第一阶段

面向所有前端

所有条形码生成的逻辑由前端负责。

客户端

前置业务流程：注册和登录

1. 用户注册。

   调用`customer/sendEmailVerifyCode`发送邮箱验证码，调用`customer/register`验证邮箱验证码正确性并注册。**返回的JSON Web Token应当被存储在前端，并在前端发出的所有request的请求头中添加JWT。**

   > 发送验证码涉及`EmailDTO`

   ```
   public class EmailDTO implements Serializable {
       private String email;
   }
   ```

   > 注册涉及`CustomerRegisterDTO`

   ```
   public class CustomerRegisterDTO implements Serializable {
       private String email;    // 邮箱
       private String password;     // 密码
       private String code;     // 验证码
   }
   ```

2. 用户登录。

   调用`customer/login`利用邮箱登录。**返回的JSON Web Token应当被存储在前端，并在前端发出的所有request的请求头中添加JWT。**

   > 登录涉及`CustomerLoginDTO`

   ```
   public class CustomerLoginDTO implements Serializable {
       private String email;    // 邮箱
       private String password;    // 密码
   }
   ```

辅助业务流程：客户地址簿

TODO

核心业务流程：下单

1. 用户创建运单。

   调用`shipment/createShipment`[1]来创建运单。

   > 涉及`CreateShipmentDTO`

   ```
   public class CreateShipmentDTO implements Serializable {
       private Long origin; // 出发地编号
       private Long destination;     // 目的地编号
       private Long customerId; // 客户ID
       private Integer type;    // 0: 标快，1: 特快
       private String payMethod;    // "cod_pending": 货到付款，"pending": 预先支付
   }
   ```

2. 用户向运单添加包裹。**注意，一个运单可以添加多个包裹，它们有独立的包裹ID，但目的地和出发地应当相同。**

   调用`package/createPackage`[2]来创建包裹。调用`package/calculatePrice`来为单个包裹计算预期价格，这个价格**不作为最终结果**，仅作面向用户展示。**应当在前端用户点击确认完成运单和包裹创建后，再向上述[1]、[2]发送请求。**

   > 创建包裹涉及`CreatePackageDTO`

   ```
   public class CreatePackageDTO implements Serializable {
       private Long shipmentId; // 运单id
       private Long receiverId; // 收件人id（如果在数据库中存在）
       private String receiverName;     // 收件人姓名
       private String receiverAddress; // 收件人地址
       private String receiverPhone;    // 收件人电话
       private Double weight;  // 包裹重量，单位kg
       private String size;     // 包裹尺寸，如 20,30,40 单位cm
   }
   ```

   > 计算价格涉及`CalculatePriceDTO`

   ```
   public class CalculatePriceDTO {
       private Long origin; // 出发地编号
       private Long destination;     // 目的地编号
       private Double weight;  // 包裹重量，单位kg
       private String size;     // 包裹尺寸，如 20,30,40 单位cm
       private Integer type;    // 包裹类型，0为标快，1为特快
   }
   ```

辅助业务流程：查询

1. 查询运单和附属包裹信息。

   调用`shipment/getShipmentInfo`来获取运单和包裹的信息。

   > 涉及URL查询参数`@RequestParam(required = true) Long id // 运单ID`。

2. 查询特定包裹地理位置历史信息。

   调用`package/getPackageLocation`来获取特定包裹的位置历史信息。

> 涉及URL查询参数`@RequestParam(required = true) Long id // 包裹ID`

3. 查询特定用户的所有运单号。

   调用`shipment/getShipmentIds`来获取用户的所有运单号。

   > 涉及URL查询参数`@RequestParam(required = true) Long id // 用户id`

4. 根据邮箱查询用户信息。

   调用`customer/getCustomerByEmail`来查询用户信息。

   > 涉及URL查询参数`@RequestParam(required = true) String email // 用户邮箱`

员工端：底层网点类

核心业务流程：揽收

1. 揽收包裹

   调用`package/pickupPackage`来揽收包裹。

   > 涉及请求体单个参数`@RequestBody Long id // 包裹ID`

核心业务流程：派送

1. 派送包裹

   调用`package/deliverPackage`来揽收包裹。

   > 涉及请求体单个参数`@RequestBody Long id // 包裹ID`

核心业务流程：签收

1. 调用`package/signedPackage`来揽收包裹。

   > 涉及请求体单个参数`@RequestBody Long id // 包裹ID`

员工端：转运类

核心业务流程：转运

1. 创建转运批次。

   调用`batch/createBatch`来创建转运批次，确定该批次的载具信息，并且设定负责人（负责人应当随同载具移动）。

   > 涉及`CreateBatchDTO`

```
public class CreateBatchDTO implements Serializable {
    private Long origin; // 出发地转运中心编号
    private Long destination;    // 目的地转运中心编号
    private Long responsible;    // 负责人id
    private Long vehicleId;  // 载具id
}
```

2. 将包裹加入转运批次。

   调用`package/addPackageToBatch`来将包裹添加进转运批次。

> 涉及`PackageBatchDTO`

```
public class PackageBatchDTO {
    private List<Long> PackageIds; // 要添加的包裹id
    private Long BatchId; // 要添加到的批次id
}
```

3. 转运过程中。

   定时调用`vehicle/updateCoordinate`来更新载具位置信息。该过程会同步更新载具上的包裹的信息。

> 涉及`VehicleDTO`

```
public class VehicleDto {
    private Long id; // 载具id
    private Point coordinate;   // 坐标
}

// 点类定义
public class Point {
    private Double latitude; // 经度
    private Double longitude; // 纬度
}
```

4. 更新批次状态。

   调用`batch/updateBatchStatus`来更新批次状态（转运中/已到达）。

> 涉及`UpdateBatchStatusDTO`

```
public class UpdateBatchStatusDTO implements Serializable {
    private Long batchId; // 批次id
    private String status;  // 状态
}

// 批次状态定义
public enum statusEnum {
    IN_TRANS("in_trans"), // 转运中
    ARRIVE("arrive");   // 已到达
}
```

第二阶段

员工api

员工登录

> 基本信息

- **Path**: /employee/login
- **Method**: Post

> 请求参数

- EmployeeLoginDTO实体
    - ```
      public Result<EmployeeLoginVO> login(@RequestBody EmployeeLoginDTO employeeLoginDTO)
      ```
    - ```
      public class EmployeeLoginDTO implements Serializable {

          private String email;

          private String password;

      }
      ```

> 返回数据

- - EmployeeLoginVO实体
        - ```
          public Result<EmployeeLoginVO> login(@RequestBody EmployeeLoginDTO employeeLoginDTO)
          ```
        - ```
          @ApiModel(description = "员工登录返回的数据格式")
          public class EmployeeLoginVO implements Serializable {

              @ApiModelProperty("主键值")
              private Long id;

              @ApiModelProperty("邮箱")
              private String eamil;

              @ApiModelProperty("jwt令牌")
              private String token;

          }
          ```

员工退出

> 基本信息

- **Path**: /employee/logout
- **Method**: Post

> 请求参数

- 无

> 返回数据

- 无

员工注册

> 基本信息

- **Path**: /employee/enroll
- **Method**: Post

> 请求参数

- EmployeeDTO实体
  - ```
    public Result save(@RequestBody EmployeeDTO employeeDTO)
    ```
  - ```
    public class EmployeeDTO {
        private String name;
        private String phone;
        private String password;
        private String email;
        private Long serveAt;
    }
    ```

> 返回数据

- - result实体
    - ```
      return Result.ok();
      ```
    - ```
      public static <T> Result<T> build(T body, ResultCodeEnum
      resultCodeEnum) {
              Result<T> result = build(body);
              result.setCode(resultCodeEnum.getCode());
              result.setMessage(resultCodeEnum.getMessage());
              return result;
          }
      ```

根据id查询员工信息

> 基本信息

- **Path**: /employee/{id}
- **Method**: Get

> 请求参数

- ```
  public Result<Employee> getById(@PathVariable Long id)
  ```

> 返回数据

- - 封装employee的result实体

查询所有员工信息

> 基本信息

- **Path**: /employee/findAllEmployees
- **Method**: Get

> 请求参数

- 无

> 返回数据

- - employee的List数据
  ```

- ```
  return Result.ok(employeeService.list());
  ```

修改员工信息

> 基本信息

- **Path**: /employee/login
- **Method**: Post

> 请求参数

- Employee实体
  - ```
    public Result update(@RequestBody Employee employee)
    ```
  - ```
    public class Employee {
        private Long id;
        private String name;
        private String phone;
        private String email;
        private String passwordHash;
        private Long serveAt;
    }
    ```

> 返回数据

- - ```
    return Result.ok();
    ```

管理员工端

查询所有员工分页信息

> 基本信息

- **Path**: /adminEmp/getAllPages/{pageNo}/{pageSize}
- **Method**: Get

> 请求参数

- 第几页，每页的数量
- ```
  BackPage<Employee> queryEmployeesPage(@RequestParam("pageNo") Long pageNo, @RequestParam("pageSize") Long pageSize)
  ```

> 返回数据

- - BackPage，里面封装了总页数，当前页数，总数和list内容
  - ```
    public class BackPage<T> {

        private static final long serialVersionUID=1L;

        /**
         * 总页数
         */
        private long totalPage;
    ```

```
    /**
     * 当前页数
     */
    private long currentPage;

    /**
     * 总数
     */
    private long totalNum;

    /**
     * 内容
     */
    private List<T> contentList;
}
```

根据id查询员工信息

> 基本信息

- **Path**: /adminEmp/{id}
- **Method**: Get

> 请求参数

- ```
  public Result<Employee> getById(@PathVariable Long id)
  ```

> 返回数据

- - 封装employee的result实体

修改员工信息

> 基本信息

- **Path**: /adminEmp/update
- **Method**: Post

> 请求参数

- ```
  public Result updateEmployee(@RequestBody Employee employee)
  ```

> 返回数据

- - ```
    return Result.ok();
    ```
    ```

删除员工

> 基本信息

- **Path**: /adminEmp/delete/{id}
- **Method**: Post

> 请求参数

- ```
  public Result deleteEmployee(@RequestParam("id") Long id)
  ```

> 返回数据

- - ```
    return Result.ok();
    ```

添加员工信息

> 基本信息

- **Path**: /adminEmp/create
- **Method**: Post

> 请求参数

- ```
  Result createEmployee(@RequestBody EmployeeDTO employeeDTO)
  ```

- ```
  public class EmployeeDTO {
      private String name;
      private String phone;
      private String password;
      private String email;
      private Long serveAt;
  }
  ```

> 返回数据

- - ```
    return Result.ok();
    ```

管理客户端

查询所有客户分页信息

> 基本信息

- **Path**: /adminCus/getAllPages/{pageNo}/{pageSize}
- **Method**: Get

> 请求参数

- 第几页，每页的数量
- ```
  public BackPage<Customer> queryCustomerPage(@RequestParam("pageNo") Long pageNo, @RequestParam("pageSize") Long pageSize)
  ```

> 返回数据

- BackPage
  - ```
    public class BackPage<T> {

        private static final long serialVersionUID=1L;

        /**
         * 总页数
         */
        private long totalPage;

        /**
         * 当前页数
         */
        private long currentPage;

        /**
         * 总数
         */
        private long totalNum;

        /**
         * 内容
         */
        private List<T> contentList;
    }
    ```

根据id查询用户信息

> 基本信息

- **Path**: /adminCus/{id}
- **Method**: Get

> 请求参数

- ```
  public Result<Customer> getById(@PathVariable Long id)
  ```

> 返回数据

- 封装employee的result实体

修改用户信息

> 基本信息

- **Path**: adminCus/update
- **Method**: Post

> 请求参数

- ```
  Result updateCustomer(@RequestBody CustomerInfoDTO customerInfoDTO)
  ```
  ```

- ```
  public class CustomerInfoDTO implements Serializable {
      private Long id;
      private String username;
      private String phone;
      private String email;
  }
  ```

> 返回数据

- - ```
    return Result.ok();
    ```

删除用户

> 基本信息

- **Path**: /adminCus/delete/{id}
- **Method**: Post

> 请求参数

- ```
  public Result deleteCustomer(@RequestParam("id") Long id)
  ```

> 返回数据

- - ```
    return Result.ok();
    ```

报表统计