EShop网上商城系统 - 控制台版

需求分析

项目需求：

使用控制台作为用户交互界面，实现用户进入购物网站后从首页到最终下单支付的流程中的一系列动作

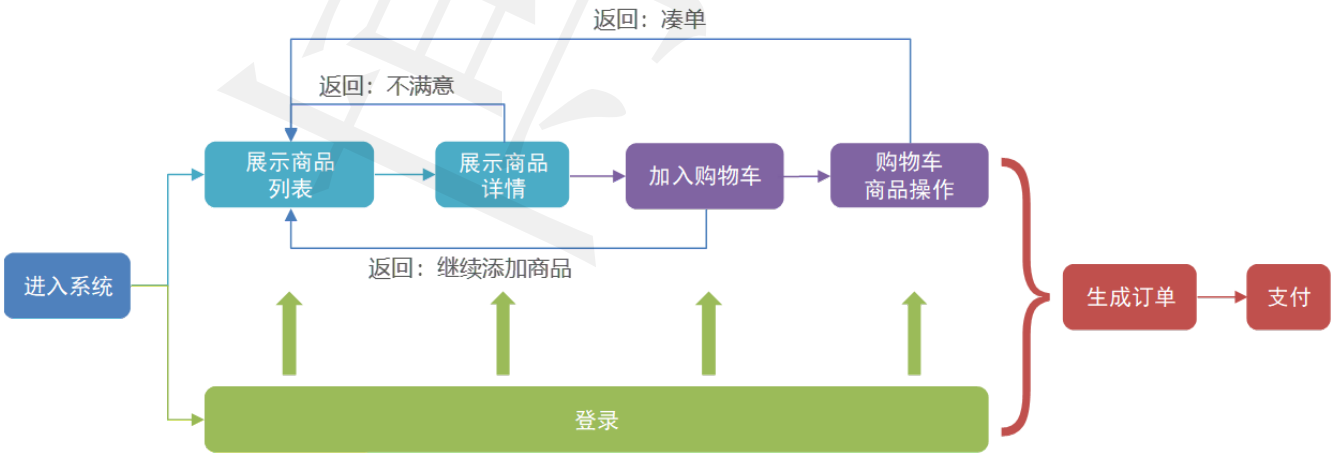
需求分析的目标：

分析出要开发的系统的模块，及每个模块下的功能

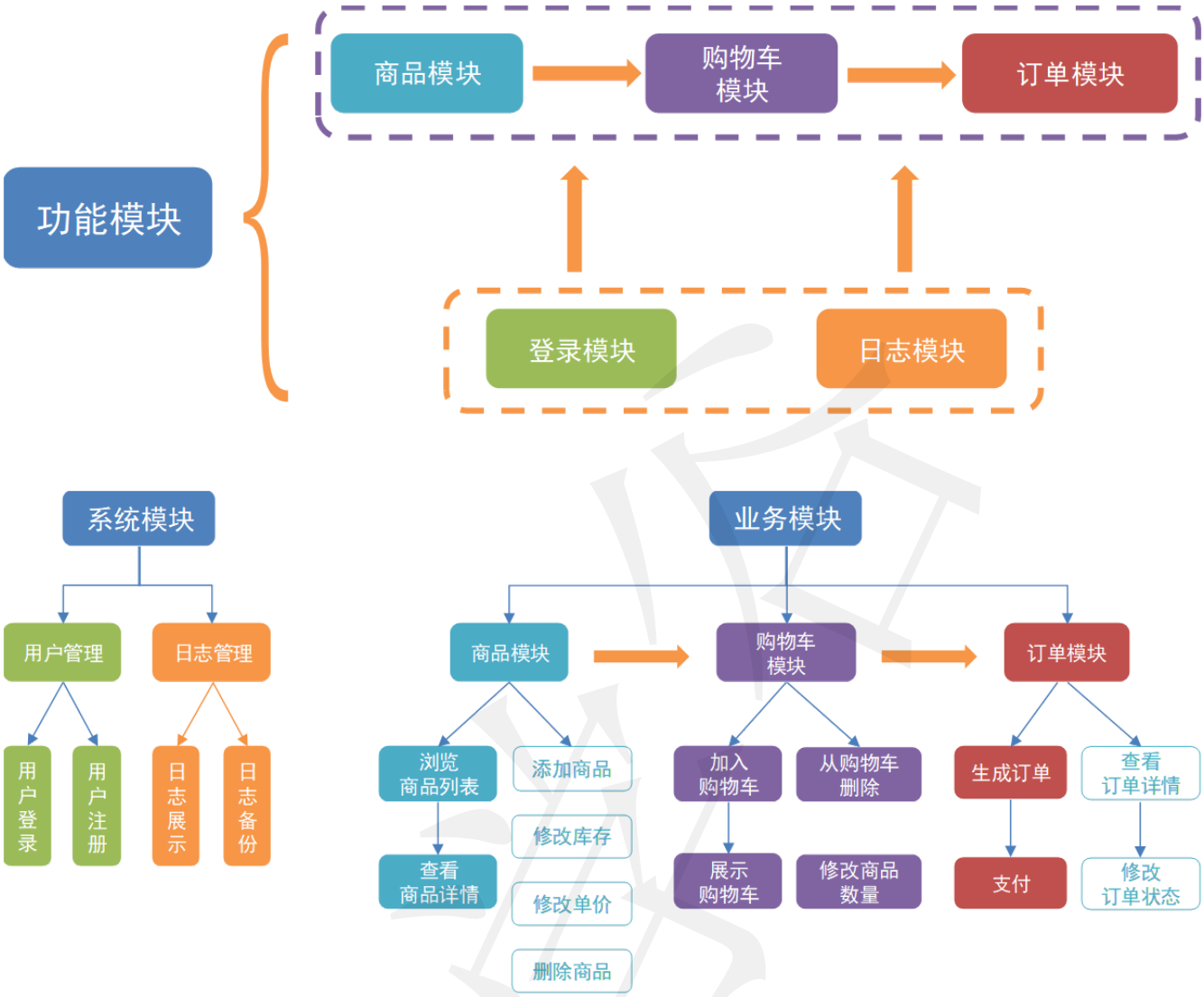
购物网站举例：

https://www.jd.com/

本系统操作流程

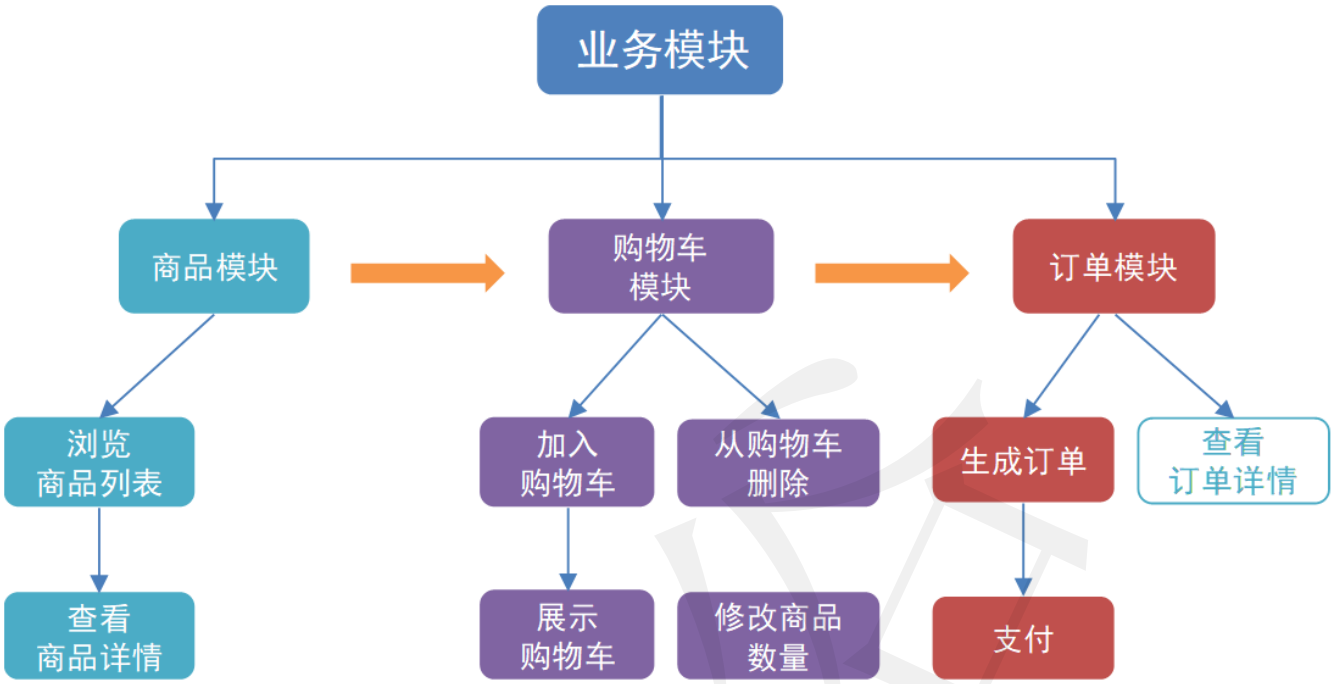


本系统功能模块

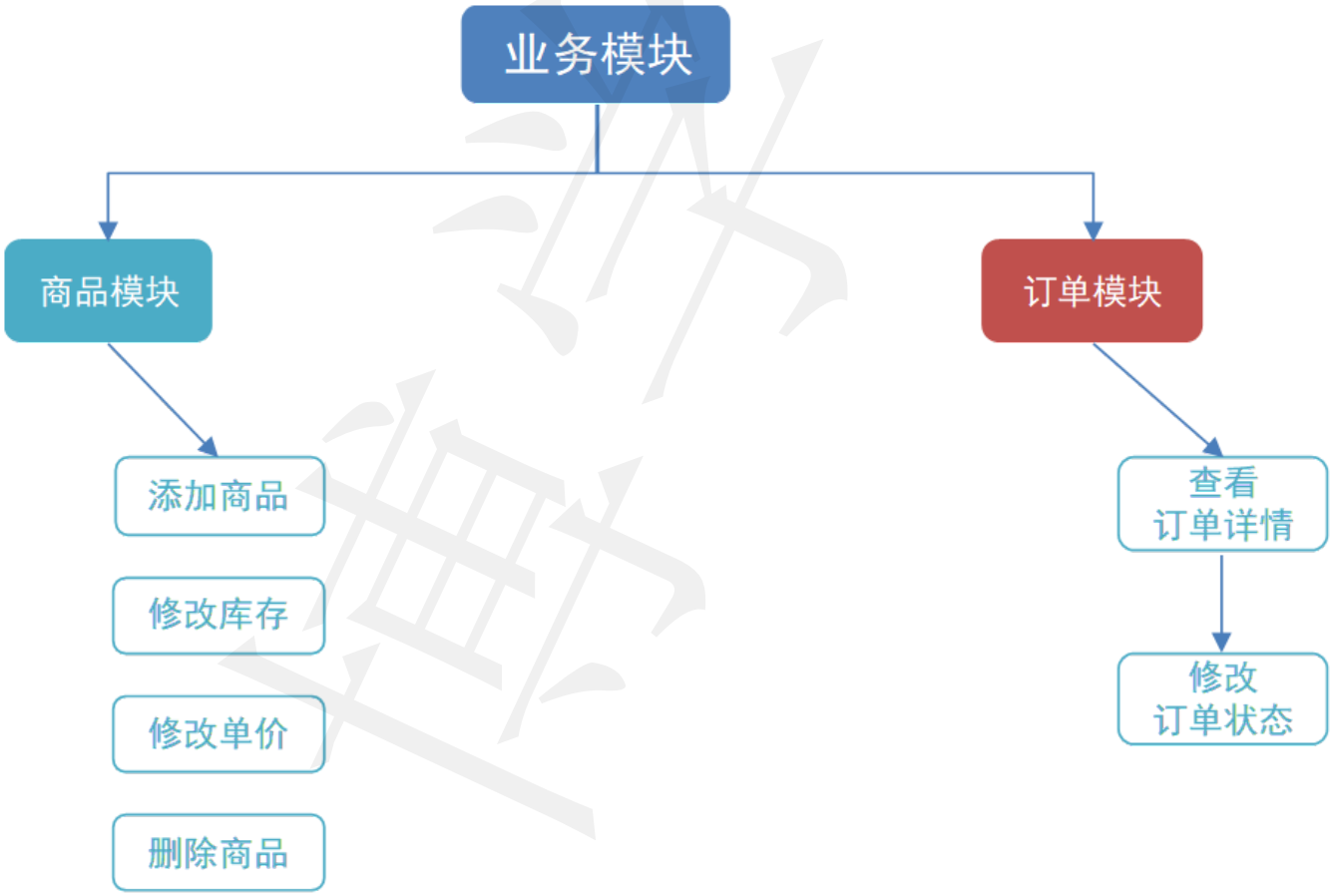


用户角色

普通用户（买家）业务模块



管理员用户（商家）业务模块



概要设计

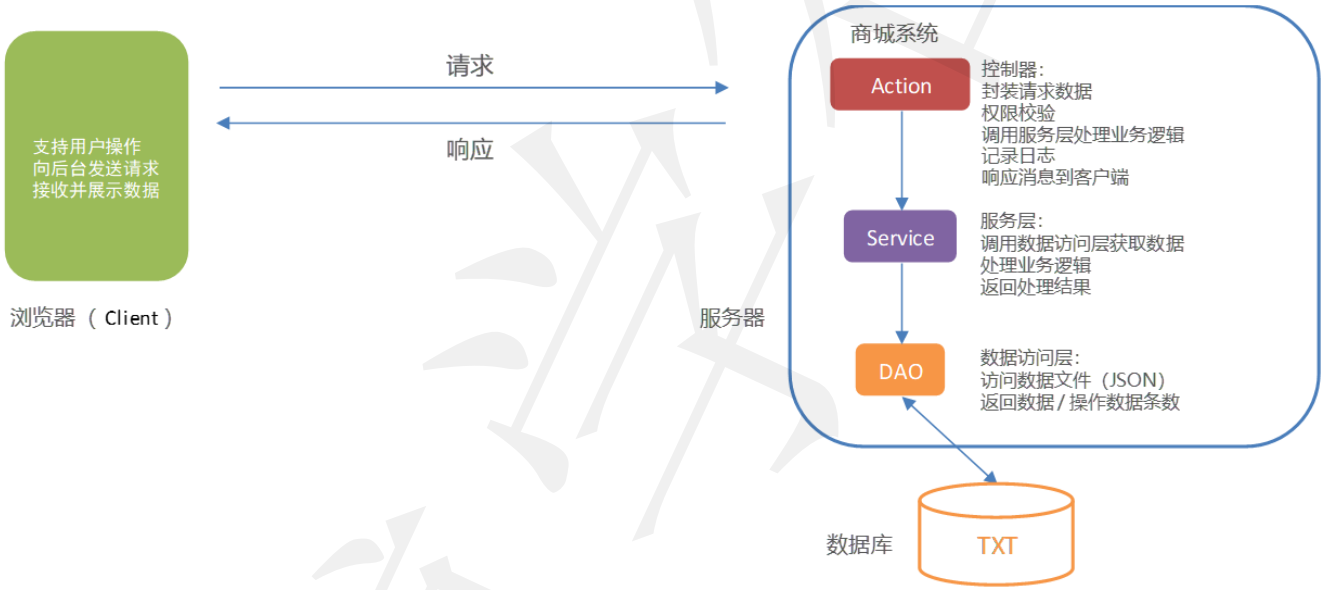
对要开发的系统功能进行设计，包括以下方面：

- 运行流程
- 功能分配
- 接口设计
- 数据结构
- 出错处理
- 日志设计

商城系统的运行流程

一个web网站系统大体由三部分构成：客户端（浏览器、手机端等移动端）、服务器、数据库。客户端主要负责用户交互，向用户展示信息和数据，支持用户操作，并根据用户的操作进行动态响应；服务器端主要负责处理业务数据，进行业务逻辑的处理，并对客户端响应数据；数据库则是用来存储数据的，一般的操作分为添、删、改、查四大类。

客户端访问服务端，一般称为客户端向服务端发送“请求”，反过来，服务端向客户端发消息时，我们称之为向客户端“响应”。请求和响应的过程，是用网络编程相关的技术实现的，涉及到很多网络相关的协议和过程。



商城系统的功能分配

客户端（Client）

- Client 客户端页面的顶层父类
- UserClient 用户（登录/注册）操作页面
- GoodsClient 商品操作页面
- CartClient 购物车操作页面
- OrderClient 订单操作页面

服务端

- BaseAction 控制器基类（ Controller ） UserAction 用户控制器类 GoodsAction 商品控制器类 CartAction 购物车控制器类 OrderAction 订单控制器类
- Service 服务层
- DAO 数据访问层（Data Access Object）

数据库

- TXT文件
- JSON格式存储
- IO流读写

商城系统的接口设计

Service层的接口：可能向多个模块提供服务

- BaseService 服务层的顶层接口，所有模块的服务层接口的父接口
- UserService 用户模块的接口，处理所有与用户数据相关的业务逻辑
- GoodsService 商品模块的接口，处理所有与商品数据相关的业务逻辑
- CartService 购物车模块的接口，处理所有与购物车数据相关的业务逻辑
- OrderService 订单模块的接口，处理所有与订单数据相关的业务逻辑

DAO层的接口：可能向多个服务层提供数据

- BaseDAO 数据访问层的顶层接口，所有模块的数据访问层接口的父接口
- UserDAO 用户模块的接口，处理所有与用户相关的数据
- GoodsDAO 商品模块的接口，处理所有与商品相关的数据
- CartDAO 购物车模块的接口，处理所有与购物车数据相关的数据
- OrderDAO 订单模块的接口，处理所有与订单数据相关的数据

数据库的接口：可能操作不同的数据库

IDataAccess 定义数据的基本操作

日志模块的接口：支持控制台打印和写入本地文件

ISysLog 定义日志的基本操作

商城系统的数据结构设计

实体类：描述现实事物的类

- Entity 实体类，所有子模块的实体类的父类 id 数据的唯一标识，String createTime 数据的创建时间，String deleteTime 数据的删除时间，String isDel 数据的删除状态，String，0已删除，1正常，默认1
- User 用户模块的实体类，用于描述用户的基本信息
- Goods 商品模块的实体类，用于描述商品的基本信息
- Cart 购物车模块的实体类，用于描述购物车的基本信息
- Order 订单模块的实体类，用于描述订单的基本信息

数据文件：和每个实体类是一一对应的

- User db_user.txt
- Goods db_goods.txt
- Cart db_cart.txt
- Order db_order.txt

商城系统的出错处理和日志设计

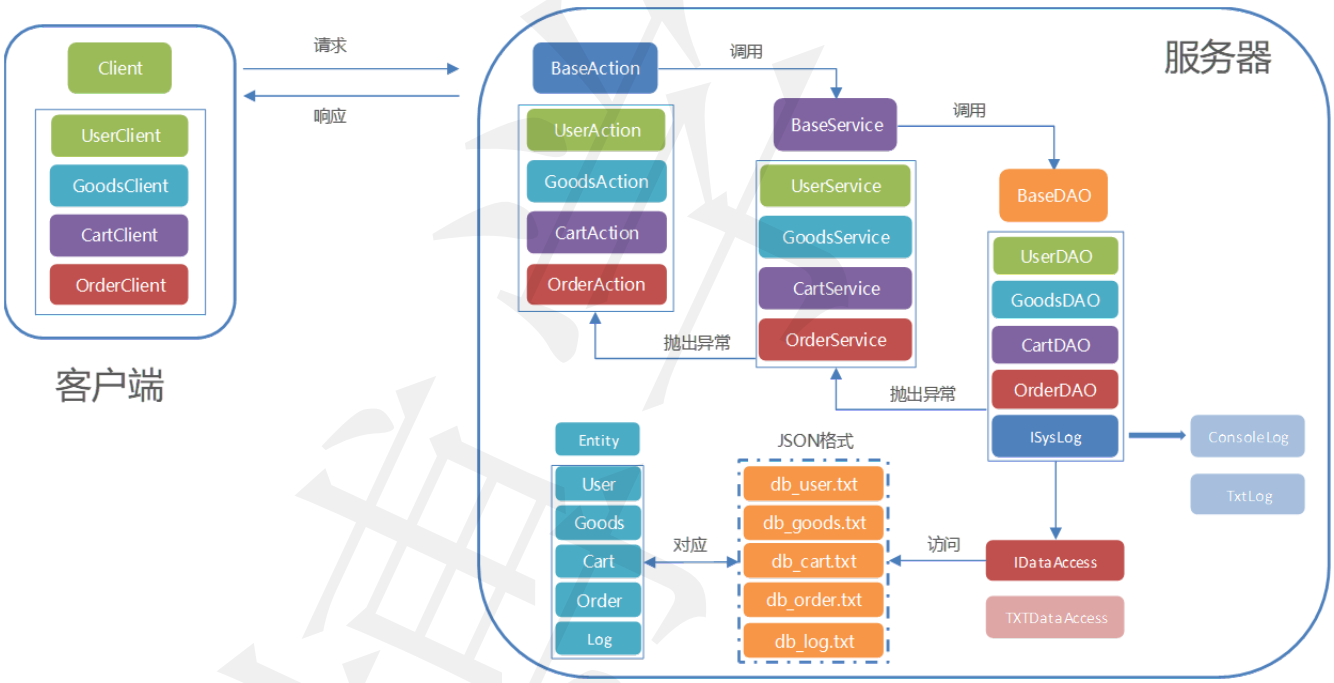
出错处理：顶层调用者处理异常，其它抛出

- Action 在控制器中捕获异常
- Service 服务层的所有异常全部抛出
- DAO 数据访问层的所有异常全部抛出

日志设计：支持控制台打印和写入本地文件

- ISysLog 日志模块接口 info 普通信息 warn 警告信息 error 错误信息
- ConsoleLog 实现类，打印日志信息到控制台
- TxtLog 实现类，打印日志信息到txt文件

概要设计 – 小结



开发前的准备工作

项目模块和分包

创建项目EShop，并添加如下模块

- user 用户模块
- log 日志模块
- goods 商品模块
- cart 购物车模块
- order 购物车模块

- client 客户端模块
- common 公共模块， 存放所有顶层父类、接口，公共bean、工具类等

各模块下的包：

- cn.itcast.eshop.user
- cn.itcast.eshop.log
- cn.itcast.eshop.goods
- cn.itcast.eshop.cart
- cn.itcast.eshop.order
- cn.itcast.eshop.client
- cn.itcast.eshop.common

公共模块的基类和顶层接口

common模块下的基类和接口

- cn.itcast.eshop.common.action.BaseAction
- cn.itcast.eshop.common.service.BaseService
- cn.itcast.eshop.common.dao.BaseDAO
- cn.itcast.eshop.common.dao.IDataAccess
- cn.itcast.eshop.common.entity.Entity

client模块下的基类

cn.itcast.eshop.client.Client

数据交换格式-JSON技术简介

JSON的概念

JSON（ JavaScript Object Notation ）是一种使用文本存储和表示数据的格式

JSON格式数据的表现形式

- { } ： 对象
- [ ] ： 数组
- 对象的数据表示为键值对
- 数据之间用逗号隔开

JSON支持的数据类型（任意组合）

- 数值： 整数对应int，浮点数对应double
- boolean： 布尔型，true， false
- 字符串： 如 “小黑"
- null： Java中的null

- array： 数组 [ ]
- object： 对象 { }

```
{
  "dates": {
    "date": [{
        "id": 1,
        "name": "JSON",
        "abb": "JavaScript Object Notation"
        },
        {
        "id": 2,
        "name": "XML",
        "abb": "eXtensible Markup Language"
        },
        {
        "id": 3,
        "name": "HTML",
        "abb": "HyperText Markup Language"
        }
    ]
  }
}
```

JSONUtil工具类封装

```
// 把对象转为JSON格式的字符串
public static String entity2JSON(Object entity)
```

```
// 把实体对象列表转换成JSON字符串
public static String entityList2JSON(List<?> entityList)
```

```
// 将JSON字符串转换成指定类型对象
public static <T> T JSON2Entity(String json, Class<T> clazz)
public static Object JSON2Entity(String json, Class<?> clazz)
```

```
// 将JSON数组转换成指定类型对象列表
public static <T> List<T> JSONArray2List(String json, Class<T> clazz)
public static List<?> JSONArray2List(String json, Class<?> clazz)
```

用户管理

用户管理模块主要功能

用户登录

输入用户名、密码，并根据用户名、密码获取用户，若该用户存在，则登录成功，否则登录失败

用户注册

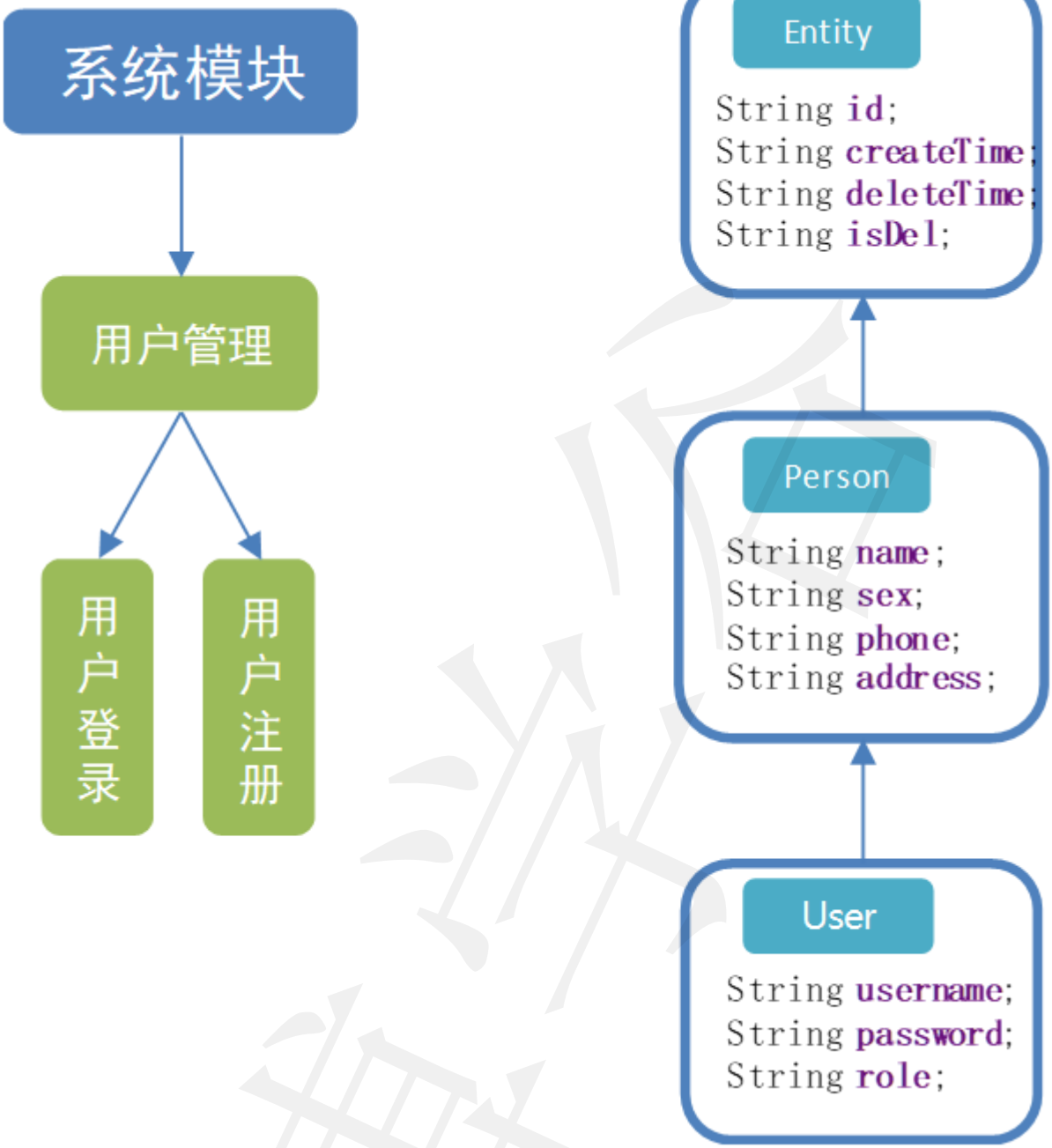
提示用户输入用户信息和个人基本信息，封装成用户对象，向数据库添加一条数据

用户实体类

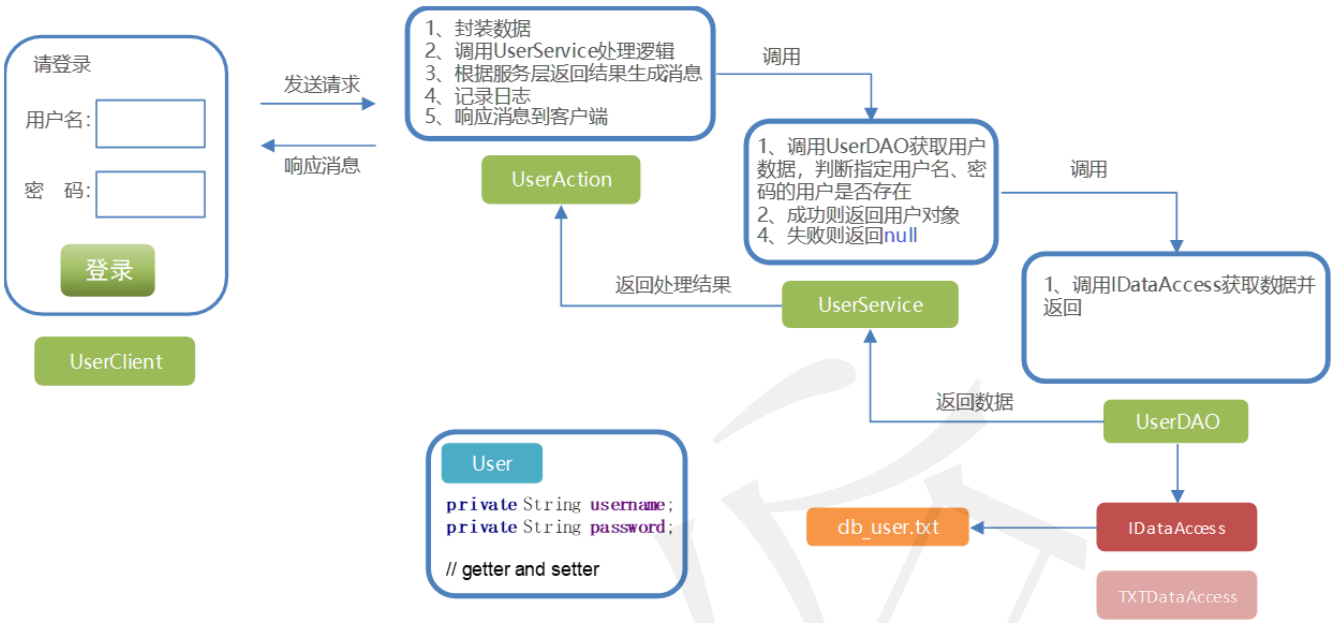
username，password，role（NORMAL，ADMIN）

Person实体类

name，sex，phone，address


系统模块
用户管理
用户登录
用户注册
Entity
String id;
String createTime;
String deleteTime;
String isDel;
Person
String name;
String sex;
String phone;
String address;
User
String username;
String password;
String role;

登录功能分析



客户端登录界面

分析步骤

1. 使用控制台提示用户输入用户名、密码
2. 向服务器发送请求，并接收返回消息字符串 使用setter方法把数据传递给Action 调用Action的登录功能
3. 解析消息字符串，提示用户信息
4. 页面跳转：使用字符串常量作为跳转标记 成功：返回上一次操作的页面 失败：返回登录页面

```
public class UserAction {
    private String username;
    private String password;

    public String login() {
        // return 消息;
    }
    // getter and setter...
}
```

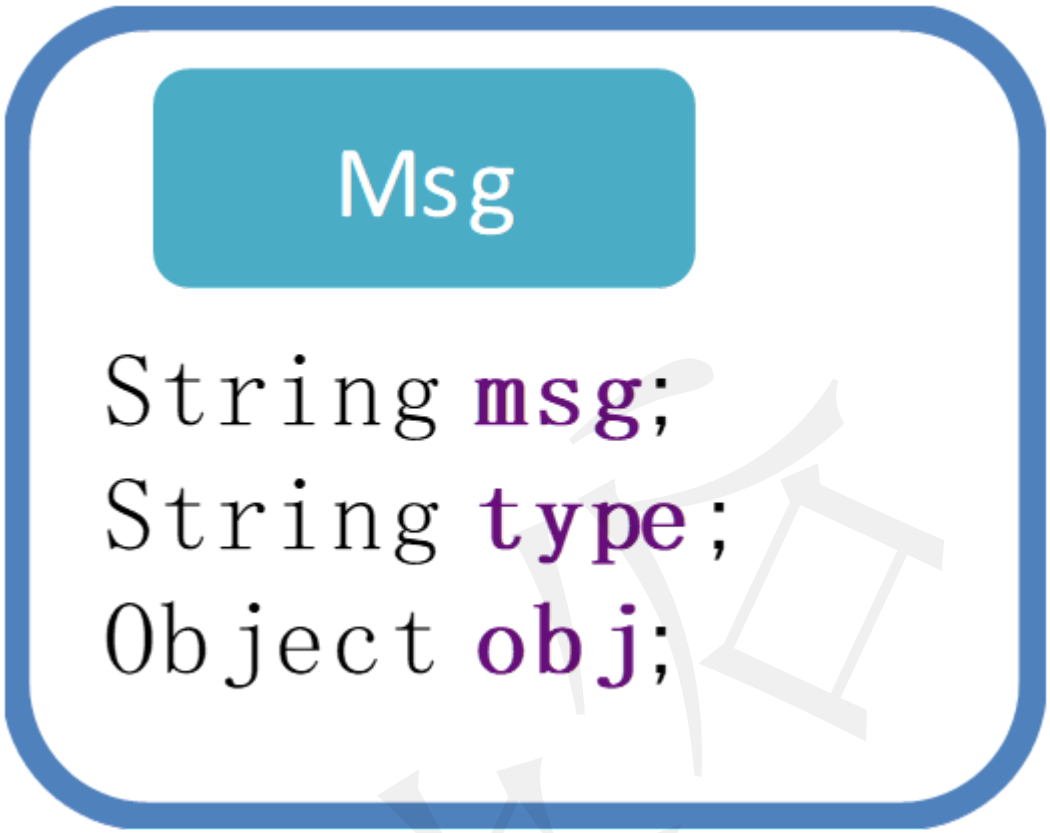
```
/**
 * 用户登录操作页面
 * 1.使用控制台提示用户输入用户名、密码
 * 2.向服务器发送请求，并接收返回消息字符串
 *   使用setter方法把数据传递给Action
```

```
     *  调用Action的登录功能
     * 3.解析消息字符串，提示用户信息
     * 4.页面跳转：使用字符串常量作为跳转标记
     *  成功：返回上一次操作的页面
     *  失败：返回登录页面
     * @return
     */
    public String showLogin() {
        System.out.println("==============欢迎登录==============");
        // 1.使用控制台提示用户输入用户名、密码
        System.out.println("请输入用户名：");
        String username = sc.nextLine();
        System.out.println("请输入密码：");
        String password = sc.nextLine();

        // 2.向服务器发送请求，并接收返回消息字符串
        //     2.1  使用setter方法把数据传递给Action
        userAction.setUsername(username);
        userAction.setPassword(password);
        //     2.2  调用Action的登录功能
        String result = userAction.login();

        // 3.解析消息字符串，提示用户信息
        Msg msg = JSONUtil.JSON2Entity(result, Msg.class);
        if(msg.getType().equals(Msg.SUCCESS)) { // 登录成功
            System.out.println(msg.getMsg());
            // 4.页面跳转：成功：返回上一次操作的页面
            return HISTORY;
        } else {
            System.out.println(msg.getMsg());
            // 4.页面跳转：失败：返回登录页面
            return LOGIN;
        }
    }
```

服务端登录 - Cotroller层的实现

分析步骤

1. 封装数据到User对象
2. 调用UserService处理逻辑 User login(User user) throws Exception;
3. 异常处理
4. 根据服务层返回结果生成消息 消息实体类Msg
5. 记录日志（待开发）
6. 响应消息到客户端




```
/**
 * 用户登录功能
 * 1.封装数据到User对象
 * 2.调用UserService处理逻辑
 *  User login(User user) throws Exception;
 * 3.异常处理
 * 4.根据服务层返回结果生成消息
 *  消息实体类Msg
 * 5.记录日志（待开发）
 * 6.响应消息到客户端
 * @return 返回处理消息，JSON格式
 */
public String login() {
    Msg msg = new Msg();
    try {
        // 1.封装数据到User对象
        User user = new User();
        user.setUsername(username);
        user.setPassword(password);

        // 2.调用UserService处理逻辑
        //     User login(User user) throws Exception;
        UserService userService = new UserServiceImpl();
        user = userService.login(user);

        // 3.异常处理

        // 4.根据服务层返回结果生成消息
        //          消息实体类Msg
```

```
            if(user != null) {
                // 把当前登录用户对象放到上下文对象中
                context.put(LOGIN_USER, user);
                msg.setType(Msg.SUCCESS); // 登录成功
                msg.setMsg("登录成功");
                // 5.记录日志
                log.info(username + "同学已登录");
            } else {
                msg.setType(Msg.FAIL); // 登录成功
                msg.setMsg("用户名或密码不正确");
            }
            // 6.响应消息到客户端
            return JSONUtil.entity2JSON(msg);
        } catch (Exception e) {
            log.error(e.getMessage());
            msg.setType(Msg.FAIL); // 登录失败
            msg.setMsg("服务器异常");
            return JSONUtil.entity2JSON(msg);
        }
    }
```

服务端登录 - Service层的实现

分析步骤

1. 调用UserDAO获取用户列表数据 List getEntityList() throws Exception;
2. 遍历用户列表，逐个与给定用户对象的用户名、密码进行匹配
3. 匹配成功则返回该用户对象，失败返回null
4. 注意事项 实际生产环境中，使用操作数据库的语言（SQL）将用户名、密码作为条件进行查询，本系统暂不支持

```
[
    {
        "username": "admin",
        "password": "123456",
        "role": "ADMIN"
    },
    {
        "username": "xiaohei",
        "password": "xiaohei",
        "role": "NORMAL"
    }
]
```


```java
/**
 * 用户登录，根据用户名、密码获取用户对象
 * 1.调用UserDAO获取用户列表数据
 *   List<User> getEntityList() throws Exception;
 * 2.遍历用户列表，逐个与给定用户对象的用户名、密码进行匹配
 * 3.匹配成功则返回该用户对象，失败返回null
 * @param user 封装了用户名、密码的实体对象
 * @return 返回User对象，或者当用户名/密码错误时返回null
 * @throws Exception
 */
public User login(User user) throws Exception {
    // 1.调用UserDAO获取用户列表数据
    userDAO = new UserDAOImpl();
    List<User> userList = userDAO.getEntityList();

    // 2.遍历用户列表，逐个与给定用户对象的用户名、密码进行匹配
    if(userList != null) {
        for (User u : userList) {
            if(u.getUsername().equals(user.getUsername())
```

```
                    && u.getPassword().equals(user.getPassword())) {
                return u; // 3.匹配成功则返回该用户对象
            }
        }
    }
    return null; // 失败返回null
}
```

服务端登录 - DAO层的实现

分析步骤

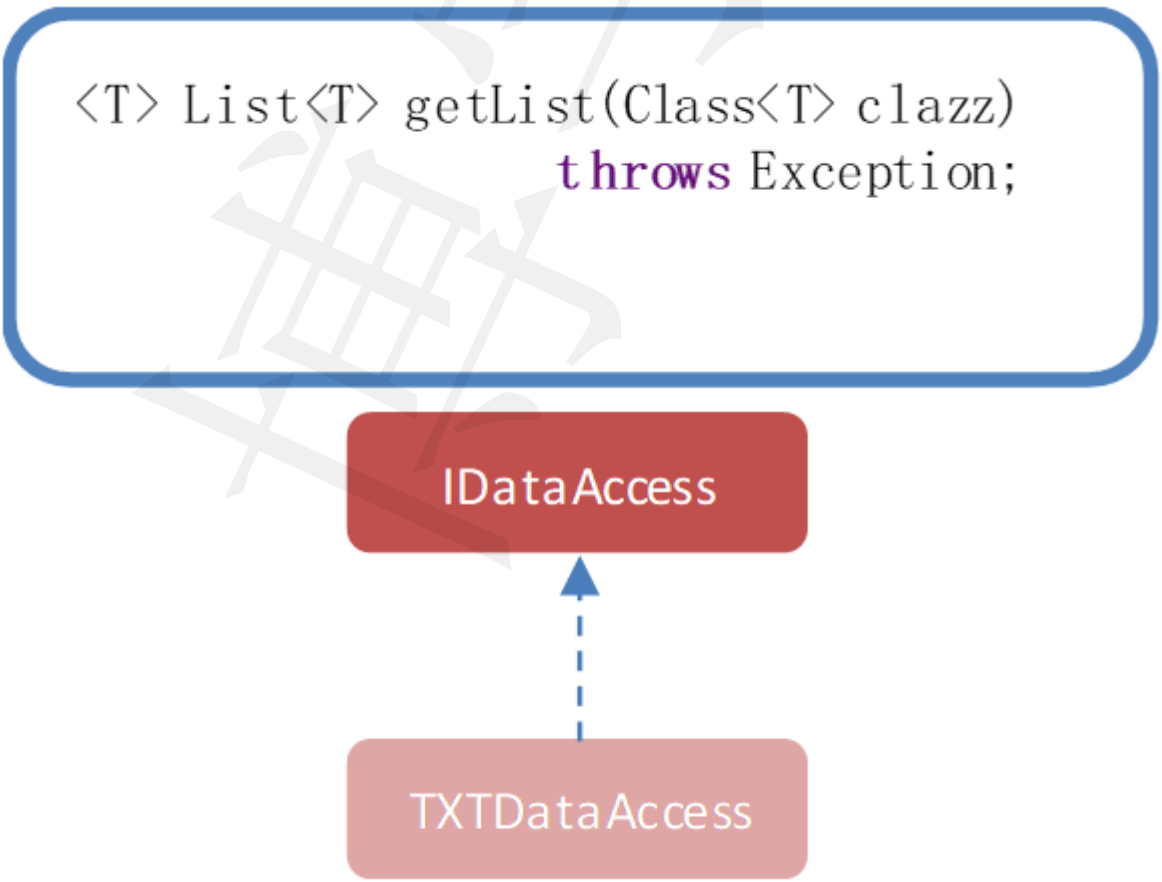
1. 创建IDataAccess子类TXTDataAccess的对象 IDataAccess dataAccess = new TXTDataAccess();
2. 调用该对象的方法获取所有用户数据并返回 List userList = dataAccess.getList(User.class);

getList()方法定义分析

- 作用： 获取全部用户数据
- 返回值： List 用户对象列表
- 参数： Class（JSON转换需要）

getList()方法实现步骤

1. 根据实体类的字节码文件对象获取类名
2. 根据类名获取用户数据文件名
3. 合成数据文件路径
4. 使用文件操作工具类读取文件中的字符串数据
5. 将字符串数据转换成List对象并返回



```
public class BaseDAOImpl implements BaseDAO {

    // 1.创建IDataAccess子类TXTDataAccess的对象
    protected IDataAccess dataAccess = new TXTDataAccess();

}
```

```
   /**
    * 调用IDataAccess获取数据并返回
    * 1.创建IDataAccess子类TXTDataAccess的对象
    *     IDataAccess dataAccess = new TXTDataAccess();
    * 2.调用该对象的方法获取所有用户数据并返回
    *      List<User> userList = dataAccess.getList(User.class);
    * @return 返回用户对象列表
    * @throws Exception
    */
   public List<User> getEntityList() throws Exception {
       // 2.调用该对象的方法获取所有用户数据并返回
       return dataAccess.getList(User.class);
   }
```

日志管理

日志的概念

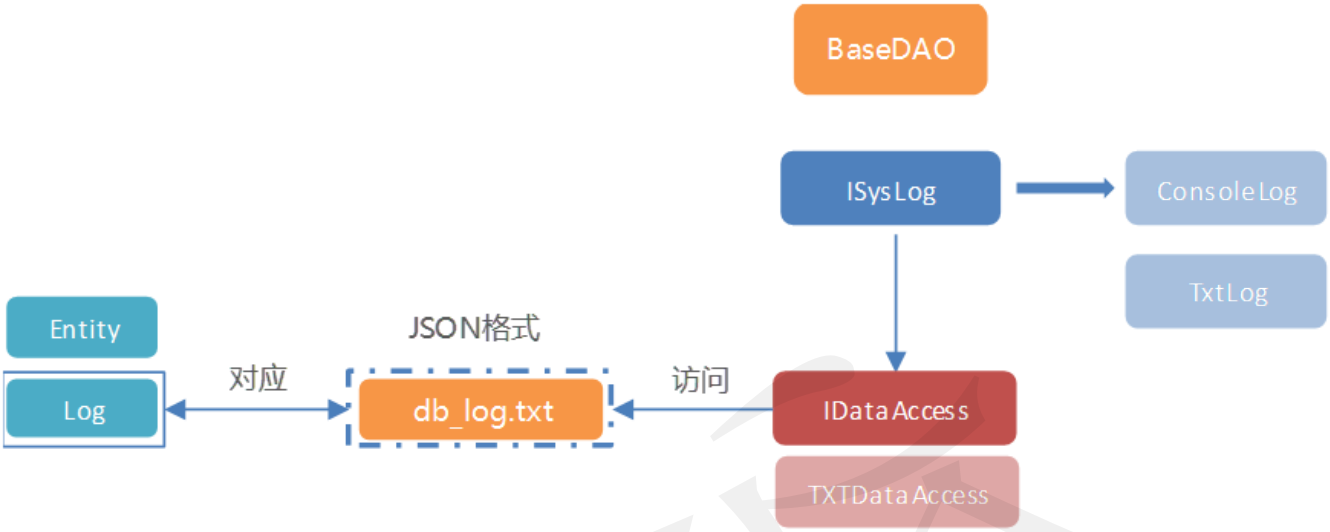
记录系统操作或消息的数据文件，具有处理历史数据、诊断问题等作用

常见日志级别（level）

- info 普通信息
- warn 警告信息
- error 错误信息

实体类Log

- String msg： 日志内容
- String level： 日志级别
- String time： 日志发生时间



ISysLog接口定义

◆ 级别常量

```
public static final String INFO = "INFO";
public static final String WARN = "WARN";
public static final String ERROR = "ERROR";
```

◆ 方法定义

void info(String msg)
void warn(String msg)
void error(String msg)

◆ 日志操作步骤

1.封装日志对象
2.打印日志数据到控制台

```
/**
 * 日志实体类
 */
public class Log {

    // 日志内容
    private String msg;

    // 日志级别
    private String level;

    // 日志发生时间
    private String time;

    public Log() {}

    /**
```

```
     * 构造方法封装日志数据
     * @param msg 日志内容
     * @param level 日志级别
     * @param time 日志发生时间
     */
    public Log(String msg, String level, String time) {
        this.msg = msg;
        this.level = level;
        this.time = time;
    }
    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
    }

    public String getLevel() {
        return level;
    }

    public void setLevel(String level) {
        this.level = level;
    }
    public String getTime() {
        return time;
    }
    public void setTime(String time) {
        this.time = time;
    }
    @Override
    public String toString() {
        // [时间] 级别: 消息内容
        return "["+ time +"] "+ level +": " + msg;
    }
}
```

```
public interface ISysLog {

    /** 日志级别 普通消息 */
    public static final String INFO = "INFO";
    /** 日志级别 警告消息 */
    public static final String WARN = "WARN";
    /** 日志级别 错误消息 */
    public static final String ERROR = "ERROR";


    void info(String msg);

    void warn(String msg);

    void error(String msg);
```

```
}
```

```
import cn.itcast.eshop.log.dao.ISysLog;
import cn.itcast.eshop.log.entity.Log;

import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * 日志实现类
 * 在控制台打印日志信息
 *
 * 步骤:
 *  1.封装日志对象
 *  2.打印日志数据到控制台
 */
public class ConsoleLog implements ISysLog {

    SimpleDateFormat sdf = new SimpleDateFormat("h:mm a"); // 12:08 PM

    @Override
    public void info(String msg) {
        // 1.封装日志对象
        String log = new Log(msg, INFO, sdf.format(new Date())).toString();
        // 2.打印日志数据到控制台
        System.out.println(log);
    }

    @Override
    public void warn(String msg) {
        // 1.封装日志对象
        String log = new Log(msg, WARN, sdf.format(new Date())).toString();
        // 2.打印日志数据到控制台
        System.out.println(log);
    }

    @Override
    public void error(String msg) {
        // 1.封装日志对象
        String log = new Log(msg, ERROR, sdf.format(new Date())).toString();
        // 2.打印日志数据到控制台
        System.out.println(log);
    }

}
```

商品管理

商城首页

```
=======欢迎进入网上商城管理系统=======

【商品列表】
编号        商品名称                单价        库存
-----------------------------------------------
1.          橘子                    3.3         20
2.          香蕉                    3.8         90
3.          山竹                    21.3        20
4.          凤梨                    22.5        99
请根据序号查看商品详情（或 L登录； E退出；）：
```

GoodsClient步骤分析

1. 向后台发送请求，获取商品数据
2. 解析响应消息字符串
3. 展示商品列表

GoodsAction步骤分析

1. 获取所有商品的对象列表
2. 将商品对象列表转换成字符串并返回
3. 异常处理
4. 记录日志
5. 响应消息到客户端

公共用户操作

String userOperate(String msg, String... oprs) msg：提示信息，如“请根据序号...” oprs：用户操作 可变参数，本质是个数组 返回值：用户录入的操作

```
    /**
     * 展示商品列表
     *  1.向后台发送请求，获取商品数据
     *  2.解析响应消息字符串
     *  3.展示商品列表
     */
    public void showGoodsList() {
        // 1.向后台发送请求，获取商品数据
        String msg = goodsAction.getGoodsList();
        // 2.解析响应消息字符串
        Msg msgObj = JSONUtil.JSON2Entity(msg, Msg.class);
        Object obj = msgObj.getObj(); // [{}, {}]
        List<?> goodsListObj = (List<?>) obj;
        System.out.println("【商品列表】");
        System.out.println("编号\t\t商品名称\t\t单价\t\t库存");
        System.out.println("---------------------------------");
        int index = 1; // 编号
        for (Object o : goodsListObj) {
```

```
            // o : {...}
            String goodsJson = o.toString(); // {,,,}
            Goods goods = JSONUtil.JSON2Entity(goodsJson, Goods.class);
            String num = goods.getNumber() + ""; // 库存
            String name = goods.getName(); // 名称
            String price = goods.getPrice() + ""; // 单价
            // 3.展示商品列表
            System.out.println(index + ".\t\t" + name + "\t\t\t" + price + "\t\t" + num);
            code2Goods.put(index + "", goods);
            index ++;
        }
    }
```

```
    /**
     * 获取商品列表
     *
     * 1.获取所有商品的对象列表
     * 2.将商品对象列表转换成字符串并返回
     * 3.异常处理
     * 4.记录日志
     * 5.响应消息到客户端
     * @return
     */
    public String getGoodsList() {
        Msg msg = new Msg();
        // 3.异常处理
        try {
            // 1.获取所有商品的对象列表
            List<Goods> goodsList = goodsService.getGoodsList();
            // 2.将商品对象列表转换成字符串并返回
            msg.setObj(goodsList);
            msg.setType(Msg.SUCCESS);

            String result = JSONUtil.entity2JSON(msg);
            // 4.记录日志
            log.info("获取商品列表");
            // 5.响应消息到客户端
            return result;
        } catch (Exception e) {
            msg.setType(Msg.FAIL);
            msg.setMsg("获取商品列表失败，服务器异常");
            // 4.记录日志
            log.error("获取商品列表失败。" + e.getMessage());
        }
        return JSONUtil.entity2JSON(msg);
    }
```

```
    /**
     * 需求：创建公共的用户操作的方法
     * 主要功能:
```

```
     *  1.提示用户信息和用户操作
     *      请根据编号进行操作（或 L登录；E退出）：
     *  2.接收用户的录入
     *      sc.nextLine()
     * 方法的分析：
     *  方法名：          userOperate
     *  返回值：          String
     *  参数列表：
     *      String msg:      用户信息
     *      String... oprs:  用户操作
     *          可变参数，本质是一个数组
     */
    public String userOperate(String msg, String... oprs) {
        // oprs == String[]
        String opr = Arrays.toString(oprs); // [opr1, opr2, opr3,,]
        opr = opr.substring(1, opr.length() - 1); // opr1, opr2, opr3,,
        msg = msg + "（或 "+ opr +" E退出）：";
        System.out.println(msg); // 请根据编号进行操作（或 L登录；E退出）：
        return sc.nextLine().trim().toUpperCase(); // 去掉空格，转成大写
    }
```

查看商品详情

思路：

商品列表页 --> 输入商品编号 --> 根据编号从Map中获取商品对象 --> 展示商品详细信息

```
=======欢迎进入网上商城管理系统=======

【商品列表】
编号      商品名称            单价      库存
-----------------------------------------------
1.        橘子              3.3       20
2.        香蕉              3.8       90
3.        山竹              21.3      20
4.        凤梨              22.5      99
请根据序号查看商品详情（或L登录；E退出；）：
1

【商品详情】
编号      商品名称            单价      库存      品牌
-----------------------------------------------
1.        橘子              3.3       20        柑橘
```

```
    Map<String, Goods> code2Goods = new HashMap<>();
    /**
     * 查看商品详情
```

```
     * 1.通过数据ID获取数据，然后进行展示
     * 2.在展示商品的时候，把商品列表存储起来，然后，当选择商品编号，就把对应的商品展示出来
     *  Map<String, Goods> :key --> 编号，value --> Goods对象
     */
    public void showGoodsDetail() {
        System.out.println("【商品详情】");
        System.out.println("编号\t\t商品名称\t\t单价\t\t库存\t\t品牌");
        System.out.println("---------------------------------");
        String num = currentGoods.getNumber() + ""; // 库存
        String name = currentGoods.getName();       // 名称
        String price = currentGoods.getPrice() + "";// 单价
        String brand = currentGoods.getBrand();     // 品牌
        // 3.展示商品列表
        System.out.println(1 + ".\t\t" + name + "\t\t\t" + price + "\t\t" + num + "\t\t" +
brand);

    }
```

购物车管理

添加购物车

思路分析：

是实际开发中，购物车的实现大致有两种情况：

第一，以Map集合作为购物车对象，将相关数据临时存储在系统中（内存）

第二，将用户添加到购物车中的商品，以购物车实体类对象的形式存储到数据库中

本系统实现第一种方式，你可以尝试实现第二种方式。

购物车Map集合：

key：商品ID

value：商品数量

```
【商品详情】
编号        商品名称                单价        库存        品牌
-----------------------------------------------------
1.          橘子                    3.3         20          柑橘
输入A加入购物车（或 I返回首页；L登录； E退出；）：
a
添加购物车成功！
请选择操作（或 I继续浏览；C购物车；L登录； E退出；）：
i

【商品列表】
编号        商品名称                单价        库存
-----------------------------------------------------
1.          橘子                    3.3         20
2.          香蕉                    3.8         90
3.          山竹                    21.3        20
4.          凤梨                    22.5        99
请根据序号查看商品详情（或 C购物车；L登录； E退出；）：
```

```
/**
 * 添加购物车
 *
 * 把当前正在操作的商品对象添加到购物车中
 * 1.把currentGoods对象发送请求到Aciton
 * 2.接收Action响应消息
 * @return
 */
public String addCart() {
    // 1.把currentGoods对象发送请求到Aciton
    cartAction.setGoods(currentGoods);
    // 2.接收Action响应消息
    String msgJson = cartAction.addCart();
    Msg msg = JSONUtil.JSON2Entity(msgJson, Msg.class);
    if(msg.getType().equals(Msg.SUCCESS)) {
        System.out.println("添加购物车成功");
    } else {
        System.out.println(msg.getMsg());
    }
    return userOperate("请选择操作", "I继续浏览", "C购物车", "L登录");
}
```

```
/**
 * 添加购物车
 *
 * 把数据存放到cart中
 * 1.如果购物车中已存在该商品，把数量 +1
 * 2.如果不存在，则放进去数字1
 * @return 返回成功或者失败的消息
```

```
     */
    public String addCart() {
        Msg msg = new Msg();
        try {
            Integer num = cart.get(goods.getId());
            if(num != null && num > 0) {
                cart.put(goods.getId(), num + 1);
            } else {
                cart.put(goods.getId(), 1);
            }
            msg.setType(Msg.SUCCESS);
        } catch (Exception e) {
            msg.setType(Msg.FAIL);
            msg.setMsg("服务器异常");
        }
        return JSONUtil.entity2JSON(msg);
    }
```

我的购物车

展示购物车中商品列表，包含每种商品的数量

```
【我的购物车】
编号        商品名称                单价        数量
-----------------------------------------------
1.          香蕉                    3.8         1
2.          橘子                    3.3         1
总金额：7.1
请根据序号选择操作（或I去凑单；O去结算；L登录； E退出；）：
```

```
    /**
     * 展示购物车功能
     * 1.从服务器获取购物车数据List<CartGoods>
     * 2.展示商品列表
     * 3.展示购物车总金额
     * @return
     */
    public String showCart() {
        System.out.println("\n【我的购物车】");
        System.out.println("编号\t商品名称\t\t单价\t\t数量");
        System.out.println("-----------------------------------------------");

        String result = cartAction.showCart();
        Msg msg = JSONUtil.JSON2Entity(result, Msg.class);
        List<?> cartGoodsList = (List<?>)msg.getObj(); // 购物车商品列表
        int i = 1; // 序号
        double sum = 0; // 总金额
        code2Goods = new HashMap<>(); // 将展示编号和数据ID关联起来
```

```
        for (Object obj: cartGoodsList) {
            String json = obj.toString();
            CartGoods cartGoods = JSONUtil.JSON2Entity(json, CartGoods.class);
            int num = cartGoods.getGoodsNum();      // 商品数量
            String name = cartGoods.getName();      // 商品名称
            double price = cartGoods.getPrice();    // 商品单价
            System.out.println(i + "." + "\t" + name + "\t\t\t"
                    + price +  "\t\t" + num);
            sum += price * num; // BigDecimal
            code2Goods.put(i++, cartGoods);
        }
        this.setCartAmount(sum + "");
        System.out.println("总金额: " + sum);

        return userOperate("请根据序号选择操作", "I去凑单 ", "L登录 ", "O去结算 ");
    }
```

```
    /**
     * 展示购物车
     * 1.获取购物车所有商品IDs，并转成字符串，用逗号隔开
     * 2.通过GoodsService对象获取对应的商品列表
     * 3.将Goods对象转成CartGoods对象
     * 4.封装到Msg对象并返回
     * @return
     */
    public String showCart() {
        Msg msg = new Msg();
        try {
            // 购物车中所有的商品IDs
            String ids = Arrays.toString(cart.keySet().toArray()); // [id1, id2, id3]
            List<Goods> goodsList = goodsService.getGoodsList(ids);
            List<CartGoods> cgList = new ArrayList<>();
            for (Goods g: goodsList) {
                CartGoods cg = new CartGoods();
                cg.setId(g.getId());                    // id
                cg.setGoodsNum(cart.get(g.getId()));// 数量
                cg.setName(g.getName());                // 名称
                cg.setPrice(g.getPrice());              // 单价
                cgList.add(cg);
            }
            msg.setType(Msg.SUCCESS);
            msg.setObj(cgList);
        } catch (Exception e) {
            msg.setType(Msg.FAIL);
            msg.setMsg(e.getMessage());
        }
        return JSONUtil.entity2JSON(msg);
    }
```

```
    /**
     * 根据商品ID获取商品对象列表
     * @param ids 商品id，多个用逗号隔开
     * @return 返回商品列表
     * @throws Exception
     */
    public List<Goods> getGoodsList(String ids) throws Exception {
        List<Goods> goodsList = new ArrayList<>();
        List<Goods> allGoodsList = getGoodsList();
        for (Goods goods: allGoodsList) {
            String id = goods.getId();
            if(ids.contains(id)) {
                goodsList.add(goods);
            }
        }
        return goodsList;
    }

    /**
     * 获取商品列表
     *
     * 1.调用GoodsDAO对象的方法获取数据并返回
     * @return 商品列表
     * @throws Exception
     */
    public List<Goods> getGoodsList() throws Exception {
        return goodsDAO.getEntityList();
    }
```

订单管理

生成订单

生成订单的过程，其实就是封装订单实体类的过程。

生成订单前，要验证是否登录。

将订单基本信息如收货人姓名、收货地址、联系方式等封装到订单对象中，根据购物车中商品列表和数量计算出订单总金额。生成订单后展示订单基本信息。

```
【我的购物车】
编号        商品名称                单价        数量
-----------------------------------------------
1.          香蕉                    3.8         10
2.          橘子                    3.3         1
总金额：41.3
请根据序号选择操作（或 I去凑单；O去结算；L登录； E退出；）：
o
------------ 正在登录 ------------
请输入用户名：
xiaohei
请输入密码：
xiaohei
登录成功
------------ 正在生成订单 ------------
请输入收货人：
小黑
请输入收货地址：
北京市昌平区建材城西路金燕龙办公楼一层
请输入联系电话：
1310
------------ 订单已生成 ------------
订单号：    2019    4014-1557674364
总金额：    41.3
状态：      待支付。
```

```java
/* 当前操作订单对象 */
private Order currentOrder;
/**
 * 生成订单
 *  1.填写订单信息
 *  2.封装成Order对象
 *  3.生成订单 --> 待支付状态
 */
public String generateOrder() {
    CartClient cartClient = new CartClient();
    /* 填写订单信息 */
    System.out.print("------------ 正在生成订单 ------------\n");

    System.out.println("请输入收货人：");
    String consignee = sc.nextLine();
    System.out.println("请输入收货地址：");
    String consigneeAddress = sc.nextLine();
    System.out.println("请输入联系电话：");
    String phone = sc.nextLine();

    /* 生成订单 --> 待支付状态 */
```

```
        currentOrder = new Order();
        currentOrder.setId(Entity.getUUID());                // 订单ID
        currentOrder.setCreateTime(sdf.format(new Date())); // 订单生成日期
        currentOrder.setAmount(cartClient.getCartAmount()); // 订单总金额
        currentOrder.setConsigneeAddress(consigneeAddress); // 收货地址
        currentOrder.setConsignee(consignee);                // 收货人
        currentOrder.setPhone(phone);                        // 收货人手机号
        String serialNumber = getSerialNumber();
        currentOrder.setSerialNumber(serialNumber);          // 订单序列号
        currentOrder.setState(Order.WAITING_PAY);            // 待支付

        System.out.println("------------- 订单已生成 -------------\n订单号: \t" + serialNumber
                + " \n总金额: \t" + cartClient.getCartAmount() + "\n状态: \t待支付。");

        return userOperate("请选择操作", "P去支付", "SD查看订单详情 ", "I回首页");
    }
```

```
    /**
     * 获取订单编号，根据当前系统日期生成
     * @return
     */
    public String getSerialNumber() {
        String serialNumber = new Date().getTime() + "";
        int cartHash = this.hashCode();
        return serialNumber + cartHash;
    }
```

```
import cn.itcast.eshop.common.entity.Entity;

public class Order extends Entity {

    /* 订单状态 待支付 */
    public static final String WAITING_PAY = "WAITING_PAY";
    /* 订单状态 待发货 */
    public static final String WAITING_SEND = "WAITING_SEND";
    /* 订单状态 待收货 */
    public static final String WAITING_RECEIVE = "WAITING_RECEIVE";
    /* 订单状态 已完成 */
    public static final String COMPLETED = "COMPLETED";
    /* 订单状态 已取消 */
    public static final String CANCELLED = "CANCELLED";

    /* 订单序列号 根据当前系统日期生成 */
    private String serialNumber;
    /* 收货人 */
    private String consignee;
    /* 收货地址 */
    private String consigneeAddress;
    /* 联系电话 */
    private String phone;
```

```
    /* 订单金额 */
    private String amount;
    /* 订单状态：
     *  待付款：订单生成之后，立即进入待付款状态
     *      支付完成之后，进入支付完成状态，之后需要商家发货
     *  待收货：商家发货之后，进入发货状态
     *  已完成：客户查收商品，进入已完成状态
     *  已取消：订单生成后，任意环节都有可能取消该订单（也可以超时后，系统自动取消）
     */
    private String state;

    public String getSerialNumber() {
        return serialNumber;
    }
    public void setSerialNumber(String serialNumber) {
        this.serialNumber = serialNumber;
    }
    public String getConsignee() {
        return consignee;
    }
    public void setConsignee(String consignee) {
        this.consignee = consignee;
    }
    public String getConsigneeAddress() {
        return consigneeAddress;
    }
    public void setConsigneeAddress(String consigneeAddress) {
        this.consigneeAddress = consigneeAddress;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public String getAmount() {
        return amount;
    }
    public void setAmount(String amount) {
        this.amount = amount;
    }
    public String getState() {
        return state;
    }
    public void setState(String state) {
        this.state = state;
    }
}
```

去支付，订单详情

去支付。根据订单总金额，提示用户输入对应数值，输入数值大于等于总金额则支付成功，否则支付失败，提示余额不足。

订单详情。打印订单信息，和订单状态。

```
------------ 订单已生成 ------------
订单号：   20[redacted]4014-1557674364
总金额：   41.3
状态：     待支付。
请输入支付金额（或 SD查看订单详情；I回首页；E退出；）：
41.3
支付成功！
请选择操作（或 SD查看订单详情；I回首页；E退出；）：
sd
【我的订单】
------------------------------------------------
订单号：20[redacted]4014-1557674364          下单时间：20[redacted] 11:40:14          总金额：41.3          状态：待发货
请选择操作（或 I回首页；E退出；）：
```

```
/**
 * 订单支付
 * 1.在控制台录入一个数值，若该数值大于等于订单总金额，则支付成功，否则余额不足
 * @return
 */
public String pay() { // PAY
    String opr = userOperate("请输入支付金额", "SD查看订单详情 ", "I回首页; ");
    while(true) {
        try {
            Double num = Double.parseDouble(opr); // 将金额转成double类型
            Double sum = Double.parseDouble(currentOrder.getAmount());
            if(num >= sum) {
                System.out.println("支付成功！");
                currentOrder.setState(Order.WAITING_SEND); // 待发货
                return userOperate("请选择操作",  "SD查看订单详情 ", "I回首页 ");
            } else {
                System.out.println("余额不足，请重新支付。");
                opr = userOperate("请输入支付金额");
            }
        } catch(Exception e) {
            return opr; // 未知操作，返回上层
        }
    }
}
```

```
/**
 * 查看订单详情
 * 1.展示订单信息
 *      将订单中商品展示出来
 * 2.根据订单状态，显示正确的描述信息
 * @return
 */
public String showOrder() {
    System.out.println("【我的订单】");
    System.out.println("------------------------------------------------");
    String createTime = currentOrder.getCreateTime();
```

```
        String amount = currentOrder.getAmount();
        String consigneeAddress = currentOrder.getConsigneeAddress();
        String consignee = currentOrder.getConsignee();
        String phone = currentOrder.getPhone();
        String serialNumber = currentOrder.getSerialNumber();
        String state = currentOrder.getState(); // 订单状态

        System.out.println("订单号: " + serialNumber + "\t\t下单时间: " + createTime
                + "\t\t总金额: " + amount + "\t\t状态: "+ getStateDescribtion(state));
        // TODO other infos of order

        String oprs = "I回首页 ";
        if(state.equals(Order.WAITING_PAY)) { // 如果订单未支付，则提供支付功能
            oprs += " P去支付 ";
        }

        return userOperate("请选择操作", oprs);
    }
```

```
    /**
     * 根据订单状态常量，获取对应的描述
     * @param state 订单状态常量
     * @return
     */
    public String getStateDescribtion(String state) {
        switch(state) {
            case Order.CANCELLED:
                return "已取消";
            case Order.COMPLETED:
                return "已完成";
            case Order.WAITING_PAY:
                return "待支付";
            case Order.WAITING_RECEIVE:
                return "待收货";
            case Order.WAITING_SEND:
                return "待发货";
            default:
                return "";
        }
    }
```

注意：请勿拨打本系统中任意位置出现的电话号码，避免给别人带来不必要的打扰。