修改测试环境，线上环境并不会受到影响）
> 2. 区分不同的阶段 开发 、测试 、生产
> 3. 对项目进行优化
>   1. 本地日志级别
>   2. 精简依赖，节省项目体积
>   3. 动态调整项目的环境 / 参数 （JVM参数调整）

多环境分类：

1. 本地环境 （个人电脑）
2. 开发环境 （例如 一些大型公司的内网服务器，类似远程开发）
3. 测试环境 （测试） 开发 / 测试 / 产品，一般为独立的数据库，独立的服务器
4. 预发布环境 （体验服） 与正式环境一直
5. 正式环境 （正式服）
6. 沙箱环境 （实验环境） 测试新功能

前端多环境

- 请求地址
  - 开发环境：localhost:8000
  - 线上环境：backend.user-center.top

```
const request = extend({
  credentials: 'include',
  prefix: process.env.NODE_ENV === 'production' ? 'http://backend.user-center.top'
: undefined
})
```

后端多环境

```
spring:
  application:
    name: usercenter-backend
  # 配置数据库连接信息
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://xxxxx:3306/user_center_backend
    username: xxx
    password: xxxxx
  session:
    timeout: PT15M
server:
  port: 8080
  servlet:
    context-path: /api
mybatis-plus:
  configuration:
    # 配置日志信息
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
    # 关闭驼峰映射
    map-underscore-to-camel-case: false
  # 配置id策略
  global-config:
    db-config:
      id-type: auto
      # 配置逻辑删除
      logic-delete-value: 0
      logic-not-delete-value: 1
```

> 主要是更改数据库/redis/消息队列等工具的链接地址以及一些配置

image-20230501221214235

> 然后使用maven进行打包，打包完成后使用下列命令启动项目

```
java -jar xxxx.jar --spring.profiles.active=prod
```

原生部署

前端：

1. 安装nginx服务器

这里使用最原始的方式，到官网进行安装

2. 部署nginx服务器 CentOS如何安装nginx - 知乎 (zhihu.com) （这里可以配置成环境变量方便后续启动）
3. 将前端项目打包然后上传到服务器
4. 配置nginx.conf指向前端打包的项目

后端：

1. 在服务器上安装jdk1.8和maven
2. 这里有两个选择
   1. 从git上拉取代码，然后在服务端进行打包
   2. 在本地打包，然后直接上传到服务器（这里考虑到服务器第一次安装maven，打包会比较慢，所以选择本地打包然后上传）
3. 使用nohup命令，挂起运行后端jar包

至此，原生部署方式完毕！

宝塔部署（最简单）

1. 安装宝塔面板 宝塔面板下载，免费全能的服务器运维软件 (bt.cn)

2. 进入宝塔面板，在软件商店中安装nginx，tomcat(为了安装java)，docker，mysql

3. 将前端项目打包上传

4. 将后端项目打包上传

5. 一键部署前端

6. 一键部署后端

docker部署：

docker是容器，将项目的环境和代码打包成一个镜像，镜像更容易分发和移植

再次启动项目可直接运行镜像

1. 宝塔安装docker
2. idea下载docker插件，编写Dockerfile文件

```
FROM maven:3.5-jdk-8-alpine as builder

# 指定工作目录
WORKDIR /app
# 将pom文件复制到工作目录
COPY pom.xml .
# 将src目录复制到工作目录
COPY src ./src
# 指定打包命令
RUN mvn package -DskipTests
```

```
CMD ["java","-jar","app/target/backend-0.0.1-SNAPSHOT.jar","--
spring.profiles.active=prod"]
```

docker run 运行镜像

Docker平台部署

1. 腾讯云
2. 后端微信云托管
3. 前端webify

项目上线

1. 域名解析（去云平台进行解析）
2. 在宝塔中网站部署地方绑定解析后的域名
3. 修改前端项目，请求后端的域名

接下来的事情很重要，在宝塔中部署两个网站，一个前端一个后端

image-20230504195925255

image-20230504200042327-1683201644859-2

```
location ^~ /api/ {

        add_header 'Access-Control-Allow-Origin' $http_origin;
        add_header 'Access-Control-Allow-Credentials' 'true';
        add_header Access-Control-Allow-Methods 'GET, POST, OPTIONS';
        add_header Access-Control-Allow-Headers '*';

        if ($request_method = 'OPTIONS') {
         add_header 'Access-Control-Allow-Credentials' 'true';
         add_header 'Access-Control-Allow-Origin' $http_origin;
         add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
         add_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-
With,If-Modified-Since,Cache-Control,Content-Type,Range';
         add_header 'Access-Control-Max-Age' 1728000;
         add_header 'Content-Type' 'text/plain; charset=utf-8';
         add_header 'Content-Length' 0;
            return 204;
        }
          proxy_pass http://127.0.0.1:8082;
    }
```

接下来访问前端网站即可，至此，项目已成功上线