[http://www.yangeit.cn:21010/skytakeoutproject2023/day02.html#](http://www.yangeit.cn:21010/skytakeoutproject2023/day02.html#_1-2-%E4%BB%A3%E7%A0%81%E5%AE%8C%E5%96%84)

粗体就是在给出的网页然后ctrl+F 找到对应位置

**用户名已存在问题:**

1. 插入数据，username在员工表为唯一 但是后端插入相同用户名如何后端正确报错

可以在每个controller代码下先查询是否有这个数据 如果相同报错，但每个都要添加很麻烦

解决：使用全局异常处理器



1. 插入数据传的参数是employeeDTO 而不是employee

可以使用这个代码：//对象属性拷贝

BeanUtils.copyProperties(employeeDTO, employee);

**员工分页查询:**

1. 设计对应的EmployeePageQueryDTO

封装pageResult，以及service代码也和平时不一样， PageResult pageQuery

（参数不用一个一个传，封装成dto/vo就可以了）

PS：VO是后端通过数据库查询传给前端的数据 DTO是前端post的数据

**操作时间字段显示有问题**：

1. 日期返回数据格式化，可以用注解，但每个都要添加很麻烦

解决：在WebMvcConfiguration中扩展SpringMVC的消息转换器，统一对日期类型进行格式处理

[http://www.yangeit.cn:21010/skytakeoutproject2023/day03.html#](http://www.yangeit.cn:21010/skytakeoutproject2023/day03.html)

**公共字段 实现思路**：

1. 插入数据时作为几个表的公共字段。并且不会作为数据从前端传到后端，得后端自己添加，可以手动在每一个service里写但是太麻烦

解决：使用AOP切面编程

a)先自定义注解 AutoFill，用于标识需要进行公共字段自动填充的方法

b)自定义切面类 AutoFillAspect，统一拦截加入了 AutoFill注解的方法，通过反射为公共字段赋值

c)在 Mapper的方法 上加入 AutoFill 注解,并注释业务层赋值时间和用户Id的代码 @AutoFill(value = OperationType.INSERT)

void insert(Category category); <-category就是AutoFillAspect 里用到的entity

**文件上传实现**：

1. 阿里云的OSS服务 我还没有配置这个所以先跳过

上传文件本质就是IO流

[http://www.yangeit.cn:21010/skytakeoutproject2023/day07.html#](http://www.yangeit.cn:21010/skytakeoutproject2023/day07.html)

**数据缓存**

(用户端访问量比较大的时候，使用缓存减少数据库查询)

1. Redis缓存

1). 用户端controller，加载菜品：

第一次查询，如果redis不为空则从redis中加载，如果数据更新则先从数据库中加载再保存到redis

获取：redisTemplate.opsForValue().get(key);

存储：redisTemplate.opsForValue().set(key, list);

2). 保障数据一致性

管理端controller，增删改菜品

增删改后为了保障数据一致性，需要清空redis的数据

用户再次访问时 从数据库加载再存储到redis即可

String key = "dish\_" + dishDTO.getCategoryId();

cleanCache(key);



1. Spring Cache 缓存

加注解即可

同上

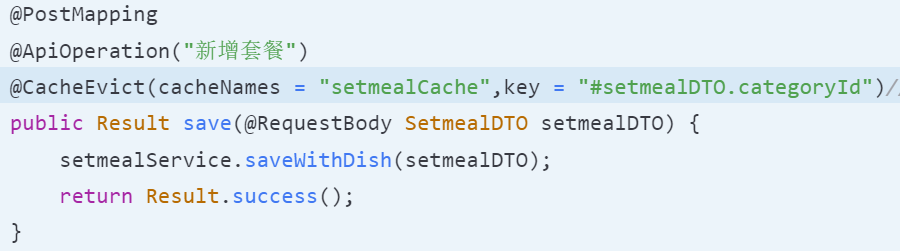
用户端访问用：

@Cacheable(cacheNames = "setmealCache",key = "#categoryId")



管理端增删改删除缓存保证数据一致性用：

@CacheEvict(cacheNames = "setmealCache",key = "#setmealDTO.categoryId")*//key: setmealCache::100*



http://www.yangeit.cn:21010/skytakeoutproject2023/day10.html#

**Spring Task(定时任务)**

应用场景如发送生日祝福，定期发送邮件等

在这里解决的是用户下单后可能存在的问题。

1.下单后未支付，订单一直处于待支付状态

2.用户收货后管理端未点击完成按钮，订单一直处于派送中状态

@Scheduled(cron = "0/5 \* \* \* \* ?") //添加这个注解即可

public void executeTask(){

log.info("定时任务开始执行：{}",new Date());

//Code…

}

**Websocket应用**

用于客户端(前端)和服务端(后端)的交互

写法类似这样：



a.如实时聊天室，

前端负责页面发送消息和接收并显示消息。

后端负责监听端口，当客户端连接上来时，将收到的消息广播给所有客户端:

webSocketServer.sendToAllClient(message);

b.天气实时预报，

当客户端连接时后端负责将客户端发送实时天气预报

wss.on('connection', function connection(ws) {

console.log('Client connected');

// 当客户端连接时，向客户端发送实时天气预报

fetch('https://api.weatherapi.com/v1/current.json?key=YOUR\_API\_KEY&q=London')

.then(response => response.json())

.then(data => {

ws.send(JSON.stringify(data));

});

}

而前端通过连接到后端的服务器，并通过后端返回的数据在页面上显示实时天气预报

ws.onmessage = function(event) {

const weatherData = JSON.parse(event.data);

const { location, current } = weatherData;

const message = `Current weather in ${location.name}: ${current.condition.text}, Temperature: ${current.temp\_c}°C`;

weatherDiv.textContent = message;

};

http://www.yangeit.cn:21010/skytakeoutproject2023/day11.html#

没什么难度，但group by的sql语句还是第一次写

以及代码有些写法还是很有意思的

1. 如得到最小的date:00:00 和 date:23:59

LocalDateTime beginTime = LocalDateTime.of(date, LocalTime.MIN);

LocalDateTime endTime = LocalDateTime.of(date, LocalTime.MAX);

1. 不再是封装成对象，而是构造成map传参查询

Map map = new HashMap();

map.put("begin",beginTime);

map.put("end", endTime);

return userMapper.countByMap(map);

3. 将查询得到的数据封装成DTO对象

List<GoodsSalesDTO> goodsSalesDTOList = orderMapper.getSalesTop10(beginTime, endTime);

但是最后返回的结果是VO，因为要传给前端

return SalesTop10ReportVO.builder()

.nameList(nameList)

.numberList(numberList)

.build()

1. stream流

Integer totalOrderCount = orderCountList.stream().reduce(Integer::sum).get();

String nameList=

StringUtils.join(goodsSalesDTOList.stream().map(GoodsSalesDTO::getName).collect(Collectors.toList()),",");

(当然我看不懂啦)

1. group by写sql

**Jwt令牌登录校验(具体看代码)**

1)配置JwtUtil工具包，有生成密钥和解密的函数

2)在用户controller的login函数里，在登录成功后生成jwt令牌

(登录前，用户密码使用MD5算法并加盐保存在数据库)

3)设置拦截器JwtTokenAdminInterceptor，用于处理HTTP请求

拦截器负责从请求头中获得令牌并解析是否正确，若正确才会正常响应