

# 面向对象分析与设计

## Object Oriented Analysis and Design

### ——详细设计

#### Detail Design

邱明 博士

厦门大学信息学院

mingqiu@xmu.edu.cn

2023年秋季学期

# 1. 对象设计

**Object Design**



# 1.1 软件设计的目标

## Software Design Goals

- 为什么要做软件设计？
  - 是解决软件复杂性的必要手段
    - 将软件分为不同的部分，分而治之。
    - 是团队分工协作的前提
  - 是降低开发成本，提高软件质量的必要手段
    - 提高软件的重用性
    - 增强软件的扩展性

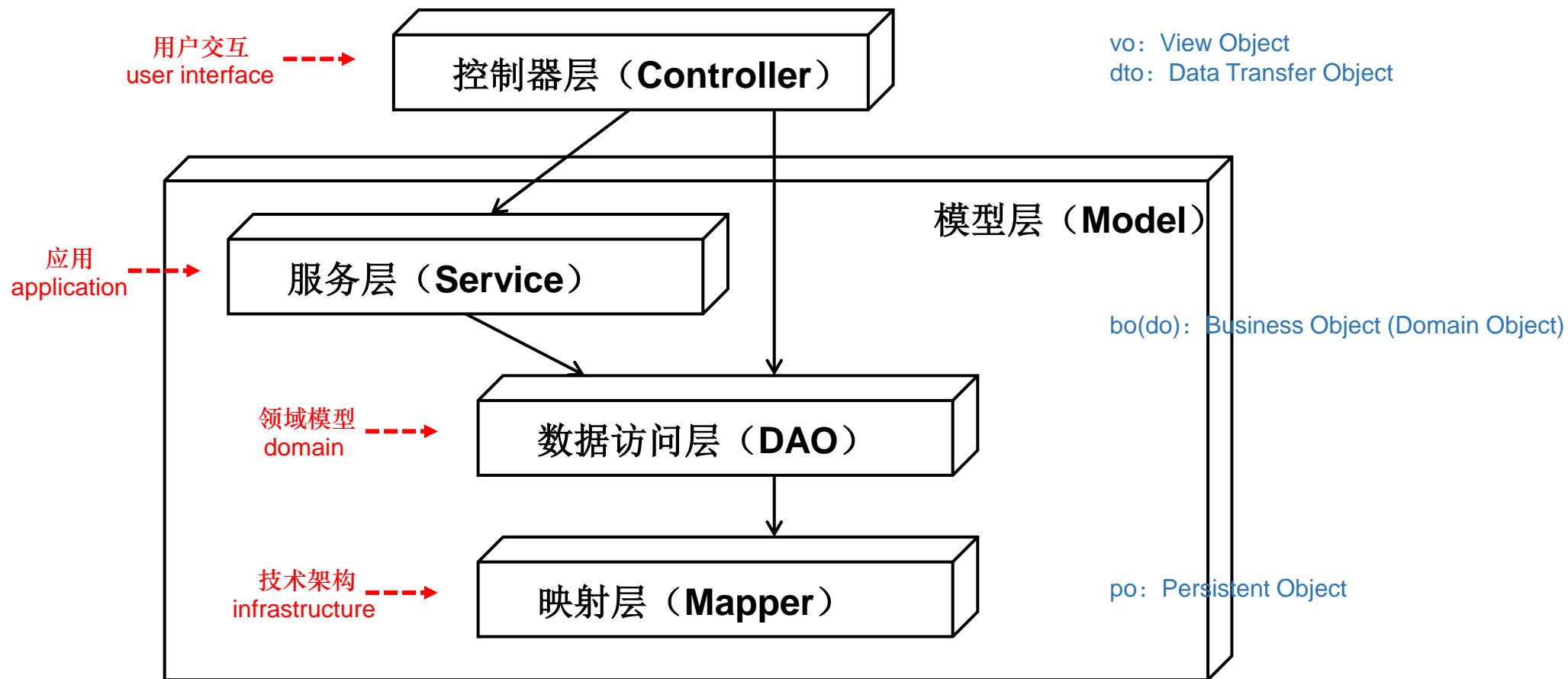
我们虽然无法找到一个绝对理想的圆，但这并不妨碍我们理解什么是圆。

—— 源自罗翔的《圆圈正义》



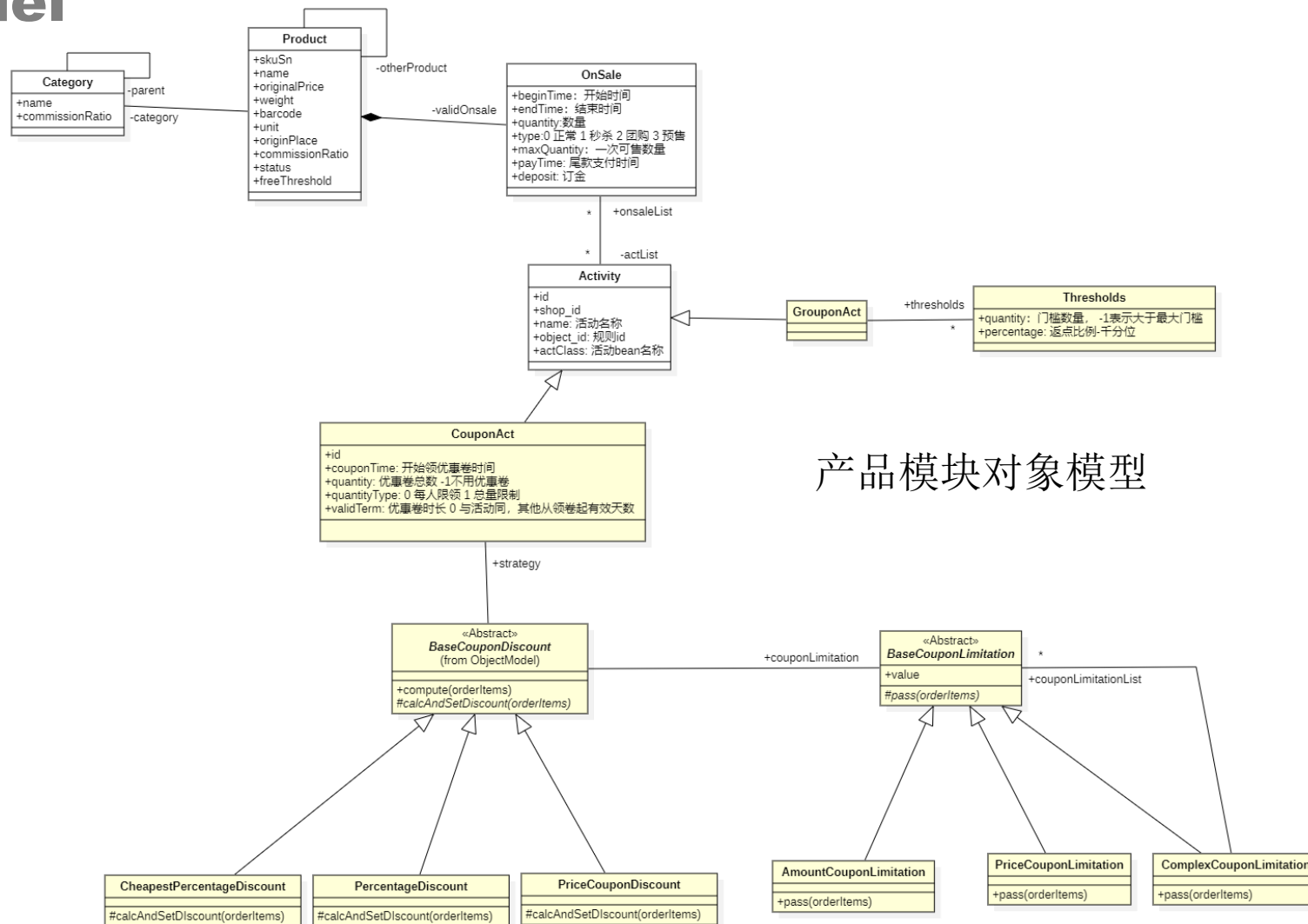
# 1.2 对象模型

## Object Model



# 1.2 对象模型

## Object Model

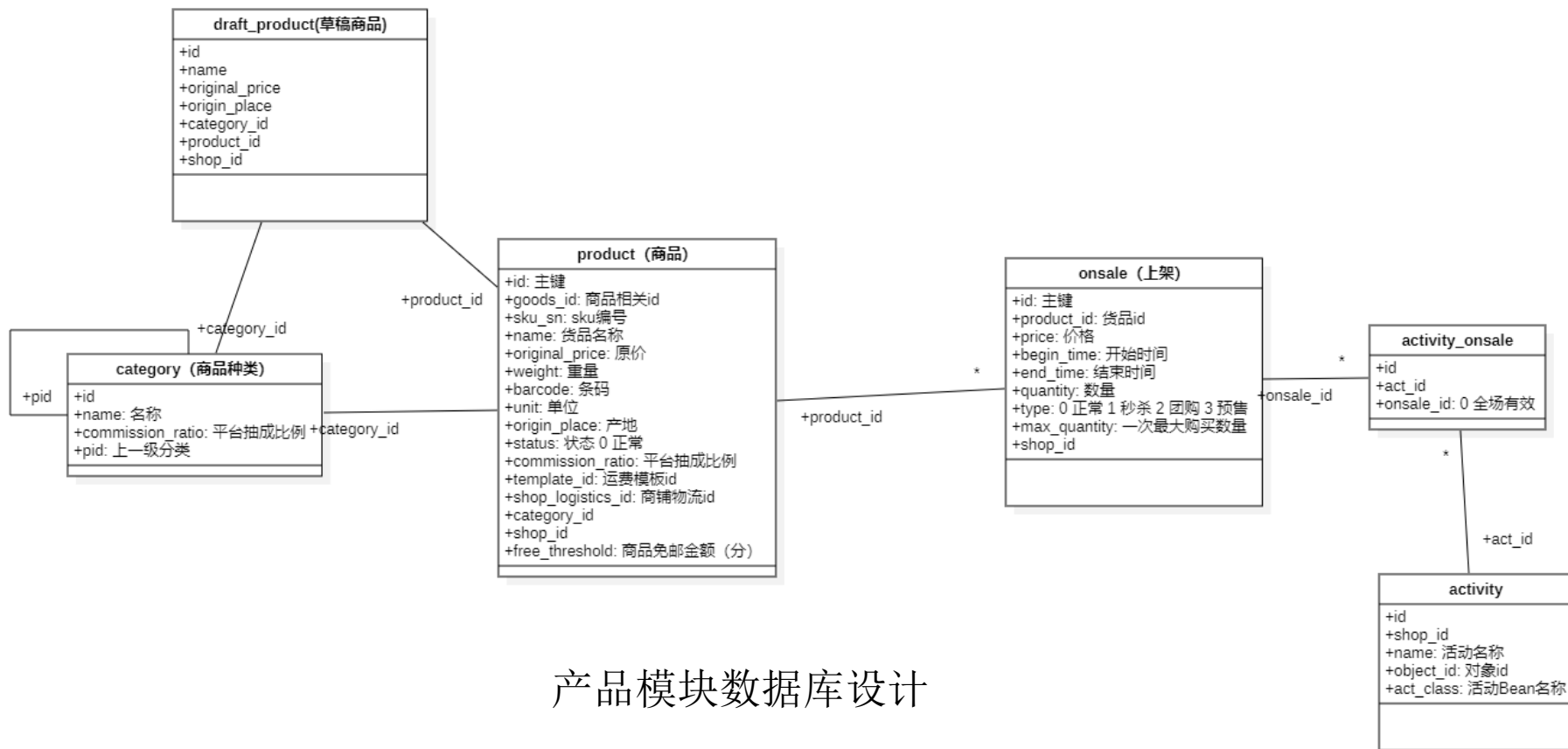


产品模块对象模型



# 1.2 对象模型

## Object Model



产品模块数据库设计



# 1.2 对象模型

## Object Model

- 贫血模型 (Anemic Model)
  - 对象模型只有getter和setter方法，没有任何业务逻辑
  - 所有的逻辑都在Service层和Dao层，同getter和setter方法访问对象模型



# 1.2 对象模型

## Object Model

### • 2021年取消团购

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class GroupOnActivity implements Serializable {

    private Long id;
    private String name;
    private Long shopId;
    private String shopName;
    private List<GroupOnStrategyVo> strategy;
    private ZonedDateTime beginTime;
    private ZonedDateTime endTime;
    private Long creatorId;
    private String creatorName;
    private Long modifierId;
    private String modifierName;
    private ZonedDateTime gmtCreate;
    private ZonedDateTime gmtModified;
    private Byte state;

    public GroupOnState getState() {
        return GroupOnState.valueOf(state);
    }

    public void setState(GroupOnState state) {
        this.state = state.getCode().byteValue();
    }
}
```

```
@Service
public class GroupOnActivityService {

    @Autowired
    private GroupActivityDao groupActivityDao;

    @Autowired
    private GoodsService goodsService;

    @Transactional(rollbackFor = Exception.class)
    public ReturnObject offlineGroupOnActivity(long id, long shopId, long loginUser, String loginUsername) {
        ReturnObject<GroupOnActivity> findObj = groupActivityDao.getGroupOnActivity(id);
        if(!findObj.getCode().equals(ReturnNo.OK))
        {
            return findObj;
        }
        if(findObj.getData()==null)
        {
            return new ReturnObject(ReturnNo.RESOURCE_ID_NOTEXIST);
        }
        if(findObj.getData().getState().equals(GroupOnState.OFFLINE))
        {
            return new ReturnObject(ReturnNo.STATENOTALLOW);
        }

        GroupOnActivity groupOnActivity = new GroupOnActivity();
        setPoModifiedFields(groupOnActivity, loginUser, loginUsername);
        groupOnActivity.setId(id);
        groupOnActivity.setState(GroupOnState.OFFLINE);

        ReturnObject obj = groupActivityDao.modifyGroupOnActivity(groupOnActivity);
        if(!obj.getCode().equals(ReturnNo.OK)) {
            return obj;
        }

        InternalReturnObject result = goodsService.offlineOnsale(id, shopId);
        if(result.getErrno()!=0){
            obj=new ReturnObject(ReturnNo.getByCode(result.getErrno()),result.getErrmsg());
        }else{
            obj=new ReturnObject();
        }
        return obj;
    }
}
```





# 1.2 对象模型

## Object Model

- 2021年取消团购

```
@Repository
public class GroupActivityDao {
    private Logger logger = LoggerFactory.getLogger(GroupActivityDao.class);
    @Autowired
    private GroupOnActivityPoMapper groupOnActivityPoMapper;

    public ReturnObject<GroupOnActivity> getGroupOnActivity(long id){
        GroupOnActivityPo g1;
        try {
            g1 = groupOnActivityPoMapper.selectByPrimaryKey(id);
        }
        catch(Exception e) {
            logger.error(e.getMessage());
            return new ReturnObject<>(ReturnNo.INTERNAL_SERVER_ERR,e.getMessage());
        }
        if(g1==null)
        {
            return new ReturnObject<>(ReturnNo.RESOURCE_ID_NOTEXIST);
        }
        GroupOnActivity groupOnActivity = (GroupOnActivity) cloneVo(g1,GroupOnActivity.class);
        return new ReturnObject<GroupOnActivity>(groupOnActivity);
    }

    public ReturnObject modifyGroupOnActivity(GroupOnActivity groupOnActivity)
    {
        ReturnObject retObj;
        GroupOnActivityPo groupOnActivityPo = (GroupOnActivityPo) cloneVo(groupOnActivity,GroupOnActivityPo.class);
        setPoModifiedFields(groupOnActivityPo,groupOnActivity.getModifierId(),groupOnActivity.getModifierName());
        int ret;
        try
        {
            ret = groupOnActivityPoMapper.updateByPrimaryKeySelective(groupOnActivityPo);
        }
        catch(Exception e)
        {
            logger.error(e.getMessage());
            return new ReturnObject(ReturnNo.INTERNAL_SERVER_ERR,e.getMessage());
        }
        if (ret == 0){
            retObj = new ReturnObject(ReturnNo.RESOURCE_ID_NOTEXIST);
        } else {
            retObj = new ReturnObject(ReturnNo.OK);
        }
        return retObj;
    }
}
```



# 1.2 对象模型

## Object Model

- 2021年取消团购

```
public ReturnObject onlineOrOfflineOnSaleAct(Long actId, Long userId, String userName, OnSale.State cntState, OnSale.State finalState) {
    try {
        OnSalePoExample oe = new OnSalePoExample();
        OnSalePoExample.Criteria cr = oe.createCriteria();
        cr.andActivityIdEqualTo(actId);
        Byte s1 = cntState.getCode().byteValue();
        cr.andStateEqualTo(s1);

        Byte s2 = finalState.getCode().byteValue();
        List<OnSalePo> pos = onSalePoMapper.selectByExample(oe);

        for (OnSalePo po : pos) {
            po.setState(s2);
            setPoModifiedFields(po, userId, userName);

            if (finalState == OnSale.State.OFFLINE) {
                //如果结束时间晚于当前时间且开始时间早于当前时间, 修改结束时间为当前时间
                if (po.getEndTime().isAfter(LocalDateTime.now()) && po.getBeginTime().isBefore(LocalDateTime.now())) {
                    po.setEndTime(LocalDateTime.now());
                }
            } else if (finalState == OnSale.State.ONLINE) {
                //如果开始时间早于当前时间且结束时间晚于当前时间, 修改开始时间为当前时间
                if (po.getBeginTime().isBefore(LocalDateTime.now()) && po.getEndTime().isAfter(LocalDateTime.now())) {
                    po.setBeginTime(LocalDateTime.now());
                }
            }

            onSalePoMapper.updateByPrimaryKeySelective(po);
        }
        return new ReturnObject();
    } catch (Exception e) {
        logger.error(e.getMessage());
        return new ReturnObject(ReturnNo.INTERNAL_SERVER_ERR, e.getMessage());
    }
}
```



# 1.2 对象模型

## Object Model

- 充血模型 (Fat Model)
  - 数据和对应的业务逻辑被封装到同一个类中, 满足面向对象的封装特性
  - 逻辑被分配到Service层, Dao层和bo对象中



# 1.2 对象模型

## Object Model

- 2023年取消团购

```
@Service
@Transactional(propagation = Propagation.REQUIRED)
public class GrouponActService {

    private Logger logger = LoggerFactory.getLogger(GrouponActService.class);

    private RedisUtil redisUtil;

    private ProductDao productDao;

    private OnsaleDao onsaleDao;

    private ActivityDao activityDao;

    @Autowired
    public GrouponActService(RedisUtil redisUtil, OnsaleDao onsaleDao, ActivityDao activityDao, ProductDao productDao) {
        this.redisUtil = redisUtil;
        this.onsaleDao = onsaleDao;
        this.activityDao = activityDao;
        this.productDao = productDao;
    }

    public void cancel(Long shopId, Long id, UserDto user) {
        GrouponAct activity = this.activityDao.findById(id, shopId, GrouponAct.ACTCLASS);
        List<String> keys = activity.cancel(user);
        this.redisUtil.del(keys.toArray(new String[keys.size()]));
    }
}
```



# 1.2 对象模型

## Object Model

@Repository

public class ActivityDao {

public <T extends Activity> T findById(Long id, Long shopId, String actClass) throws RuntimeException{

assert(null != id && null != shopId) : "ActivityDao.findById: activity id and shopId can not be null";

String key = String.format(KEY, id);

if (redisUtil.hasKey(key)){

Activity act = (Activity) redisUtil.get(key);

if (!shopId.equals(act.getShopId()) && PLATFORM != shopId) {

throw new BusinessException(ReturnNo.RESOURCE\_ID\_OUTSCOPE, String.format(ReturnNo.RESOURCE\_ID\_OUTSCOPE.getMessage(), "活动", id, shopId));

}

if (act.getActClass().equals(actClass)){

setBo(act);

return (T) act;

} else {

throw new BusinessException(ReturnNo.RESOURCE\_ID\_NOTEXIST, String.format(ReturnNo.RESOURCE\_ID\_NOTEXIST.getMessage(), "活动", id));

}

}

Optional<ActivityPo> ret = this.activityPoMapper.findById(id);

if (ret.isPresent()){

ActivityPo po = ret.get();

if (!shopId.equals(po.getShopId()) && PLATFORM != shopId) {

throw new BusinessException(ReturnNo.RESOURCE\_ID\_OUTSCOPE, String.format(ReturnNo.RESOURCE\_ID\_OUTSCOPE.getMessage(), "活动", id, shopId));

}

if (po.getActClass().equals(actClass)) {

return (T) this.getBo(po, Optional.ofNullable(key));

}else{

throw new BusinessException(ReturnNo.RESOURCE\_ID\_NOTEXIST, String.format(ReturnNo.RESOURCE\_ID\_NOTEXIST.getMessage(), "活动", id));

}

}else{

throw new BusinessException(ReturnNo.RESOURCE\_ID\_NOTEXIST, String.format(ReturnNo.RESOURCE\_ID\_NOTEXIST.getMessage(), "活动", id));

}

}

}

# 1.2 对象模型

## Object Model

```
@AllArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
@ToString(callSuper = true)
@CopyFrom({GrouponActVo.class, GrouponActModVo.class, GrouponActPo.class, ActivityPo.class})
public class GrouponAct extends Activity{

    public static final String ACTCLASS = "grouponActDao";

    private List<Threshold> thresholds;

    public GrouponAct() {
        super();
        this.actClass = ACTCLASS;
    }

    public List<String> cancel(UserDto user){
        List<Onsale> onsaleList = this.getOnsaleList();
        LocalDateTime now = LocalDateTime.now();
        List<String> keys = new ArrayList<>(onsaleList.size());
        for (Onsale obj : onsaleList){
            Onsale updateObj = obj.cancel(now);
            if (null != updateObj) {
                keys.add(this.onsaleDao.save(updateObj, user));
            }
        }
        return keys;
    }
}
```



# 1.2 对象模型

## Object Model

@Repository

@RefreshScope

public class OnsaleDao {

```
private void hasConflictOnsale(Onsale onsale) throws BusinessException{
    logger.debug("hasConflictOnsale: onsale = {}", onsale);
    if ( null == onsale.getProductId() && null == onsale.getBeginTime() && null == onsale.getEndTime()){
        throw new IllegalArgumentException("OnsaleDao.hasConflictOnsale: onsale's productId, beginTime and endTime can not be null");
    }

    PageRequest pageable = PageRequest.of(0, MAX_RETURN, Sort.by(Sort.Direction.ASC, "beginTime"));
    List<OnsalePo> poList = this.onsalePoMapper.findOverlap(onsale.getProductId(), onsale.getBeginTime(), onsale.getEndTime(), pageable);
    if ( poList.size() > 0){
        logger.debug("hasConflictOnsale: poList Size = {}, onsale's id = {}", poList.size(), onsale.getId());
        if (null != onsale.getId()) {
            //修改的目标onsale 不计算在重复范围内
            poList = poList.stream().filter(o -> !onsale.getId().equals(o.getId())).collect(Collectors.toList());
        }

        if (poList.size() > 0) {
            throw new BusinessException(ReturnNo.GOODS_ONSALE_CONFLICT, String.format(ReturnNo.GOODS_ONSALE_CONFLICT.getMessage(), poList.get(0).getId()));
        }
    }
}

public String save(Onsale onsale, UserDto user) throws BusinessException{
    logger.debug("save: onsale={}", onsale);
    this.hasConflictOnsale(onsale);
    onsale.setModifier(user);
    onsale.setGmtModified(LocalDateTime.now());
    OnsalePo onsalePo = CloneFactory.copy(new OnsalePo(), onsale);
    OnsalePo newPo = this.onsalePoMapper.save(onsalePo);
    if (Onsale.NOTEXIST == newPo.getId()){
        throw new BusinessException(ReturnNo.RESOURCE_ID_NOTEXIST, String.format(ReturnNo.RESOURCE_ID_NOTEXIST.getMessage(), "销售", onsalePo.getId()));
    }
    return String.format(KEY, newPo.getId());
}
}
```

# 1.3 静态建模和动态建模

## Static Modeling and Dynamic Modeling

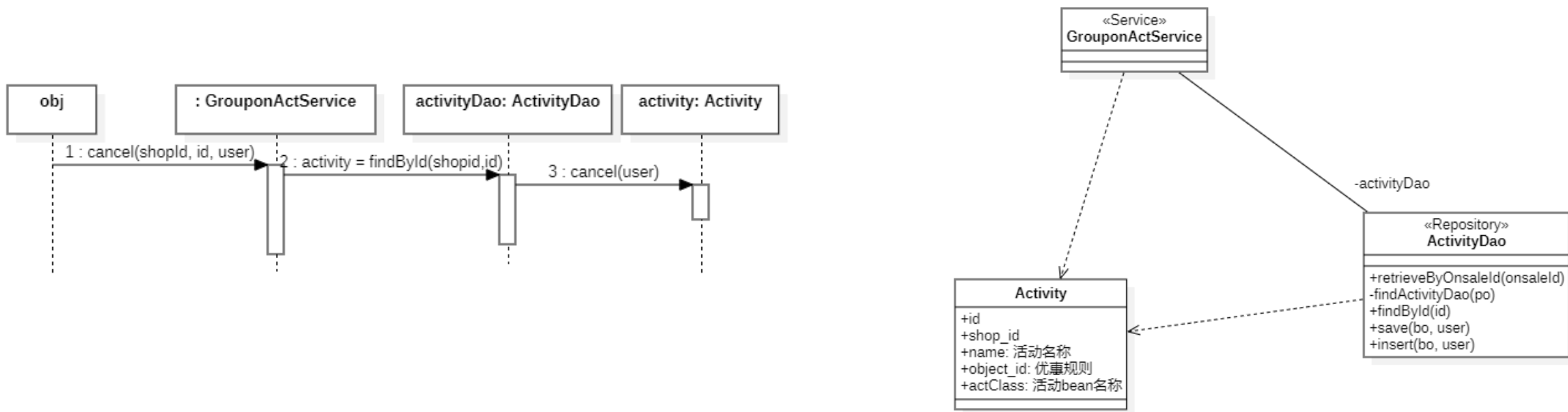
- 动态建模：使用GRASP原则，设计代码的行为，业务逻辑
  - UML 交互图 (Interaction Diagram)
    - 顺序图 (Sequence Diagram)和沟通图 (Communication Diagram)
  - 状态机图 (State Machine Diagram)
  - 活动图 (Activity Diagram)
- 静态建模：设计系统的结构，包括包结构，类，属性和方法定义
  - UML 类图 (Class Diagram)
  - 部署图 (Deployment Diagram)
  - 组件图 (Component Diagram)





# 1.3 静态建模和动态建模

## Static Modeling and Dynamic Modeling



静态建模和动态建模是同步进行的



# 1.3 静态建模和动态建模

## Static Modeling and Dynamic Modeling

- 动态建模
  - 思考对象交互的细节，描述实现逻辑的方式。
  - 是真正面向对象设计的细节。 “rubber hits the road”



# 2. UML交互图

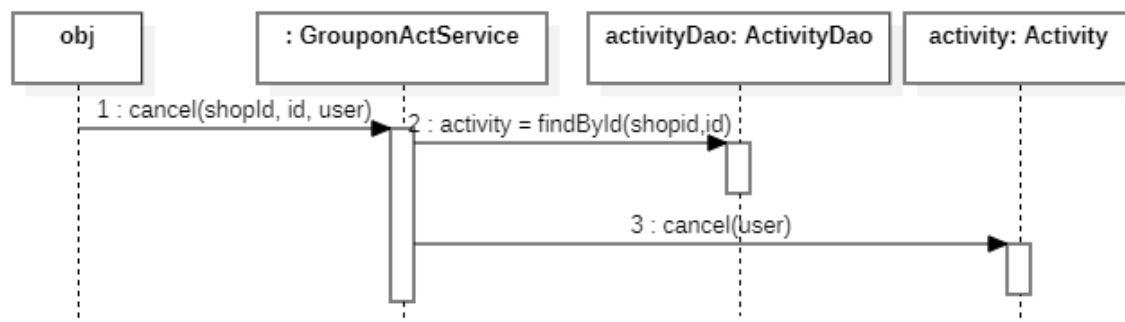
## Interaction Diagram



## 2.1 顺序图和通信图

### Sequence Diagram and Communication Diagram

- 顺序图 (Sequence Diagram)
  - 用于藩篱格式描述对象之间的交互



```
@Service
@Transactional(propagation = Propagation.REQUIRED)
public class GrouponActService {

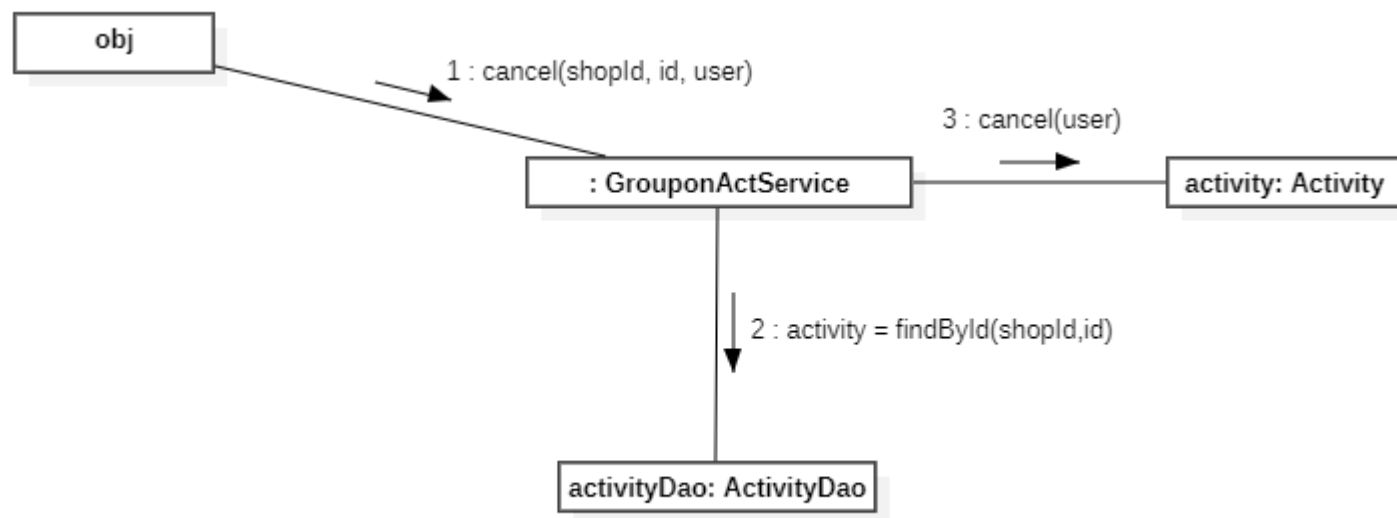
    public void cancel(Long shopId, Long id, UserDto user) {
        GrouponAct activity = this.activityDao.findById(id, shopId, GrouponAct.ACTCLASS);
        List<String> keys = activity.cancel(user);
        this.redisUtil.del(keys.toArray(new String[keys.size()]));
    }
}
```



## 2.1 顺序图和通信图

### Sequence Diagram and Communication Diagram

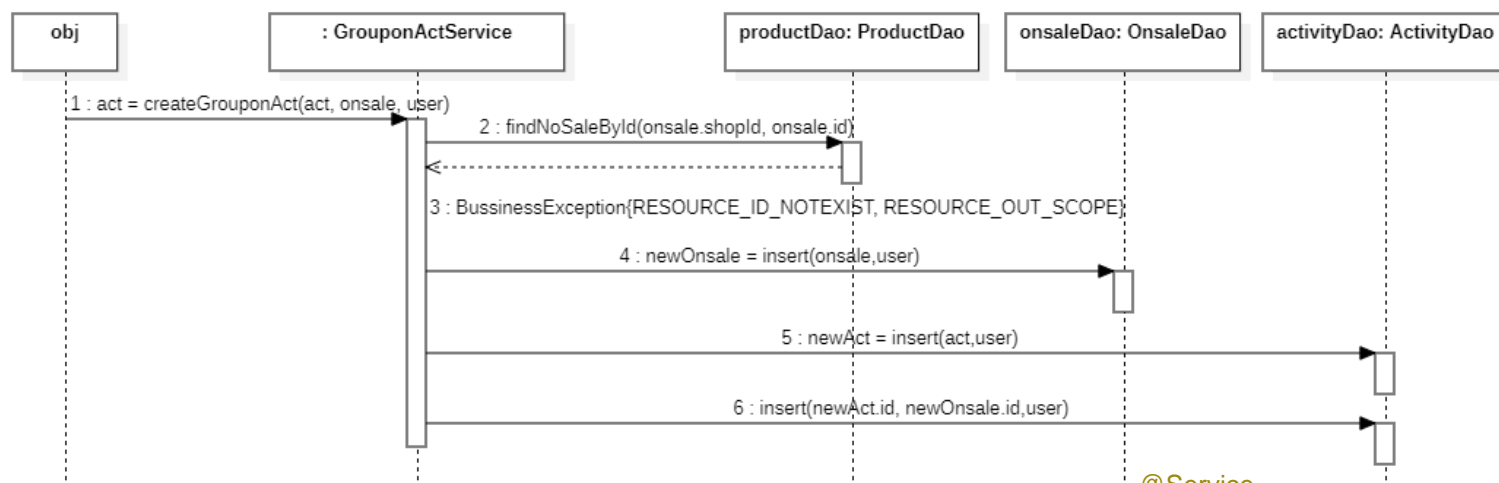
- 通信图（Communication Diagram）
  - 用网络图的格式描述对象之间的交互



## 2.1 顺序图和通信图

### Sequence Diagram and Communication Diagram

- 顺序图 (Sequence Diagram)



@Service

@Transactional(propagation = Propagation.REQUIRED)

public class GrouponActService {

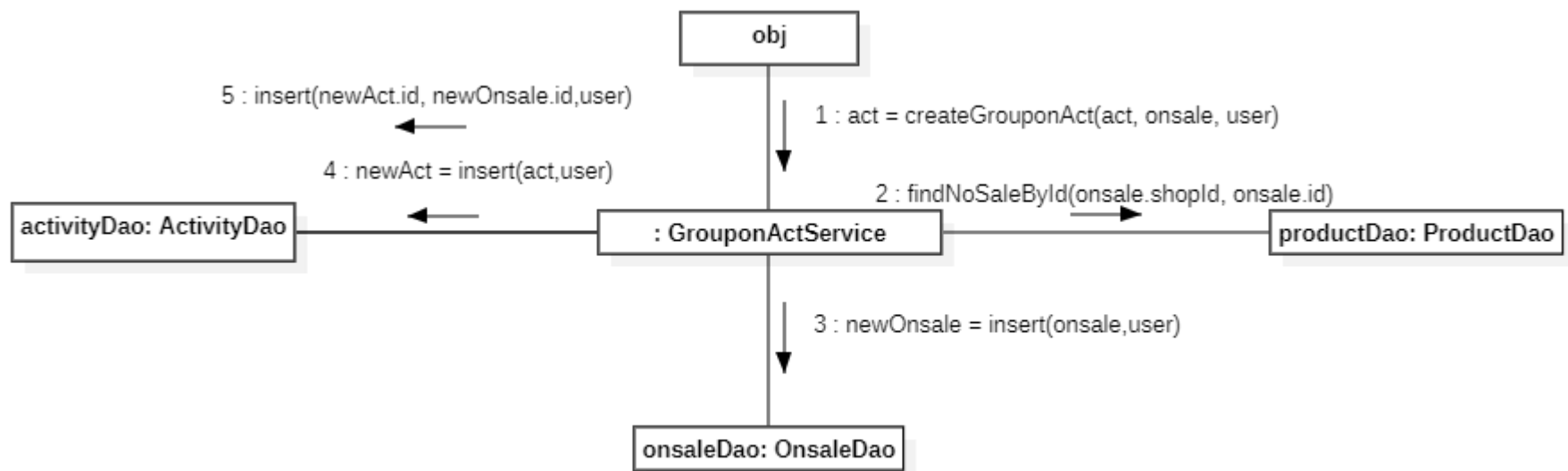
```
public GrouponAct createGrouponAct(GrouponAct act, Onsale onsale, UserDto user) {
    this.productDao.findNoOnsaleById(onsale.getShopId(), onsale.getProductId());
    onsale.setType(Onsale.GROUPON);
    Onsale newOnsale = this.onsaleDao.insert(onsale, user);
    GrouponAct newAct = this.activityDao.insert(act, user);
    this.activityDao.insertActivityOnsale(newAct.getId(), newOnsale.getId(), user);
    act.setId(newAct.getId());
    return act;
}
```



## 2.1 顺序图和通信图

### Sequence Diagram and Communication Diagram

- 通信图 (Communication Diagram)



## 2.1 顺序图和通信图

### Sequence Diagram and Communication Diagram

- 顺序图与通信图的区别

顺序图	清晰描述消息的顺序	需从左至右描述，特别在交互对象较多时比较占版面
通信图	节省版面	难以区分消息的顺序

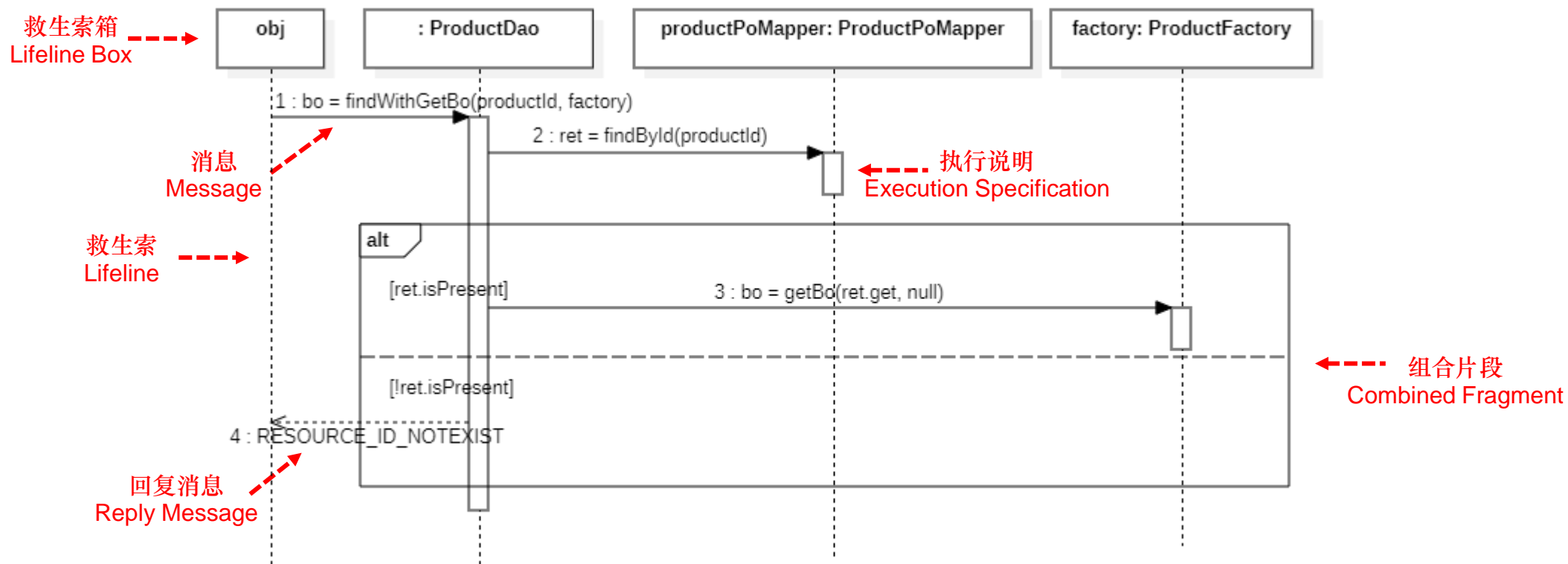




## 2.2 顺序图

### Sequence Diagram

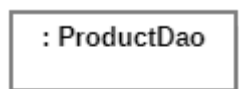
- 顺序图的符号



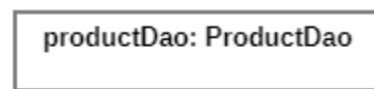
## 2.2 顺序图

### Sequence Diagram

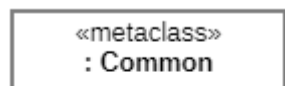
- 用Lifeline Box描述对象



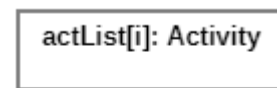
匿名ProductDao对象



变量名为productDao的ProductDao对象



Common的静态类



actList集合中的一个Activity对象

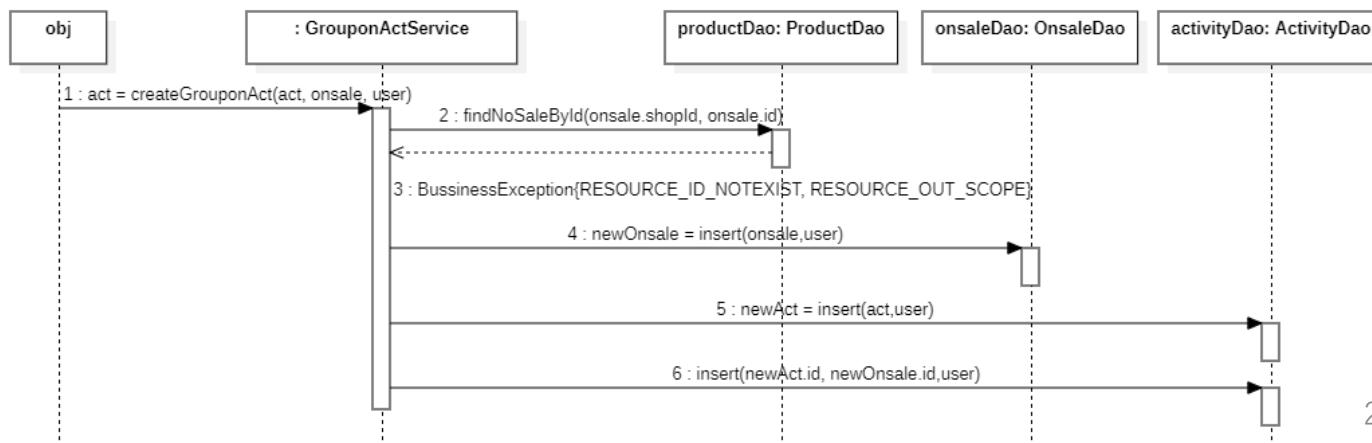
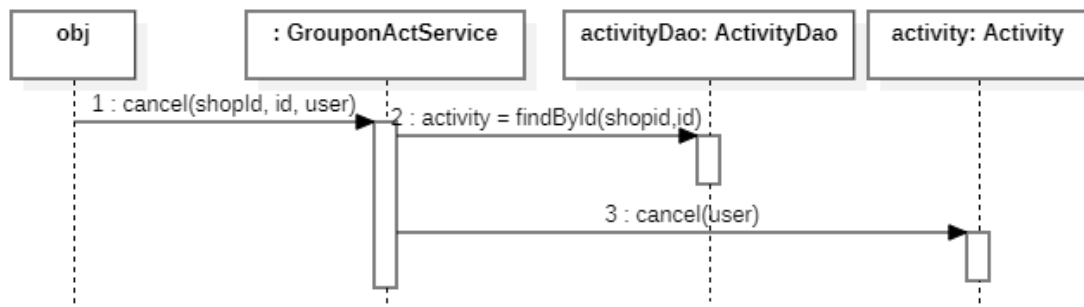


## 2.2 顺序图

### Sequence Diagram

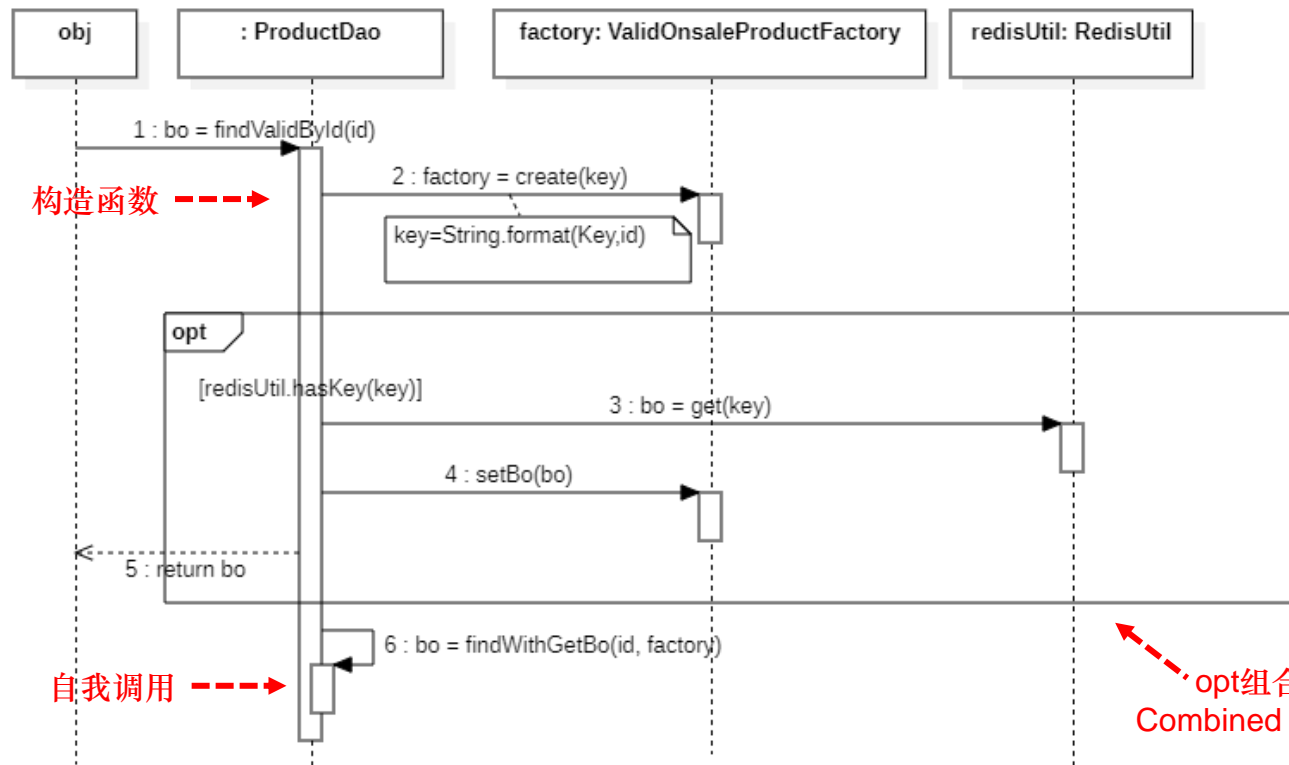
- 消息格式

- `return = message(parameter : parameterType) : returnType`



## 2.2 顺序图

### Sequence Diagram



```
@Repository
@RefreshScope
public class ProductDao {

    private final static Logger logger = LoggerFactory.getLogger(ProductDao.class);

    private RedisUtil redisUtil;

    public Product findValidById(Long id) throws RuntimeException {
        logger.debug("findValidById: id = {}", id);
        String key = String.format(KEY, id);
        ValidOnsaleProductFactory factory = new ValidOnsaleProductFactory(id);
        if (this.redisUtil.hasKey(key)) {
            Product bo = (Product) redisUtil.get(key);
            factory.setBo(bo);
            return bo;
        }
        return this.findWithGetBo(PLATFORM, id, factory, true);
    }
}
```



## 2.2 顺序图

### Sequence Diagram

- 组合片段

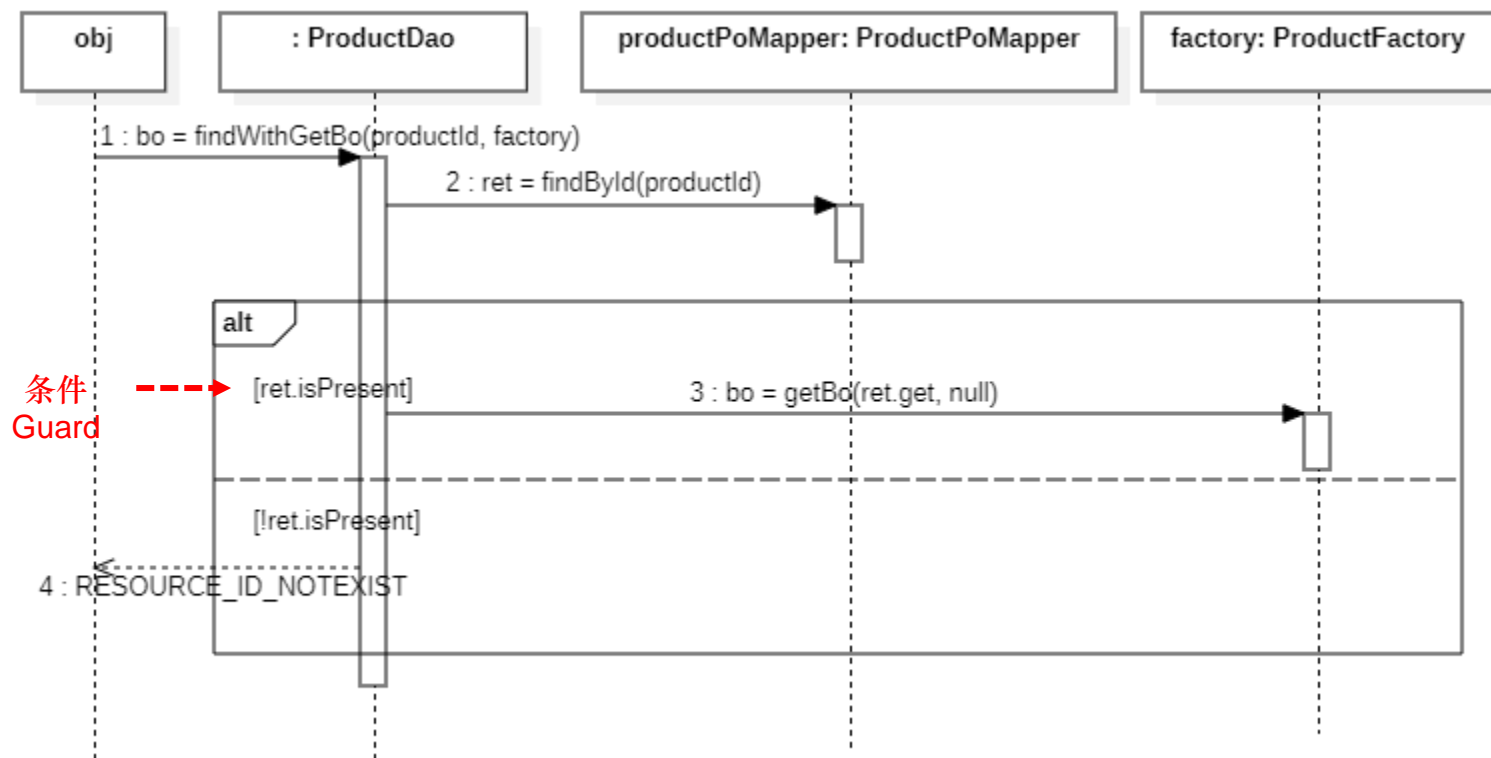
类型	解释
alt	if ... else
loop	循环
opt	if
region	关键区
par	并行代码区



## 2.2 顺序图

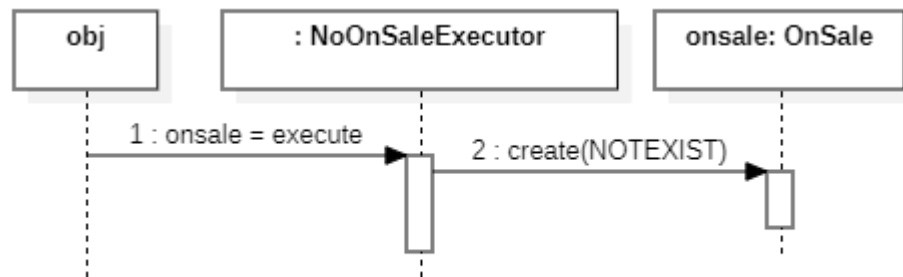
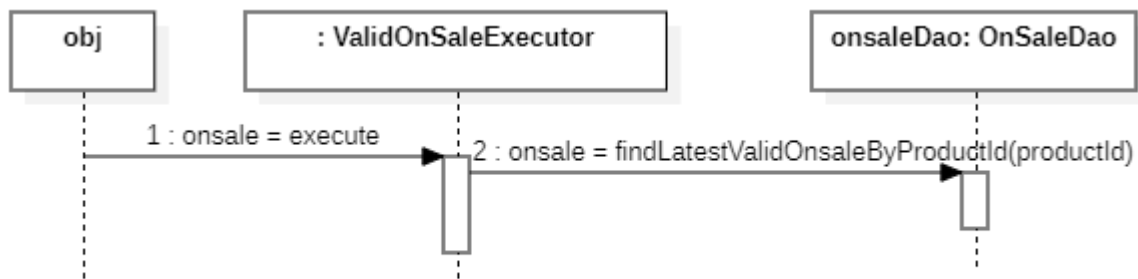
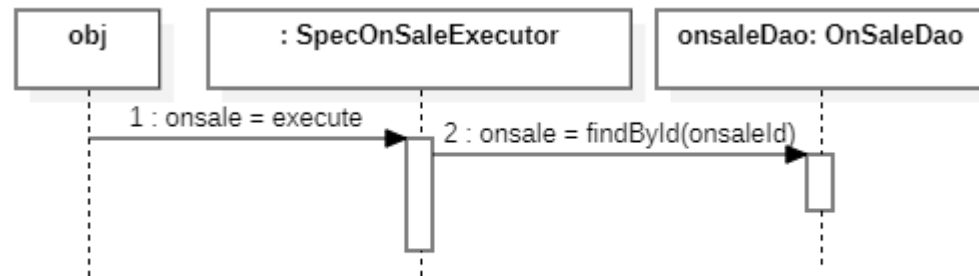
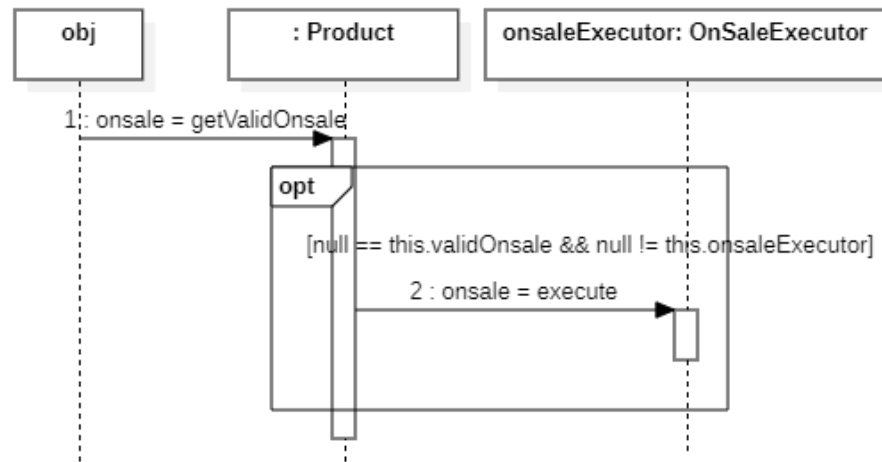
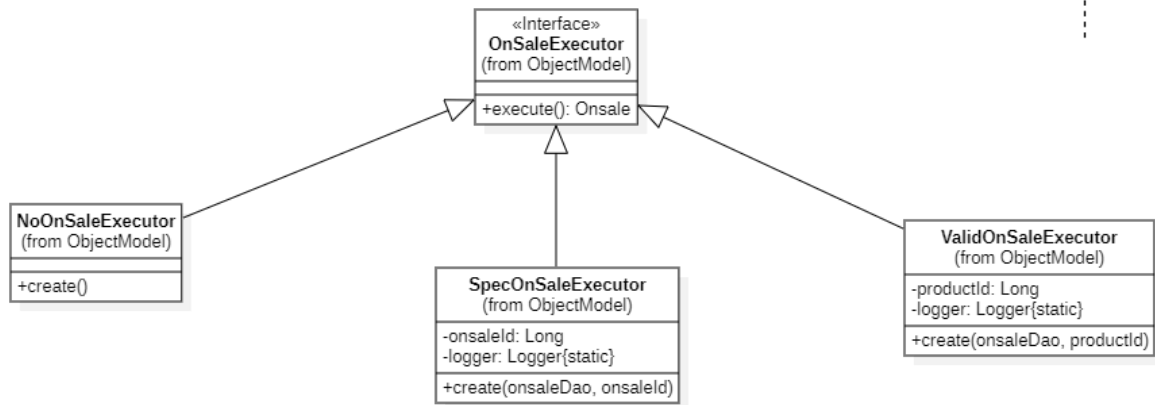
### Sequence Diagram

- 组合片段



## 2.2 顺序图 Sequence Diagram

- 多态



# 3. UML 状态图

**UML State Machine Diagram**





# 3.1 对象状态建模

## Object State Modeling

- 系统中的对象可以分为两类
  - 状态无关的对象：始终保持相同的行为特性.
  - 状态有关的对象：在不同状态下行为特性不同



## 3.2 状态机图

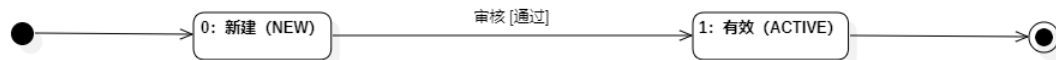
### State Machine Diagram

- 事件 (Event )
  - 事件指的是发生在时间和空间上的对状态机来讲有意义的那些事情。事件通常会引起状态的变迁，促使对象从一种状态切换到另一种状态
- 状态 (State)
  - 状态指的是对象在其生命周期中的一种状况.
- 转换 (Transition)
  - 是两个不同状态之间的迁移关系，表明处于某个状态的对象由某个事件触发并且在满足某个特定条件下进入一个新的状态。

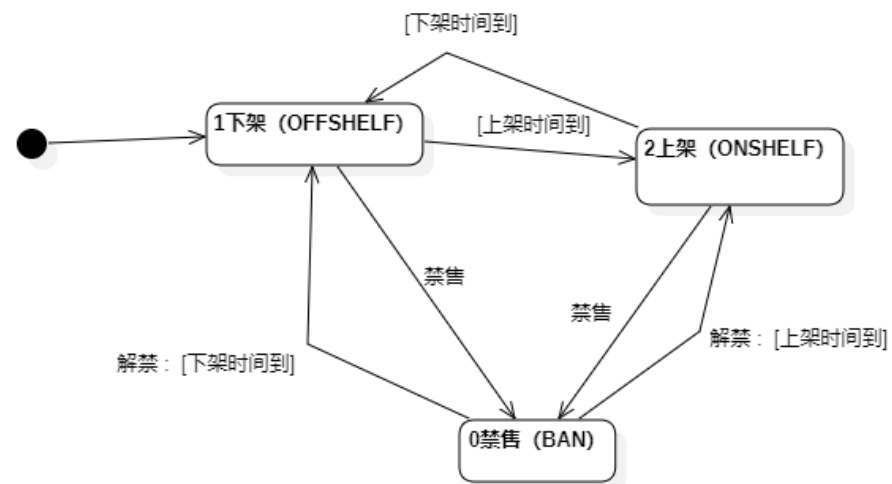


## 3.2 状态机图

### State Machine Diagram



活动状态图



产品状态图



# 4. GRASP: 职责驱动的设计

**GRASP: Designing Objects With Responsibilities**



# 4.1 职责驱动的设计

## Response-Driven Design

- 职责 (Responsibilities)
- 角色 (Roles)
- 协作 (Collaborations)

General Responsibility Assignment Software Patterns



# 4.1 职责驱动的设计

## Response-Driven Design

- GRASP方法
  - 创建者 (Creator)
  - 信息专家 (Information Expert)
  - 低耦合 (Low Coupling)
  - 高内聚 (High Cohesion)
  - 多态 (Polymorphism)
  - 间接 (Indirection)
  - 虚构 (Pure Fabrication)



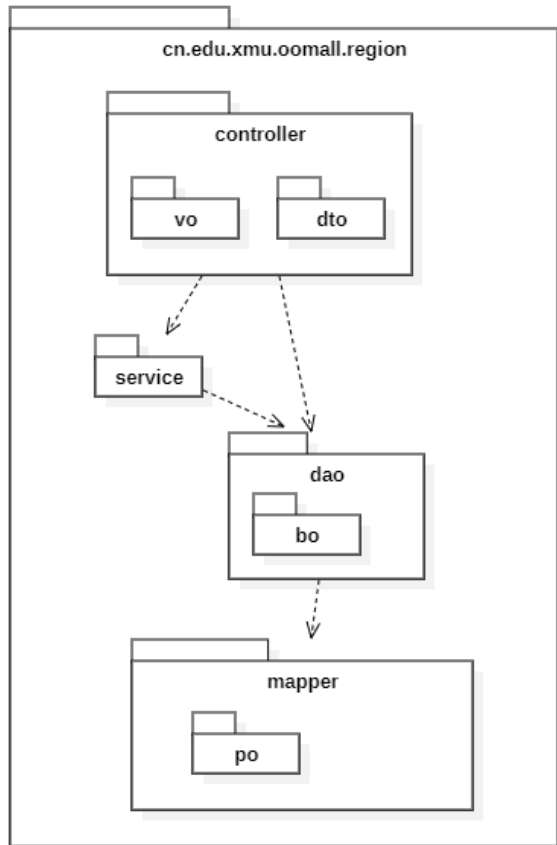
## 4.2 创建者

### Creator

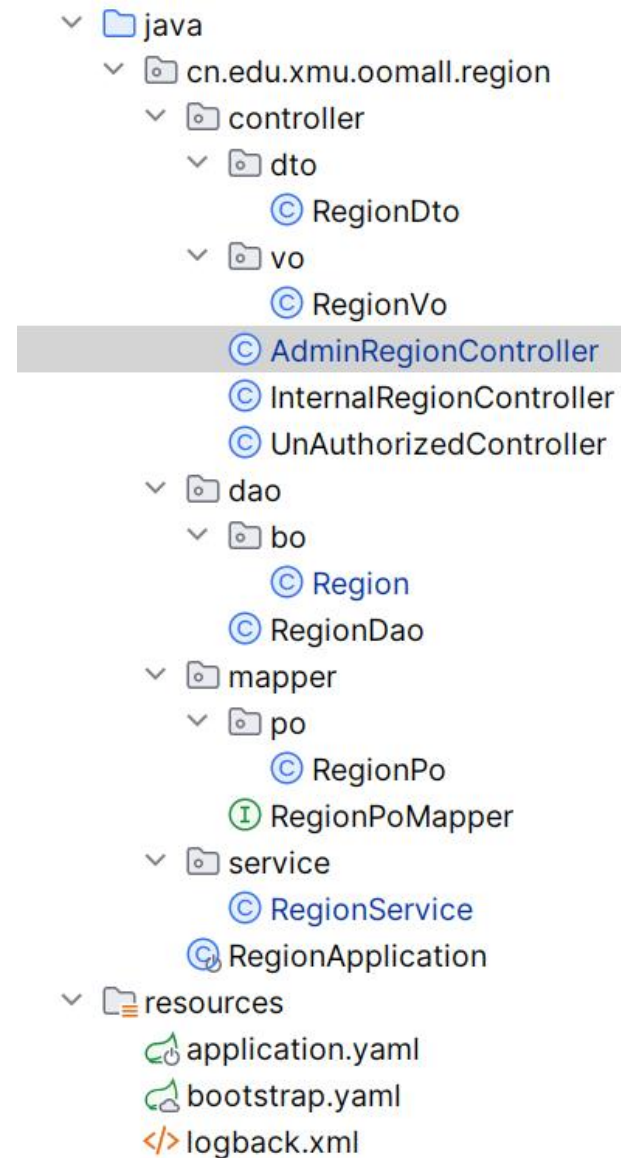
- 问题：对象A由谁创建？
- 方案： 如果对象B符合以下条件，则由B对象创建对象A：
  - B 包含 A。
  - B 记录 A。
  - B 用到 A。
  - B 有 A的初始化数据



## 4.2 创建者 Creator



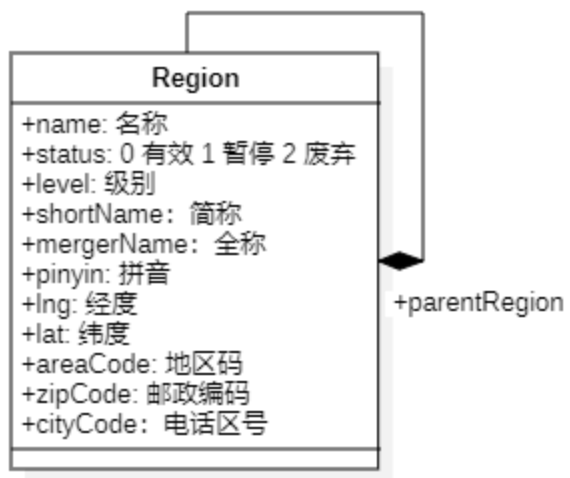
Region模块包图





## 4.2 创建者

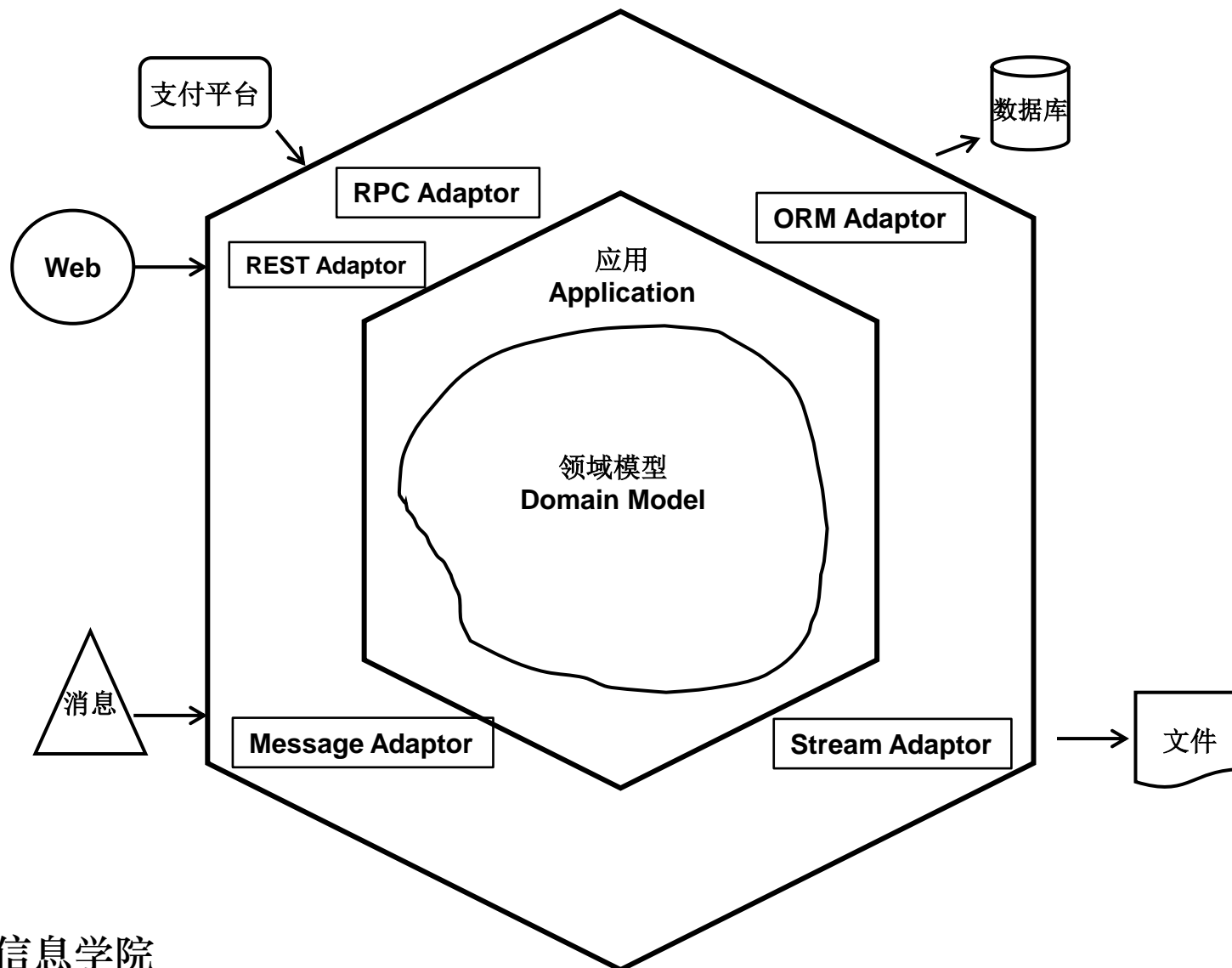
### Creator



Region模块对象模型



## 4.2 创建者 Creator



## 4.2 创建者 Creator

POST

/shops/{did}/regions/{id}/subregions 管理员在地区下新增子地区

Try it out

Parameters

Name	Description
<b>authorization</b> * required string (header)	用户token authorization
<b>did</b> * required integer(\$int64) (path)	只能为0,否则出17错误 did
<b>id</b> * required integer (path)	地区id id
<b>body</b> * required (body)	地区信息 Example Value   Model <pre>{  "name": "string",  "shortName": "string",  "mergerName": "string",  "pinyin": "string",  "lng": "string",  "lat": "string",  "areaCode": "string",  "zipCode": "string",  "cityCode": "string"}</pre> <div>Parameter content type application/json</div>

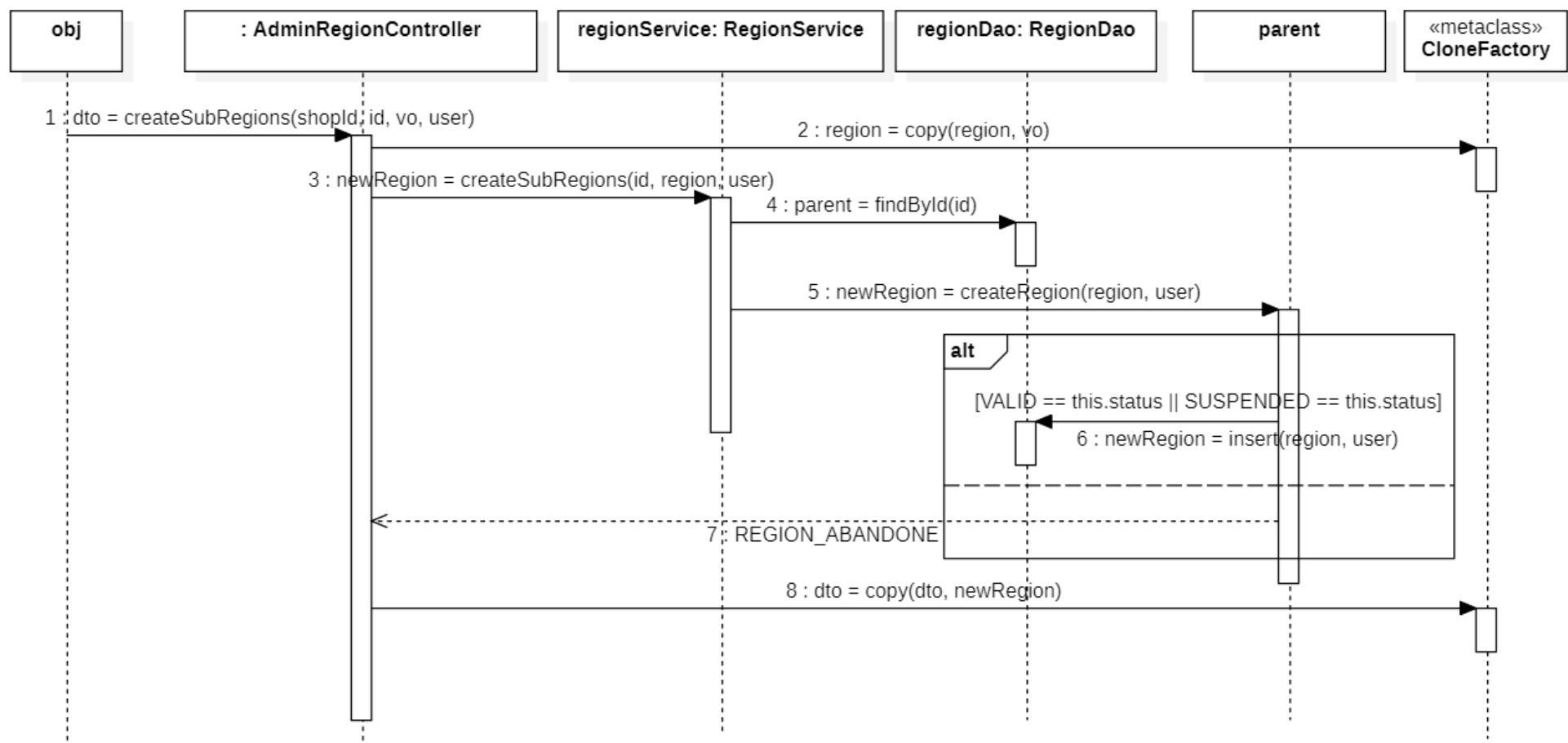
Responses

Response content type application/json

Code	Description
901	地区(id=%d)已废弃
default	成功 Example Value   Model <pre>{  "errno": 0,  "errmsg": "成功",  "data": {    "id": 0,    "name": "string"  }}</pre>



## 4.2 创建者 Creator



创建下级地区顺序图



## 4.3 信息专家

### Information Expert

- 问题： 分配职责给对象的基本原则是什么？
- 方案：“知者为之”-谁具备完成职责所需的信息，就由谁来承担职责



## 4.3 信息专家

### Information Expert

- 问题： 分配职责给对象的基本原则是什么？
- 方案：“知者为之”-谁具备完成职责所需的信息，就由谁来承担职责



## 4.3 信息专家

### Information Expert

**GET** /internal/regions/{id}/parents 查询地区的上级地区

- 无需登录
- 从最近到最远按序返回最多返回10条
- 需要缓存

Parameters

Try it out

Name	Description
<b>id</b> <small>★ required</small>	地区id
integer (path)	<input type="text" value="id"/>

Responses

Response content type application/json

Code	Description
default	成功

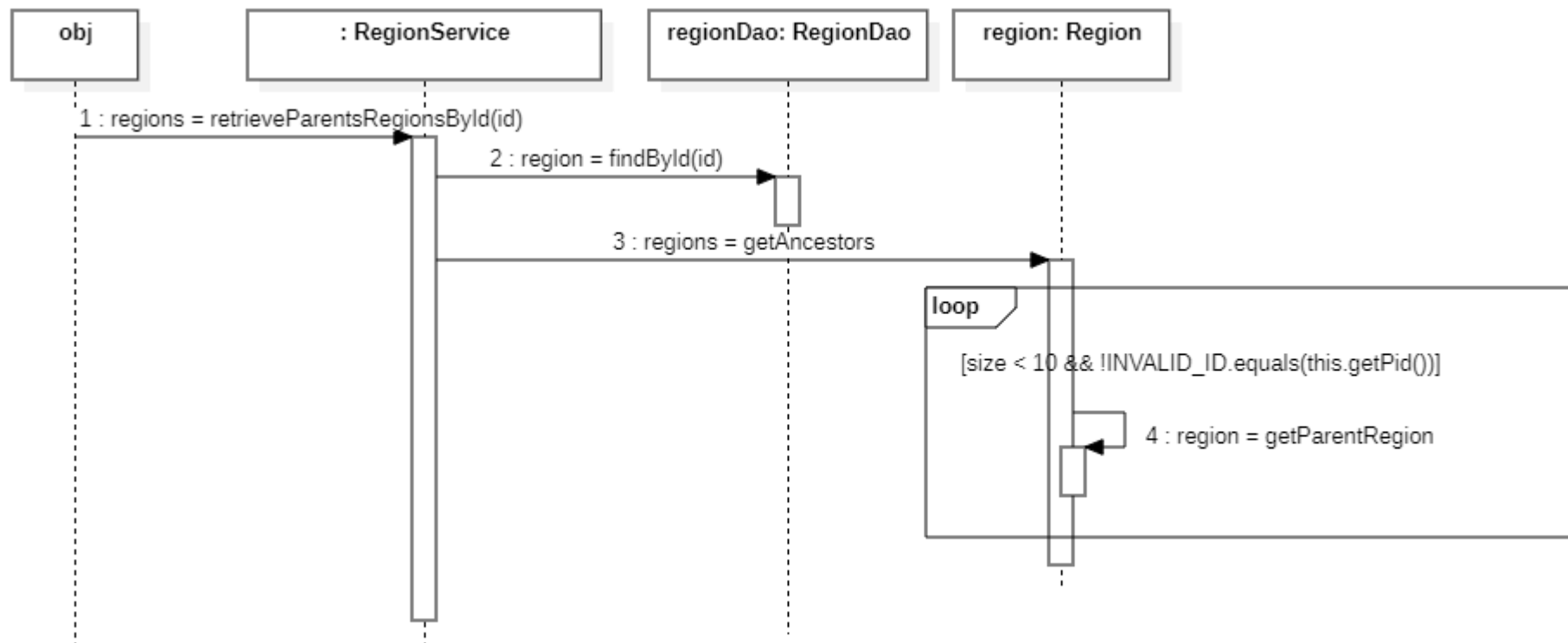
Example Value | Model

```
{  "errno": 0,  "errmsg": "成功",  "data": [    {      "id": 0,      "name": "string"    }  ]}
```

查询上级地区 (<https://app.swaggerhub.com/apis/mingqcn/OOMALL/1.3.1#/region/parentRegions>)



## 4.3 信息专家 Information Expert



查询上级地区顺序图





## 4.3 信息专家

### Information Expert

**PUT** /shops/{did}/regions/{id}/suspend 管理员停用某个地区

• 下级地区一并停用

Parameters

Try it out

Name	Description
<b>authorization</b> * required string (header)	用户token <input type="text" value="authorization"/>
<b>did</b> * required integer (\$int64) (path)	只能为0,否则出17错误 <input type="text" value="did"/>
<b>id</b> * required integer (path)	地区id <input type="text" value="id"/>

Responses

Response content type application/json

Code	Description
507	当前状态禁止此操作
default	成功

Example Value | Model

```
{  "errno": 0,  "errmsg": "成功"}
```

停用某个地区 (<https://app.swaggerhub.com/apis/mingqcn/OOMALL/1.3.1#/region/suspendRegions>)



## 4.3 信息专家

### Information Expert

**PUT** /shops/{did}/regions/{id}/resume 管理员恢复某个地区

- 下级地区一并恢复

Parameters

Try it out

Name	Description
<b>authorization</b> * required string (header)	用户token authorization
<b>did</b> * required integer (\$int64) (path)	只能为0,否则出17错误 did
<b>id</b> * required integer (path)	地区id id

Responses

Response content type application/json

Code	Description
default	成功

Example Value | Model

```
{  "errno": 0,  "errmsg": "成功"}
```

恢复某个地区 (<https://app.swaggerhub.com/apis/mingqcn/OOMALL/1.3.1#/region/resumeRegions>)



## 4.3 信息专家

### Information Expert

**DELETE** /shops/{did}/regions/{id} 管理员废弃某个地区

• 下级地区一并废弃

Parameters

Try it out

Name	Description
<b>authorization</b> * required string (header)	用户token <input type="text" value="authorization"/>
<b>did</b> * required integer (\$int64) (path)	只能为0,否则出17错误 <input type="text" value="did"/>
<b>id</b> * required integer (path)	地区id <input type="text" value="id"/>

Responses

Response content type application/json

Code	Description
507	当前状态禁止此操作
default	成功

Example Value | Model

```
{  "errno": 0,  "errmsg": "成功"}
```

废弃某个地区 (<https://app.swaggerhub.com/apis/mingqcn/OOMALL/1.3.1#/region/delRegions>)



## 4.3 信息专家

### Information Expert

**DELETE** /shops/{did}/regions/{id} 管理员废弃某个地区

- 下级地区一并废弃

Parameters

Try it out

Name	Description
<b>authorization</b> * required string (header)	用户token <input type="text" value="authorization"/>
<b>did</b> * required integer (\$int64) (path)	只能为0,否则出17错误 <input type="text" value="did"/>
<b>id</b> * required integer (path)	地区id <input type="text" value="id"/>

Responses

Response content type application/json

Code	Description
507	当前状态禁止此操作
default	成功

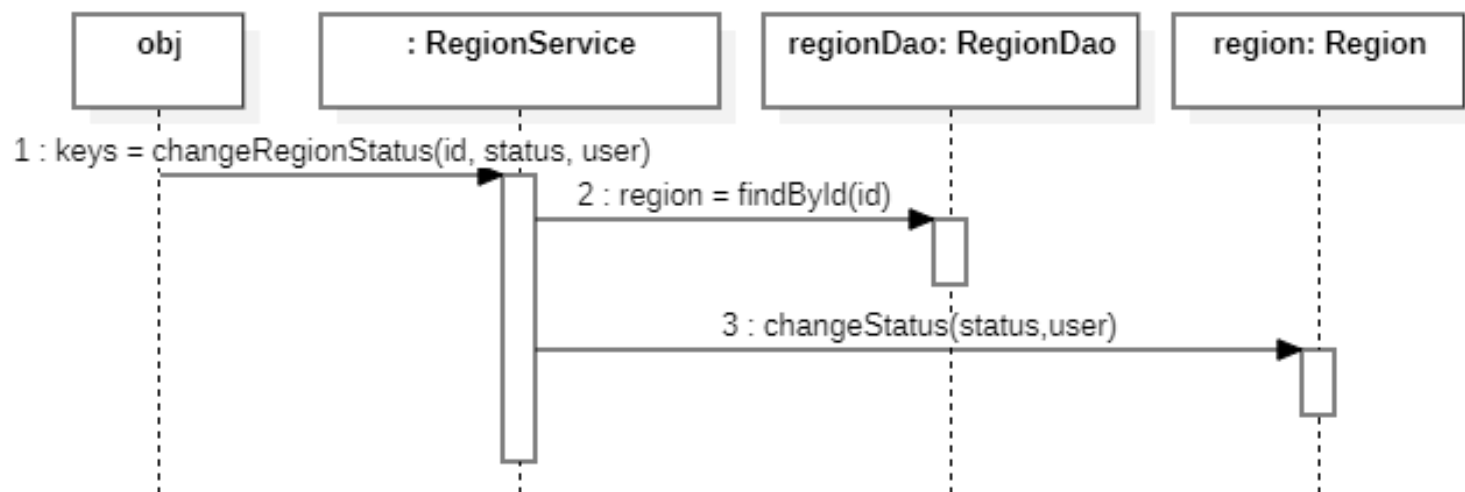
Example Value | Model

```
{  "errno": 0,  "errmsg": "成功"}
```

废弃某个地区 (<https://app.swaggerhub.com/apis/mingqcn/OOMALL/1.3.1#/region/delRegions>)



## 4.3 信息专家 Information Expert

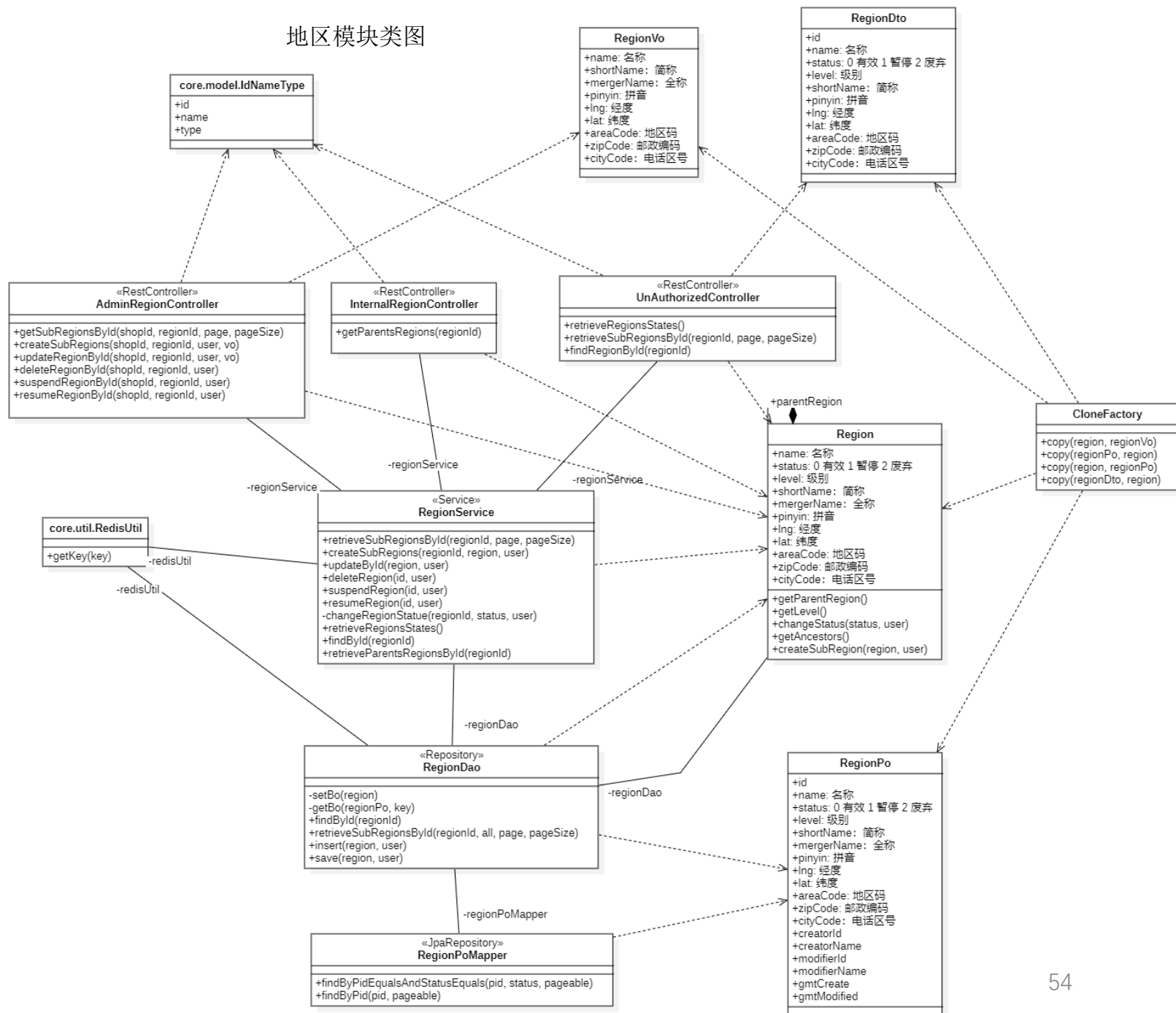


修改地区状态顺序图



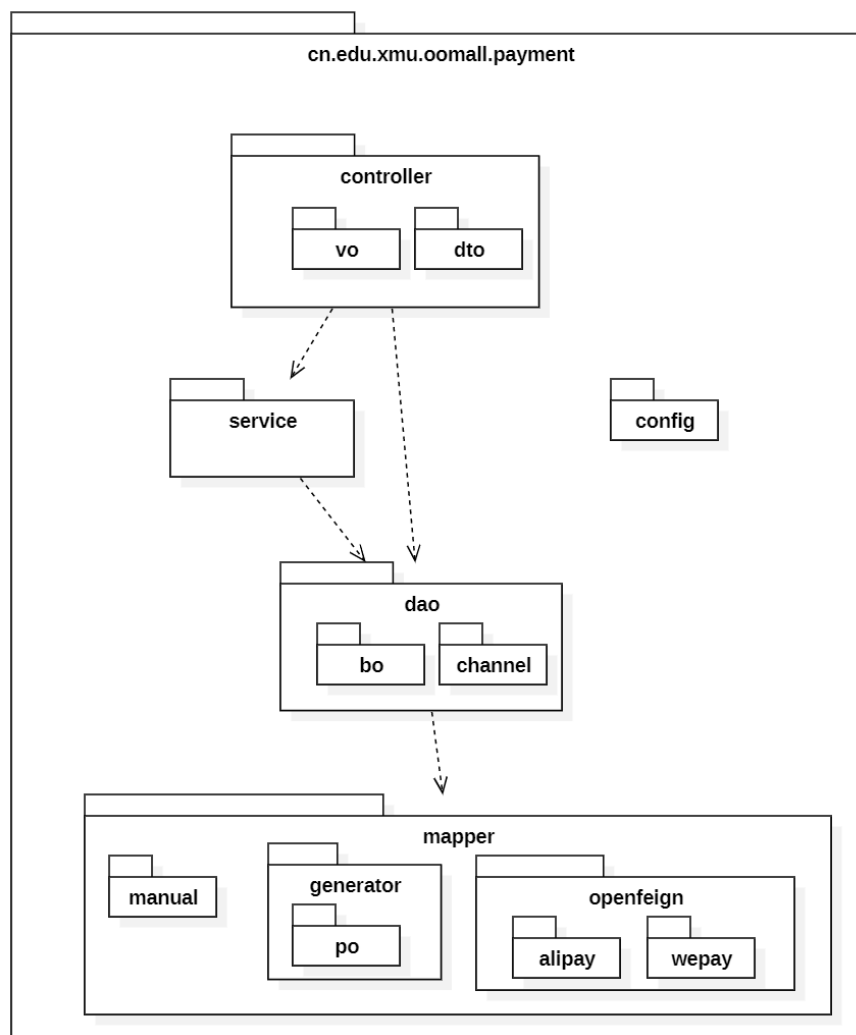
# 4.3 信息专家 Information Expert

地区模块类图



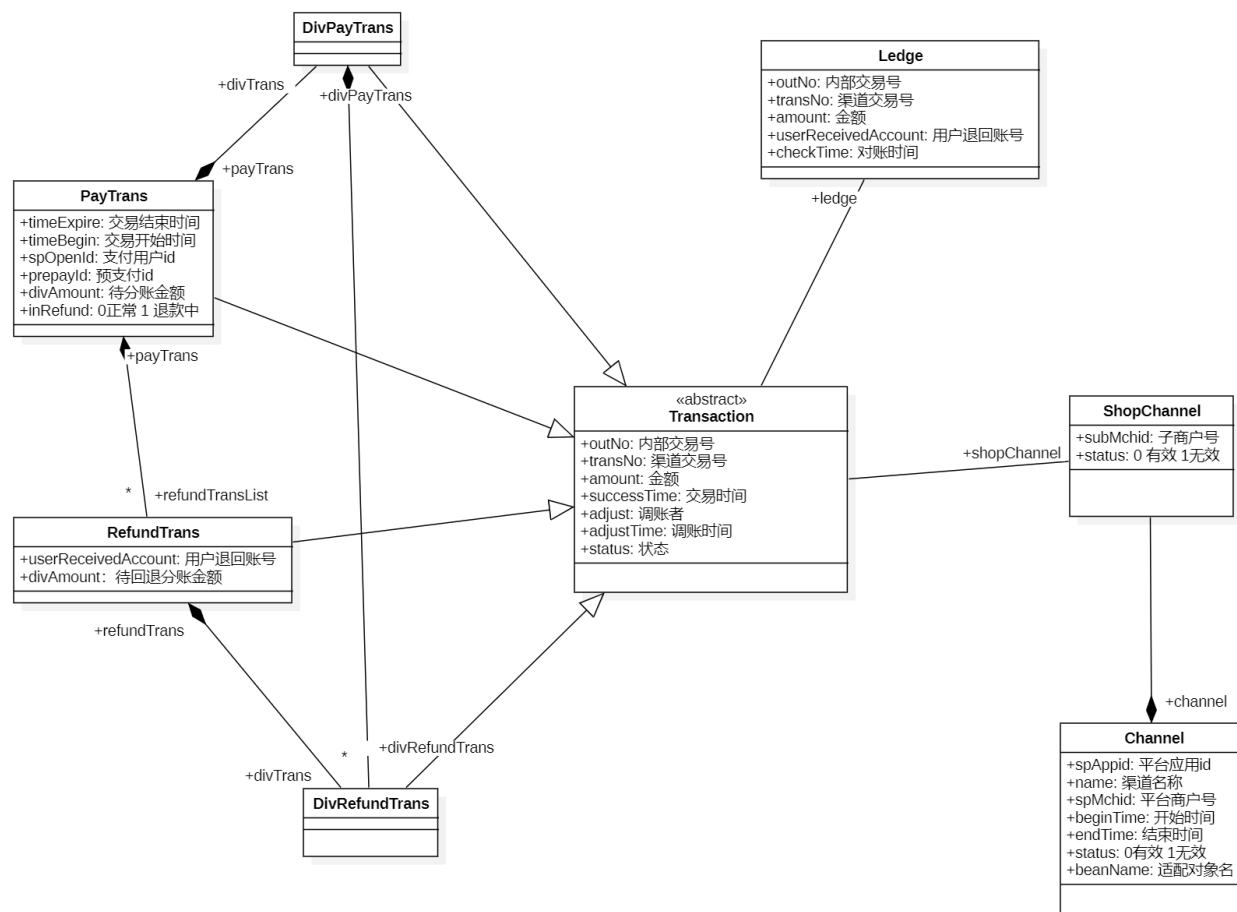
## 4.4 支付模块中创建者和信息专家

### Creator and Information Expert in Payment Module



# 4.4 支付模块中创建者和信息专家

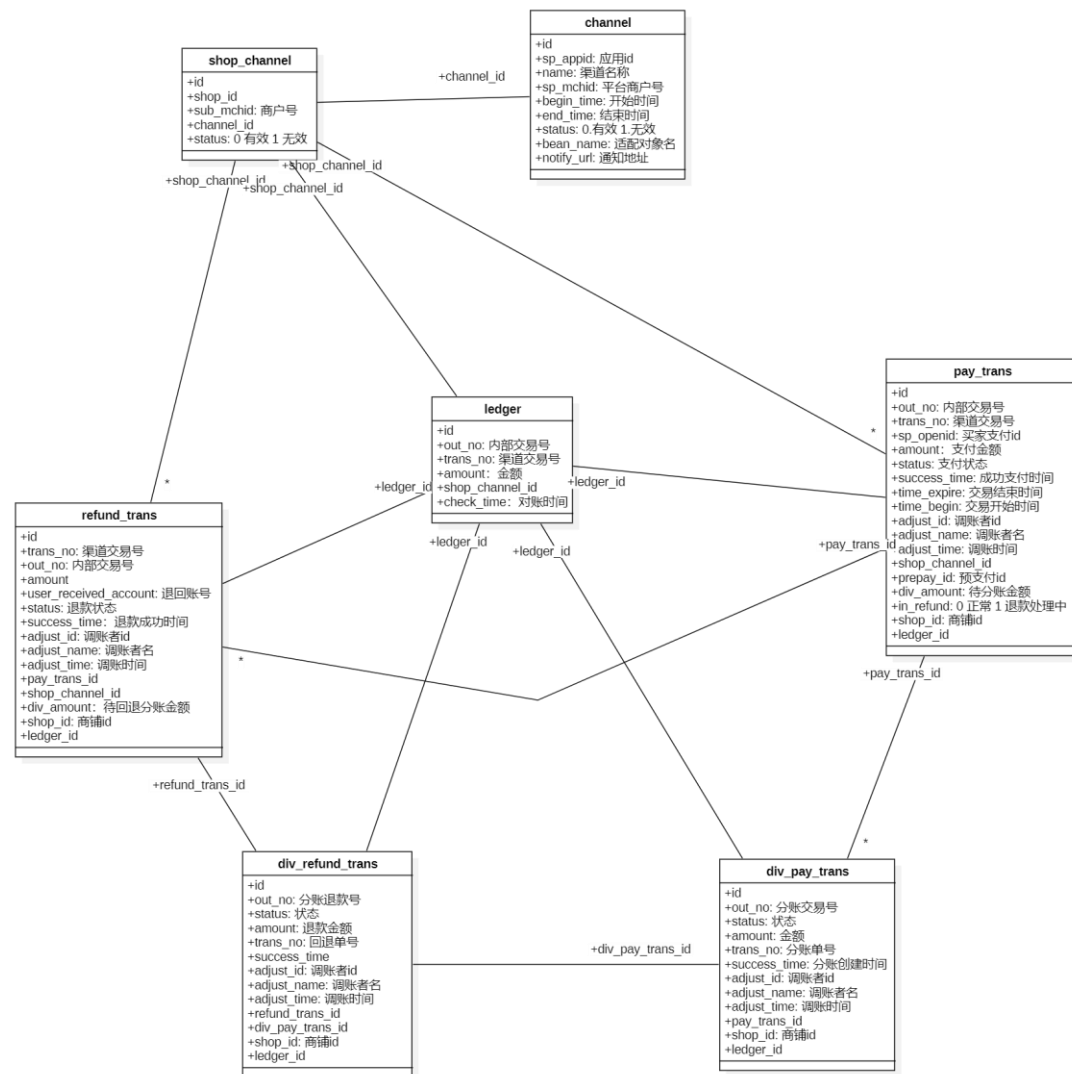
## Creator and Information Expert in Payment Module





# 4.4 支付模块中创建者和信息专家

## Creator and Information Expert in Payment Module



# 4.4 支付模块中创建者和信息专家

## Creator and Information Expert in Payment Module

PUT

/shops/{shopId}/channels/{id}/invalid 无效平台支付渠道

📄 ↶ ↷

-有效状态设为无效

https://app.swaggerhub.com/apis/mingqcn/00MALL/1.3.2#/payment/invalidChannel

Parameters

Try it out

Name	Description
<b>authorization</b> * required string (header)	用户token
<b>shopId</b> * required integer(\$int64) (path)	商铺id, 只能为0
<b>id</b> * required integer (path)	渠道id

Responses

Response content type application/json

Code	Description
default	成功

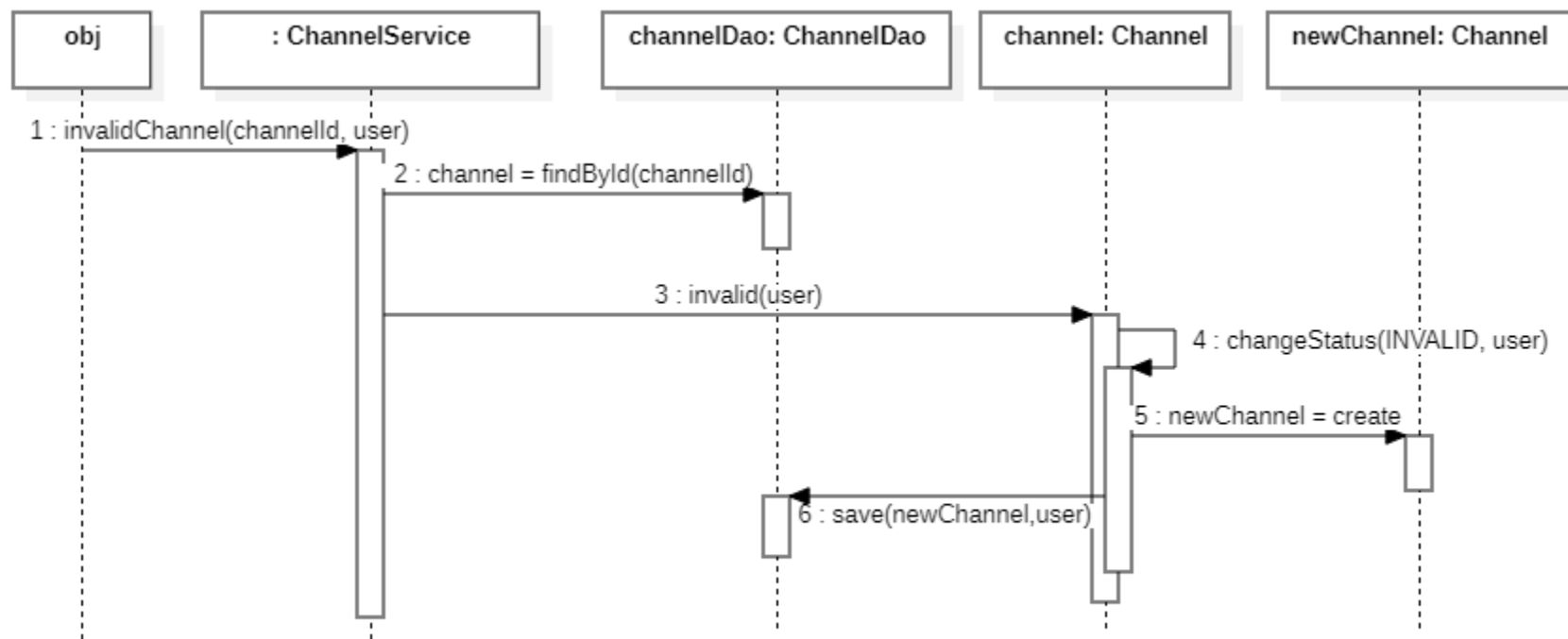
Example Value | Model

```
{  "errno": 0,  "errmsg": "成功"}
```



## 4.4 支付模块中创建者和信息专家

### Creator and Information Expert in Payment Module



## 4.4 支付模块 Creator and Inform

GET

/shops/{shopId}/shopchannels 获得商铺的所有支付渠道

📄 ↶ ↷

• 返回商铺的所有支付渠道，有效和无效

https://app.swaggerhub.com/apis/mingqcn/OOMALL/1.3.2#/payment/retrieveShopChannel

Parameters

Try it out

Name	Description
<b>authorization</b> * required string (header)	用户token <input type="text" value="authorization"/>
<b>shopId</b> * required integer(\$int64) (path)	店铺id <input type="text" value="shopId"/>
page integer (query)	页码 <input type="text" value="page"/>
pageSize integer (query)	每页数目 <input type="text" value="pageSize"/>

Responses

Response content type application/json

Code	Description
default	成功

Example Value | Model

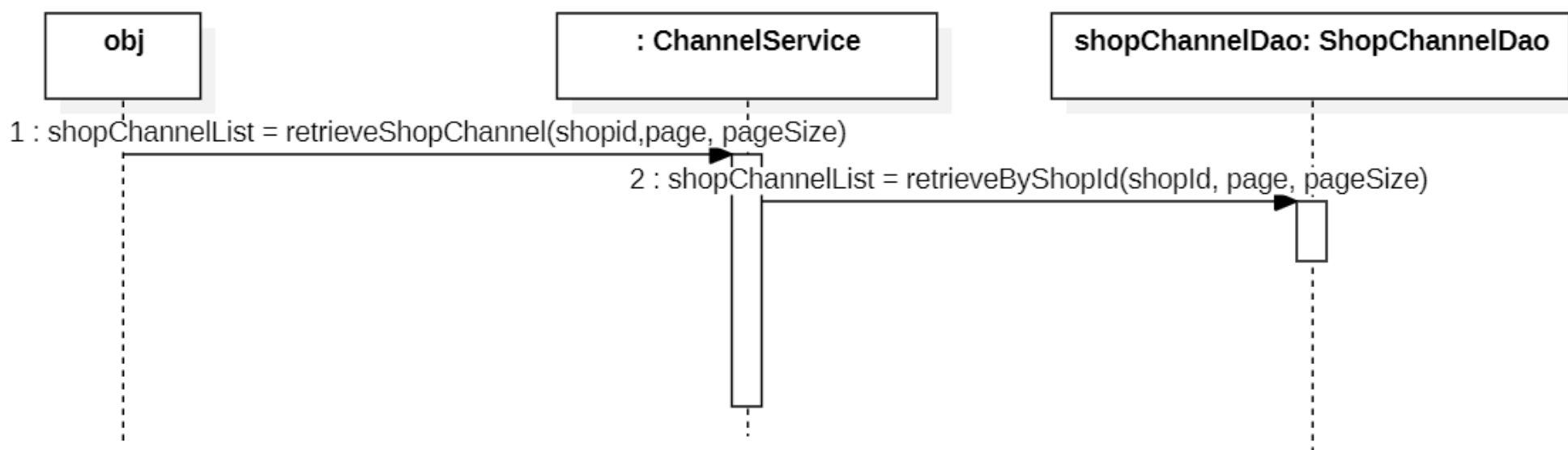
```
{  "errno": 0,  "errmsg": "string",  "data": {    "page": 0,    "pageSize": 0,    "list": [      {        "id": 0,        "subMchid": 0,        "status": 0      }    ]  }}
```

 厦门大学信息学院

60

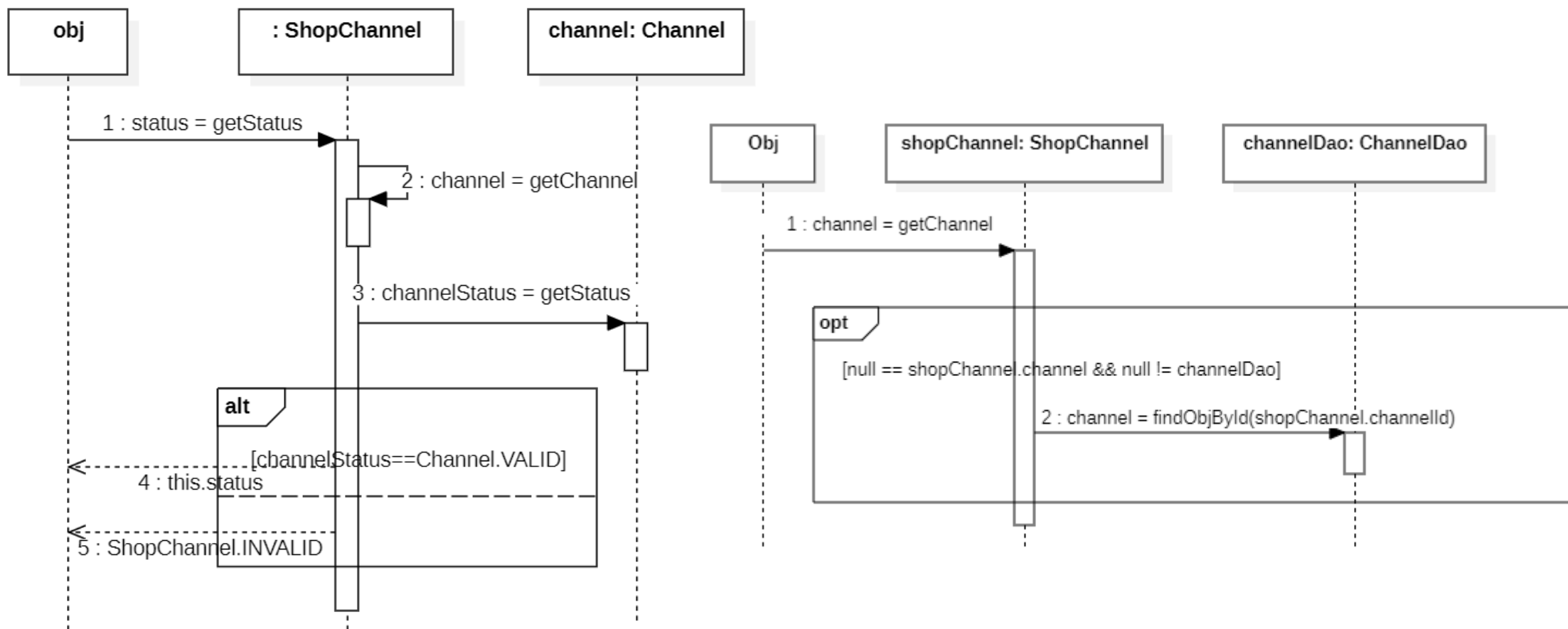
## 4.4 支付模块中创建者和信息专家

### Creator and Information Expert in Payment Module



## 4.4 支付模块中创建者和信息专家

### Creator and Information Expert in Payment Module



# 4.5 多态

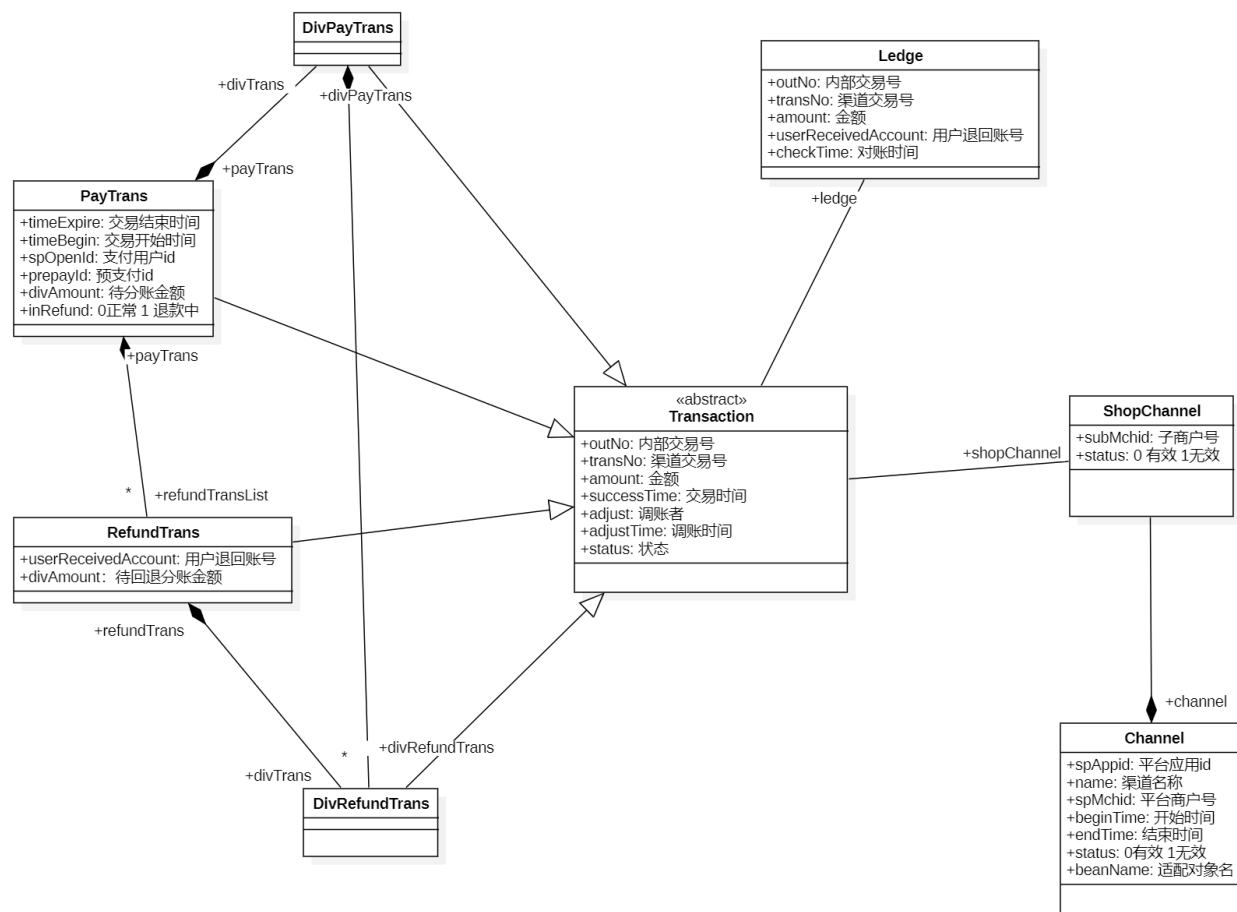
## Polymorphism

- 问题
  - 支付模块中的支付，退款，支付分账和退款分账交易
  - 这些类型有共性的逻辑
  - 如何处理多种类型？
- 方案
  - 使用多态实现不同的类型行为



# 4.5 多态

## Polymorphism

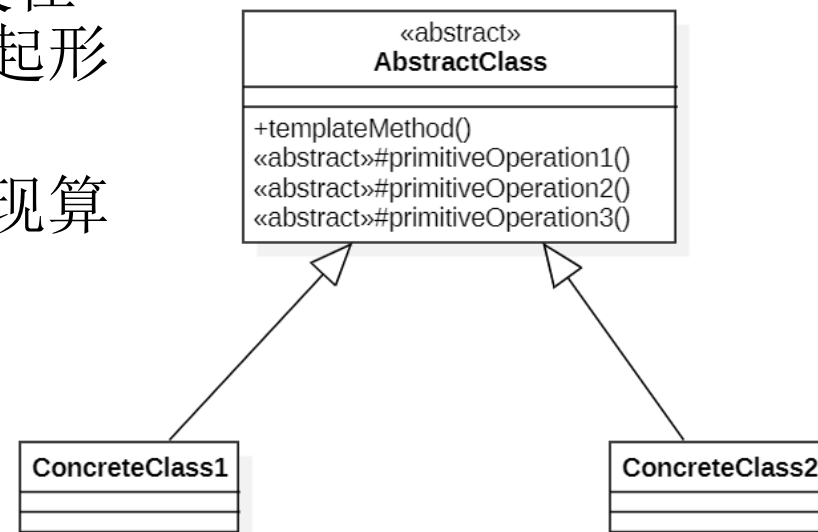




## 4.5 多态

### Polymorphism

- 模板方式设计模式 (Template Method)
  - 模板方法 ( Template Method ) 是定义在抽象类中的、把基本操作方法组合在一起形成一个总算法或一个总行为的方法。
  - 基本方法 ( Primitive Method ) 是实现算法各个步骤的方法。



## 4.6 间接 Indirection

- 问题
  - 支付模块中需要集成多个支付渠道，这些支付渠道会逐步增加，且API会发生变化
  - 如何避免支付模块与这些支付渠道的直接耦合？
- 方案
  - 我不用你，我通过其他对象用你



# 4.6 间接 Indirection



POST

/shops/{shopId}/channels/{id}/shopchannels 签约支付渠道

📋 ↶ ↷

• 建好为无效态

https://app.swaggerhub.com/apis/mingqcn/OOMALL/1.3.2#/payment/createShopChannel

Parameters

Try it out

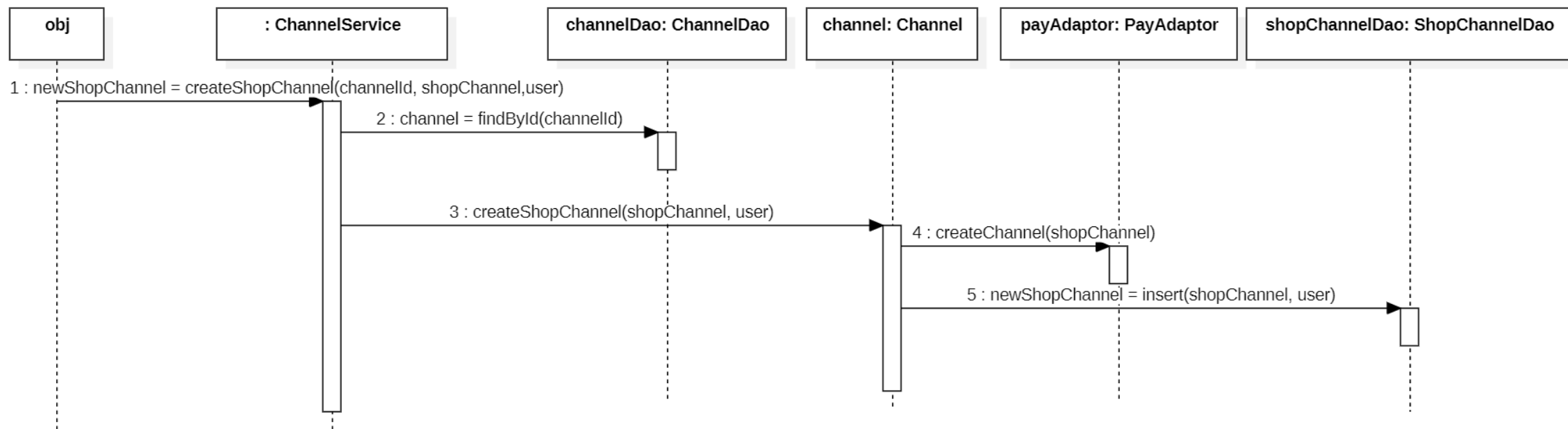
Name	Description
<b>authorization</b> <small>★ required</small> string <i>(header)</i>	用户token <input type="text" value="authorization"/>
<b>shopId</b> <small>★ required</small> integer(\$int64) <i>(path)</i>	店铺id <input type="text" value="shopId"/>
<b>id</b> <small>★ required</small> integer(\$int64) <i>(path)</i>	支付渠道id <input type="text" value="id"/>
<b>body</b> <small>★ required</small> object <i>(body)</i>	支付信息 Example Value   Model <pre>{   "subMchid": "string" }</pre> <div>Parameter content type <input type="text" value="application/json"/></div>

Responses

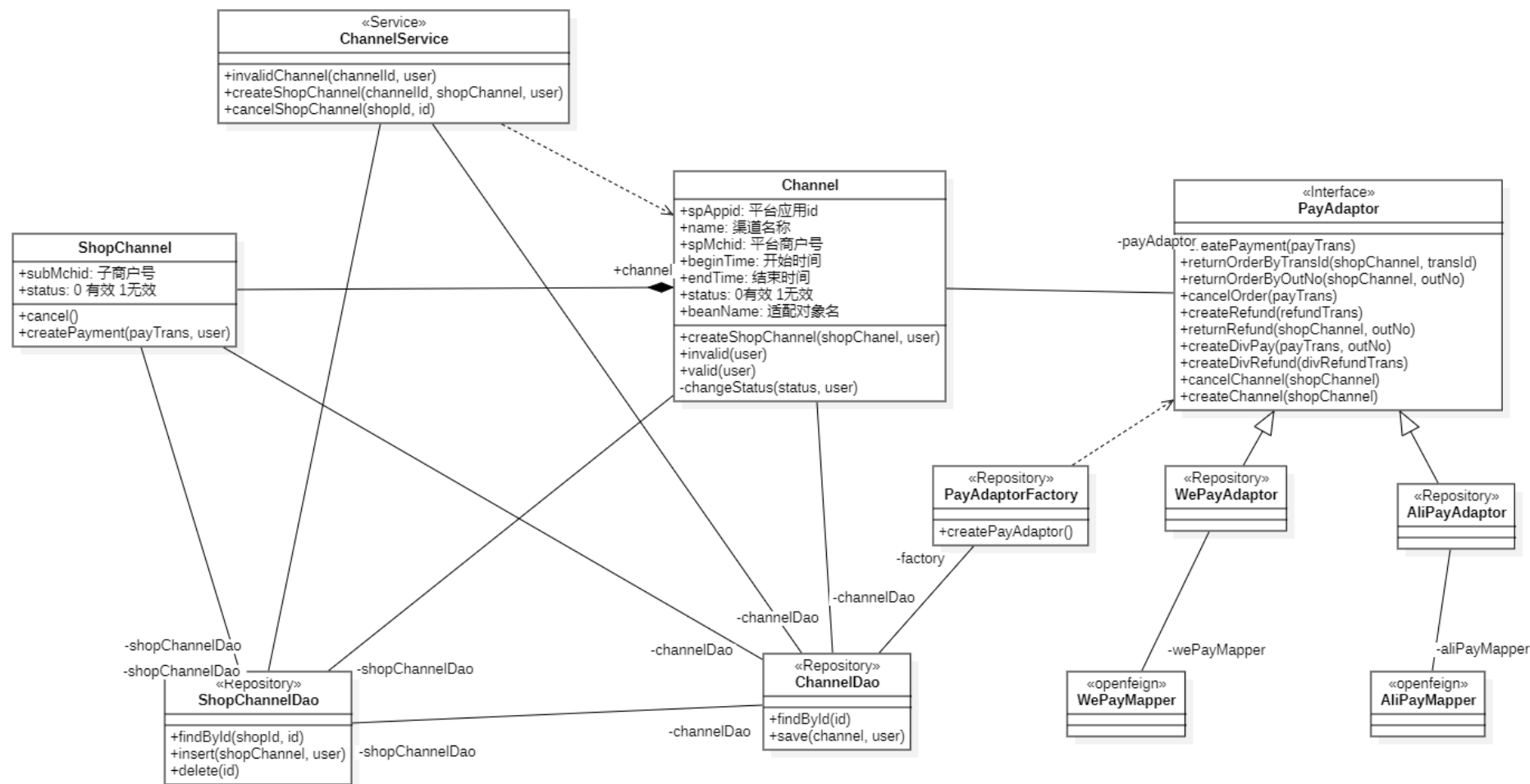
Response content type

Code	Description
103	商铺(id=%d)的支付渠道(id=%d)已经存在
default	成功 Example Value   Model <pre>{   "errno": 0,   "errmsg": "成功",   "data": {     "id": 0,     "subMchid": 0,     "status": 0   } }</pre>

## 4.6 间接 Indirection

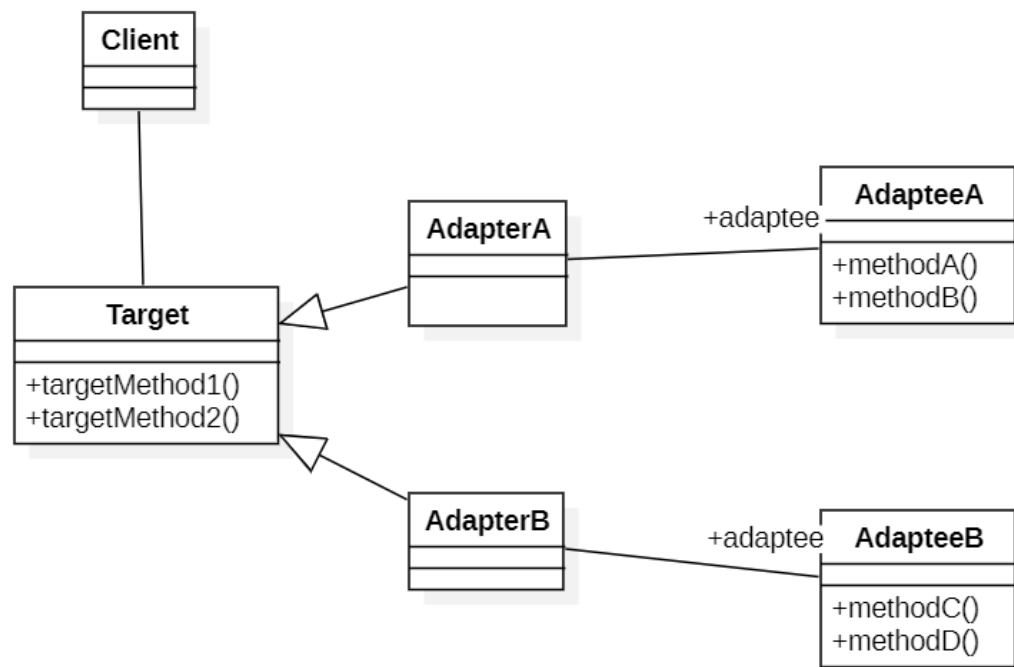


## 4.6 间接 Indirection



## 4.6 间接 Indirection

- 适配器模式 (Adaptor)
  - 将一个类的接口转换成客户希望的另外一个接口。 Adapter模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

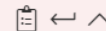


## 4.6 间接 Indirection



DELETE

/shops/{shopId}/shopchannels/{id} 解约店铺的账户



无效状态才能删除

https://app.swaggerhub.com/apis/mingqcn/OOMALL/1.3.2#/payment/cnancelShopChannel

Parameters

Try it out

Name

Description

authorization \* required

用户token

string

(header)

authorization

shopId \* required

店铺id

integer

(path)

shopId

id \* required

店铺账户id

integer

(path)

id

Responses

Response content type

application/json



Code

Description

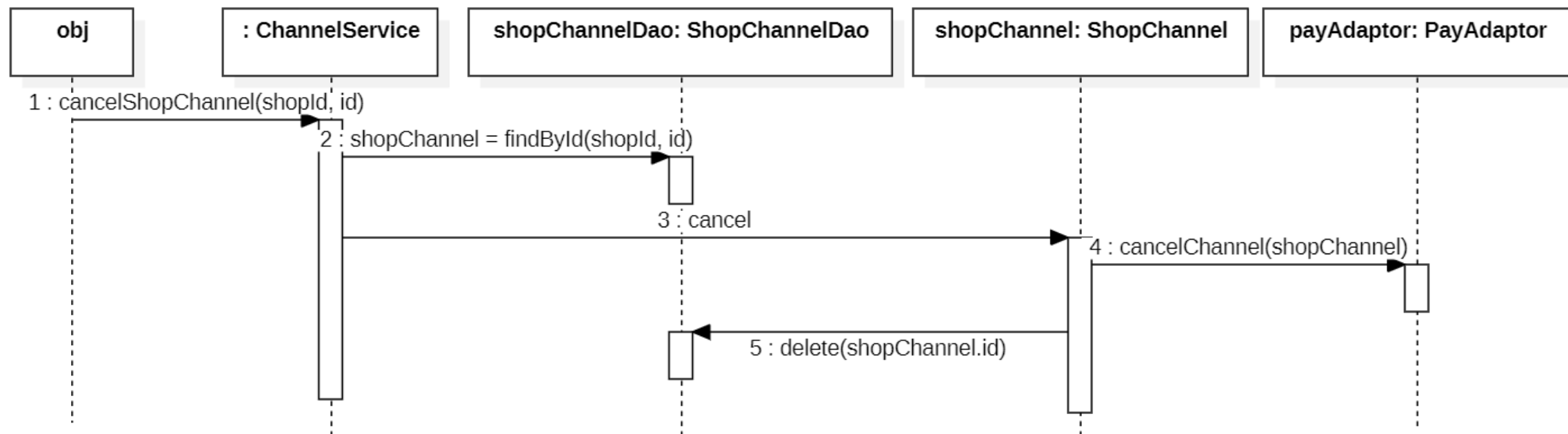
default

成功

Example Value | Model

```
{
  "errno": 0,
  "errmsg": "成功"
}
```

## 4.6 间接 Indirection





## 4.7 虚构

### Pure Fabrication

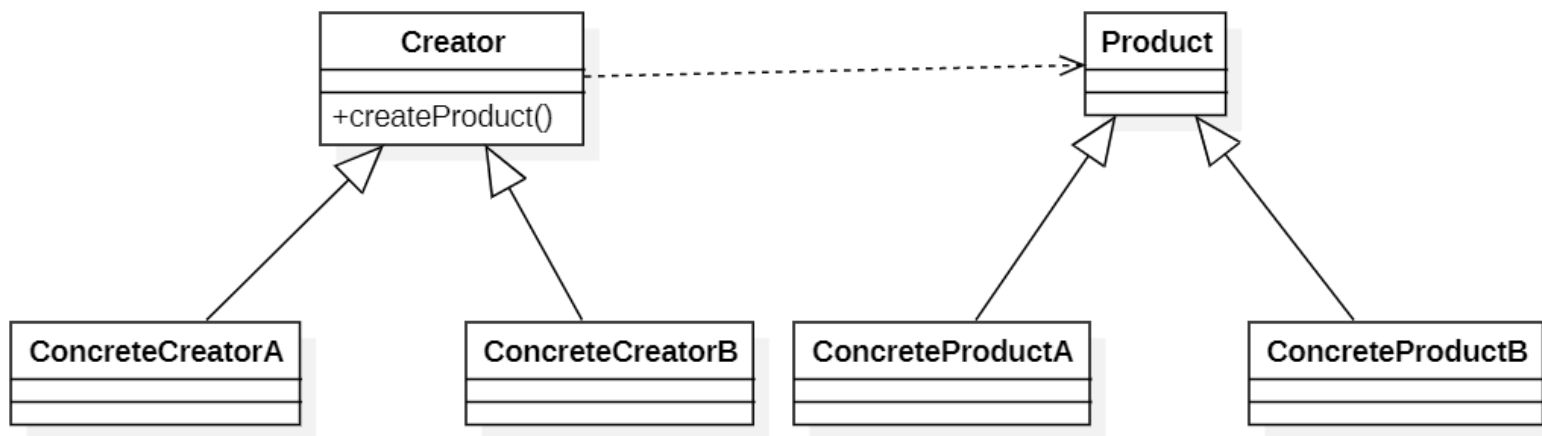
- 问题
  - 如何创建支付渠道的适配器？
  - 当没有两全之策时
- 方案
  - 可以考虑专门虚构一个对象来创建适配器



## 4.7 虚构

### Pure Fabrication

- 工厂方法 (Factory Method)
  - 负责生产一个对象。
  - 定义一个创建对象的接口，让子类决定实例化哪个类，使得一个类的实例化延迟到其子类。



## 4.8 低耦合和高内聚

### Low Coupling and High Cohesion

- 低耦合
  - 问题：如何减轻变更的影响范围？
  - 方案：尽量采用耦合度低的职责分配方案。
- 高内聚
  - 问题：如何让对象易理解，好管理？
  - 方案：尽量采用内聚性高的职责分配方案



# 4.9 软件设计的七大原则

## Seven Principles

- 开闭原则 (Open Closed Principle, OCP)
  - 由勃兰特·梅耶 (Bertrand Meyer) 在 1988 年的著作《面向对象软件构造》 (Object Oriented Software Construction) 中提出:
  - 软件实体应当对扩展开放, 对修改关闭 (Software entities should be open for extension, but closed for modification) 。



# 4.9 软件设计的七大原则

## Seven Principles

- 开闭原则是面向对象程序设计的终极目标，它使软件实体拥有一定的适应性和灵活性的同时具备稳定性和延续性。
  - 对软件测试的影响 -- 软件遵守开闭原则的话，软件测试时只需要对扩展的代码进行测试就可以了，因为原有的测试代码仍然能够正常运行。
  - 可以提高代码的可复用性 -- 粒度越小，被复用的可能性就越大；
  - 可以提高软件的可维护性 -- 遵守开闭原则的软件，其稳定性高和延续性强，从而易于扩展和维护。
- 通过“抽象约束、封装变化”来实现开闭原则
  - 即通过接口或者抽象类为软件实体定义一个相对稳定的抽象层，而将相同的可变因素封装在具体实现类中。



# 4.9 软件设计的七大原则

## Seven Principles

- Liskov替换原则 (Liskov Substitution Principle, LSP)
  - 由麻省理工学院计算机科学实验室的里斯科夫 (Liskov) 女士在 1987 年的“面向对象技术的高峰会议” (OOPSLA) 上发表的一篇文章《数据抽象和层次》 (Data Abstraction and Hierarchy) 里提出
  - 继承必须确保超类所拥有的性质在子类中仍然成立 (Inheritance should ensure that any property proved about supertype objects also holds for subtype objects) 。



# 4.9 软件设计的七大原则

## Seven Principles

- Liskov替换原则是实现开闭原则的重要方式之一。
  - 它克服了继承中重写父类造成的可复用性变差的缺点。
  - 它是动作正确性的保证。即类的扩展不会给已有的系统引入新的错误，降低了代码出错的可能性。
  - 加强程序的健壮性，同时变更时可以做到非常好的兼容性，提高程序的维护性、可扩展性，降低需求变更时引入的风险。



# 4.9 软件设计的七大原则

## Seven Principles

- 依赖倒置原则 (Dependence Inversion Principle, DIP)
  - 由Object Mentor 公司总裁罗伯特·马丁 (Robert C.Martin) 于 1996 年在 C++ Report 上发表的文章中提出：高层模块不应该依赖低层模块，两者都应该依赖其抽象；抽象不应该依赖细节，细节应该依赖抽象  
(High level modules should not depend upon low level modules. Both should depend upon abstractions. Abstractions should not depend upon details. Details should depend upon abstractions)。
  - 其核心思想是：要面向接口编程，不要面向实现编程。





# 4.9 软件设计的七大原则

## Seven Principles

- 单一职责原则 (Single Responsibility Principle, SRP)
  - 由罗伯特·C. 马丁 (Robert C. Martin) 于《敏捷软件开发：原则、模式和实践》一书中提出的。
  - 单一职责原则规定一个类应该有且仅有一个引起它变化的原因，否则类应该被拆分 (There should never be more than one reason for a class to change)



# 4.9 软件设计的七大原则

## Seven Principles

- 单一职责原则的核心就是控制类的粒度大小、将对象解耦、提高其内聚性。如果遵循单一职责原则将有以下优点。
  - 降低类的复杂度。一个类只负责一项职责，其逻辑肯定要比负责多项职责简单得多。
  - 提高类的可读性。复杂性降低，自然其可读性会提高。
  - 提高系统的可维护性。可读性提高，那自然更容易维护了。
  - 变更引起的风险降低。变更是必然的，如果单一职责原则遵守得好，当修改一个功能时，可以显著降低对其他功能的影响。



# 4.9 软件设计的七大原则

## Seven Principles

- 接口隔离原则（Interface Segregation Principle, ISP）
  - 客户端不应该被迫依赖于它不使用的方法（Clients should not be forced to depend on methods they do not use）。
  - 一个类对另一个类的依赖应该建立在最小的接口上（The dependency of one class to another one should depend on the smallest possible interface）。



# 4.9 软件设计的七大原则

## Seven Principles

- 迪米特法则 (Law of Demeter, LoD) / 最少知识原则 (Least Knowledge Principle, LKP)
  - 由伊恩·荷兰 (Ian Holland) 提出，被 UML 创始者之一的布奇 (Booch) 普及，后来又因为在经典著作《程序员修炼之道》 (The Pragmatic Programmer) 提及而广为人知。
  - 只与你的直接朋友交谈，不跟“陌生人”说话 (Talk only to your immediate friends and not to strangers)。
  - 如果两个软件实体无须直接通信，那么就不应当发生直接的相互调用，可以通过第三方转发该调用。其目的是降低类之间的耦合度，提高模块的相对独立性。



# 4.9 软件设计的七大原则

## Seven Principles

- 合成复用原则 (Composite Reuse Principle, CRP) /组合/聚合  
复用原则 (Composition/Aggregate Reuse Principle, CARP)
  - 在软件复用时，要尽量先使用组合或者聚合等关联关系来实现，其次才考虑使用继承关系来实现。



# 4.9 软件设计的七大原则

## Seven Principles

设计原则	内容	目的
开闭原则	对扩展开放，对修改关闭	降低维护带来的新风险
依赖倒置原则	高层不应该依赖低层，要面向接口编程	更利于代码结构的升级扩展
单一职责原则	一个类只干一件事，实现类要单一	便于理解，提高代码的可读性
接口隔离原则	一个接口只干一件事，接口要精简单一	功能解耦，高聚合、低耦合
迪米特法则	不该知道的不要知道，一个类应该保持对其它对象最少的了解，降低耦合度	只和朋友交流，不和陌生人说话，减少代码臃肿
Liskov 替换原则	不要破坏继承体系，子类重写方法功能发生改变，不应该影响父类方法的含义	防止继承泛滥
合成复用原则	尽量使用组合或者聚合关系实现代码复用，少使用继承	降低代码耦合



# 4.10 支付模块中的例子

## Examples in Payment Module

<https://app.swaggerhub.com/apis/mingqcn/OOMALL/1.3.2#/payment/createPayment>



POST

/internal/shopchannels/{id}/payments 创建支付单

• 此API为模拟API, 即时返回支付成功, 生成paysn

Parameters

Try it out

Name	Description
<b>authorization</b> <span>★ required</span> string (header)	用户token <div>authorization</div>
<b>id</b> <span>★ required</span> integer (path)	商户支付渠道id <div>id</div>
<b>body</b> <span>★ required</span> object (body)	支付信息 Example Value   Model <div><pre>{  "outNo": "string",  "description": "string",  "timeExpire": "string",  "timeBegin": "string",  "amount": 0,  "divAmount": 0}</pre></div>

Parameter content type

application/json

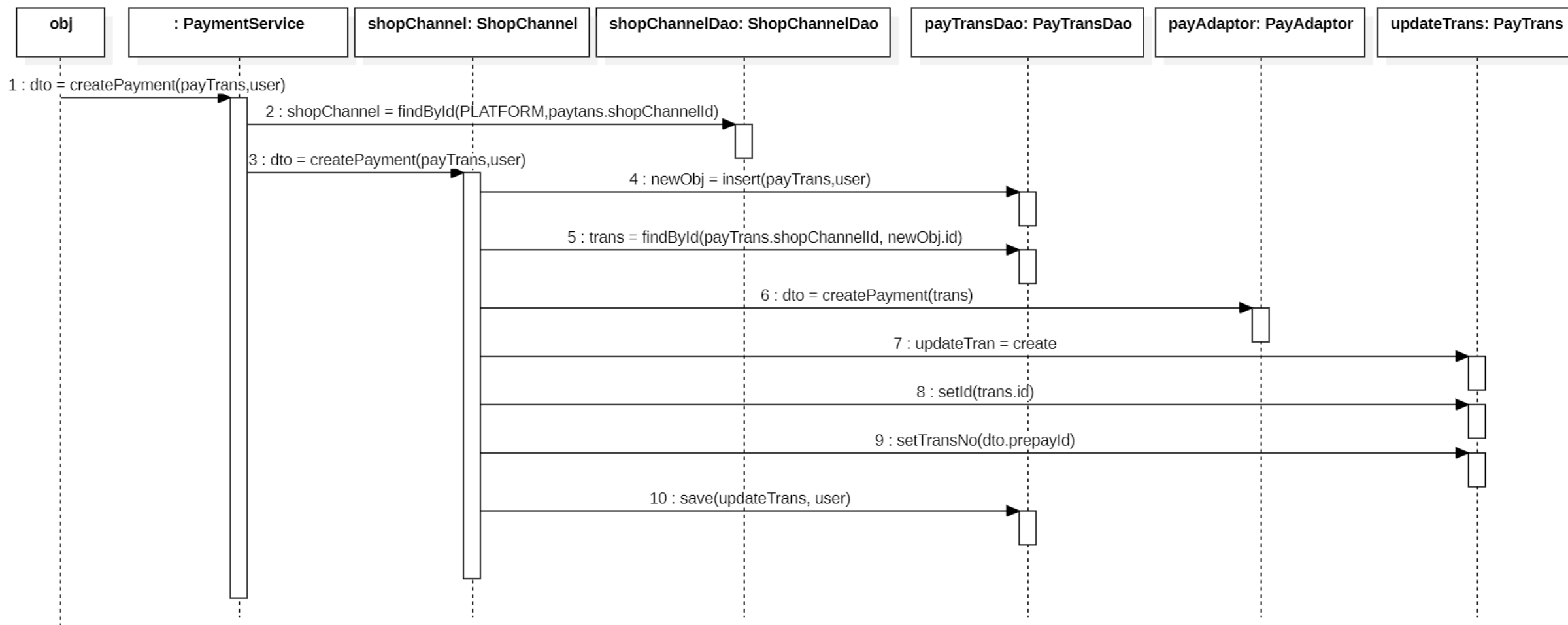
Responses

Response content type application/json

Code	Description
default	成功 Example Value   Model <div><pre>{  "errno": 0,  "errmsg": "成功",  "data": {    "outTradeNo": "string",    "prepayId": "string",    "totalAmount": 0,    "sellerId": "string",    "merchantOrderNo": "string"  }}</pre></div>

## 4.10 支付模块中的例子

### Examples in Payment Module





## 4.10 支付模块中的例子

<https://app.swaggerhub.com/apis/mingqcn/OOMALL/1.3.2#/payment/getPayment>

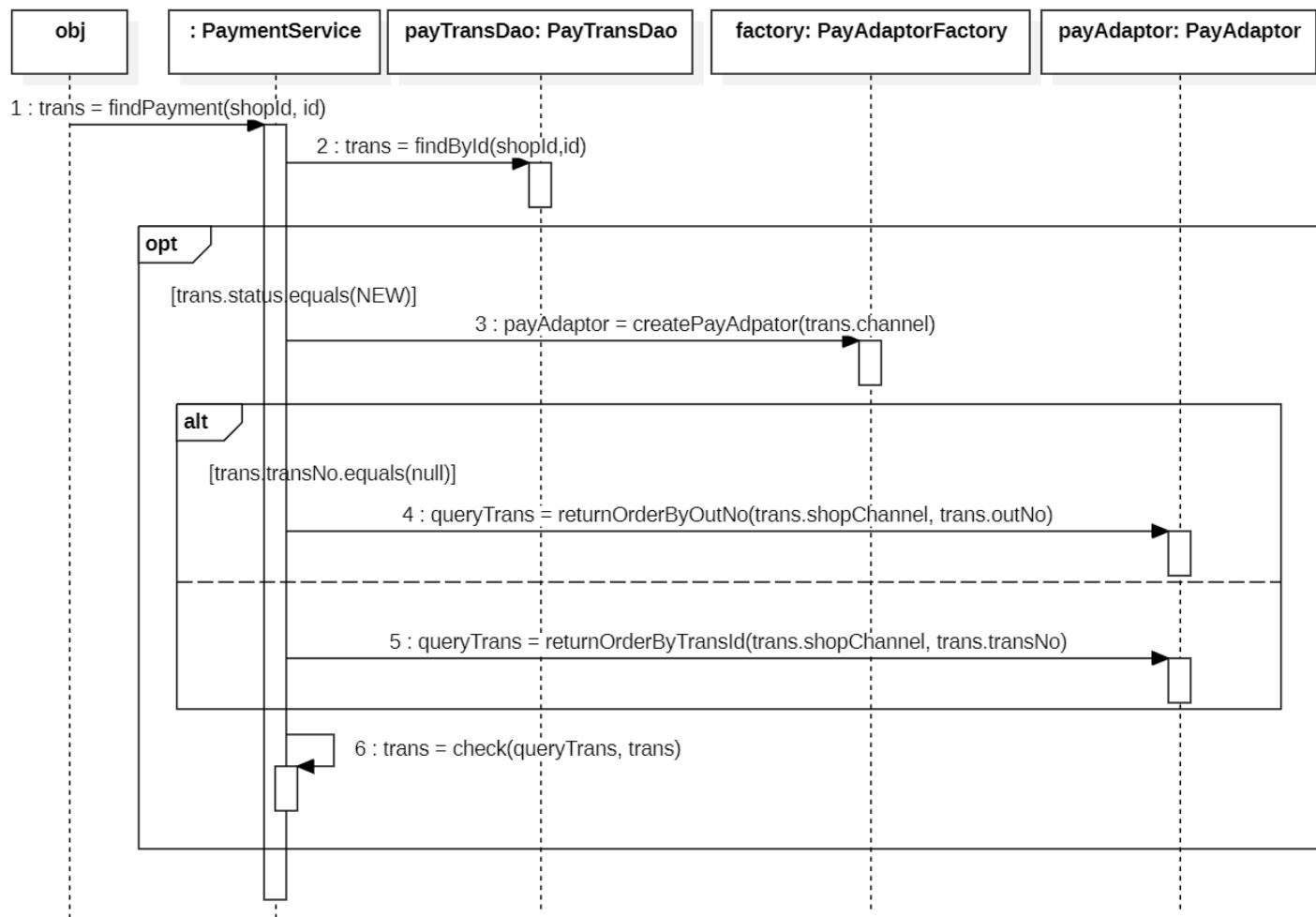


厦门大学信息学院

GET /shops/{shopId}/payments/{id} 查看支付单的详情信息									
如果状态不是最终状态，需要查询渠道api获取最新状态									
Parameters Try it out									
<table><thead><tr><th>Name</th><th>Description</th></tr></thead><tbody><tr><td><b>authorization</b> * required string (header)</td><td>用户token authorization</td></tr><tr><td><b>shopId</b> * required integer(\$int64) (path)</td><td>店铺id shopId</td></tr><tr><td><b>id</b> * required integer(\$int64) (path)</td><td>支付单id id</td></tr></tbody></table>		Name	Description	<b>authorization</b> * required string (header)	用户token authorization	<b>shopId</b> * required integer(\$int64) (path)	店铺id shopId	<b>id</b> * required integer(\$int64) (path)	支付单id id
Name	Description								
<b>authorization</b> * required string (header)	用户token authorization								
<b>shopId</b> * required integer(\$int64) (path)	店铺id shopId								
<b>id</b> * required integer(\$int64) (path)	支付单id id								
Responses Response content type application/json									
<table><thead><tr><th>Code</th><th>Description</th></tr></thead><tbody><tr><td>default</td><td>成功</td></tr></tbody></table> <div>Example Value   Model</div> <pre>{  "schema": {    "errno": 0,    "errmsg": "string",    "data": {      "id": 0,      "outNo": "string",      "transNo": "string",      "amount": 0,      "divAmount": 0,      "successTime": "string",      "prepayId": "string",      "inRefund": 0,      "channel": {        "id": 0,        "name": "string"      },      "status": 0,      "timeBegin": "string",      "timeExpire": "string",      "creator": {        "id": 0,        "name": "string"      }    }  }</pre>		Code	Description	default	成功				
Code	Description								
default	成功								

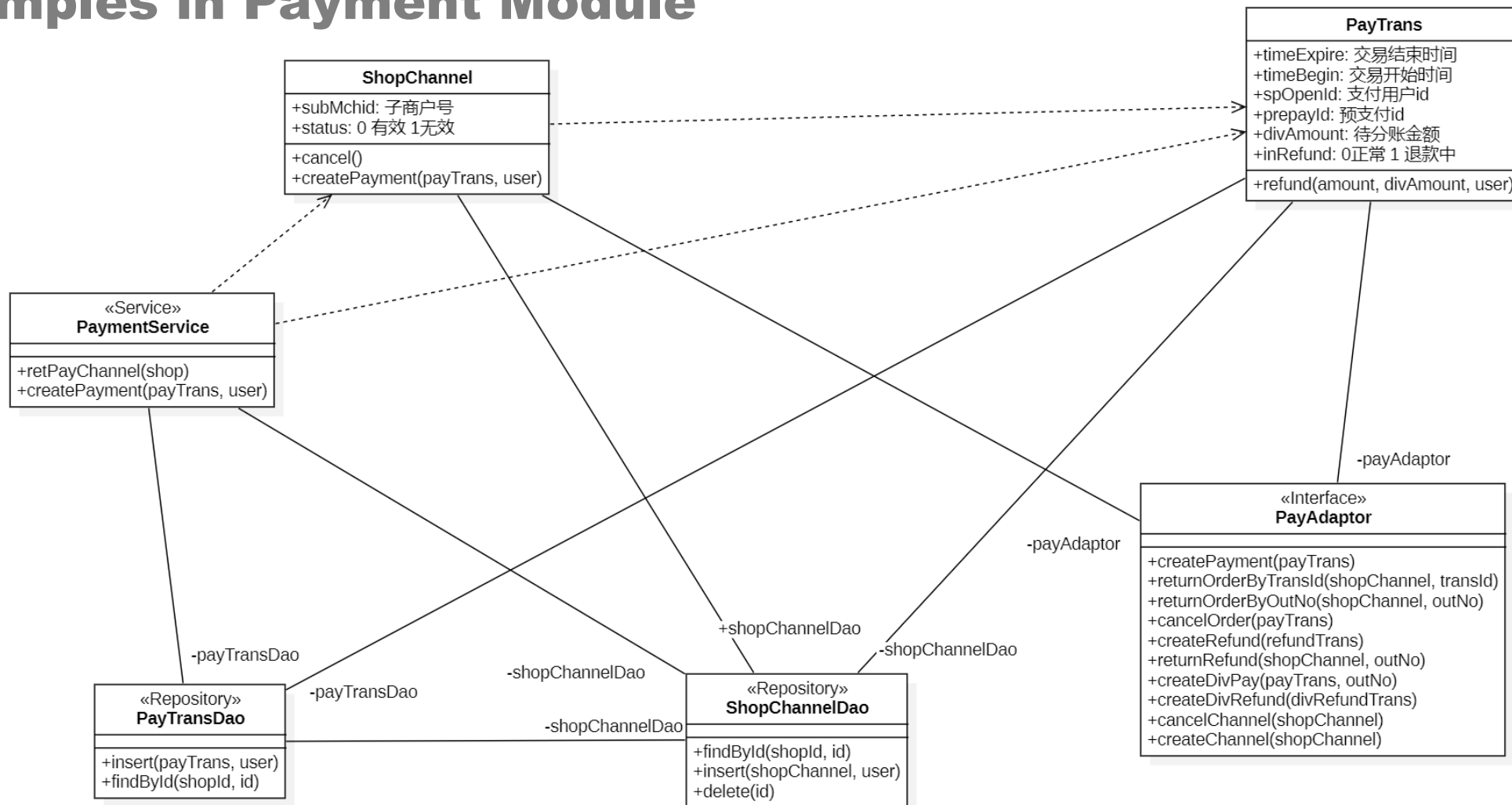
## 4.10 支付模块中的例子

### Examples in Payment Module



## 4.10 支付模块中的例子

### Examples in Payment Module



# 4.10 支付模块中的例子

## Examples in Payment Module

<https://app.swaggerhub.com/apis/mingqcn/OOMALL/1.3.2#/payment/createRefund>

POST

/internal/shops/{shopId}/payments/{id}/refunds 管理员创建退款信息，需检查Payment是否是此商铺的payment

• 此API为模拟API，即时返回支付成功，生成paysn

Parameters

Try it out

Name	Description
<b>authorization</b> * required string (header)	用户token <input type="text" value="authorization"/>
<b>shopId</b> * required integer (\$int64) (path)	店铺id <input type="text" value="shopId"/>
<b>id</b> * required integer (path)	支付id <input type="text" value="id"/>
<b>body</b> * required object (body)	退款金额 Example Value   Model <pre>{  "amount": 0,  "divAmount": 0}</pre> Parameter content type <div>application/json</div>

Responses

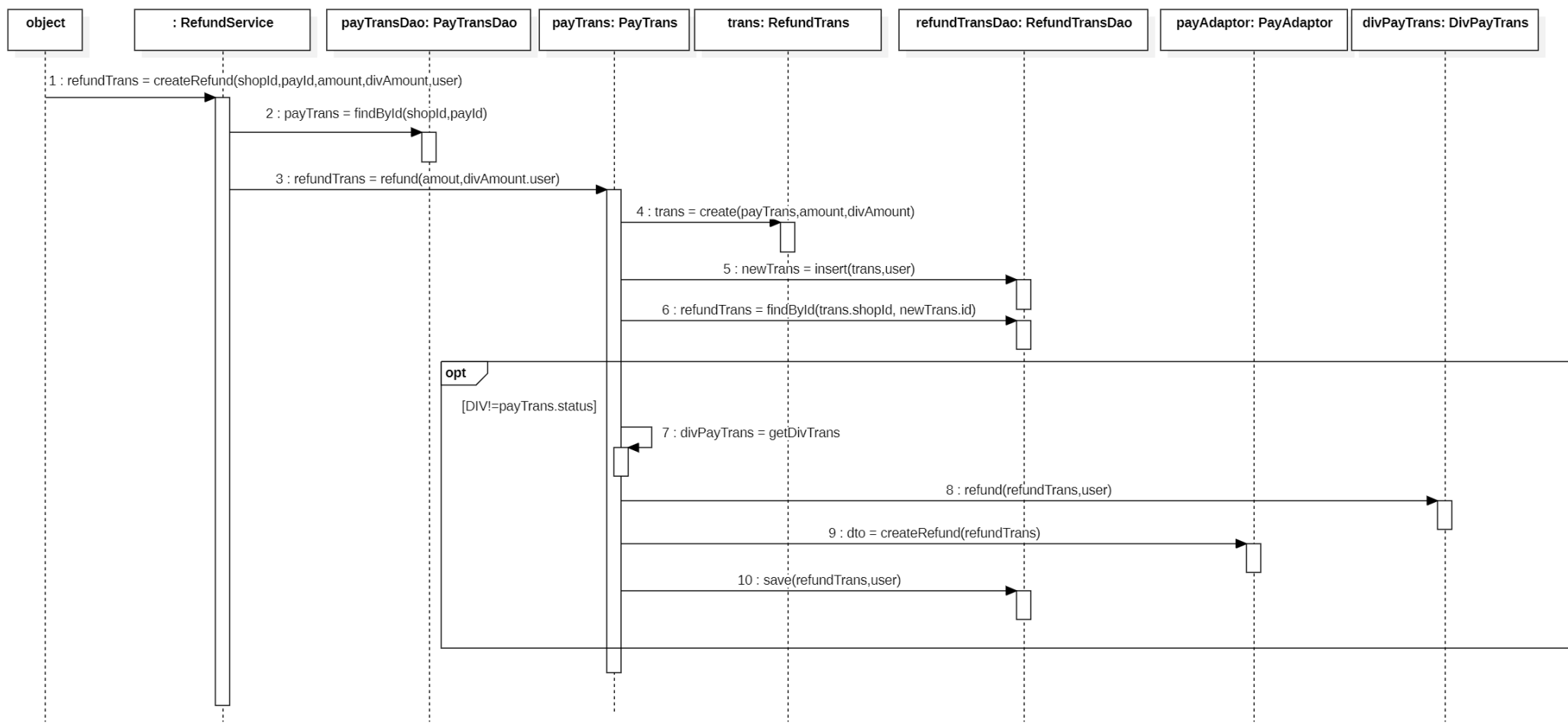
Response content type application/json

Code	Description
default	Example Value   Model <pre>{  "errno": 0,  "errmsg": "成功",  "data": {    "id": 0,    "outNo": "string",    "transNo": "string",    "amount": 0,    "successTime": "string",    "channel": {      "id": 0,      "name": "string"    },  },  "status": 0,  "adjustor": {    "id": 0,    "name": "string"  },  "adjustTime": "string",  "ledger": {    "id": 0,    "outNo": "string",</pre>



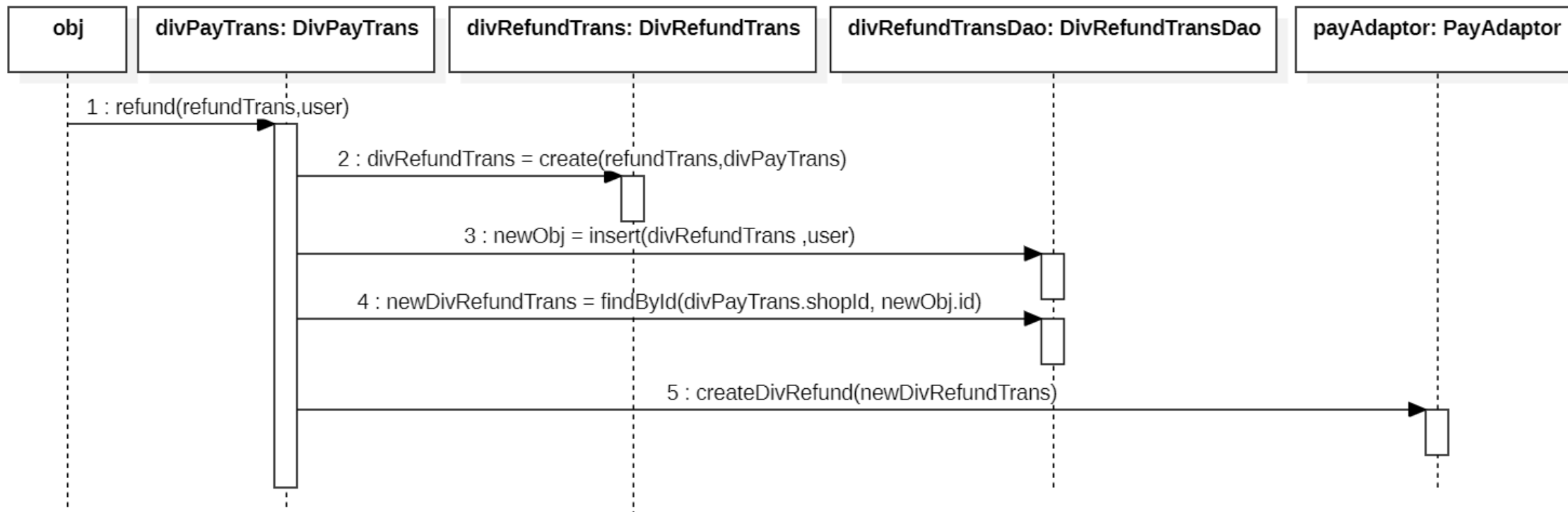
## 4.10 支付模块中的例子

### Examples in Payment Module



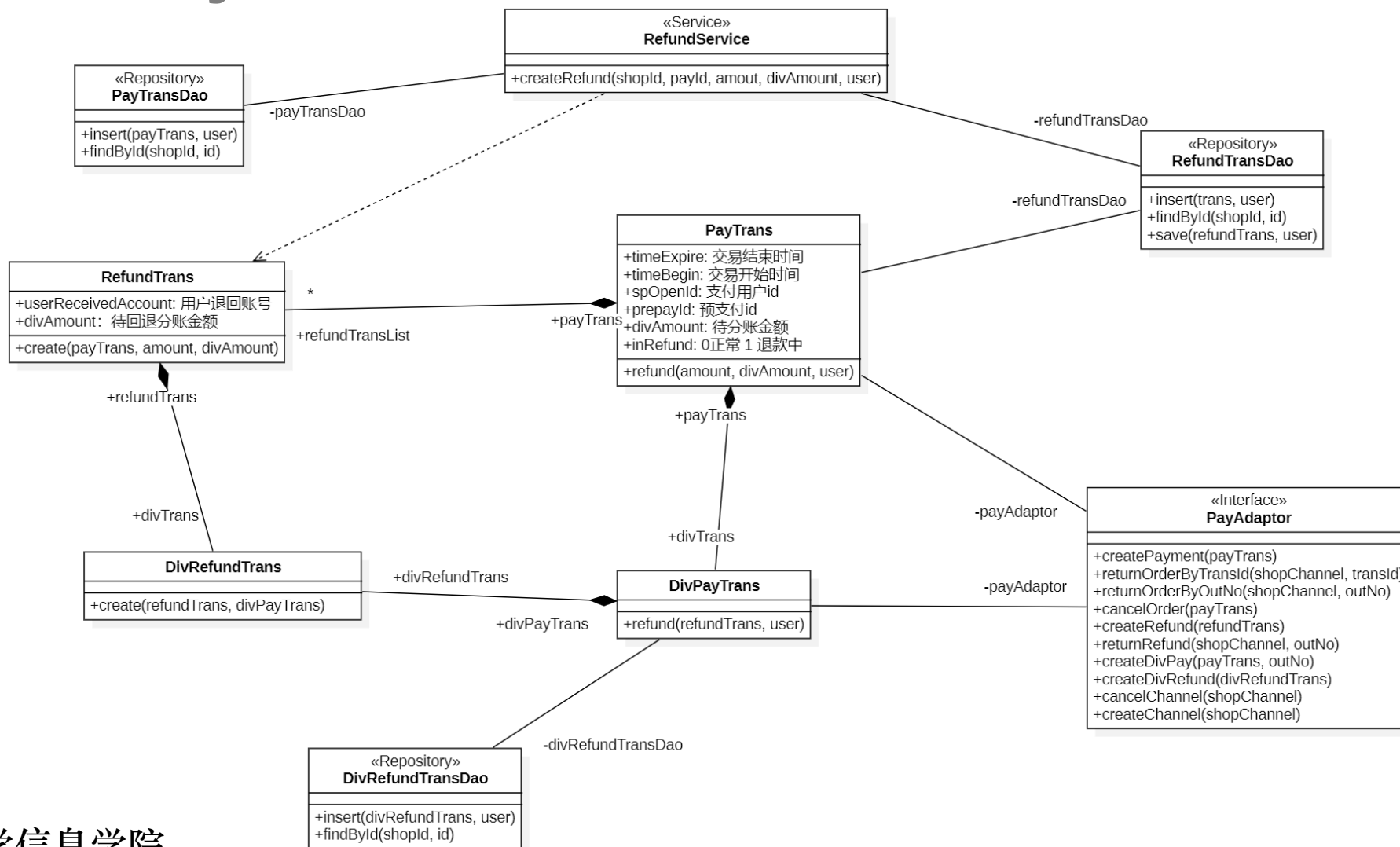
## 4.10 支付模块中的例子

### Examples in Payment Module



## 4.10 支付模块中的例子

### Examples in Payment Module



# 4.10 支付模块中的例子

## Examples in Payment Module

