```php
<?php
declare (strict_types = 1);
namespace app\terminal\service\equipment;
use app\common\enum\Setting as SettingEnum;
use app\common\model\store\Setting as SettingModel;
use app\common\MQTTServer;
use app\terminal\model\Equipment as EquipmentModel;
use app\terminal\model\Store as StoreModel;
use app\common\service\BaseService;
class Equipment extends BaseService
{
    public function matecode(string $matecode): array
    {
        $model = EquipmentModel::withoutGlobalScopeGetOne(['matecode' => $matecode, 'is_delete' => 0]);
        if (empty($model)) {
            throwError('设备不存在');
        }
        $equipmenInfo = $model->toArray();
        if ($equipmenInfo['mate_flag'] == 1) {
            throwError('设备已配对');
        }
        $mate = [
            'mate_flag' => 1,
            'mate_time' => date('Y-m-d H:i:s')
        ];
        $res = EquipmentModel::withoutGlobalScope()->where(['id' => $equipmenInfo['id']])->update($mate);
        if (!$res) {
            throwError('配对失败');
        }
        $store_id = $equipmenInfo['store_id'];
        $mate_time = $mate['mate_time'];
        $data = [
            'id' => $equipmenInfo['id'],
            'code' => $equipmenInfo['code'],
            'name' => $equipmenInfo['name'],
            'type' => $equipmenInfo['type'],
            'matecode' => $equipmenInfo['matecode'],
            'store_id' => $store_id,
            'mate_time' => $mate_time,
        ];
        $MQTTServer = new MQTTServer();
        $data['ali_mqtt'] = $MQTTServer->get_config();
        $data['baidu_dishes']= SettingModel::getItem(SettingEnum::BAIDU_DISHES, $store_id);
        $message = config('message');
        $data['webhook'] = TP_ENV == 'prod' ? $message['webhook_prod'] : $message['webhook_test'];
        $data['msg'] = TP_ENV == 'prod' ? $message['msg_prod'] : $message['msg_test'];
        $data['store_host'] = '';
        $data['store_name'] = '';
        $store = StoreModel::withoutGlobalScope()
            ->where(['store_id' => $store_id])
```

```
            ->find();
        if (!empty($store)) {
            $store = $store->toArray();
            $data['store_host'] = trim($store['host'], '/store');
            $data['store_name'] = $store['store_name'];
        }
        return $data;
    }
}
<?php
declare (strict_types = 1);
namespace app\terminal\service\dishes;
use app\common\enum\dishes\FoodsStatus;
use app\terminal\model\dishes\Dishes as DishesModel;
use app\terminal\model\dishes\DishesCate as DishesCateModel;
use app\terminal\model\UploadFile as UploadFileModel;
use app\terminal\service\store\User as StoreUserService;
use app\terminal\validate\Dishes as DishesValidate;
use app\common\service\BaseService;
use cores\exception\BaseException;
class Dishes extends BaseService
{
    public $id;
    public function getList(array $param = []): array
    {
        $model = new DishesModel();
        $field = 'id,name,category,weight,energy,carbohydrate,protein,fat,file_id,create_by,create_time';
        $list = $model->getList($param, $field);
        if (empty($list['data'])) {
            return $list;
        }
        $categorys = DishesCateModel::getAllData(['pid' => 0]);
        $categorys = array_column($categorys, null, 'id');
        $file_ids = array_column($list['data'], 'file_id');
        $files = UploadFileModel::getAllData(['file_id' => $file_ids]);
        $files = array_column($files, null, 'file_id');
        $list['data'] = array_map(function ($item) use ($categorys, $files) {
            $item['category_name'] = $categorys[$item['category']]['name'] ?? '';
            $item['file_url'] = $files[$item['file_id']]['preview_url'] ?? '';
            return $item;
        }, $list['data']);
        return $list;
    }
    public function getAll(array $param = []): array
    {
        $model = new DishesModel();
        $field = 'id,name,category';
        $list = $model->getAll($param, $field);
        $data = [
            'list' => $list,
```

```
            'count' => count($list),
        ];
        return $data;
    }
    public function add(array $data): bool
    {
        $this->validate($data);
        if (DishesModel::checkExist($data['name'])) {
            $this->error = '菜品名称已存在';
            return false;
        }
        $create_by = StoreUserService::getLoginUserName();
        $model = new DishesModel;
        $res = $model->add($data, $create_by);
        $this->id = $model['id'] ?: 0;
        return $res;
    }
    public function detailMore(string $dishes_ids)
    {
        $dishes_ids = explode(',', $dishes_ids);
        return DishesModel::getAllData(['id' => $dishes_ids, 'is_delete' => 0], 'id,name');
    }
    public function detail(int $id)
    {
        $with = ['cate', 'file', 'unsuitable', 'foods'];
        $detail = DishesModel::detail($id, $with);
        if (empty($detail)) {
            throwError('数据不存在');
        }
        $detail = $detail->toArray();
        $detail['category_name'] = $detail['cate']['name'] ?? '';
        unset($detail['cate']);
        $detail['file_url'] = $detail['file']['preview_url'] ?? '';
        if (!empty($detail['unsuitable'])) {
            $detail['unsuitable'] = array_column($detail['unsuitable'], 'meal_times');
        }
        if (!empty($detail['foods'])) {
            $foods = [];
            foreach ($detail['foods'] as $food) {
                $item = [
                    'foods_id' => $food['pivot']['foods_id'],
                    'foods_weight' => $food['pivot']['foods_weight'],
                    'foods_ratio' => $food['pivot']['foods_ratio'],
                    'foods_status' => $food['foods_status'],
                    'foods_status_name' => FoodsStatus::getName($food['foods_status']),
                    'name' => $food['name'],
                ];
                $foods[] = $item;
            }
            $detail['foods'] = $foods;
```

```
        }
        return $detail;
    }
    public function edit(int $id, array $data): bool
    {
        $this->validate($data);
        $model = DishesModel::detail($id);
        if (empty($model)) {
            throwError('数据不存在');
        }
        if ($model['name'] !== $data['name'] && DishesModel::checkExist($data['name'])) {
            $this->error = '菜品名称已存在';
            return false;
        }
        return $model->edit($data);
    }
    public function setDelete(int $id): bool
    {
        $model = DishesModel::detail($id);
        if (empty($model)) {
            throwError('数据不存在');
        }
        return $model->setDelete();
    }
    private function validate(array $data): void
    {
        $validate = new DishesValidate;
        if (!$validate->check($data)) {
            throwError($validate->getError());
        }
    }
    public function cate(): array
    {
        $model = new DishesCateModel;
        return $model->getAll();
    }
}
<?php
declare (strict_types = 1);
namespace app\terminal\service\dishes;
use app\common\service\BaiduDishes;
use app\terminal\model\dishes\Dishes as DishesModel;
use app\terminal\model\dishes\DishesImg as DishesImgModel;
use app\terminal\model\UploadFile as UploadFileModel;
use app\terminal\service\store\User as StoreUserService;
use app\terminal\validate\DishesImg as DishesImgValidate;
use app\common\service\BaseService;
use cores\exception\BaseException;
class Img extends BaseService
{
```

```
public $id;
public function getAll(int $dishes_id): array
{
    $dishes = DishesModel::detail($dishes_id);
    if (empty($dishes)) {
        throwError('菜品数据不存在');
    }
    $imgs = DishesImgModel::getAllData(['dishes_id' => $dishes_id, 'is_delete' => 0]);
    if (empty($imgs)) {
        return [];
    }
    $file_ids = array_column($imgs, 'file_id');
    $files = UploadFileModel::getAllData(['file_id' => $file_ids]);
    $files = array_column($files, null, 'file_id');
    $imgs = array_map(function ($item) use ($files) {
        $item['file_url'] = $files[$item['file_id']]['preview_url'] ?? '';
        return $item;
    }, $imgs);
    return $imgs;
}
public function add(array $data): bool
{
    $this->validate($data);
    $dishes = DishesModel::detail($data['dishes_id']);
    if (empty($dishes)) {
        throwError('菜品数据不存在');
    }
    $where = [
        'dishes_id' => $data['dishes_id'],
        'file_id' => $data['file_id'],
        'is_delete' => 0,
    ];
    $img = DishesImgModel::getOne($where);
    if (!empty($img)) {
        throwError('图片已存在');
    }
    $file = UploadFileModel::getOne(['file_id' => $data['file_id']]);
    if (empty($file)) {
        throwError('图片不存在');
    }
    $baiduDishes = new BaiduDishes;
    $cont_sign = $baiduDishes->add($dishes['id'], $dishes['name'], $file['preview_url']);
    if (empty($cont_sign)) {
        throwError('菜品图片学习失败');
    }
    $data['cont_sign'] = $cont_sign;
    $create_by = StoreUserService::getLoginUserName();
    $model = new DishesImgModel;
    $res = $model->add($data, $create_by);
    $this->id = $model['id'] ?: 0;
```

```
        return $res;
    }
    public function setDelete(int $dishes_id, string $ids): bool
    {
        $dishes = DishesModel::detail($dishes_id);
        if (empty($dishes)) {
            throwError('菜品数据不存在');
        }
        $ids = explode(',', $ids);
        $imgs = DishesImgModel::getAllData(['id' => $ids, 'is_delete' => 0, 'dishes_id' => $dishes_id]);
        if (empty($imgs)) {
            throwError('图片数据不存在');
        }
        $baiduDishes = new BaiduDishes;
        foreach ($imgs as $item) {
            if (!empty($item['cont_sign'])) {
                $baiduDishes->delete($item['cont_sign']);
            }
        }
        $ids = array_column($imgs, 'id');
        return DishesImgModel::updateBase(['is_delete' => 1], ['id' => $ids]);
    }
    private function validate(array $data): void
    {
        $validate = new DishesImgValidate;
        if (!$validate->check($data)) {
            throwError($validate->getError());
        }
    }
}
<?php
declare (strict_types = 1);
namespace app\terminal\service\meal;
use app\terminal\model\meal\Meal as MealModel;
use app\terminal\model\staff\Staff as StaffModel;
use app\terminal\service\store\User as StoreUserService;
use app\terminal\validate\Meal as MealValidate;
use app\common\enum\dishes\MealTimes as MealTimesEnum;
use app\common\service\BaseService;
use cores\exception\BaseException;
class Meal extends BaseService
{
    public function infos(int $staff_id): array
    {
        $model = new MealModel;
        $lastMeal = $model::getOneOrder(['staff_id' => $staff_id, 'is_delete' => 0], ['meal_time' => 'desc']);
        if (empty($lastMeal)) {
            return [];
        }
        $data['list'] = $model->getInfos((int)$lastMeal['id']);
```

```
        $data['meal_date'] = $lastMeal['meal_date'];
        $data['meal_times'] = $lastMeal['meal_times'];
        $data['meal_time'] = $lastMeal['meal_time'];
        $data['meal_times_name'] = MealTimesEnum::getName($lastMeal['meal_times']);
        $data['is_loading'] = $lastMeal['is_loading'];
        return $data;
    }
    public function add(array $data): bool
    {
        $this->validate($data);
        $staffInfo = StaffModel::info($data['staff_id']);
        if (empty($staffInfo)) {
            throwError('用户不存在');
        }
        $create_by = StoreUserService::getLoginUserName();
        $model = new MealModel;
        $res = $model->add($data, $create_by);
        return $res;
    }
    public function addMeal(array $data): bool
    {
        $this->validate($data);
        $staffInfo = StaffModel::info($data['staff_id']);
        if (empty($staffInfo)) {
            throwError('用户不存在');
        }
        $create_by = StoreUserService::getLoginUserName();
        $model = new MealModel;
        $res = $model->addCurrentMeal($data, $create_by);
        return $res;
    }
    private function validate(array $data): void
    {
        $validate = new MealValidate;
        if (!$validate->check($data)) {
            throwError($validate->getError());
        }
    }
}
<?php
declare (strict_types=1);
namespace app\api\controller;
use think\response\Json;
use app\api\model\Order as OrderModel;
use app\api\service\Cart as CartService;
use app\api\service\order\Checkout as CheckoutService;
use app\api\validate\order\Checkout as CheckoutValidate;
class Checkout extends Controller
{
    private ?CheckoutValidate $validate = null;
```

```
public function order(string $mode = 'buyNow'): Json
{
    if ($mode === 'buyNow') {
        return $this->buyNow();
    } elseif ($mode === 'cart') {
        return $this->cart();
    }
    return $this->renderError('结算模式不合法');
}
public function submit(string $mode = 'buyNow'): Json
{
    return $this->order($mode);
}
private function buyNow(): Json
{
    $Checkout = new CheckoutService;
    $params = $Checkout->setParam($this->getParam([
        'goodsId' => 0,
        'goodsSkuId' => '',
        'goodsNum' => 0,
    ]));
    if (!$this->getValidate()->scene('buyNow')->check($params)) {
        return $this->renderError($this->getValidate()->getError(), ['isCreated' => false]);
    }
    $model = new OrderModel;
    $goodsList = $model->getOrderGoodsListByNow(
        (int)$params['goodsId'],
        (string)$params['goodsSkuId'],
        (int)$params['goodsNum']
    );
    $orderInfo = $Checkout->onCheckout($goodsList);
    if ($this->request->isGet()) {
        return $this->renderSuccess([
            'order' => $orderInfo,
            'personal' => $Checkout->getPersonal(),
            'setting' => $Checkout->getSetting(),
        ]);
    }
    if ($Checkout->hasError()) {
        return $this->renderError($Checkout->getError(), ['isCreated' => false]);
    }
    if (!$Checkout->createOrder($orderInfo)) {
        return $this->renderError($Checkout->getError() ?: '订单创建失败', ['isCreated' => false]);
    }
    return $this->renderSuccess(['orderId' => $Checkout->model['order_id']], '订单创建成功');
}
private function cart(): Json
{
    $Checkout = new CheckoutService;
    $params = $Checkout->setParam($this->getParam());
```

```
        $cartIds = $this->getCartIds();
        $CartModel = new CartService;
        $goodsList = $CartModel->getOrderGoodsList($cartIds);
        $orderInfo = $Checkout->onCheckout($goodsList);
        if ($this->request->isGet()) {
            return $this->renderSuccess([
                'order' => $orderInfo,
                'personal' => $Checkout->getPersonal(),
                'setting' => $Checkout->getSetting(),
            ]);
        }
        if ($Checkout->hasError()) {
            return $this->renderError($Checkout->getError(), ['isCreated' => false]);
        }
        if (!$Checkout->createOrder($orderInfo)) {
            return $this->renderError($Checkout->getError() ?: '订单创建失败');
        }
        $CartModel->clear($cartIds);
        return $this->renderSuccess(['orderId' => $Checkout->model['order_id']], '订单创建成功');
    }
    private function getValidate(): CheckoutValidate
    {
        if (is_null($this->validate)) {
            $this->validate = new CheckoutValidate;
        }
        return $this->validate;
    }
    private function getCartIds()
    {
        $cartIds = $this->request->param('cartIds');
        return explode(',', $cartIds);
    }
    private function getParam(array $define = []): array
    {
        return array_merge($define, $this->request->param());
    }
}
<?php
declare (strict_types=1);
namespace app\api\service\meal;
use app\api\model\User as UserModel;
use app\api\model\Restaurant as RestaurantModel;
use app\common\model\zhct\Staff as ZhctStaffModel;
use app\common\model\zhct\User as ZhctUserModel;
use app\api\model\MealOrder as OrderModel;
use app\api\model\UserSubsidy as UserSubsidyModel;
use app\api\service\User as UserService;
use app\common\enum\equipment\EquipmentType as EquipmentTypeEnum;
use app\common\enum\order\OrderSource as OrderSourceEnum;
use app\common\service\BaseService;
```

```
use app\common\library\helper;
use cores\exception\BaseException;
use app\common\enum\payment\Method as PaymentMethodEnum;
use app\common\enum\order\PayStatus as PayStatusEnum;
use app\common\enum\order\PrintStatus as PrintStatusEnum;
use app\common\enum\order\LockerStatus as LockerStatusEnum;
use app\common\enum\order\TakeMethod as TakeMethodEnum;
class Checkout extends BaseService
{
    private const PACKAGE_FEE_UUID = 'package_fee';
    public OrderModel $model;
    private $user;
    private $ZhctUser = [];
    private $ZhctStaff = [];
    private $dishesList;
    private array $param = [];
    private array $orderData = [];
    public function __construct()
    {
        parent::__construct();
        $this->user = UserService::getCurrentLoginUser(true);
        $this->model = new OrderModel;
        $this->storeId = $this->getStoreId();
    }
    public function setParam($param): array
    {
        $this->param = $param;
        return $this->getParam();
    }
    public function getParam(): array
    {
        return $this->param;
    }
    public function onCheckout($dishesList): array
    {
        $this->dishesList = $dishesList;
        return $this->checkout();
    }
    private function checkout(): array
    {
        $this->orderData = $this->getOrderData();
        $this->setZhctUser();
        $orderTotalNum = (int)helper::getArrayColumnSum($this->dishesList, 'quantity');
        $this->setOrderTotalPrice();
        $this->setOrderPayPrice();
        return array_merge([
            'dishesList' => $this->dishesList,
            'orderTotalNum' => $orderTotalNum,
            'hasError' => $this->hasError(),
            'errorMsg' => $this->getError(),
```

```
    ], $this->orderData);
}
private function getOrderData(): array
{
    $takeMethod = $this->getTakeMethod();
    $packageFee = $this->getPackageFeeByMethod($takeMethod);
    $locker = $takeMethod === TakeMethodEnum::LOCKER
        ? $this->getLockerByCode((string)($this->param['locker_code'] ?? ''))
        : [];
    return [
        'pay_method' => PaymentMethodEnum::BALANCE,
        'pay_status' => PayStatusEnum::PENDING,
        'meal_date' => $this->param['meal_date'],
        'meal_times' => $this->param['meal_times'],
        'bind_id' => 0,
        'bind_time' => date('Y-m-d H:i:s'),
        'last_meal_time' => date('Y-m-d H:i:s'),
        'equipment_code' => '',
        'source' => OrderSourceEnum::ORDERFOOD,
        'restaurant_id' => $this->param['restaurant_id'],
        'user_mobile' => (string)($this->param['user_mobile'] ?? ''),
        'user_name' => (string)($this->param['user_name'] ?? ''),
        'take_method_text' => $this->getTakeMethodText($takeMethod),
        'restaurant_method' => $takeMethod,
        'locker_code' => (string)($locker['code'] ?? ''),
        'locker_address' => (string)($locker['address'] ?? ''),
        'locker_status' => $takeMethod === TakeMethodEnum::LOCKER ? LockerStatusEnum::WAIT_STORE : LockerStatusEnum::NONE,
        'package_fee' => helper::number2((string)$packageFee),
        'remark' => $this->param['remark'],
        'zhaodai_num' => $this->param['zhaodai_num'] ?? 0,
        'peitong_num' => $this->param['peitong_num'] ?? 0,
        'take_time_start' => $this->param['take_time_start'] ?? '',
        'take_time_end' => $this->param['take_time_end'] ?? '',
        'print_status' => PrintStatusEnum::PENDING,
    ];
}
private function setZhctUser()
{
    $ZhctUserModel = new ZhctUserModel;
    $ZhctUser = $ZhctUserModel->getInfoBYAIID($this->user['user_id']);
    $this->ZhctUser = $ZhctUser;
    $ZhctStaffModel = new ZhctStaffModel;
    $ZhctStaff = $ZhctStaffModel->getInfoBYAIID($this->user['user_id']);
    $this->ZhctStaff = $ZhctStaff;
}
private function setOrderTotalPrice()
{
    $dishesTotal = helper::getArrayColumnSum($this->dishesList, 'total_price');
    $packageFee = (float)$this->orderData['package_fee'];
    $this->orderData['total_price'] = helper::number2(helper::bcadd((string)$dishesTotal, (string)$packageFee));
```

```
}
private function setOrderPayPrice()
{
    $this->orderData['order_price'] = $this->orderData['total_price'];
    $this->orderData['pay_price'] = $this->orderData['order_price'];
}
public function getPersonal(): array
{
    $cahs_balance = $this->user['balance'];
    $UserSubsidyModel = new UserSubsidyModel;
    $subsidy_balance = $UserSubsidyModel->getCount($this->user['user_id']);
    $balance = $cahs_balance + $subsidy_balance;
    return [
        'user_id' => $this->user['user_id'],
        'user_mobile' => $this->user['mobile'],
        'user_name' => $this->user['nick_name'],
        'user_balance' => $balance,
        'user_cahs_balance' => $cahs_balance,
        'user_subsidy_balance' => $subsidy_balance,
    ];
}
public function createOrder(array $order): bool
{
    if (!$this->validateOrderForm($order)) {
        return false;
    }
    return $this->model->transaction(function () use ($order) {
        return $this->createOrderEvent($order);
    });
}
private function createOrderEvent($order): bool
{
    $this->add($order);
    $this->saveOrderDishes($order);
    return true;
}
private function validateOrderForm(array $order): bool
{
    if (empty($this->ZhctUser) || empty($this->ZhctStaff)) {
        $this->error = '人员数据错误';
        return false;
    }
    $takeMethod = (int)($order['restaurant_method'] ?? $order['take_method'] ?? $this->getTakeMethod());
    if ($takeMethod == TakeMethodEnum::PICKUP) {
        return $this->validateOrderFormByPickup($order);
    }
    if ($takeMethod == TakeMethodEnum::LOCKER) {
        return $this->validateOrderFormByLocker($order);
    }
    return true;
```

```
}
private function getTakeMethod(): int
{
    $restaurantMethod = (int)($this->param['restaurant_method'] ?? 0);
    if (in_array($restaurantMethod, [TakeMethodEnum::DINE_IN, TakeMethodEnum::PICKUP, TakeMethodEnum::LOCKER], true)) {
        return $restaurantMethod;
    }
    $takeMethod = (int)($this->param['take_method'] ?? 0);
    if (in_array($takeMethod, [TakeMethodEnum::DINE_IN, TakeMethodEnum::PICKUP, TakeMethodEnum::LOCKER], true)) {
        return $takeMethod;
    }
    return TakeMethodEnum::DINE_IN;
}
private function getTakeMethodText(int $takeMethod): string
{
    return TakeMethodEnum::getName($takeMethod) ?: TakeMethodEnum::getName(TakeMethodEnum::DINE_IN);
}
private function getPackageFeeByMethod(int $takeMethod): float
{
    if (!in_array($takeMethod, [TakeMethodEnum::PICKUP, TakeMethodEnum::LOCKER], true)) {
        return 0.0;
    }
    $restaurantId = (int)($this->param['restaurant_id'] ?? 0);
    if ($restaurantId <= 0) {
        return 0.0;
    }
    $restaurant = $this->getRestaurantById($restaurantId);
    if (empty($restaurant)) {
        return 0.0;
    }
    if ($takeMethod === TakeMethodEnum::LOCKER) {
        return (float)($restaurant['locker_package_fee'] ?? 0);
    }
    return (float)($restaurant['package_fee'] ?? 0);
}
private function getRestaurantById(int $restaurantId): array
{
    $RestaurantModel = new RestaurantModel;
    $restaurant = $RestaurantModel->getDetail($restaurantId);
    if (is_array($restaurant)) {
        return $restaurant;
    }
    if (is_object($restaurant) && method_exists($restaurant, 'toArray')) {
        return $restaurant->toArray();
    }
    return [];
}
private function validateOrderFormByPickup(array $order): bool
{
    if (empty($order['restaurant_id'])) {
```

```
            $this->error = '您还没有选择取餐餐厅';
            return false;
        }
        if (empty($order['user_name'])) {
            $this->error = '取餐人姓名不能为空';
            return false;
        }
        if (empty($order['user_mobile'])) {
            $this->error = '取餐人手机号不能为空';
            return false;
        }
        return true;
    }
    private function validateOrderFormByLocker(array $order): bool
    {
        if (empty($order['restaurant_id'])) {
            $this->error = '您还没有选择取餐餐厅';
            return false;
        }
        if (empty($order['locker_code'])) {
            $this->error = '请选择取餐柜';
            return false;
        }
        $locker = $this->getLockerByCode((string)$order['locker_code']);
        if (empty($locker)) {
            $this->error = '取餐柜不存在或已停用';
            return false;
        }
        return true;
    }
    private function getLockerByCode(string $lockerCode): array
    {
        if ($lockerCode === '') {
            return [];
        }
        try {
            $locker = \think\facade\Db::connect('mysql_zhct')
                ->name('equipment')
                ->where('code', $lockerCode)
                ->where('type', EquipmentTypeEnum::ENUM50)
                ->where('del_flag', 0)
                ->field('id,code,name,sn,address')
                ->find();
            return is_array($locker) ? $locker : [];
        } catch (\Throwable $e) {
            return [];
        }
    }
<?php
declare (strict_types=1);
```

```php
namespace app\api\service\cashier;
use app\api\model\MealOrder as OrderModel;
use app\api\model\Payment as PaymentModel;
use app\api\model\PaymentTrade as PaymentTradeModel;
use app\api\model\UserSubsidy as UserSubsidyModel;
use app\api\service\User as UserService;
use app\api\service\Order as OrderService;
use app\api\service\meal\PaySuccess as OrderPaySuccesService;
use app\api\service\order\source\Factory as OrderSourceFactory;
use app\common\service\BaseService;
use app\common\enum\Client as ClientEnum;
use app\common\enum\OrderType as OrderTypeEnum;
use app\common\enum\payment\Method as PaymentMethodEnum;
use app\common\library\payment\Facade as PaymentFacade;
use cores\exception\BaseException;
use app\common\enum\order\OrderSource as OrderSourceEnum;
use app\common\library\helper;
class MealPayment extends BaseService
{
    private string $message = '';
    private int $orderId;
    private ?OrderModel $orderInfo;
    private string $method;
    private string $client;
    public function setOrderId(int $orderId): MealPayment
    {
        $this->orderId = $orderId;
        return $this;
    }
    public function setMethod(string $method): MealPayment
    {
        $this->method = $method;
        return $this;
    }
    public function setClient(string $client): MealPayment
    {
        $this->client = $client;
        return $this;
    }
    public function orderInfo(): array
    {
        $userInfo = UserService::getCurrentLoginUser(true);
        $cahs_balance = $userInfo['balance'];
        $UserSubsidyModel = new UserSubsidyModel;
        $subsidy_balance = $UserSubsidyModel->getCount($userInfo['user_id']);
        $balance = helper::bcadd($cahs_balance,$subsidy_balance);
        $userInfo['balance'] = $balance;
        $userInfo['cahs_balance'] = $cahs_balance;
        $userInfo['subsidy_balance'] = $subsidy_balance;
        $PaymentModel = new PaymentModel;
```

```
    $methods = $PaymentModel->getMethodsByClient($this->client);
    $MealMethods = [];
    foreach ($methods as $item) {
        if ($item['method'] == PaymentMethodEnum::BALANCE) {
            $item['is_default'] = true;
            $MealMethods[] = $item;
        }
    }
    $OrderModel = new OrderModel;
    $orderInfo = $OrderModel->getUnpaidOrderDetail($this->orderId);
    return [
        'order' => $orderInfo,
        'personal' => $userInfo,
        'paymentMethods' => $MealMethods
    ];
}
public function orderPay(array $extra = []): array
{
    $this->orderInfo = OrderModel::getDetail($this->orderId);
    $this->orderPayEvent();
    $payment = $this->unifiedorder($extra);
    $this->recordPaymentTrade($payment);
    return compact('payment');
}
public function tradeQuery(string $outTradeNo): bool
{
    if (!in_array($this->method, [PaymentMethodEnum::WECHAT, PaymentMethodEnum::ALIPAY])) {
        return false;
    }
    $options = $this->getPaymentConfig();
    $Payment = PaymentFacade::store($this->method)->setOptions($options, $this->client);
    $result = $Payment->tradeQuery($outTradeNo);
    if (!empty($result) && $result['paySuccess']) {
        $tradeInfo = PaymentTradeModel::detailByOutTradeNo($outTradeNo);
        $this->orderPaySucces($tradeInfo['order_no'], $tradeInfo['trade_id'], $result);
    }
    return $result ? $result['paySuccess'] : false;
}
private function recordPaymentTrade(array $payment): void
{
    if ($this->method != PaymentMethodEnum::BALANCE) {
        PaymentTradeModel::record($this->orderInfo, $this->method, $this->client, OrderTypeEnum::ORDER, $payment);
    }
}
public function getMessage()
{
    return $this->message;
}
private function orderPayEvent(): void
{
```

```
        $this->checkOrderStatusOnPay();
        if ($this->method == PaymentMethodEnum::BALANCE) {
            $this->orderPaySucces($this->orderInfo['order_no']);
        }
    }
    private function checkOrderStatusOnPay()
    {
        $orderSource = OrderSourceFactory::getFactory($this->orderInfo['source']);
        if (!$orderSource->checkOrderStatusOnPay($this->orderInfo)) {
            throwError($orderSource->getError() ?: '当前订单状态不允许支付');
        }
    }
    private function unifiedorder(array $extra = []): array
    {
        if (!in_array($this->method, [PaymentMethodEnum::WECHAT, PaymentMethodEnum::ALIPAY])) {
            return [];
        }
        $outTradeNo = OrderService::createOrderNo();
        $options = $this->getPaymentConfig();
        $extra = $this->extraAsUnify($extra);
        $Payment = PaymentFacade::store($this->method)->setOptions($options, $this->client);
        if (!$Payment->unify($outTradeNo, (string)$this->orderInfo['pay_price'], $extra)) {
            throwError('第三方支付下单 API 调用失败');
        }
        return $Payment->getUnifyResult();
    }
<?php
declare (strict_types = 1);
namespace app\store\service\dishes;
use app\common\enum\dishes\FoodsStatus;
use app\common\enum\dishes\EnergyRange as EnergyRangeEnum;
use app\common\enum\EnumType as EnumTypeEnum;
use app\common\model\ConfigEnum as ConfigEnumModel;
use app\common\service\BaiduDishes;
use app\store\model\dishes\Dishes as DishesModel;
use app\store\model\dishes\DishesCate as DishesCateModel;
use app\store\model\UploadFile as UploadFileModel;
use app\store\service\store\User as StoreUserService;
use app\store\validate\Dishes as DishesValidate;
use app\store\validate\CustomDishes as CustomDishesValidate;
use app\common\service\BaseService;
use app\terminal\model\dishes\DishesImg as DishesImgModel;
use cores\exception\BaseException;
use think\facade\Cache;
use app\common\Algorithm;
use app\common\enum\dishes\Nutrient as NutrientEnum;
class Dishes extends BaseService
{
    public $id;
    public function getList(array $param = []): array
```

```
{
    $model = new DishesModel();
    $field = 'id,name,category,weight,energy,carbohydrate,protein,fat,file_id,create_by,create_time,source,add_user_id,user_delete';
    $list = $model->getList($param, $field);
    if (empty($list['data'])) {
        return $list;
    }
    $categorys = DishesCateModel::getAllData(['pid' => 0]);
    $categorys = array_column($categorys, null, 'id');
    $file_ids = array_column($list['data'], 'file_id');
    $files = UploadFileModel::getAllData(['file_id' => $file_ids]);
    $files = array_column($files, null, 'file_id');
    $list['data'] = array_map(function ($item) use ($categorys, $files) {
        $item['category_name'] = $categorys[$item['category']]['name'] ?? '';
        $item['file_url'] = $files[$item['file_id']]['preview_url'] ?? '';
        return $item;
    }, $list['data']);
    return $list;
}
public function getAll(array $param = []): \think\Collection
{
    $model = new DishesModel();
    return $model->getAll($param);
}
public function ai()
{
    $last_id = Cache::get('last_id');
    $last_id = $last_id ?? 0;
    $last_id_test = 0;
    $limit = 10;
    $count = 0;
    $model = new DishesModel();
    $field = 'id,name,weight,energy,carbohydrate,protein,fat,file_id';
    $all = $model->with('file')
        ->where('is_delete', '=', 0)
        ->where('source', 'in', [1,2])
        ->where('id', '>', $last_id_test)
        ->field($field)
        ->order(['id' => 'asc'])
        ->limit($limit)
        ->select()
        ->toArray();
    $data = [];
    if (!empty($all)) {
        foreach ($all as $item) {
            $file = $item['file'] ?? null;
            $data[] = [
                'id' => $item['id'],
                'name' => $item['name'],
                'weight' => $item['weight'],
```

```
                'img_url' => $file['preview_url'] ?? '',
            ];
        }
        $count = count($all);
        if ($count == $limit) {
            $last_id = end($all)['id'];
        } else {
            $last_id = 0;
        }
    }
    Cache::set('last_id', $last_id);
    $msg = [
        '更新条数' => $count,
    ];
    \app\common\QwRobot::pushMsgFormat($msg, '扣子 AI 菜品数据库定时更新成功', 'msg');
    return $data;
}
public function add(array $data): bool
{
    $this->validate($data);
    if (DishesModel::checkExistStore($data['name'])) {
        $this->error = '菜品名称已存在';
        return false;
    }
    $create_by = StoreUserService::getLoginUserName();
    $model = new DishesModel;
    $res = $model->add($data, $create_by);
    $this->id = $model['id'] ?: 0;
    return $res;
}
public function detail(int $id)
{
    $with = ['cate', 'file', 'unsuitable', 'foods', 'tag'];
    $detail = DishesModel::detail($id, $with);
    if (empty($detail)) {
        throwError('数据不存在');
    }
    $detail = $detail->toArray();
    $detail['category_name'] = $detail['cate']['name'] ?? '';
    unset($detail['cate']);
    $detail['file_url'] = $detail['file']['preview_url'] ?? '';
    if (!empty($detail['unsuitable'])) {
        $detail['unsuitable'] = array_column($detail['unsuitable'], 'meal_times');
    }
    $cookInfo = ConfigEnumModel::getEnumOne((string)EnumTypeEnum::ENUM4, (string)$detail['cook_method']);
    $detail['cook_method_name'] = $cookInfo['enum_name'] ?? '';
    if (!empty($detail['foods'])) {
        $foods = [];
        foreach ($detail['foods'] as $food) {
            $item = [
```

```
                'foods_id' => $food['pivot']['foods_id'],
                'foods_weight' => $food['pivot']['foods_weight'],
                'foods_ratio' => $food['pivot']['foods_ratio'],
                'foods_status' => $food['foods_status'],
                'foods_status_name' => FoodsStatus::getName($food['foods_status']),
                'foods_name' => $food['name'],
            ];
            $foods[] = $item;
        }
        $detail['foods'] = $foods;
    }
    $EnergyRangeEnum = EnergyRangeEnum::data();
    $energy_lower = $EnergyRangeEnum[1]['value'];
    $energy_upper = $EnergyRangeEnum[2]['value'];
    $detail['energy_upper'] = $EnergyRangeEnum[3]['value'];
    if($detail['energy'] < $energy_lower){
        $detail['energy_status'] = 1;
    }elseif ($detail['energy'] > $energy_upper){
        $detail['energy_status'] = 3;
    }else{
        $detail['energy_status'] = 2;
    }
    return $detail;
}
public function detail100(int $id)
{
    $with = ['cate', 'file', 'unsuitable', 'foods', 'tag'];
    $detail = DishesModel::detail($id, $with);
    if (empty($detail)) {
        throwError('数据不存在');
    }
    $detail = $detail->toArray();
    $carbohydrate_ratio = 0;
    $protein_ratio = 0;
    $fat_ratio = 0;
    if ($detail['energy'] > 0) {
        $carbohydrate_ratio = round($detail['carbohydrate'] * 4 / $detail['energy'] * 100);
        $protein_ratio = round($detail['protein'] * 4 / $detail['energy'] * 100);
        if ($carbohydrate_ratio > 100) $carbohydrate_ratio = 100;
        if ($carbohydrate_ratio + $protein_ratio > 100) $protein_ratio = 100 - $carbohydrate_ratio;
        $fat_ratio = 100 - $carbohydrate_ratio - $protein_ratio;
    }
    $detail['carbohydrate_ratio'] = $carbohydrate_ratio;
    $detail['protein_ratio'] = $protein_ratio;
    $detail['fat_ratio'] = $fat_ratio;
    $nutrientEnum = NutrientEnum::data();
    foreach ($nutrientEnum as $nutrient) {
        $field = $nutrient['field'];
        if (isset($detail[$field])) {
            $detail[$field] = Algorithm::divScale($detail[$field] * 100 , $detail['weight']);
```

```
            }
        }
        $detail['category_name'] = $detail['cate']['name'] ?? '';
        unset($detail['cate']);
        $detail['file_url'] = $detail['file']['preview_url'] ?? '';
        if (!empty($detail['unsuitable'])) {
            $detail['unsuitable'] = array_column($detail['unsuitable'], 'meal_times');
        }
        $cookInfo = ConfigEnumModel::getEnumOne((string)EnumTypeEnum::ENUM4, (string)$detail['cook_method']);
        $detail['cook_method_name'] = $cookInfo['enum_name'] ?? '';
        if (!empty($detail['foods'])) {
            $foods = [];
            foreach ($detail['foods'] as $food) {
                $item = [
                    'foods_id' => $food['pivot']['foods_id'],
                    'foods_weight' => $food['pivot']['foods_weight'],
                    'foods_ratio' => $food['pivot']['foods_ratio'],
                    'foods_status' => $food['foods_status'],
                    'foods_status_name' => FoodsStatus::getName($food['foods_status']),
                    'foods_name' => $food['name'],
                ];
<?php
declare (strict_types=1);
namespace app\api\service\meal;
use app\api\model\MealOrder as OrderModel;
use app\api\model\Restaurant as RestaurantModel;
use app\api\model\User as UserModel;
use app\api\model\UserSubsidy as UserSubsidyModel;
use app\api\model\UserSubsidyUsed as UserSubsidyUsedModel;
use app\api\model\user\BalanceLog as BalanceLogModel;
use app\api\model\user\SubsidyLog as SubsidyLogModel;
use app\api\service\User as UserService;
use app\common\enum\order\OrderSource as OrderSourceEnum;
use app\common\enum\order\OrderStatus as OrderStatusEnum;
use app\common\enum\order\PayStatus as PayStatusEnum;
use app\common\enum\order\PrepareStatus as PrepareStatusEnum;
use app\common\enum\order\PrintStatus as PrintStatusEnum;
use app\common\enum\order\TakeMethod as TakeMethodEnum;
use app\common\enum\payment\Method as PaymentMethodEnum;
use app\common\enum\restaurant\StallType as StallTypeEnum;
use app\common\enum\user\balanceLog\Scene as SceneEnum;
use app\common\enum\user\subsidyLog\Scene as SubsidySceneEnum;
use app\common\library\helper;
use app\common\model\zhct\Staff as ZhctStaffModel;
use app\common\model\zhct\User as ZhctUserModel;
use app\common\service\BaseService;
use think\facade\Db;
class DirectPay extends BaseService
{
    public function pay(int $restaurantId, $money, string $remark = ''): array
```

```
{
    $payPrice = helper::number2((string)$money);
    if (helper::bccomp($payPrice, '0', 2) <= 0) {
        throwError('支付金额必须大于 0');
    }
    $loginUser = UserService::getCurrentLoginUser(true);
    $aiUserId = (int)$loginUser['user_id'];
    $repeatKey = sprintf('meal_direct_pay_%d_%d', $aiUserId, $restaurantId);
    if (!prevent_repeat_submit($repeatKey)) {
        throwError('支付中，请勿重复支付');
    }
    $restaurant = (new RestaurantModel)->getDetail($restaurantId);
    if (empty($restaurant)) {
        throwError('餐厅不存在');
    }
    $zhctUser = (new ZhctUserModel)->getInfoBYAIID($aiUserId);
    if (empty($zhctUser)) {
        throwError('人员数据错误');
    }
    $zhctStaff = (new ZhctStaffModel)->getInfoBYAIID($aiUserId);
    $user = UserModel::detail($aiUserId);
    if (empty($user)) {
        throwError('未找到买家用户信息');
    }
    $balance = $this->getAvailableBalance($aiUserId, (float)$user['balance']);
    if (helper::bccomp($balance['total'], $payPrice, 2) < 0) {
        throwError('余额不足，请充值后再尝试！', 8001);
    }
    $now = date('Y-m-d H:i:s');
    $mealDate = date('Y-m-d', strtotime($now));
    $mealTimes = $this->resolveMealTimes(is_object($restaurant) && method_exists($restaurant, 'toArray') ? $restaurant->toArray() : (array)$restaurant, $now);
    $restaurantMethod = ((int)($restaurant['stall_type'] ?? 0) === StallTypeEnum::ENUM2)
        ? TakeMethodEnum::PICKUP
        : TakeMethodEnum::DINE_IN;
    $orderModel = new OrderModel;
    $orderNo = $orderModel->orderNo();
    $order = [
        'user_id' => (int)$zhctUser['id'],
        'total_price' => (float)$payPrice,
        'order_price' => (float)$payPrice,
        'pay_price' => (float)$payPrice,
        'meal_date' => $mealDate,
        'meal_times' => $mealTimes,
        'meal_number' => 0,
        'bind_id' => 0,
        'bind_time' => $now,
        'last_meal_time' => $now,
        'equipment_code' => '',
        'paycode' => '',
        'card_id' => '',
```

```
            'order_no' => $orderNo,
            'pay_method' => PaymentMethodEnum::BALANCE,
            'pay_status' => PayStatusEnum::SUCCESS,
            'source' => OrderSourceEnum::ORDERFOOD,
            'order_status' => OrderStatusEnum::COMPLETED,
            'pay_time' => $now,
            'ori_total_price' => (float)$payPrice,
            'restaurant_id' => $restaurantId,
            'restaurant_method' => $restaurantMethod,
            'package_fee' => 0,
            'remark' => $remark,
            'user_mobile' => (string)($loginUser['mobile'] ?? ''),
            'user_name' => (string)($loginUser['nick_name'] ?? ($zhctStaff['name'] ?? '')),
            'print_status' => $this->resolvePrintStatus((int)($restaurant['stall_type'] ?? 0)),
        ];
        $result = [];
        Db::transaction(function () use ($orderModel, $order, $aiUserId, $payPrice, $restaurant, &$result) {
            $orderModel->transaction(function () use ($orderModel, $order, $aiUserId, $payPrice, $restaurant, &$result) {
                $orderModel->save($order);
                $orderId = (int)$orderModel['id'];
                $deduct = $this->deductBalance($aiUserId, $orderId, (string)$order['order_no'], $payPrice);
                $pickupNo = $this->ensurePickupRecord(array_merge($order, ['id' => $orderId]), (int)($restaurant['stall_type'] ?? 0));
                $orderModel->save([
                    'cash' => (float)$deduct['cash'],
                    'subsidy' => (float)$deduct['subsidy'],
                    'remain_balance' => (float)$deduct['remain_balance'],
                    'remain_cash' => (float)$deduct['remain_cash'],
                    'remain_subsidy' => (float)$deduct['remain_subsidy'],
                ]);
                UserModel::setIncPayMoney($aiUserId, (float)$payPrice);
                $result = [
                    'orderId' => $orderId,
                    'order_no' => (string)$order['order_no'],
                    'pay_price' => $payPrice,
                    'pay_time' => (string)$order['pay_time'],
                    'restaurant_id' => (int)$order['restaurant_id'],
                    'meal_date' => (string)$order['meal_date'],
                    'meal_times' => (int)$order['meal_times'],
                    'restaurant_method' => (int)$order['restaurant_method'],
                    'pickup_no' => $pickupNo,
                    'remain_balance' => $deduct['remain_balance'],
                    'cash' => $deduct['cash'],
                    'subsidy' => $deduct['subsidy'],
                ];
            });
        });
        return $result;
    }
    private function getAvailableBalance(int $aiUserId, float $cashBalance): array
    {
```

```
    $cash = helper::number2((string)$cashBalance);
    $subsidy = helper::number2((string)(new UserSubsidyModel)->getCount($aiUserId));
    return [
        'cash' => $cash,
        'subsidy' => $subsidy,
        'total' => helper::bcadd($cash, $subsidy, 2),
    ];
}
private function deductBalance(int $aiUserId, int $orderId, string $orderNo, string $payPrice): array
{
    $user = UserModel::detail($aiUserId);
    $initialCash = helper::number2((string)($user['balance'] ?? 0));
    $subsidyRows = (new UserSubsidyModel)
        ->where('user_id', '=', $aiUserId)
        ->where('is_delete', '=', 0)
        ->where('money', '>', 0)
        ->where('expiration_start_time', '<=', date('Y-m-d H:i:s'))
        ->where('expiration_end_time', '>=', date('Y-m-d H:i:s'))
        ->order('expiration_end_time ASC')
        ->select();
    $initialSubsidy = '0.00';
    $remainPay = $payPrice;
    $subsidyUsed = '0.00';
    $cashUsed = '0.00';
    $subsidyUsedModel = new UserSubsidyUsedModel;
    $subsidyModel = new UserSubsidyModel;
    foreach ($subsidyRows as $item) {
        $initialSubsidy = helper::bcadd($initialSubsidy, helper::number2((string)$item['money']), 2);
    }
    foreach ($subsidyRows as $item) {
        if (helper::bccomp($remainPay, '0', 2) <= 0) {
            break;
        }
        $available = helper::number2((string)$item['money']);
        if (helper::bccomp($available, '0', 2) <= 0) {
            continue;
        }
        $used = helper::bccomp($available, $remainPay, 2) >= 0 ? $remainPay : $available;
        $remainPay = helper::bcsub($remainPay, $used, 2);
        $subsidyUsed = helper::bcadd($subsidyUsed, $used, 2);
        $remainMoney = helper::bcsub($available, $used, 2);
        $subsidyModel->where('id', '=', $item['id'])->update([
            'used_money' => helper::bcadd((string)$item['used_money'], $used, 2),
            'money' => $remainMoney,
        ]);
        $subsidyUsedModel->insert([
            'user_id' => $aiUserId,
            'user_subsidy_id' => $item['id'],
            'ori_money' => $available,
            'remain_money' => $remainMoney,
```

```
                'used_money' => $used,
                'order_id' => $orderId,
                'used_time' => date('Y-m-d H:i:s'),
                'store_id' => $this->storeId,
            ]);
        }
        if (helper::bccomp($subsidyUsed, '0', 2) > 0) {
            SubsidyLogModel::add(SubsidySceneEnum::CONSUME, [
                'order_id' => $orderId,
                'user_id' => $aiUserId,
                'money' => -(float)$subsidyUsed,
            ], ['order_no' => $orderNo]);
        }
        if (helper::bccomp($remainPay, '0', 2) > 0) {
            $cashUsed = $remainPay;
            UserModel::setDecBalance($aiUserId, (float)$cashUsed);
            BalanceLogModel::add(SceneEnum::CONSUME, [
                'order_id' => $orderId,
<?php
declare (strict_types = 1);
namespace app\api\service\meal;
use app\api\model\MealOrder as OrderModel;
use app\api\model\Restaurant as RestaurantModel;
use app\common\model\zhct\InitialMenu as InitialMenuModel;
use app\common\model\zhct\OrderEvaluate as OrderEvaluateModel;
use app\common\model\zhct\OrderEvaluateDetail as OrderEvaluateDetailModel;
use app\common\model\zhct\Dishes as DishesModel;
use app\common\enum\order\OrderSource as OrderSourceEnum;
use app\common\enum\order\OrderStatus as OrderStatusEnum;
use app\common\enum\order\PrepareStatus as PrepareStatusEnum;
use app\common\enum\order\EvaluateStatus as EvaluateStatusEnum;
use app\common\enum\order\DishesTaste as DishesTasteEnum;
use app\common\enum\order\LockerStatus as LockerStatusEnum;
use app\common\enum\order\TakeMethod as TakeMethodEnum;
use app\common\service\BaseService;
use cores\exception\BaseException;
use think\App;
use app\api\service\User as UserService;
class Order extends BaseService
{
    private const TAKE_STATUS_NONE = 0;
    private const PACKAGE_FEE_UUID = 'package_fee';
    private const PACKAGE_FEE_NAME = '打包费';
    public function getList($start_date = '', $end_date= '')
    {
        $ai_user_id = UserService::getCurrentLoginUserId();
        $db = \think\facade\Db::connect('mysql_zhct');
        $userInfoZhct = $db->name('user')
            ->where('ai_user_id', $ai_user_id)
            ->where('is_delete', 0)
```

```
    ->find();
if (empty($userInfoZhct)) {
    return [];
}
$userId = $userInfoZhct['id'];
$field = ['id','order_no','bind_time','user_id','meal_date','meal_times','total_price','pay_price','meal_number','pay_status',
    'source','pay_time','order_status','quota','quota_price','evaluate_status','package_fee','restaurant_method',
    'locker_code','locker_address','locker_status','locker_cell_no'];
$query = $db->name('meal_order')
    ->where('user_id','=', $userId)
    ->where('is_delete','=', 0);
if (!empty($start_date)) {
    $query->where('meal_date', '>=', $start_date);
}
if (!empty($end_date)) {
    $query->where('meal_date', '<=', $end_date);
}
$data = $query->field($field)
    ->order(['bind_time' => 'desc'])
    ->paginate(15)
    ->toArray();
if (!empty($data['data'])) {
    $fileUrlPrefix = getFileUrlPrefix();
    $meal_order_ids = array_column($data['data'], 'id');
    $dishes = $db->name('initial_menu')->alias('initial_menu')
        ->join('dishes dishes', 'initial_menu.dishes_uuid = dishes.uuid')
        ->where('initial_menu.meal_order_id','in', $meal_order_ids)
        ->field('initial_menu.meal_order_id,dishes.id,dishes.name,dishes.file_path')
        ->select()
        ->toArray();
    $dishesArr = [];
    if (!empty($dishes)) {
        foreach ($dishes as $key => $value) {
            $value['file_path'] = !empty($value['file_path']) ?
                str_replace("\\","/",$fileUrlPrefix . $value['file_path']) : '';
            $dishesArr[$value['meal_order_id']][] = $value;
        }
    }
    $takeStatusMap = $this->getTakeStatusMap($db, $meal_order_ids);
    foreach ($data['data'] as $key => $value) {
        $data['data'][$key]['goods'] = $dishesArr[$value['id']] ?? [];
        $data['data'][$key]['take_method'] = (int)($value['take_method'] ?? $value['restaurant_method'] ?? TakeMethodEnum::DINE_IN);
        $data['data'][$key] = array_merge($data['data'][$key], $takeStatusMap[$value['id']] ?? $this->getDefaultTakeStatus());
        if ($data['data'][$key]['take_method'] === TakeMethodEnum::LOCKER) {
            $data['data'][$key] = array_merge($data['data'][$key], $this->formatLockerStatus((int)($value['locker_status'] ?? LockerStatusEnum::NONE)));
        }
        if ($value['order_status'] != OrderStatusEnum::COMPLETED && $value['quota'] == 1 && $value['total_price'] > 0) {
            $data['data'][$key]['total_price'] = $value['quota_price'];
        }
    }
```

```
        }
        return $data;
    }
    public function getUserOrderDetail(int $orderId, bool $onlyCurrentUser = true)
    {
        $ai_user_id = UserService::getCurrentLoginUserId();
        $db = \think\facade\Db::connect('mysql_zhct');
        $userInfoZhct = $db->name('user')
            ->where('ai_user_id', $ai_user_id)
            ->where('is_delete', 0)
            ->find();
        if (empty($userInfoZhct)) {
            return [];
        }
        $query = $db->name('meal_order')
            ->where('id','=', $orderId)
            ->where('is_delete','=', 0);
        if ($onlyCurrentUser) {
            $query->where('user_id', '=', $userInfoZhct['id']);
        }
        $field = ['id','order_no','bind_time','user_id','meal_date','meal_times','total_price','pay_price','meal_number','pay_status',

'source','pay_time','user_mobile','user_name','restaurant_method','package_fee','remark','order_status','restaurant_id','quota','quota_price','cash','subsidy','discount_amount',
            'locker_code','locker_address','locker_status','locker_store_time','locker_pickup_time','locker_cell_no','locker_cancel_time'];
        $order = $query->field($field)->find();
        empty($order) && throwError('订单不存在');
        $restaurant_name = '';
        $restaurant_location = '';
        $restaurant_time = '';
        if ($order['source'] == OrderSourceEnum::ORDERFOOD) {
            $RestaurantModel = new RestaurantModel;
            $Restaurant = $RestaurantModel->getDetail($order['restaurant_id']);
            $restaurant_name = $Restaurant['name'];
            $restaurant_location = $Restaurant['location'];
            $restaurant_time = '00:00 - 23:59';
            switch ($order['meal_times']) {
                case 1:
                    $restaurant_time = date('H:i', strtotime($Restaurant['morning_start'])).' - '.date('H:i', strtotime($Restaurant['morning_end']));
                    break;
                case 2:
                    $restaurant_time = date('H:i', strtotime($Restaurant['noon_start'])).' - '.date('H:i', strtotime($Restaurant['noon_end']));
                    break;
                case 3:
                    $restaurant_time = date('H:i', strtotime($Restaurant['night_start'])).' - '.date('H:i', strtotime($Restaurant['night_end']));
                    break;
            }
        }
        if ($order['order_status'] != OrderStatusEnum::COMPLETED && $order['quota'] == 1 && $order['total_price'] > 0) {
            $order['total_price'] = $order['quota_price'];
        }
        $order['take_method'] = (int)($order['take_method'] ?? $order['restaurant_method'] ?? TakeMethodEnum::DINE_IN);
```

```
$order['restaurant_method_text'] = $this->formatTakeMethodText($order['take_method']);
$order['take_method_text'] = $order['restaurant_method_text'];
$order['restaurant_name'] = $restaurant_name;
$order['restaurant_location'] = $restaurant_location;
$order['restaurant_time'] = $restaurant_time;
$packageFeeItem = $this->getPackageFeeItem();
$dishes = $db->name('initial_menu')->alias('initial_menu')
    ->leftJoin('dishes dishes', 'initial_menu.dishes_uuid = dishes.uuid')
    ->where('initial_menu.meal_order_id','=', $order['id'])
    ->field('initial_menu.meal_order_id,initial_menu.dishes_uuid,dishes.id,dishes.name,dishes.file_path,initial_menu.dishes_weight,
    initial_menu.meal_time,initial_menu.total_price,initial_menu.weight,initial_menu.dishes_price')
    ->select()
    ->toArray();
$goods = [];
if (!empty($dishes)) {
    $fileUrlPrefix = getFileUrlPrefix();
    foreach ($dishes as $key => $value) {
        if ((string)($value['dishes_uuid'] ?? '') === self::PACKAGE_FEE_UUID) {
            $value['id'] = 0;
            $value['name'] = $packageFeeItem['name'];
            $value['file_path'] = $packageFeeItem['file_path'];
        } else {
            $value['file_path'] = !empty($value['file_path']) ?
                str_replace("\\",'/',$fileUrlPrefix . $value['file_path']) : '';
        }
        $goods[] = $value;
    }
}
$pay_method_text = '未知';
if ($order['cash'] > 0 && $order['subsidy'] > 0) {
    $pay_method_text = '补贴+现金支付';
} else if ($order['subsidy'] > 0) {
    $pay_method_text = '补贴支付';
} else if ($order['cash'] > 0) {
    $pay_method_text = '现金支付';
}
$order['pay_method_text'] = $pay_method_text;
$order['goods'] = $goods;
$pickup = $db->name('meal_order_pickup')
    ->where('meal_order_id', '=', $order['id'])
    ->field('prepare_status,pickup_no')
    ->find();
$takeStatus = isset($pickup['prepare_status'])
    ? $this->formatTakeStatus((int)$pickup['prepare_status'], (string)($pickup['pickup_no'] ?? ''))
    : $this->getDefaultTakeStatus();
$order = array_merge($order, $takeStatus);
$order['pickup_qrcode'] = $this->buildLockerPickupQrcode((string)$order['order_no'], (string)($order['pickup_no'] ?? ''));
if ($order['take_method'] === TakeMethodEnum::LOCKER) {
    $order = array_merge($order, $this->formatLockerStatus((int)($order['locker_status'] ?? LockerStatusEnum::NONE)));
}
```

```
    return $order;
}
private function getTakeStatusMap($db, array $mealOrderIds): array
{
    if (empty($mealOrderIds)) {
        return [];
    }
    $pickups = $db->name('meal_order_pickup')
        ->whereIn('meal_order_id', $mealOrderIds)
        ->field('meal_order_id,prepare_status,pickup_no')
        ->select()
        ->toArray();
    if (empty($pickups)) {
        return [];
    }
    $data = [];
    foreach ($pickups as $pickup) {
        $data[(int)$pickup['meal_order_id']] = $this->formatTakeStatus(
            (int)($pickup['prepare_status'] ?? self::TAKE_STATUS_NONE),
            (string)($pickup['pickup_no'] ?? '')
        );
    }
    return $data;
}
private function getDefaultTakeStatus(): array
{
    return $this->formatTakeStatus(self::TAKE_STATUS_NONE);
}
private function formatTakeStatus(int $status, string $pickupNo = ''): array
{
    $statusTextMap = [self::TAKE_STATUS_NONE => ''];
    foreach (PrepareStatusEnum::data() as $item) {
        $statusTextMap[(int)$item['value']] = (string)$item['name'];
    }
    return [
        'prepare_status' => $status,
        'prepare_status_text' => $statusTextMap[$status] ?? '',
        'pickup_no' => $pickupNo,
    ];
}
private function formatTakeMethodText(int $takeMethod): string
{
    return TakeMethodEnum::getName($takeMethod) ?: TakeMethodEnum::getName(TakeMethodEnum::DINE_IN);
}
private function formatLockerStatus(int $status): array
{
    $statusText = LockerStatusEnum::getName($status);
    return [
        'prepare_status' => $status,
        'prepare_status_text' => $statusText,
```

```
            'locker_status' => $status,
            'locker_status_text' => $statusText,
        ];
    }
    private function buildLockerPickupQrcode(string $orderNo, string $pickupNo): string
    {
        if ($orderNo === '' || $pickupNo === '') {
            return '';
        }
        $content = json_encode([
            'action' => 'TakeByCode',
            'order' => $orderNo,
            'code' => $pickupNo,
        ], JSON_UNESCAPED_UNICODE);
        return is_string($content) ? $content : '';
    }
    private function getPackageFeeItem(): array
    {
        $item = [];
        try {
            $config = \think\facade\Db::connect('mysql_zhct')
                ->name('config')
                ->where('config_key', 'package_fee_dish')
                ->where('status', 1)
                ->order('id', 'desc')
<?php
declare (strict_types=1);
namespace app\api\controller;
use think\response\Json;
use app\api\service\MealCashier as CashierService;
class MealCashier extends Controller
{
    public function orderInfo(int $orderId, string $client): Json
    {
        $CashierService = new CashierService;
        $data = $CashierService->setOrderId($orderId)->setClient($client)->orderInfo();
        return $this->renderSuccess($data);
    }
    public function orderPay(int $orderId, string $method, string $client, array $extra = []): Json
    {
        $CashierService = new CashierService;
        $data = $CashierService->setOrderId($orderId)
            ->setMethod($method)
            ->setClient($client)
            ->orderPay($extra);
        return $this->renderSuccess($data, $CashierService->getMessage() ?: '下单成功');
    }
    public function tradeQuery(string $outTradeNo, string $method, string $client): Json
    {
        $CashierService = new CashierService;
```