



# 基于JavaWeb的超市收银系统

🔧 可拓展性、 ⚡ 快速开发、 🚀 减少开发成本

# 课题背景

在现代快节奏的生活中，超市成为人们购物的首选。为了提高超市的效率和顾客的购物体验，需要一个高效的收银系统。基于JavaWeb技术的开发背景下，我们可以开发一个功能强大的超市收银系统。它可以实现商品管理、库存管理、会员管理、购物车管理等功能。通过该系统，超市员工可以快速扫描商品条码、计算价格，并支持多种支付方式。同时，系统还可以生成销售报表和统计数据，为超市经营提供决策依据。这个基于JavaWeb的超市收银系统将提高超市的运营效率，简化工作流程，并为顾客提供便捷、高效的购物体验。



# 系统需求分析

由收银员输入顾客的会员卡卡号（若有卡），所购买的货号。从数据库中取出有关价格信息，在把这些信息返回给收银台，同时把收银台的销售总量和有关种类商品的剩余量以及该持卡顾客的消费情况交数据库存储以供查询。另外，对没有卡的消费不记录该顾客的消费情况托个人信息，如果一个未持卡顾客一次购买满200元，可围棋发放一张会员卡，以后在该商场购物可享受9折优惠。

- 会员信息的管理
- 商品信息（库存）的管理。支持导入商品信息
- 顾客购物
- 顾客订单的管理
- 超市销售情况汇总，支持导出销售情况。



# 系统架构

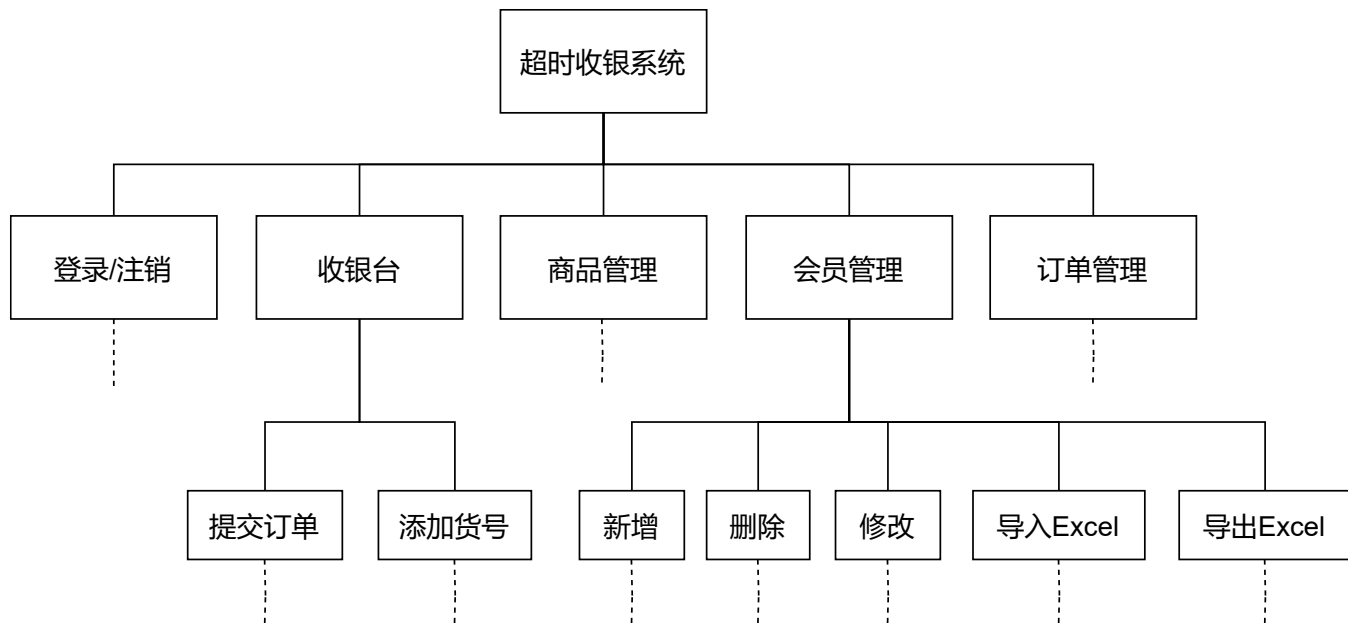


在软件体系架构中，分层式结构必不可少，本系统采用三层架构模式，可以降低层与层之间的依赖性、相互之间高聚能、低耦合，有利于相同业务功能的复用。通常将其划分为MVC架构。数据访问层实现数据持久注入数据库中，实现对数据的增删改，用户访问数据只能通过数据访问层，减少入口，提高其安全性；业务逻辑层接收前台所传递参数数据并接收数据访问层操作后的数据，最后将其返回给表示层进行展示；表示层和用户直接交互，直观、动态展示信息，其与业务逻辑层对接。

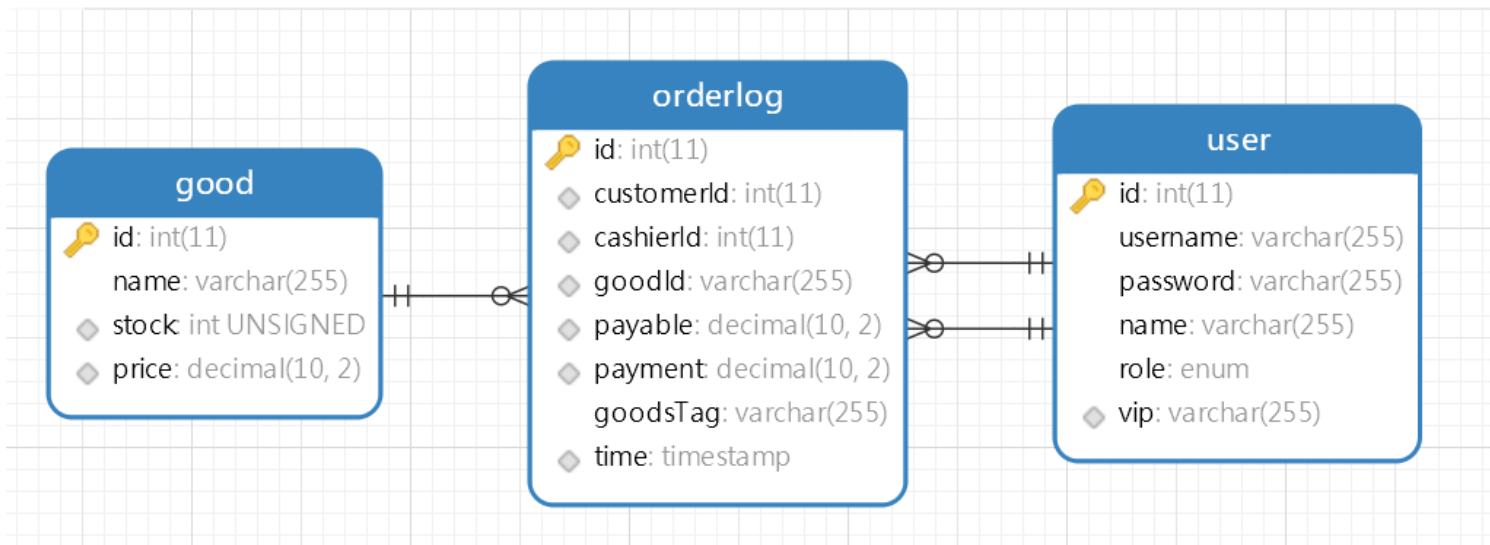
据业务架构实践，结合业界分层规范与流行技术框架分析，推荐分层结构如图所示，默认上层依赖于下层，箭头关系表示可直接依赖，如：开放API层可以依赖于Web层（Controller层），也可以直接依赖于Service层



# 系统层次图



# ER图



# ER图 注释

orderlog /* 订单记录 */		
customerId /* 顾客id */	int(11)	
cashierId /* 收银员id */	int(11)	
goodId /* 商品ids */	varchar(255)	
payable /* 应付 */	decimal(10,2)	
payment /* 实付 */	decimal(10,2)	
goodsTag /* 订单优惠标记 */	varchar(255)	
time /* 交易时间 */	timestamp	
id	int(11)	

user /* 会员信息 */		
username	varchar(255)	
password	varchar(255)	
name /* 名字 */	varchar(255)	
role /* 角色 */	enum('admin', 'customer', 'cashier')	
vip /* 会员卡 */	varchar(255)	
id	int(11)	





good /* 商品 */		
name /* 商品名 */	varchar(255)	
stock /* 库存 */	int(10) unsigned	
price	decimal(10,2)	
id	int(11)	



# 数据字典 – good – 商品表

表结构

关联视图

#	名字	类型	排序规则	属性	空	默认	注释	额外
1	<b>id</b> 	int(11)			否	无		AUTO_INCREMENT
2	<b>name</b> 	varchar(255)	utf8_general_ci		否	无	商品名	
3	<b>stock</b> 	int(10)		UNSIGNED	否	无	库存	
4	<b>price</b> 	decimal(10,2)			否	无		

索引

键名	类型	唯一	紧凑	字段	基数	排序规则	空	注释
PRIMARY	BTREE	是	否	id	1000	A	否	
good_id_stock_price_index	BTREE	否	否	id	1000	A	否	
				stock	1000	A	否	
				price	1000	A	否	





# 数据字典 – orderlog – 销售记录

#	名字	类型	排序规则	属性	空	默认	注释	额外
1	<b>id</b> 🔑	int(11)		否	无			AUTO_INCREMENT
2	<b>customerid</b> 🔑	int(11)		是	NULL		顾客id	
3	<b>cashierid</b> 🔑	int(11)		否	无		收银员id	
4	<b>goodid</b> 🔑	varchar(255)	utf8_general_ci	否	无		商品ids	
5	<b>payable</b> 🔑	decimal(10,2)		否	无		应付	
6	<b>payment</b> 🔑	decimal(10,2)		否	无		实付	
7	<b>goodsTag</b>	varchar(255)	utf8_general_ci	是	NULL		订单优惠标记	
8	<b>time</b> 🔑	timestamp		否	无		交易时间	

索引							
键名	类型	唯一	紧凑	字段	基数	排序规则	空 注释
PRIMARY	BTREE	是	否	id	0	A	否
				customerid	0	A	是
				cashierid	0	A	否
				id	0	A	否
index1	BTREE	否	否	goodid	0	A	否
				payable	0	A	否
				payment	0	A	否
				time	0	A	否



# 数据字典 – user – 用户表

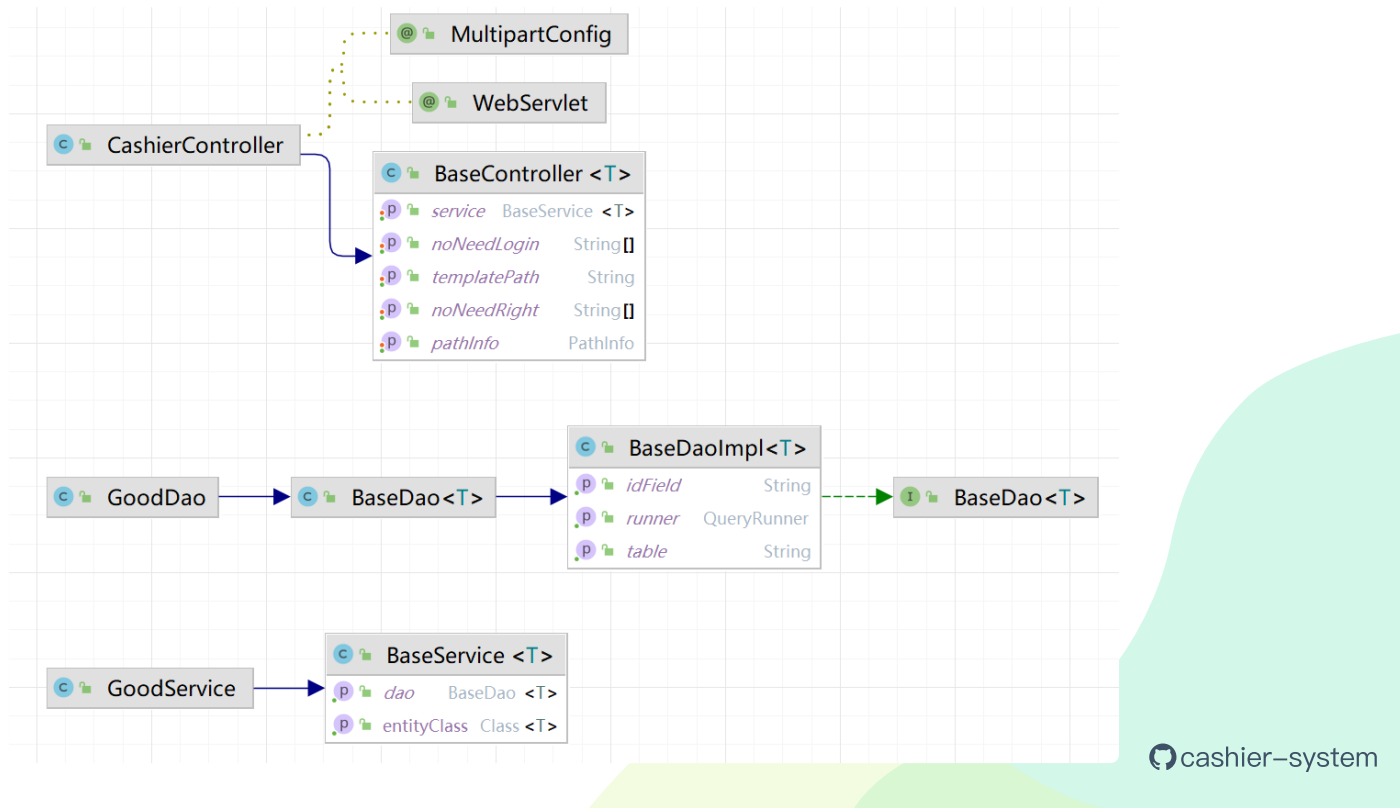
#	名字	类型	排序规则	属性	空	默认	注释	额外
1	<b>id</b> 🔑	int(11)			否	无		AUTO_INCREMENT
2	<b>username</b>	varchar(255)	utf8_general_ci		是	NULL		
3	<b>password</b>	varchar(255)	utf8_general_ci		是	NULL		
4	<b>name</b>	varchar(255)	utf8_general_ci		是	NULL	名字	
5	<b>role</b>	enum('admin', 'customer', 'cashier')	utf8_general_ci		是	NULL	角色	
6	<b>vip</b> 🔑	varchar(255)	utf8_general_ci		是	NULL	会员卡	

索引

键名	类型	唯一	紧凑	字段	基数	排序规则	空	注释
<b>PRIMARY</b>	BTREE	是	否	id	3	A	否	
<b>vip_pk</b>	BTREE	是	否	vip	3	A	是	
<b>user_id_vip_index</b>	BTREE	否	否	id	3	A	否	
				vip	3	A	是	



# 总览类图



# 详细类图和设计

BaseDao<T>	BaseService<T>	BaseController<T>
<ul style="list-style-type: none"> <li>update(T, Map&lt;String, Object&gt;) int</li> <li>insert(T) T</li> <li>selectCount(Map&lt;String, Object&gt;) Long</li> <li>selectMapsPage(P, Map&lt;String, Object&gt;) P</li> <li>updateById(T) int</li> <li>selectByMap(Map&lt;String, Object&gt;) List&lt;T&gt;</li> <li>selectList(T) List&lt;T&gt;</li> <li>selectBatchIds(Collection&lt;Serializable&gt;) List&lt;T&gt;</li> <li>deleteById(Serializable) int</li> <li>selectPage(P) P</li> <li>selectMaps(Map&lt;String, Object&gt;) List&lt;Map&lt;String, Object&gt;&gt;</li> <li>selectById(Serializable) T</li> <li>selectPage(P, Map&lt;String, Object&gt;) P</li> <li>deleteByMap(Map&lt;String, Object&gt;) int</li> <li>deleteBatchIds(Collection&lt;?&gt;) int</li> <li>selectKeyBatchIds(Collection&lt;Serializable&gt;) Map&lt;String, T&gt;</li> <li>selectOne(T, boolean) T</li> <li>exists(Map&lt;String, Object&gt;) boolean</li> </ul>	<ul style="list-style-type: none"> <li>dao</li> <li>entityClass</li> <li>keyedListByIds(Collection&lt;Serializable&gt;) Map&lt;String, T&gt;</li> <li>saveOrUpdate(T) T</li> <li>update(Map&lt;String, Object&gt;) boolean</li> <li>page(E, Map&lt;String, Object&gt;) E</li> <li>saveOrUpdateBatch(Collection&lt;T&gt;, int) boolean</li> <li>updateById(T) boolean</li> <li>updateBatchById(Collection&lt;T&gt;, int) boolean</li> <li>count(Map&lt;String, Object&gt;) long</li> <li>listMaps() List&lt;Map&lt;String, Object&gt;&gt;</li> <li>getById(Serializable) T</li> <li>page(E) E</li> <li>pageMaps(E, Map&lt;String, Object&gt;) E</li> <li>count() long</li> <li>list() List&lt;T&gt;</li> <li>remove(Map&lt;String, Object&gt;) boolean</li> <li>removeById(T) boolean</li> <li>listByIds(Collection&lt;Serializable&gt;) List&lt;T&gt;</li> <li>getOne(T, boolean) T</li> <li>update(T, Map&lt;String, Object&gt;) boolean</li> <li>removeByMap(Map&lt;String, Object&gt;) boolean</li> <li>updateBatchById(Collection&lt;T&gt;) boolean</li> <li>save(T) T</li> <li>saveBatch(Collection&lt;T&gt;, int) boolean</li> <li>listMaps(Map&lt;String, Object&gt;) List&lt;Map&lt;String, Object&gt;&gt;</li> <li>removeByIds(Collection&lt;?&gt;) boolean</li> <li>pageMaps(E) E</li> <li>listByMap(Map&lt;String, Object&gt;) List&lt;T&gt;</li> <li>list(Map&lt;String, Object&gt;) List&lt;T&gt;</li> <li>getOne(T) T</li> <li>saveBatch(Collection&lt;T&gt;) boolean</li> <li>saveOrUpdateBatch(Collection&lt;T&gt;) boolean</li> <li>removeById(Serializable) boolean</li> </ul>	<ul style="list-style-type: none"> <li>service</li> <li>noNeedLogin</li> <li>templatePath</li> <li>noNeedRight</li> <li>pathInfo</li> <li>doPost(HttpServletRequest, HttpServletResponse) void</li> <li>dispatch(HttpServletRequest, HttpServletResponse) void</li> <li>autoForward(HttpServletRequest, HttpServletResponse) void</li> <li>autoForward(HttpServletRequest, HttpServletResponse, String) void</li> <li>forwardHome(HttpServletRequest, HttpServletResponse) void</li> <li>doGet(HttpServletRequest, HttpServletResponse) void</li> <li>save(HttpServletRequest, HttpServletResponse) void</li> <li>getIndexFields(HttpServletRequest, HttpServletResponse) Map&lt;String, FieldDescriptor&gt;</li> <li>importXls(HttpServletRequest, HttpServletResponse) void</li> <li>getAddFields(HttpServletRequest, HttpServletResponse) Map&lt;String, FieldDescriptor&gt;</li> <li>getUpdateFields(HttpServletRequest, HttpServletResponse) Map&lt;String, FieldDescriptor&gt;</li> <li>delete(HttpServletRequest, HttpServletResponse) void</li> <li>add(HttpServletRequest, HttpServletResponse) void</li> <li>index(HttpServletRequest, HttpServletResponse) void</li> <li>edit(HttpServletRequest, HttpServletResponse) void</li> <li>exportXls(HttpServletRequest, HttpServletResponse) void</li> </ul>

# 系统测试

为保障该系统质量，系统的测试环节必不可少，除了要实现规定的功能以外，还要满足安全、兼容、负载性能下也能正常运行。本文具体做出以下测试：

- 1) 功能测试：针对功能测试，选择对订单查询与修改操作进行测试。登录系统后，在订单管理界面选择订单记录，返回一条对应订单，在数据库中查询得到数据一致；选择第一条001订单号进行信息修改，删除名称为“护发素”的商品，保存之后返回界面查看订单与修改一致。
- 2) 安全测试：针对安全测试，首先直接访问主页界面，系统弹出并未登录的信息对其做出有效拦截；对于订单、商品、用户数据修改和删除时，模拟中途网络出现故障，这时查看MySQL数据库，数据并未发生改变。
- 3) 兼容测试：对于系统编写完成后，分别将其部署在本地系统上与云服务器上Linux操作系统上。测试之后，本端可通过本机网络分配IP地址进行访问，远端服务器可通过其分配的IP地址进行访问。
- 4) 负载测试：本次测试中，10个用户访问系统进行一系列操作。在整个的测试过程中，系统并未出现闪退、未响应、页面加载错误等问题，在Windows操作系统下的系统出现延迟高，在远程服务器上良好。

对于整个系统测试，虽然在超高负载下出现延迟高等问题，后续需做出调整，但也达到了中小型超市的日常使用要求。

