China Conference

# 领域驱动：

# 看我如何拥抱需求变更

范钢

China Conference

# 我们现在面对的是快速变化的时代

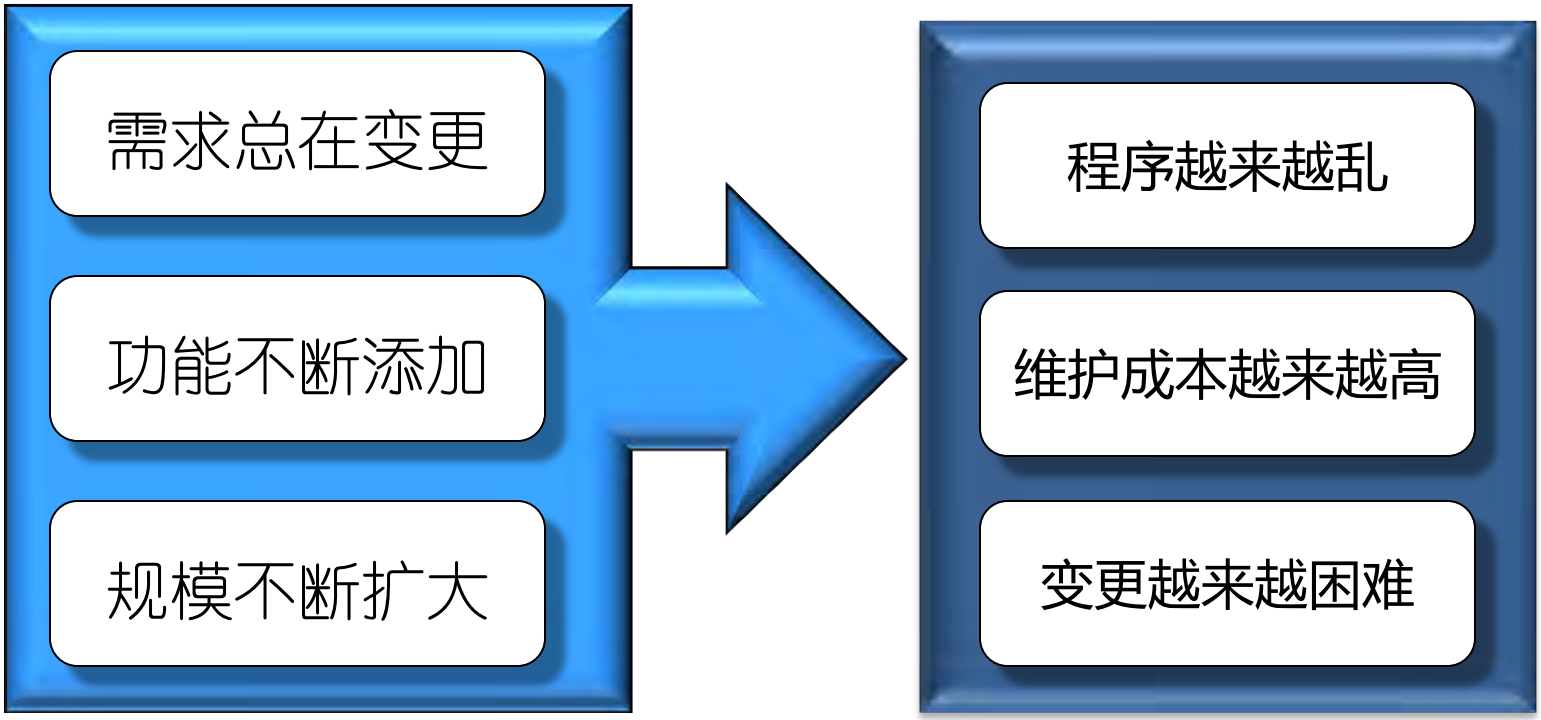
JD.京东 JD.COM

11.11 全球狂欢节 2017

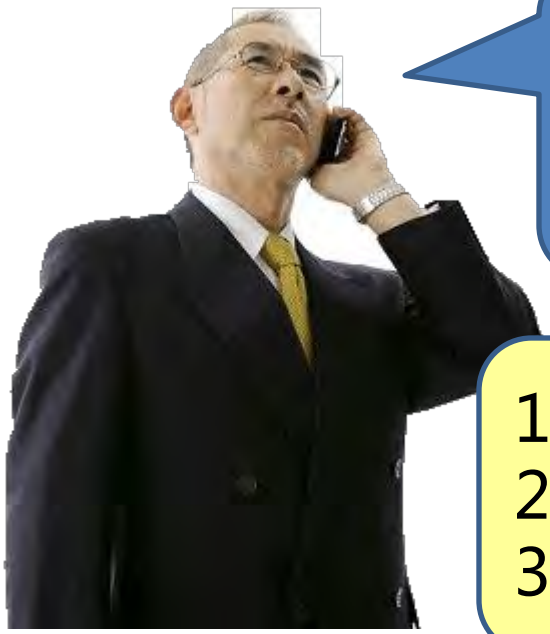

移动应用
互联网转型
架构演进

微服务
大数据转型
人工智能

# 我们的软件研发出现了问题



# 我们遇到了什么困难？

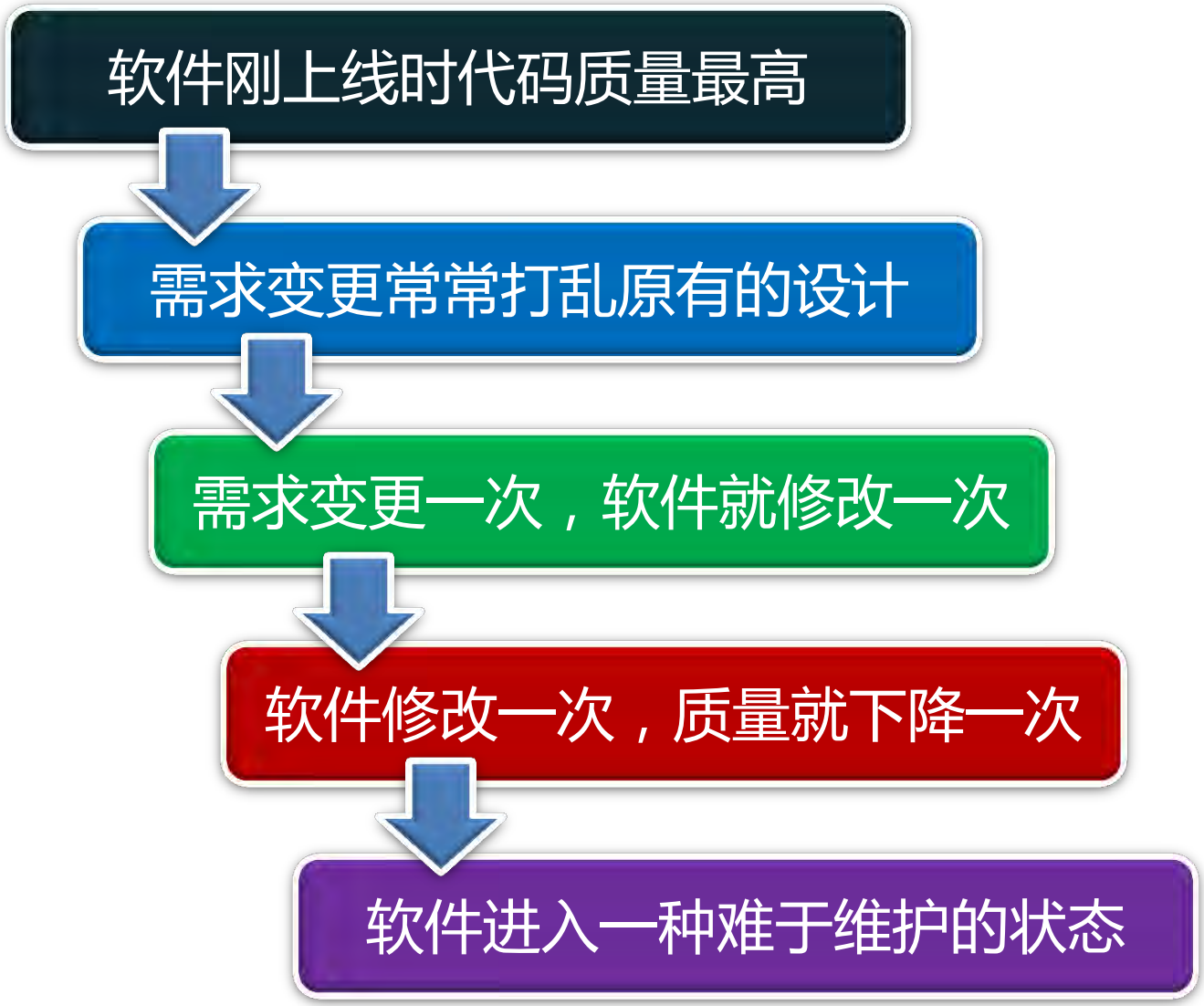


1. 维护成本越来越高
2. 测试越来越困难
3. 开发周期越来越长

1. 程序越来越乱
2. 变更越来越困难
3. 非常难引入新技术

1. 无法跟上市场需求
2. 无法跟上技术更迭

# 频繁的变更带来软件质量的下降



# 电商网站付款功能

```
public boolean payoff(Order order, String addressI
    //收集客户信息
    String customerId = SessionService.getUserId()
    Customer customer = dao.getCustomer(customerId
    order.setCustomer(customer);
    Address address = dao.getAddress(addressId);
    order.setAddress(address);

    //计算付款
    double amount = 0;
    for(OrderDetail detail : order.getDetails()) {
        amount += detail.getQuantity()*detail.getPrice();
    }
    order.setAmount(amount);
    Serializable orderId = dao.save(order);

    //跳转支付页面
    WebserviceFactory factory = new WebserviceFactory();
    PaymentWebservice payment =
        (PaymentWebservice)factory.getWebservice("aliPayment"
    return payment.payoff(orderId, customer, amount);
}
```

VIP会员：
- 金银卡会员
- 会员福利
- 会员特权

秒杀

预订

闪购

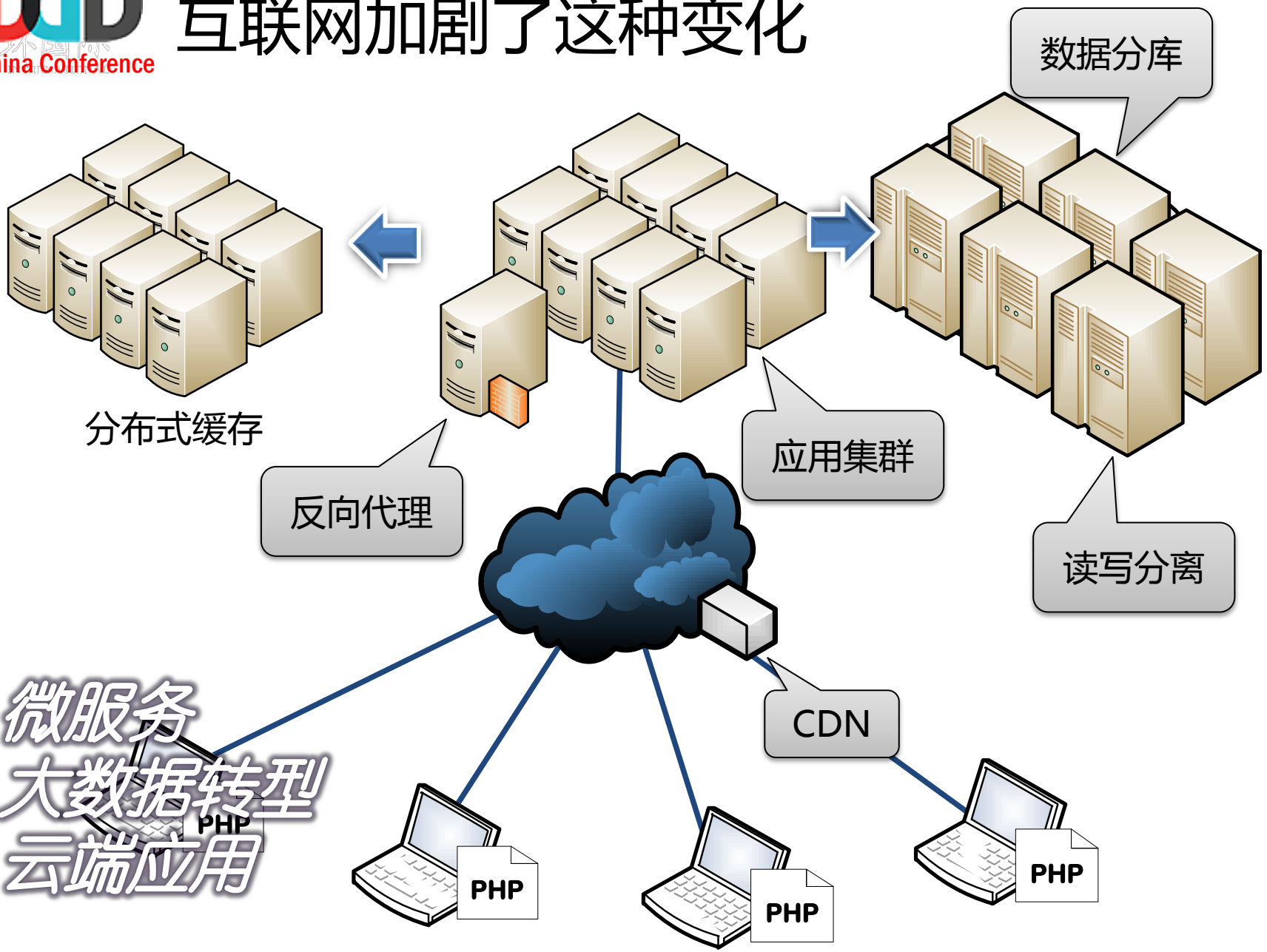
众筹

品折扣：
- 限时折扣
- 限量折扣
- 某类商品折扣

返券

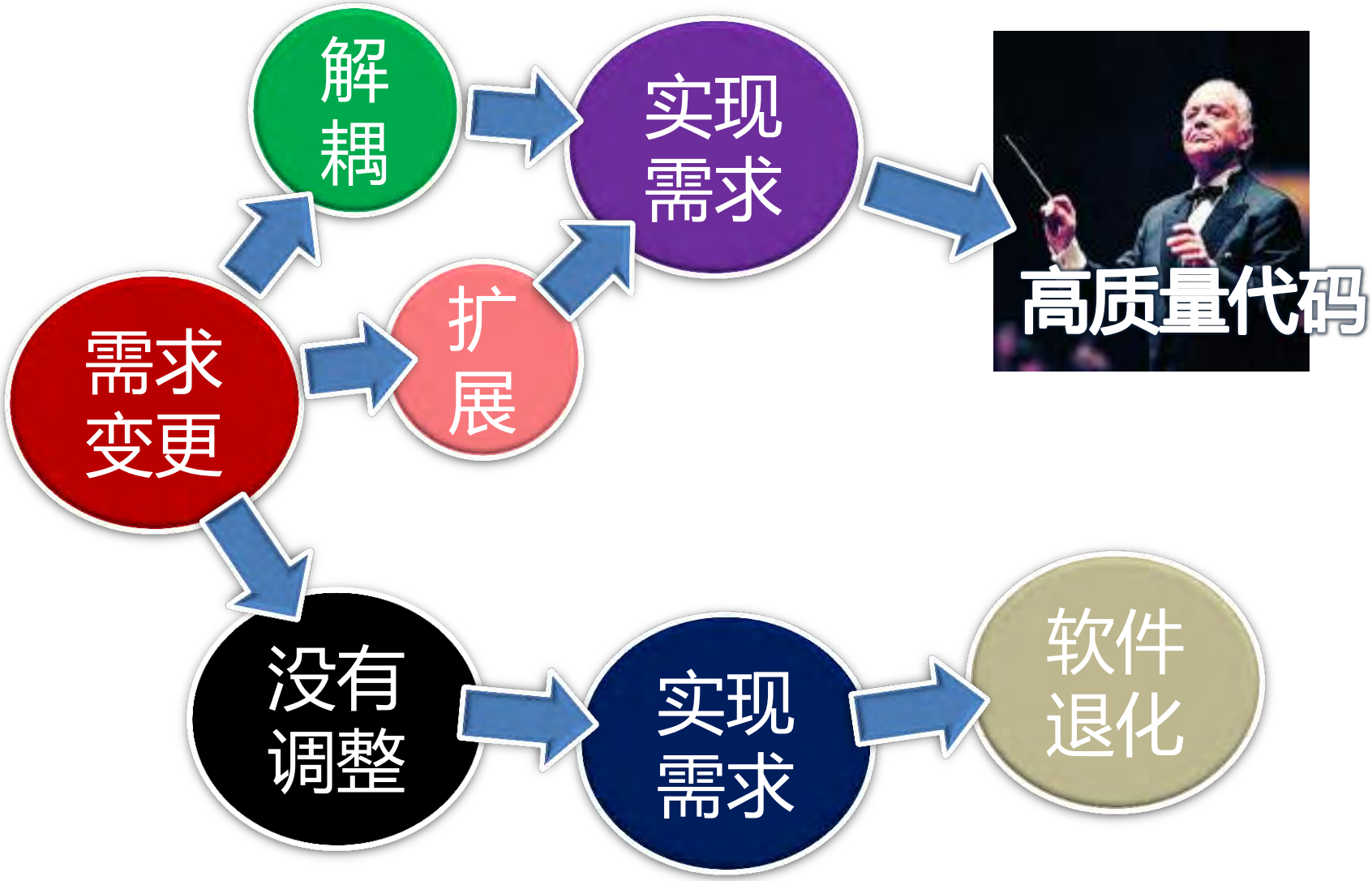
支付方式：
- 支付宝
- 微信
- 银行卡

# 互联网加剧了这种变化




微服务
大数据转型
云端应用

# 需求变更是软件退化的根源吗？



# 软件设计过程中面临的难题

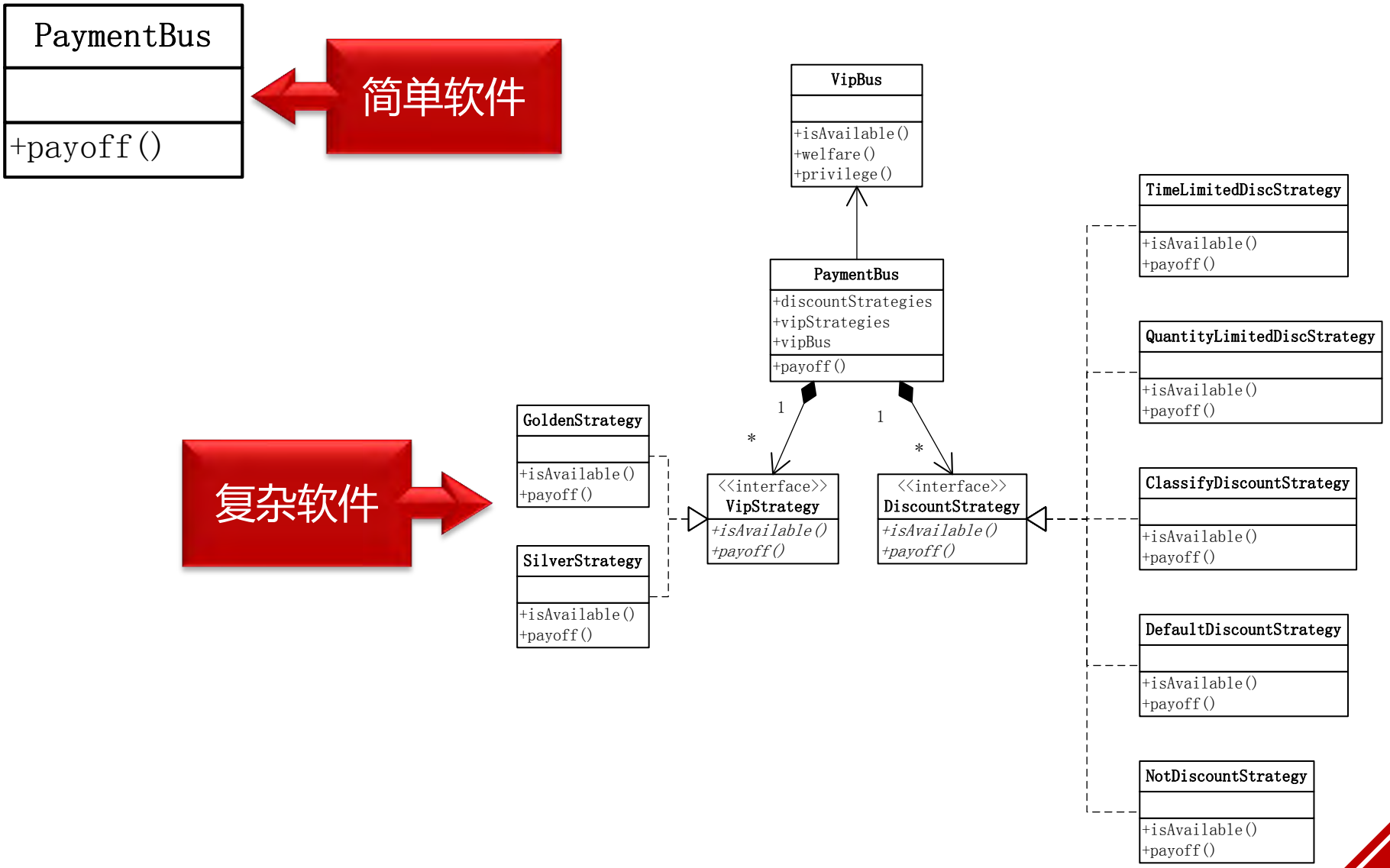
软件是对真实世界的模拟，但真实世界十分复杂

人在认识真实世界的时候总是有一个从简单到复杂的过程

因此，软件需求变更成为一种必然，并且总是由简单向复杂转变


但在软件由简单向复杂转变过程中，设计容易迷失方向

# 简单软件 vs. 复杂软件


PaymentBus
+payoff()
简单软件
复杂软件
VipBus
+isAvailable()
+welfare()
+privilege()
PaymentBus
+discountStrategies
+vipStrategies
+vipBus
+payoff()
1
*
1
*
GoldenStrategy
+isAvailable()
+payoff()
SilverStrategy
+isAvailable()
+payoff()
<<interface>>
VipStrategy
+isAvailable()
+payoff()
<<interface>>
DiscountStrategy
+isAvailable()
+payoff()
TimeLimitedDiscStrategy
+isAvailable()
+payoff()
QuantityLimitedDiscStrategy
+isAvailable()
+payoff()
ClassifyDiscountStrategy
+isAvailable()
+payoff()
DefaultDiscountStrategy
+isAvailable()
+payoff()
NotDiscountStrategy
+isAvailable()
+payoff()

# 难题的解决思路与方案

简单软件有简单软件的设计，复杂软件有复杂软件的设计

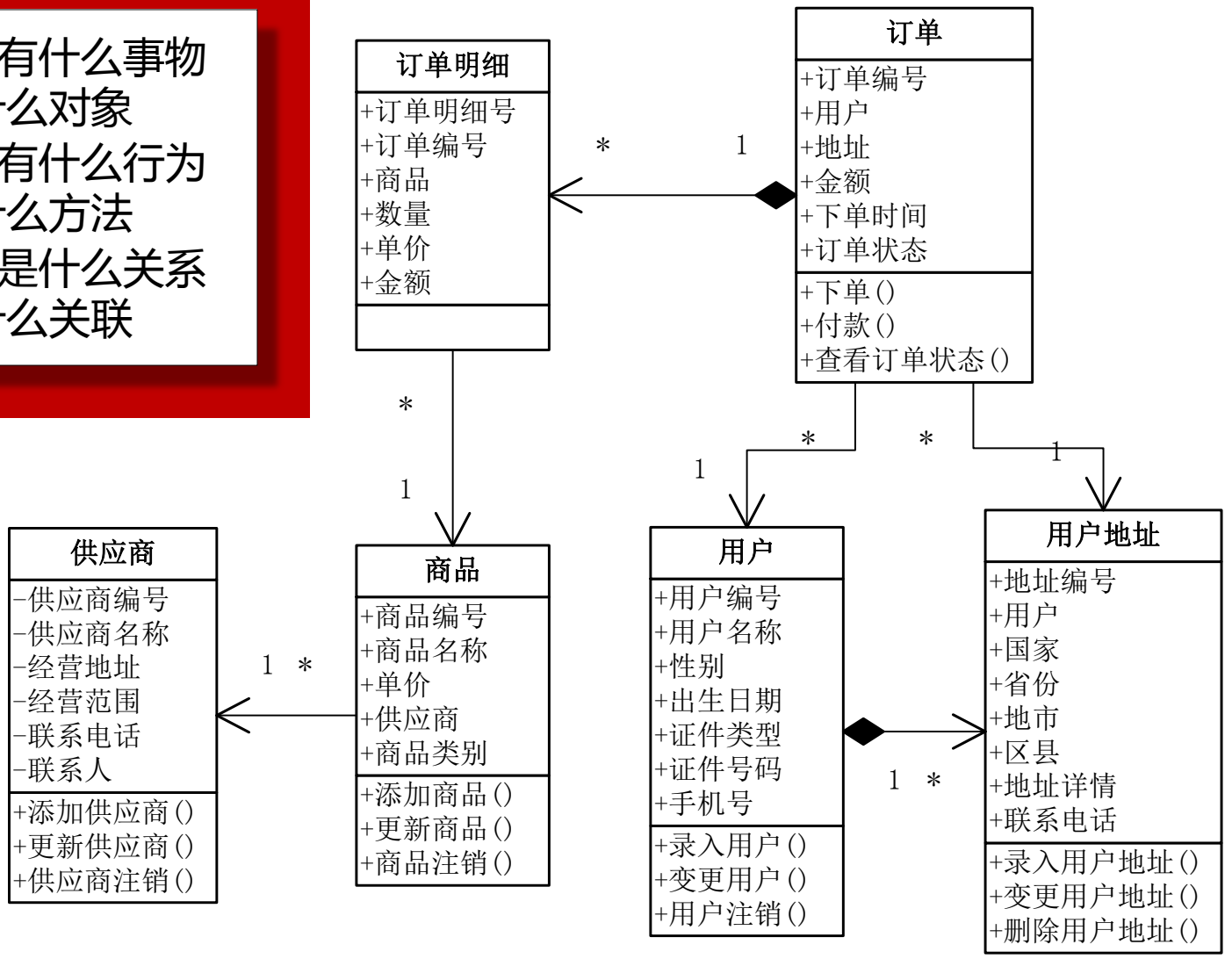
当软件由简单软件向复杂软件转变时，应当适时对程序结构进行调整

将软件设计与真实世界对应起来，建立领域模型，指导软件开发

每次变更的时候，重新回到领域模型进行变更，来指导软件变更

# 领域模型及领域建模思想

- 现实世界有什么事物
  →就有什么对象
- 现实世界有什么行为
  →就有什么方法
- 现实世界是什么关系
  →就有什么关联



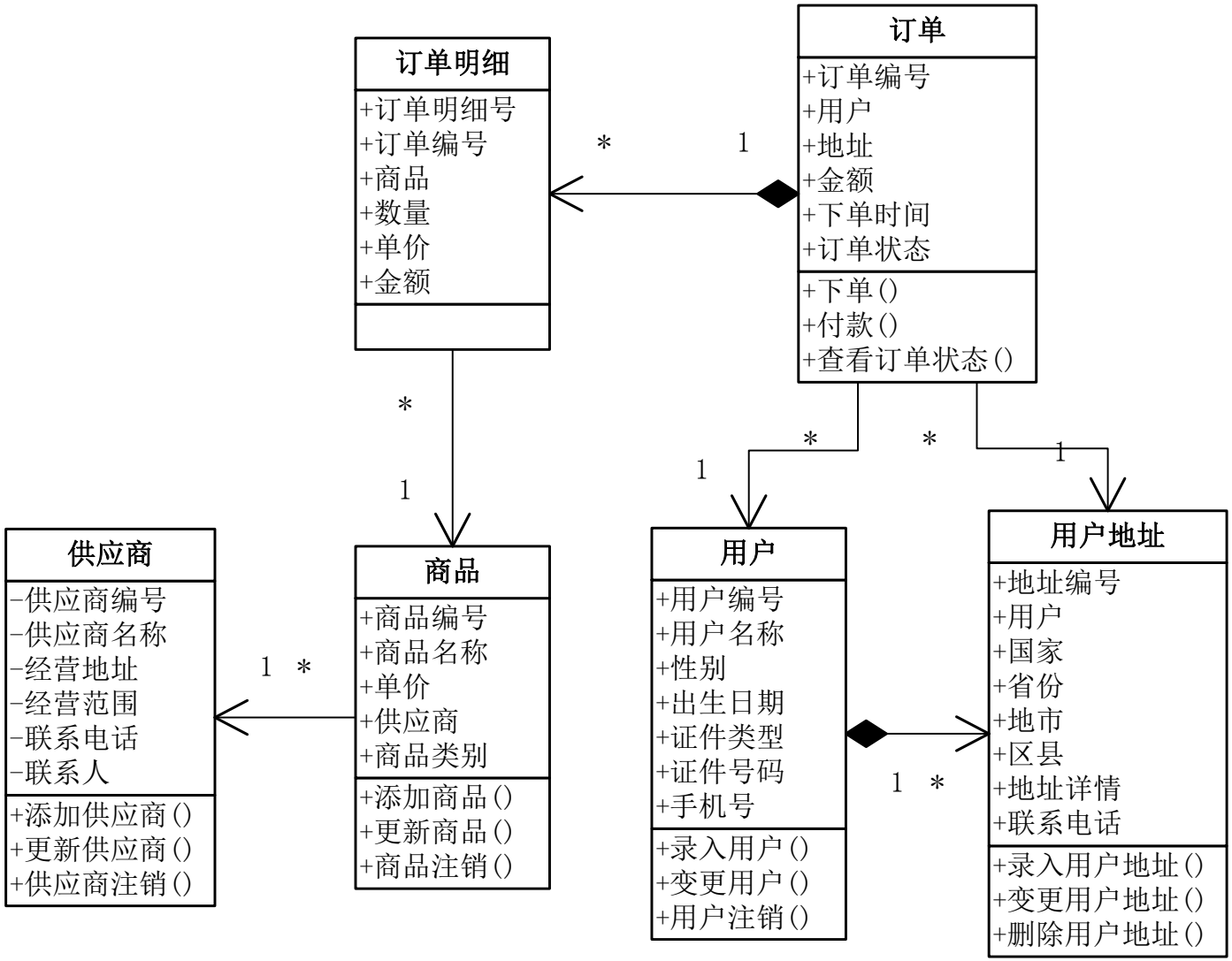
# 用慢动作的手法重新演练一次

- 需求变更应当怎样应对？
- 正确的软件设计过程是什么？
- 领域驱动是如何拥抱变化？

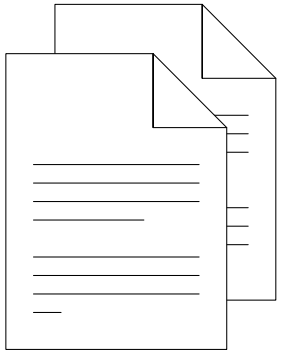
# 原始需求

- 用户付款功能
  - 用户下单以后，经过下单流程进入付款功能
  - 通过用户档案获得用户名称、地址等信息
  - 记录商品及其数量，并汇总付款金额
  - 保存订单
  - 通过远程调用支付接口，进入支付功能

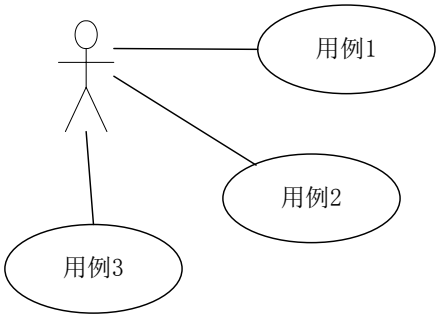
# 第一个版本的领域模型


订单明细
+订单明细号
+订单编号
+商品
+数量
+单价
+金额
订单
+订单编号
+用户
+地址
+金额
+下单时间
+订单状态
+下单()
+付款()
+查看订单状态()
供应商
-供应商编号
-供应商名称
-经营地址
-经营范围
-联系电话
-联系人
+添加供应商()
+更新供应商()
+供应商注销()
商品
+商品编号
+商品名称
+单价
+供应商
+商品类别
+添加商品()
+更新商品()
+商品注销()
用户
+用户编号
+用户名称
+性别
+出生日期
+证件类型
+证件号码
+手机号
+录入用户()
+变更用户()
+用户注销()
用户地址
+地址编号
+用户
+国家
+省份
+地市
+区县
+地址详情
+联系电话
+录入用户地址()
+变更用户地址()
+删除用户地址()
*
1
*
1
*
1
*
1
1
*
1
*

# 领域驱动的软件设计过程



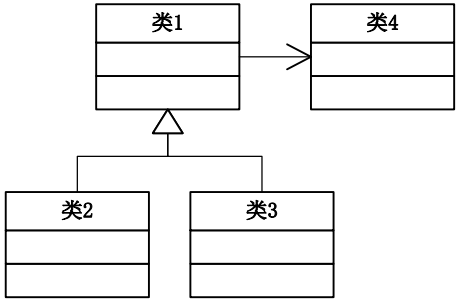
需求文档



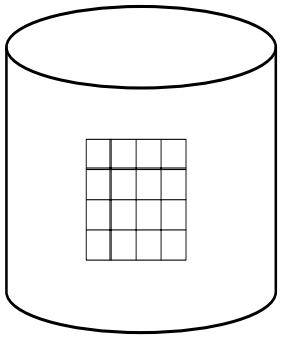

用例设计

```
import org.apache.spark.SparkConf

object Csv {
  def main(args: Array[String]) {
    val conf = (new SparkConf).setMaster("local").setAppName("Csv")
    val sc = new SparkContext(conf)
    val input = "file:///d:\\input\\aa.csv"//args(0)
    val lines = sc.wholeTextFiles(input)
    val result = lines.flatMap { case(_, x) =>
      val reader = new CSVReader(new StringReader(x))
      val rowarray = Array[Array[String]]()
      def readnext(reader: CSVReader) {
        val row = reader.readNext()
        rowarray+:row
        if(row.isEmpty) {}
        else readnext(reader)
      }
      rowarray
    }
    result.foreach { row => row.foreach { x => print(x) } }
  }
}
```

程序设计

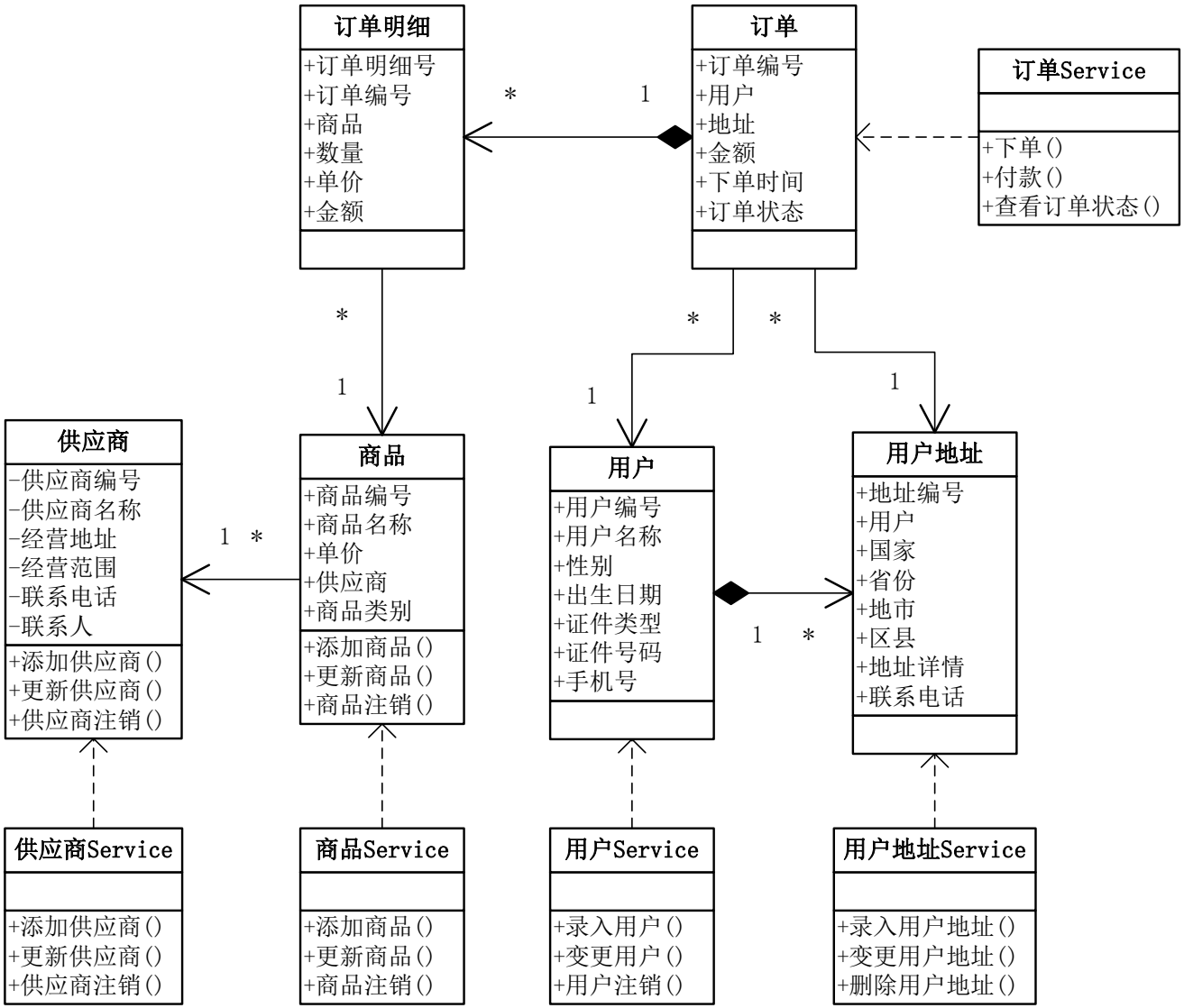



领域模型



数据库设计

# 第一个版本的程序设计


订单明细
+订单明细号
+订单编号
+商品
+数量
+单价
+金额
订单
+订单编号
+用户
+地址
+金额
+下单时间
+订单状态
订单Service
+下单()
+付款()
+查看订单状态()
供应商
-供应商编号
-供应商名称
-经营地址
-经营范围
-联系电话
-联系人
+添加供应商()
+更新供应商()
+供应商注销()
商品
+商品编号
+商品名称
+单价
+供应商
+商品类别
+添加商品()
+更新商品()
+商品注销()
用户
+用户编号
+用户名称
+性别
+出生日期
+证件类型
+证件号码
+手机号
用户地址
+地址编号
+用户
+国家
+省份
+地市
+区县
+地址详情
+联系电话
供应商Service
+添加供应商()
+更新供应商()
+供应商注销()
商品Service
+添加商品()
+更新商品()
+商品注销()
用户Service
+录入用户()
+变更用户()
+用户注销()
用户地址Service
+录入用户地址()
+变更用户地址()
+删除用户地址()

# 付款功能第一个版本的实现

```
public boolean payoff(Order order, String addressId) {
    //收集客户信息
    String customerId = SessionService.getUserId();
    Customer customer = dao.getCustomer(customerId);
    order.setCustomer(customer);
    Address address = dao.getAddress(addressId);
    order.setAddress(address);

    //计算付款
    double amount = 0;
    for(OrderDetail detail : order.getDetails()) {
        amount += detail.getQuantity()*detail.getPrice();
    }
    order.setAmount(amount);
    Serializable orderId = dao.save(order);

    //跳转支付页面
    WebserviceFactory factory = new WebserviceFactory();
    PaymentWebservice payment =
        (PaymentWebservice)factory.getWebservice("aliPayment");
    return payment.payoff(orderId, customer, amount);
}
```

# 第一次需求变更

- 增加商品折扣功能
  - 限时折扣
  - 限量折扣
  - 对某类商品进行折扣
  - 一般地对某个商品进行折扣
  - 不折扣

# 付款功能第二个版本的实现

```
public boolean payoff(Order order, String addressId) {
      //收集客户信息
      String customerId = SessionService.getUserId();
      Customer customer = dao.getCustomer(customerId);
      order.setCustomer(customer);
      Address address = dao.getAddress(addressId);
      order.setAddress(address);

      //计算付款
      double amount = 0;
      for(OrderDetail detail : order.getDetails()) {
            if(hasTimeLimited(detail)) {
              //如果有限时抢购，此处省略100字
            } else if(hasQuantityLimited(detail) {
              //如果有限量抢购，此处省略100字
            } else if(hasClassifyDiscount(detail) {
              //如果有某类商品的折扣，此处省略100字
            } else if(hasDiscount(detail) {
              //如果一般商品折扣，此处省略100字
            } else {
              amount += detail.getQuantity()*detail.getPrice();
            }
      }
      order.setAmount(amount);
      Serializable orderId = dao.save(order);

      //跳转支付页面
      WebserviceFactory factory = new WebserviceFactory();
      PaymentWebservice payment =
            (PaymentWebservice)factory.getWebservice("aliPayment");
      return payment.payoff(orderId, customer, amount);
}
```

商品折扣：

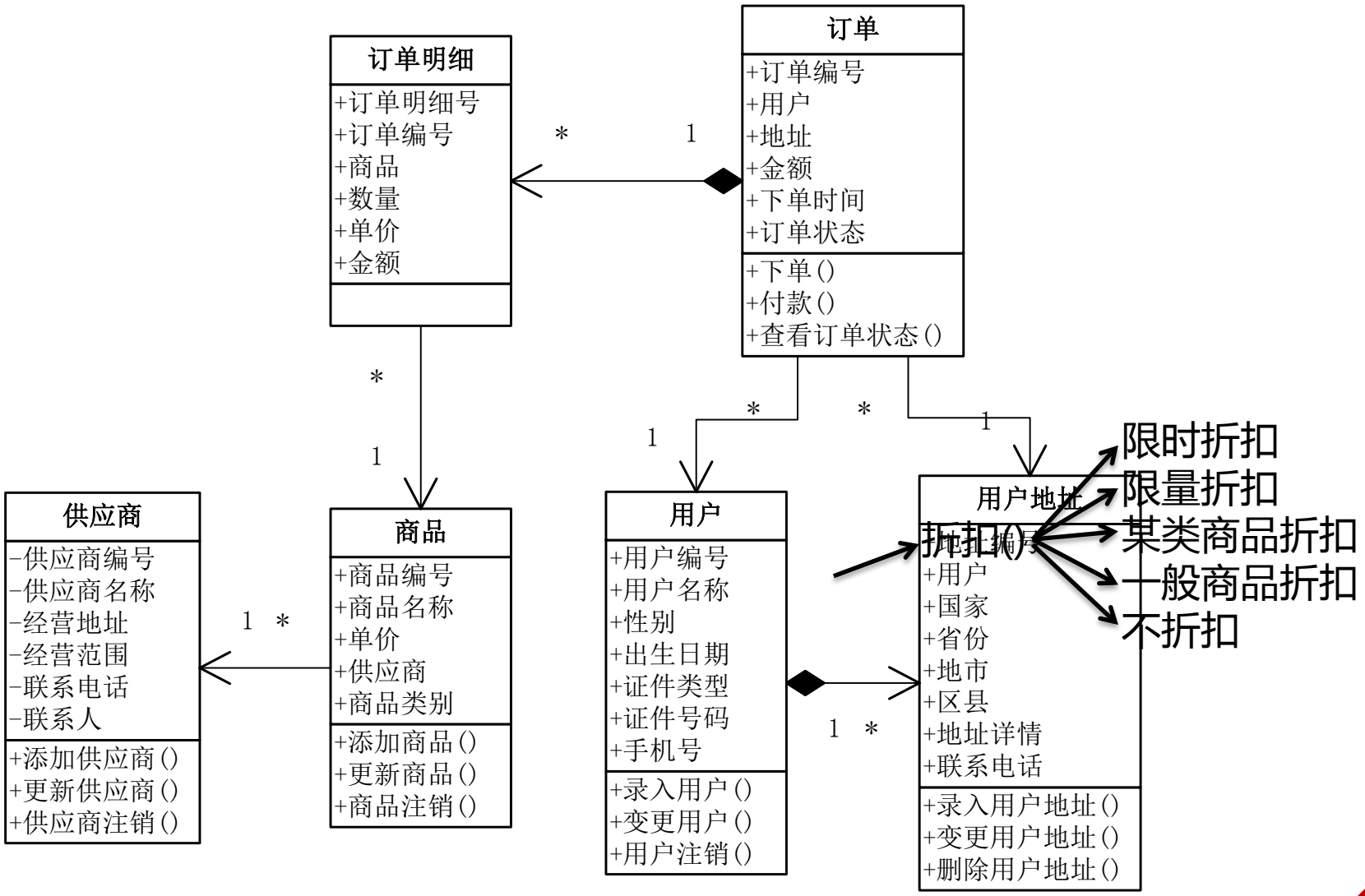
- 限时折扣
- 限量折扣
- 某类商品折扣
- 一般商品折扣
- 不折扣

# 两顶帽子的设计方式

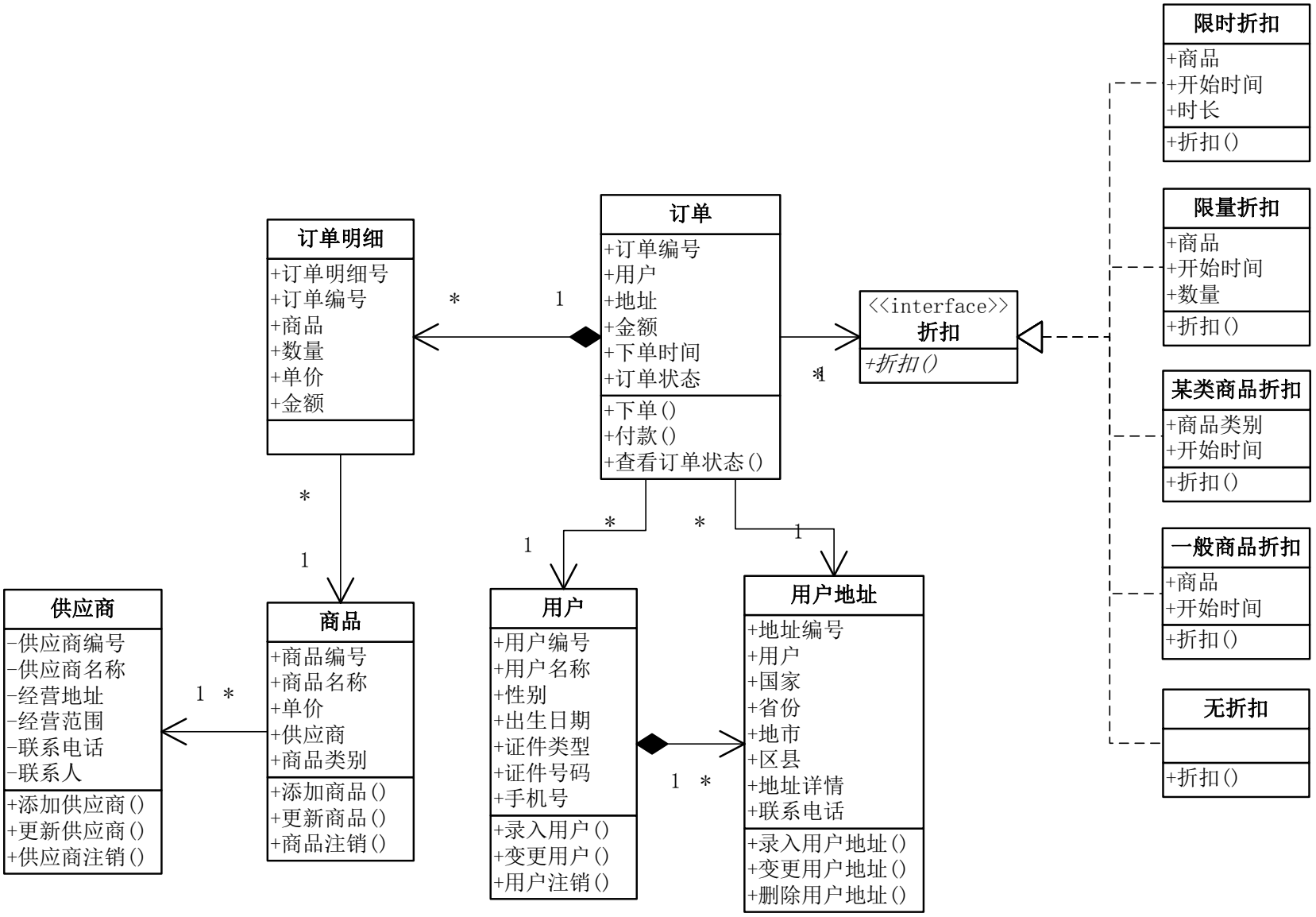

1. 在不添加新功能的前提下，
   重构代码，以适应新功能

2. 实现新功能

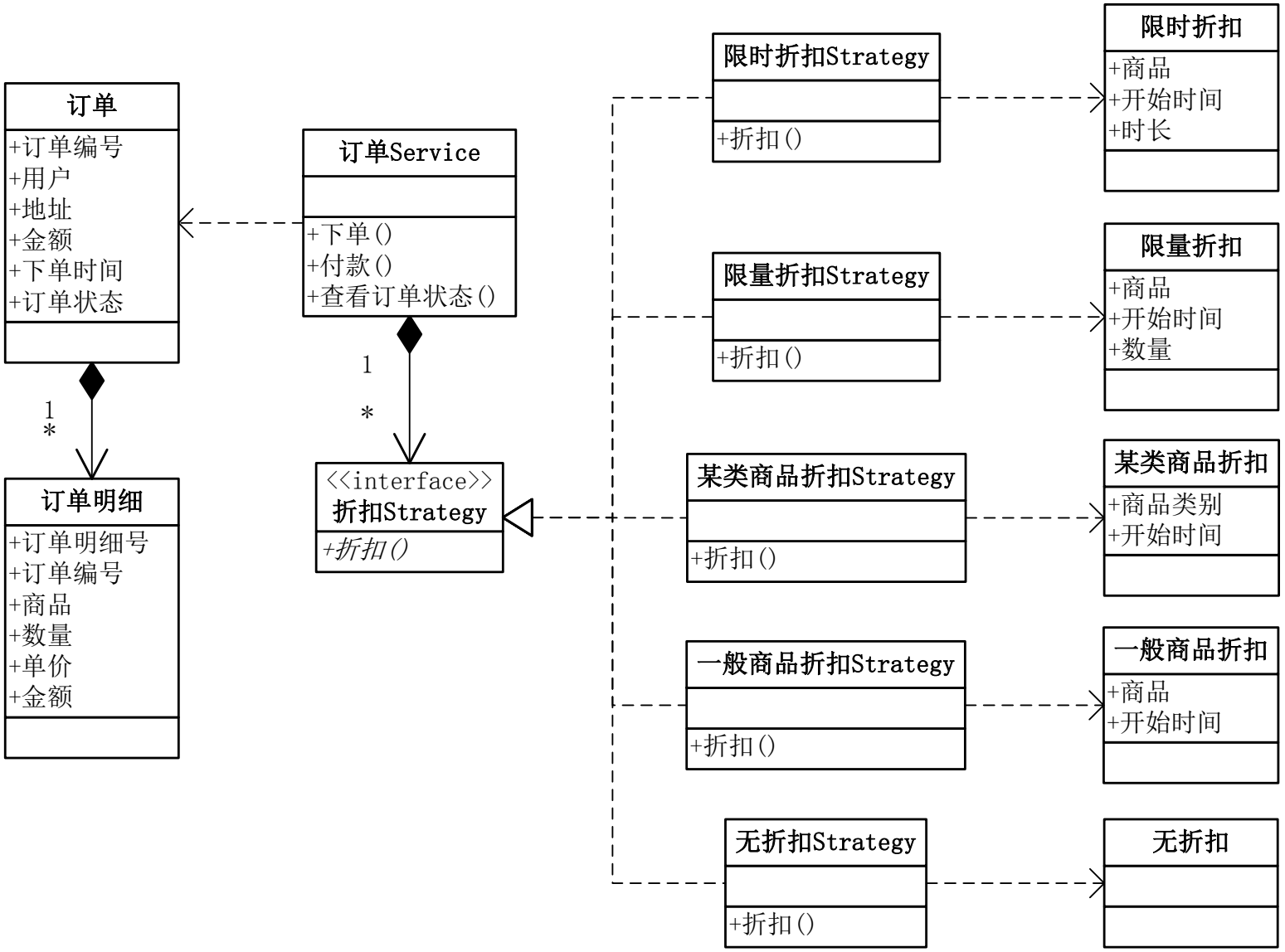
# 在第一个版本的领域模型上分析



# 第二个版本的领域模型


订单明细
+订单明细号
+订单编号
+商品
+数量
+单价
+金额
订单
+订单编号
+用户
+地址
+金额
+下单时间
+订单状态
+下单()
+付款()
+查看订单状态()
<<interface>>
折扣
+折扣()
限时折扣
+商品
+开始时间
+时长
+折扣()
限量折扣
+商品
+开始时间
+数量
+折扣()
某类商品折扣
+商品类别
+开始时间
+折扣()
一般商品折扣
+商品
+开始时间
+折扣()
无折扣
+折扣()
供应商
-供应商编号
-供应商名称
-经营地址
-经营范围
-联系电话
-联系人
+添加供应商()
+更新供应商()
+供应商注销()
商品
+商品编号
+商品名称
+单价
+供应商
+商品类别
+添加商品()
+更新商品()
+商品注销()
用户
+用户编号
+用户名称
+性别
+出生日期
+证件类型
+证件号码
+手机号
+录入用户()
+变更用户()
+用户注销()
用户地址
+地址编号
+用户
+国家
+省份
+地市
+区县
+地址详情
+联系电话
+录入用户地址()
+变更用户地址()
+删除用户地址()
*
1
1
*
*
1
1
*
*
1
1
*
*
1

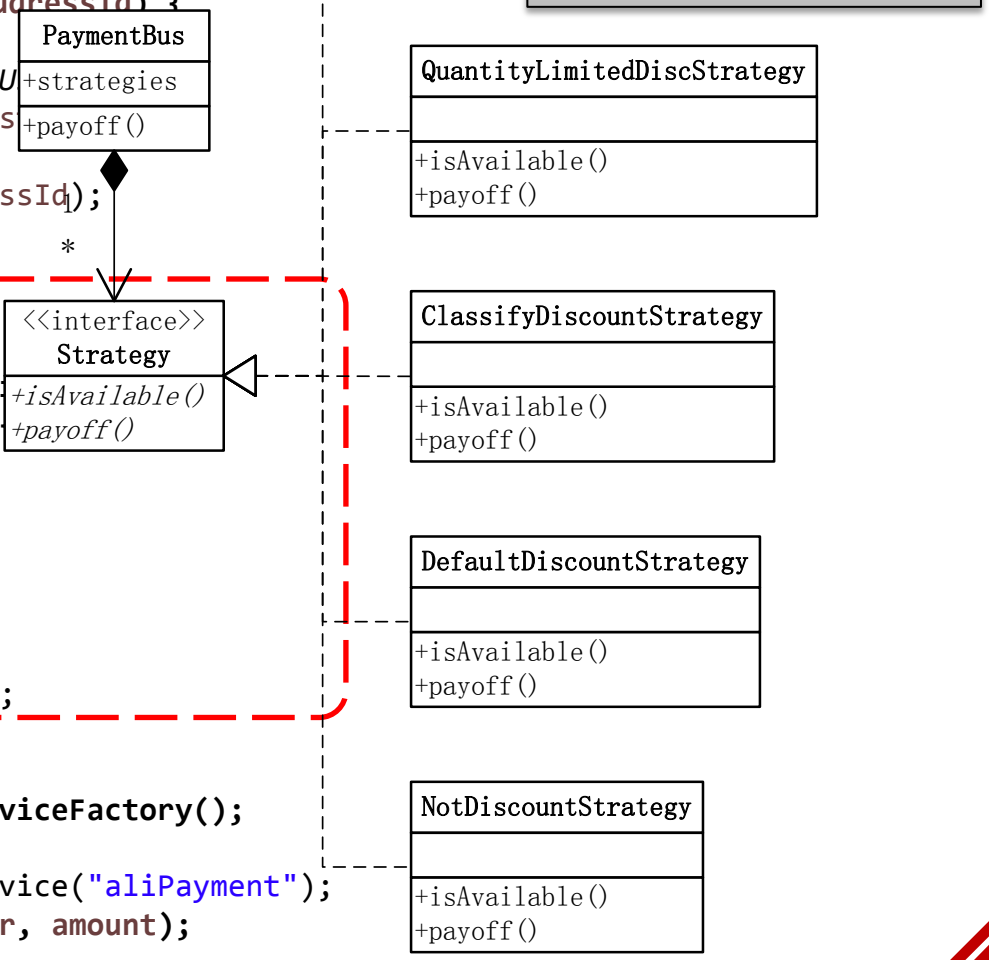
# 第二个版本的设计实现


订单
+订单编号
+用户
+地址
+金额
+下单时间
+订单状态
订单Service
+下单()
+付款()
+查看订单状态()
1
*
1
*
订单明细
+订单明细号
+订单编号
+商品
+数量
+单价
+金额
<<interface>>
折扣Strategy
+折扣()
限时折扣Strategy
+折扣()
限时折扣
+商品
+开始时间
+时长
限量折扣Strategy
+折扣()
限量折扣
+商品
+开始时间
+数量
某类商品折扣Strategy
+折扣()
某类商品折扣
+商品类别
+开始时间
一般商品折扣Strategy
+折扣()
一般商品折扣
+商品
+开始时间
无折扣Strategy
+折扣()
无折扣

# 付款功能第二个版本的实现

商品折扣：
- 限时折扣
- 限量折扣
- 分类折扣
- 默认折扣

```
private List<Strategy> strategies = null;
public List<Strategy> getStrategies() {
     return strategies;
}
public void setStrategies(List<Strategy> strategies) {
     this.strategies = strategies;
}
public boolean payoff(Order order, String addressId) {
     //收集客户信息
     String customerId = SessionService.getU
     Customer customer = dao.getCustomer(cus
     order.setCustomer(customer);
     Address address = dao.getAddress(addressId);
     order.setAddress(address);

     //计算付款
     double amount = 0;
     for(OrderDetail detail : order.getDeta
       for(Strategy strategy : strategies)
         if(strategy.isAvailable(detail))
           strategy.payoff(detail);
       }
       amount += detail.getAmount();
     }
     order.setAmount(amount);
     Serializable orderId = dao.save(order);

     //跳转支付页面
     WebserviceFactory factory = new WebserviceFactory();
     PaymentWebservice payment =
       (PaymentWebservice)factory.getWebservice("aliPayment");
     return payment.payoff(orderId, customer, amount);
}
```



# 第二次需求变更

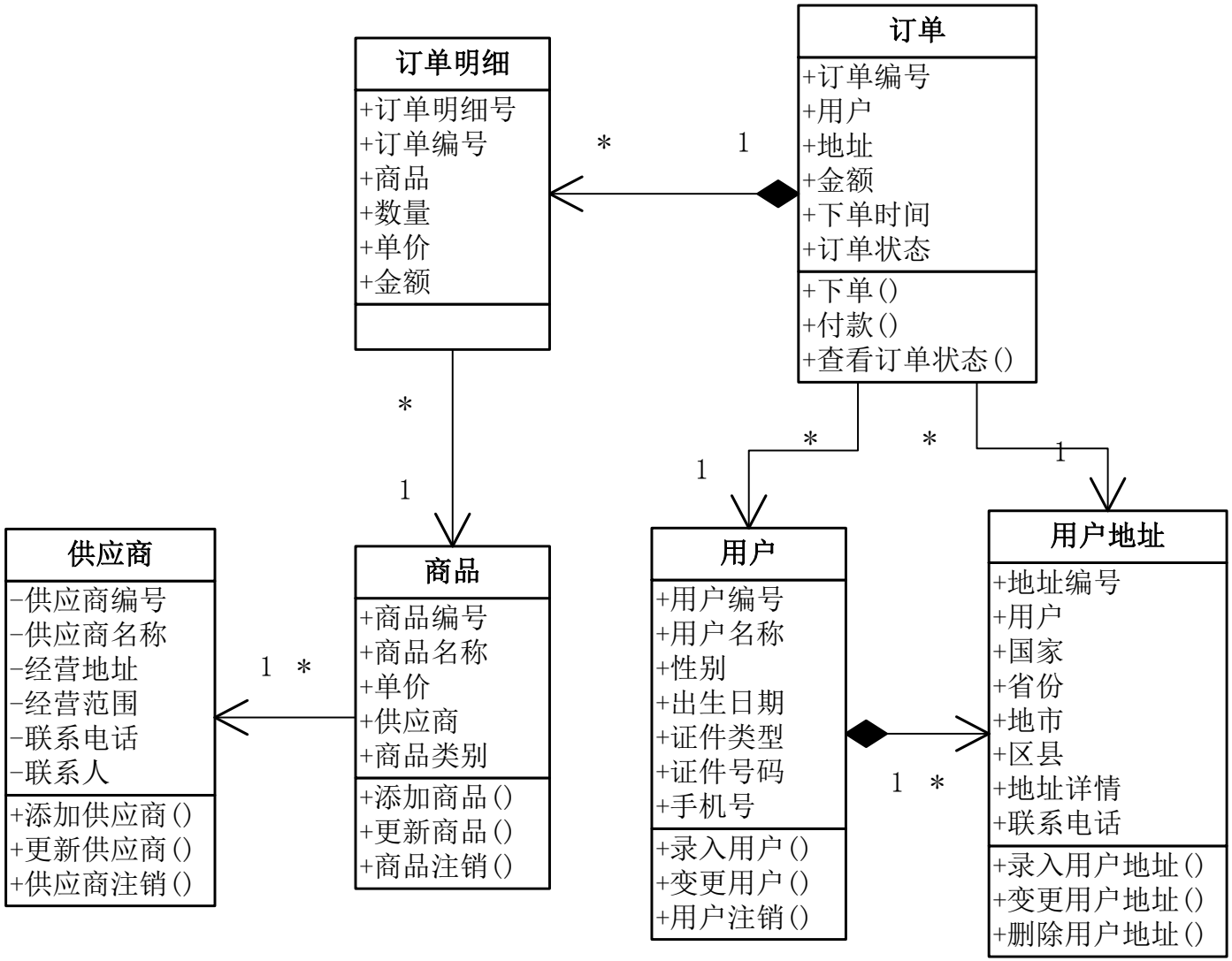
- 增加VIP会员功能
  - 对VIP金卡和银卡会员进行不同的折扣
  - 在支付时为VIP会员发放福利
  - VIP会员可以享受某些特权

# 付款功能第三个版本的实现

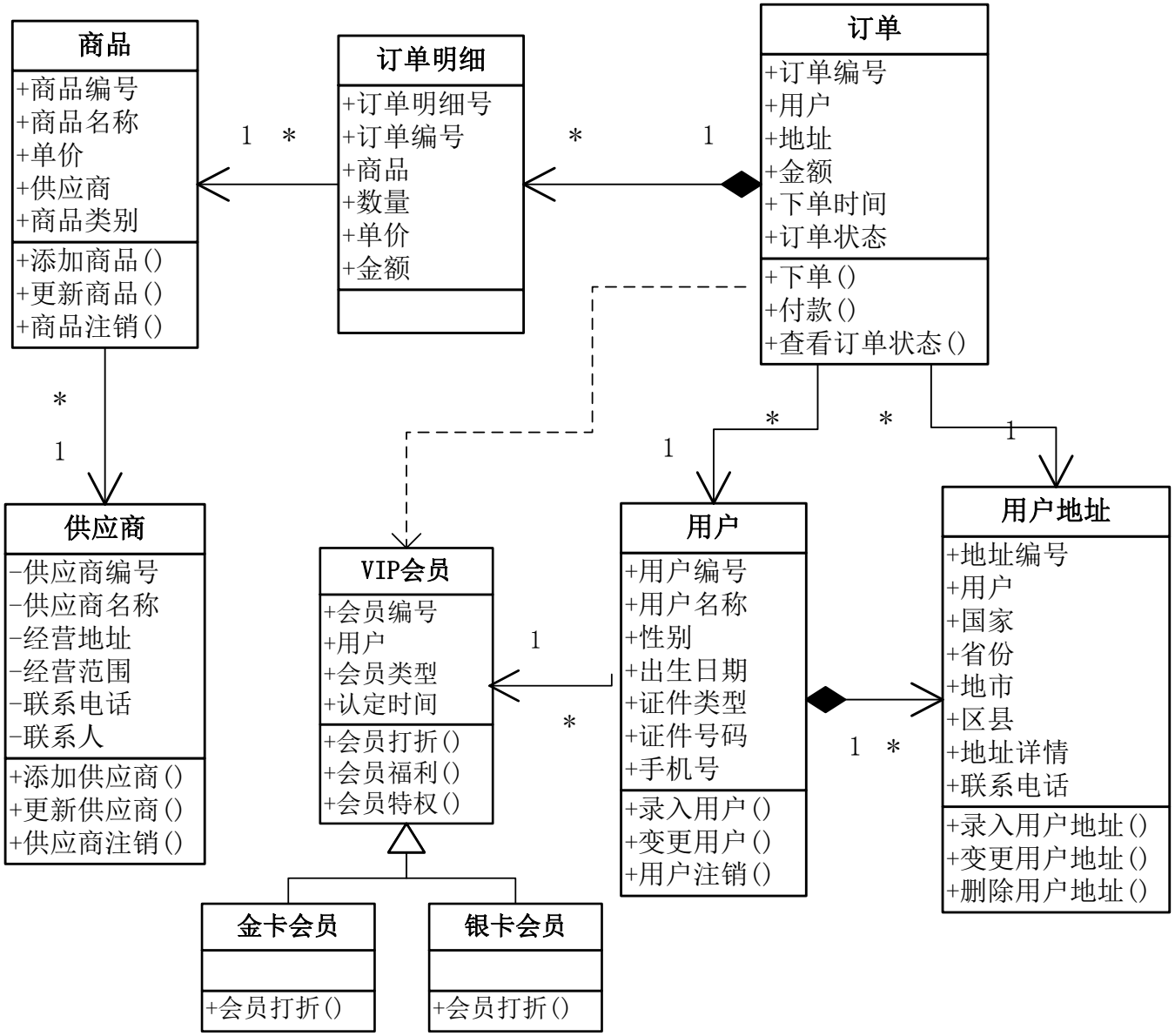
VIP会员：
- 金银卡会员
- 会员福利
- 会员特权

```
public boolean payoff(Order order, String addressId) {
    //收集客户信息
    String customerId = SessionService.getUserId();
    Customer customer = dao.getCustomer(customerId);
    order.setCustomer(customer);
    Address address = dao.getAddress(addressId);
    order.setAddress(address);

    //会员福利
    if(vipBus.isAvailable(customer)) {
      //此处省略200字
    }

    //计算付款
    double amount = 0;
    for(OrderDetail detail : order.getDetails()) {
      //获取用户信息判断是否是VIP会员
      if(isVipCustomer) {
          //此处省略200字
      }
      for(DiscountStrategy strategy : strategies) {
        if(strategy.isAvailable(detail))
          strategy.payoff(detail);
      }
      amount += detail.getAmount();
    }
    order.setAmount(amount);
    Serializable orderId = dao.save(order);

    //跳转支付页面
    WebserviceFactory factory = new WebserviceFactory();
    PaymentWebservice payment =
      (PaymentWebservice)factory.getWebservice("aliPayment");
```

# 回到上一个版本的领域模型



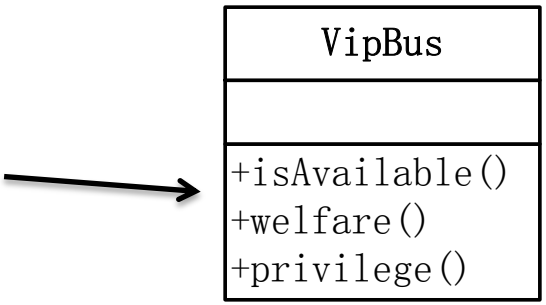
# 第三个版本的领域模型

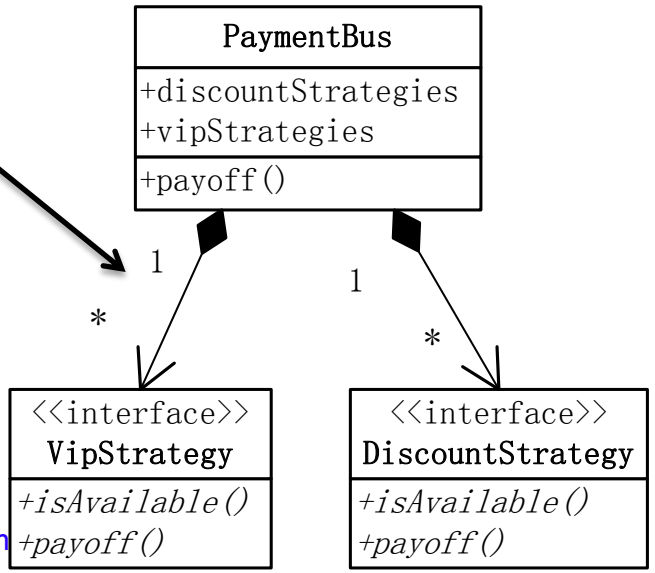

商品
+商品编号
+商品名称
+单价
+供应商
+商品类别
+添加商品()
+更新商品()
+商品注销()
订单明细
+订单明细号
+订单编号
+商品
+数量
+单价
+金额
订单
+订单编号
+用户
+地址
+金额
+下单时间
+订单状态
+下单()
+付款()
+查看订单状态()
供应商
-供应商编号
-供应商名称
-经营地址
-经营范围
-联系电话
-联系人
+添加供应商()
+更新供应商()
+供应商注销()
VIP会员
+会员编号
+用户
+会员类型
+认定时间
+会员打折()
+会员福利()
+会员特权()
用户
+用户编号
+用户名称
+性别
+出生日期
+证件类型
+证件号码
+手机号
+录入用户()
+变更用户()
+用户注销()
用户地址
+地址编号
+用户
+国家
+省份
+地市
+区县
+地址详情
+联系电话
+录入用户地址()
+变更用户地址()
+删除用户地址()
金卡会员
+会员打折()
银卡会员
+会员打折()
1
*

# 付款功能第三个版本的实现

VIP会员：
- 金银卡会员
- 会员福利
- 会员特权

```
public boolean payoff(Order order, String addressId) {
    //收集客户信息
    String customerId = SessionService.getUserId();
    Customer customer = dao.getCustomer(customerId);
    order.setCustomer(customer);
    Address address = dao.getAddress(addressId);
    order.setAddress(address);

    //会员福利
    if(vipBus.isAvailable(customer))
      vipBus.welfare(customer);

    //计算付款
    double amount = 0;
    for(OrderDetail detail : order.getDetails()) {
      for(VipStrategy strategy : strategies) {
        if(strategy.isAvailable(customer))
          strategy.payoff(customer);
      }
      for(DiscountStrategy strategy : strategies) {
        if(strategy.isAvailable(detail))
          strategy.payoff(detail);
      }
      amount += detail.getAmount();
    }
    order.setAmount(amount);
    Serializable orderId = dao.save(order);

    //跳转支付页面
    WebserviceFactory factory = new WebserviceFactory();
    PaymentWebservice payment =
      (PaymentWebservice)factory.getWebservice("aliPaymen
    return payment.payoff(orderId, customer, amount);
}
```



# 重构是软件发展的必然

- 结论
  - 软件总是由简单软件向复杂软件转变
  - 简单软件有简单软件的设计
  - 复杂软件有复杂软件的设计
  - 当软件开始由简单软件转变为复杂软件时我们就需要

# 两顶帽子的意义

- 过去，我们是这样设计的：
  - 总是担心未来的变更
  - 总是为未来的设计埋单
  - 总是在过度设计
- 今天，活在今天的格子里做今天的事儿
  - 只为当前的需求进行设计
  - 当未来变更时，先重构以适应新需求
  - 然后添加新的需求

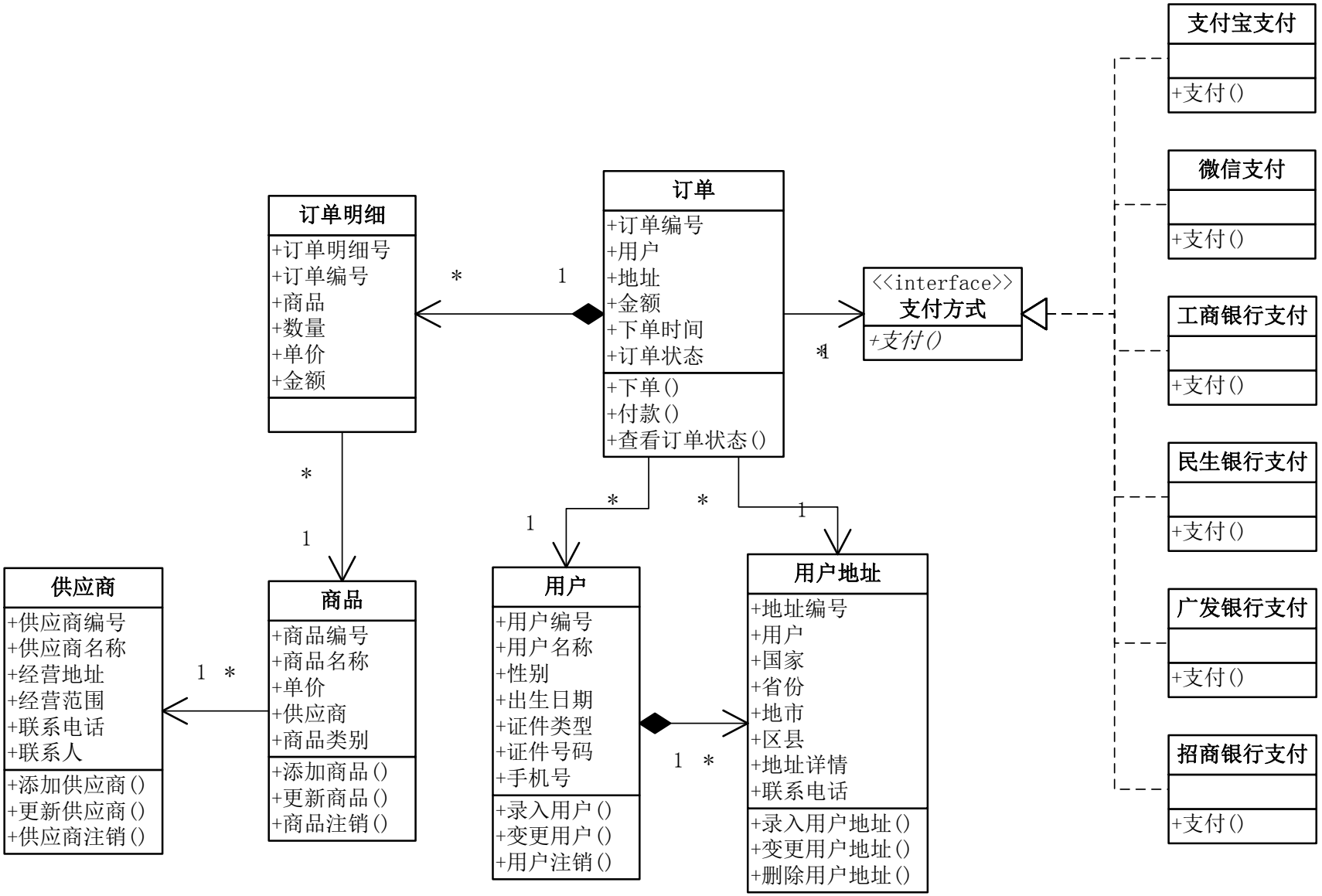
# 领域驱动设计的作用

- 保持软件质量不退化
  - 关键时间点：每次变更时的正确设计
  - 软件变更设计的方法：两顶帽子
- 关键的问题是如何正确设计
  ——领域驱动设计
  - 软件变更设计时先回到领域模型
  - 在领域模型中，结合真实世界，进行相应变更
  - 用领域模型的变更映射成软件设计的变更

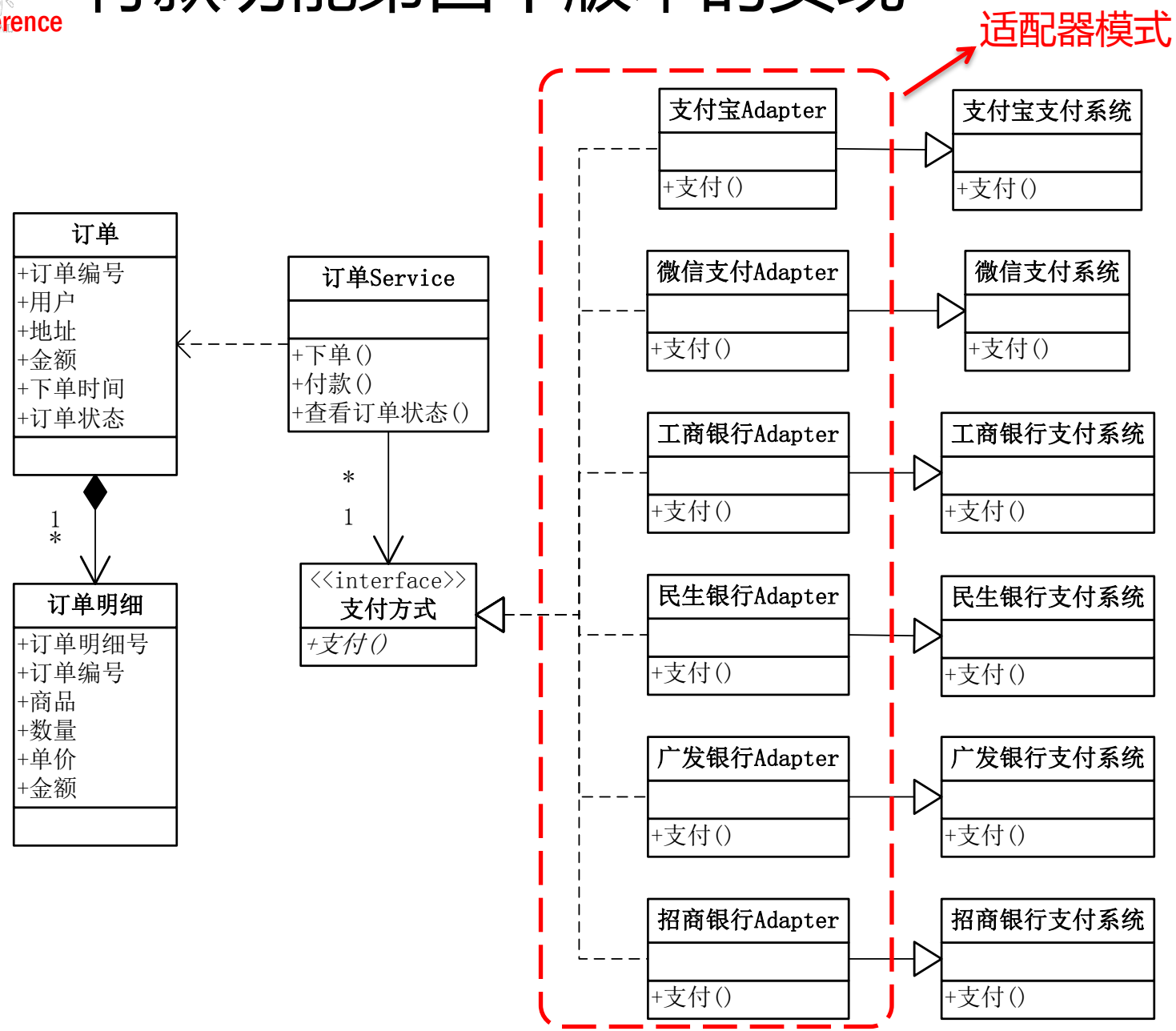
# 第三次需求变更

- 增加更多的支付方式
  - 支付宝支付
  - 微信支付
  - 工商银行支付
  - 民生银行支付
  - 广发银行支付
  - 招商银行支付

# 第四个版本的领域模型


订单明细
+订单明细号
+订单编号
+商品
+数量
+单价
+金额
订单
+订单编号
+用户
+地址
+金额
+下单时间
+订单状态
+下单()
+付款()
+查看订单状态()
<<interface>>
支付方式
+支付()
支付宝支付
+支付()
微信支付
+支付()
工商银行支付
+支付()
民生银行支付
+支付()
广发银行支付
+支付()
招商银行支付
+支付()
供应商
+供应商编号
+供应商名称
+经营地址
+经营范围
+联系电话
+联系人
+添加供应商()
+更新供应商()
+供应商注销()
商品
+商品编号
+商品名称
+单价
+供应商
+商品类别
+添加商品()
+更新商品()
+商品注销()
用户
+用户编号
+用户名称
+性别
+出生日期
+证件类型
+证件号码
+手机号
+录入用户()
+变更用户()
+用户注销()
用户地址
+地址编号
+用户
+国家
+省份
+地市
+区县
+地址详情
+联系电话
+录入用户地址()
+变更用户地址()
+删除用户地址()
*
1
1
*
*
1
*
1
1
*
1
*
1
*

# 付款功能第四个版本的实现



# 第N次需求变更

- 增加更多的功能
  - 秒杀 → 异步化操作
  - 预订 → 分布式缓存
  - 闪购 → 读写分离
  - 众筹 → 微服务架构
  - → 大数据转型

China Conference

# 软件架构如何支持领域驱动和架构演进

如何支持领域的快速变更

如何支持架构的快速演化

设计案例演示