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

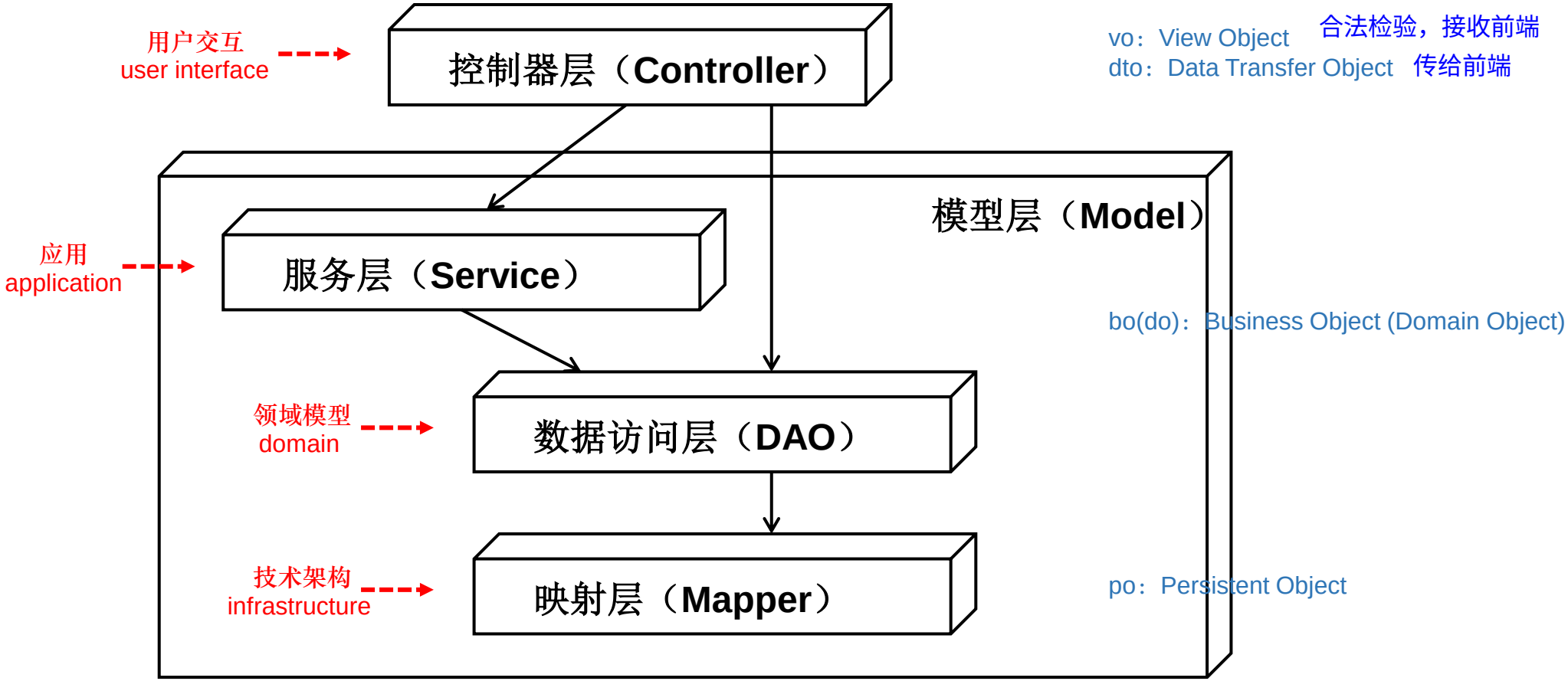
1.1 软件设计的目标
Software Design Goals

- 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

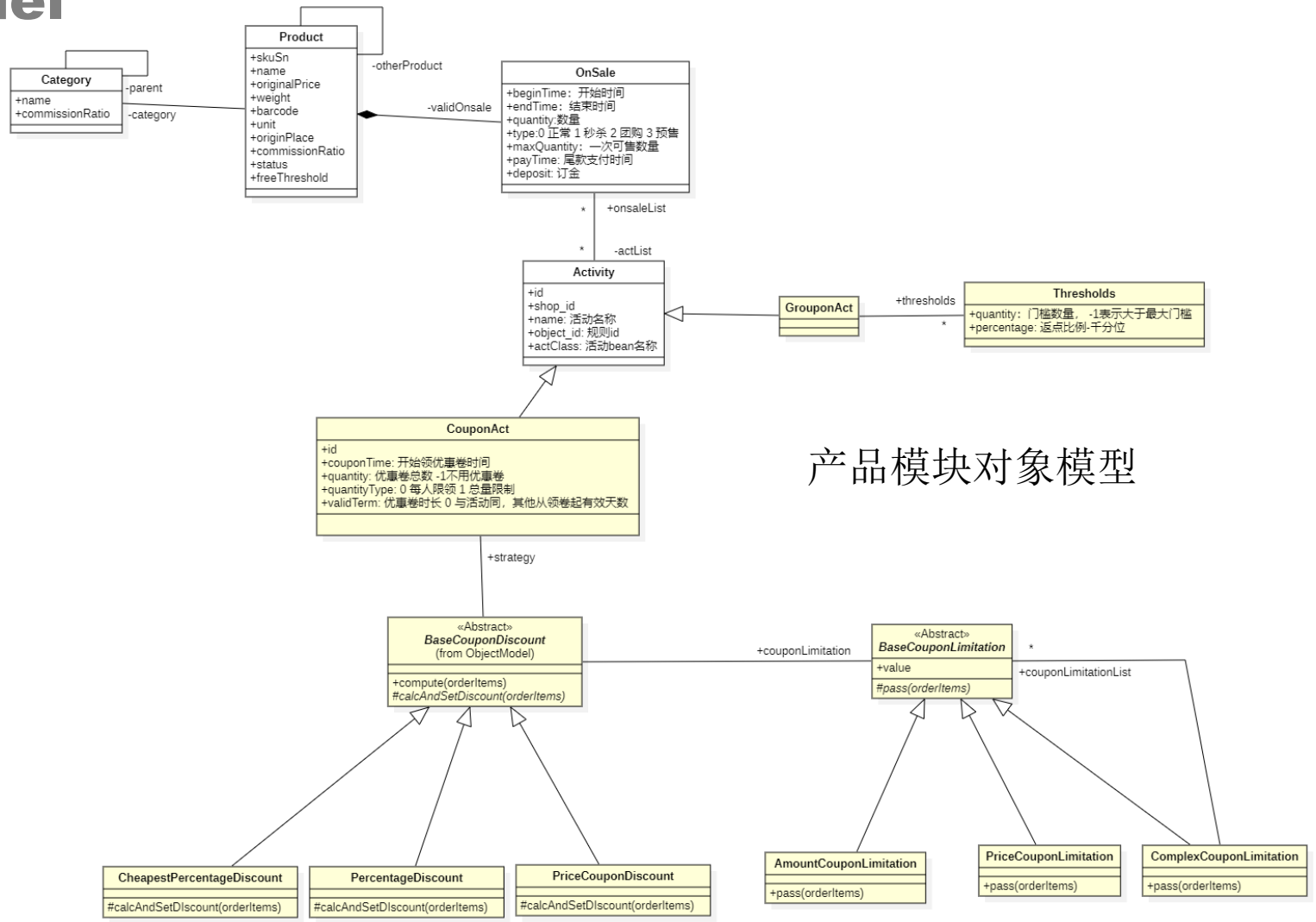
我们虽然无法找到一个绝对理想的圆，但这并不妨碍我们理解什么是圆。

--- 源自罗翔的《圆圈正义》

1.2 对象模型
Object Model

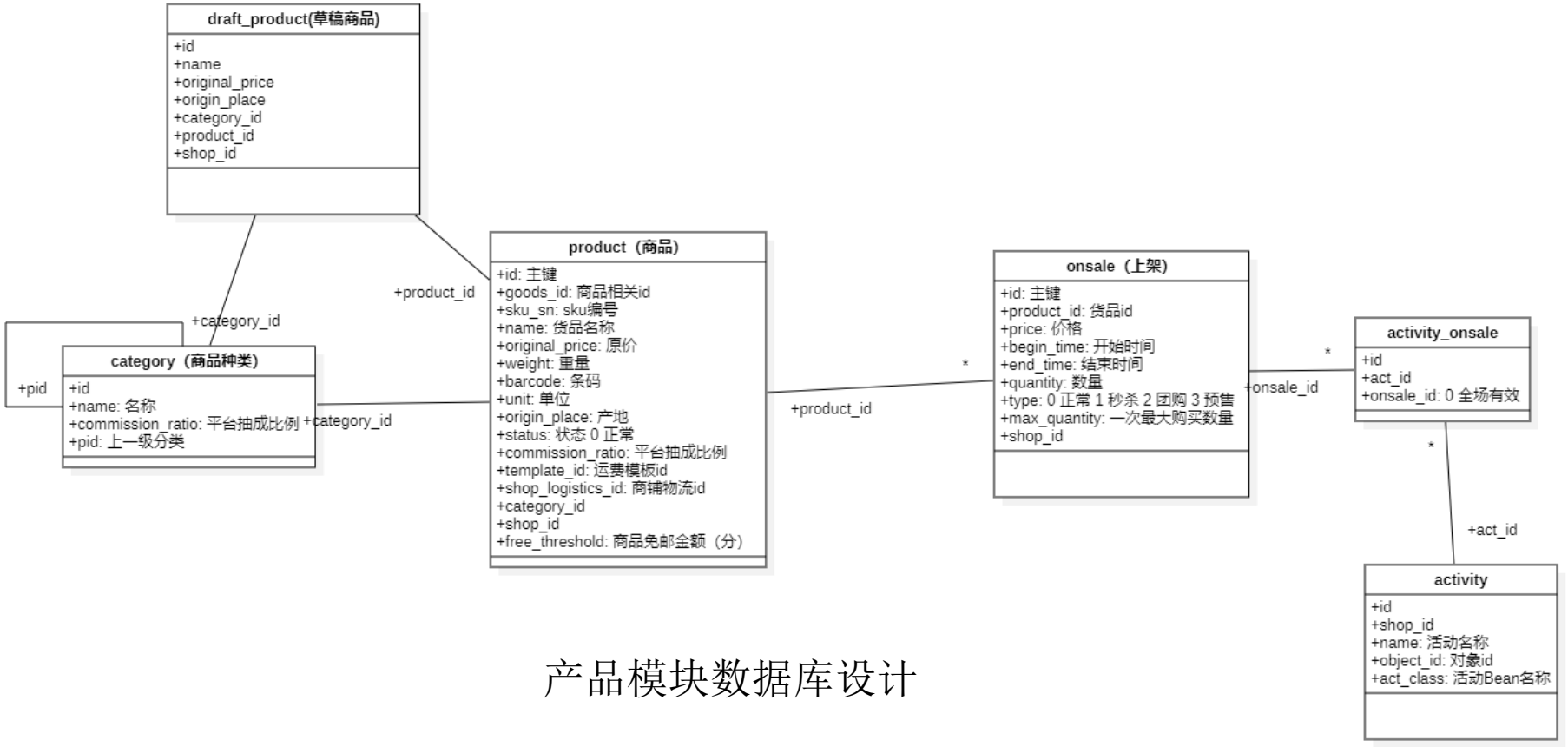


1.2 对象模型
Object Model



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

service层

- 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

DAO层

- 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);      找到团购活动、确定是不是自己店铺
        List<String> keys = activity.cancel(user);          取消
        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) : "ActitvityDao.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;
    }
}
```

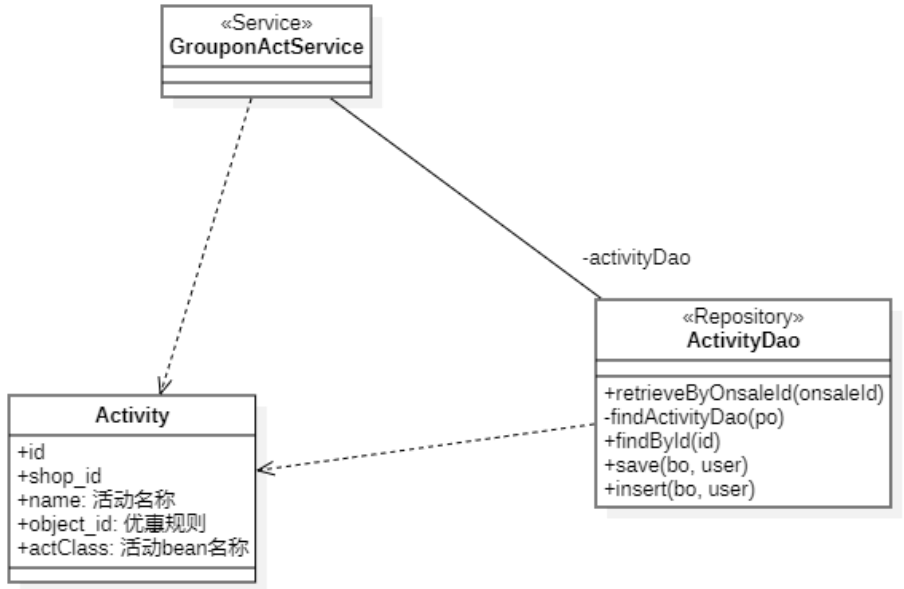
1.2 对象模型
Object Model

```
@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

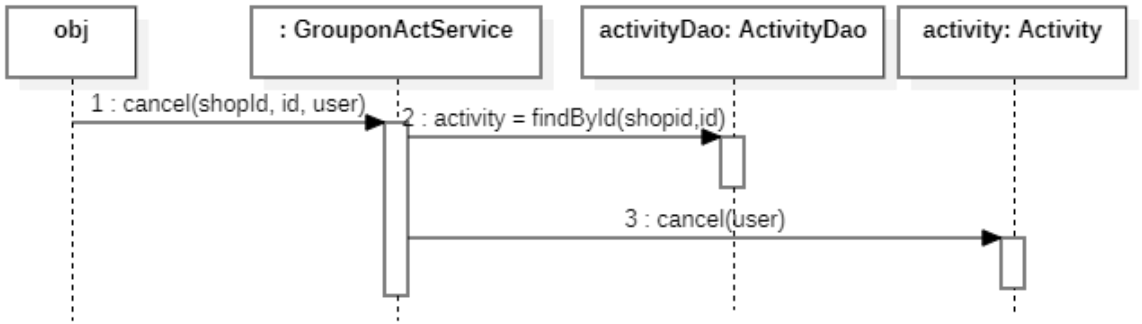
- 动态建模
  - 思考对象交互的细节，描述实现逻辑的方式。
  - 是真正面向对象设计的细节。 “rubber hits the road”

2. UML交互图

Interaction Diagram

2.1顺序图和通信图
Sequence Diagram and Communication Diagram

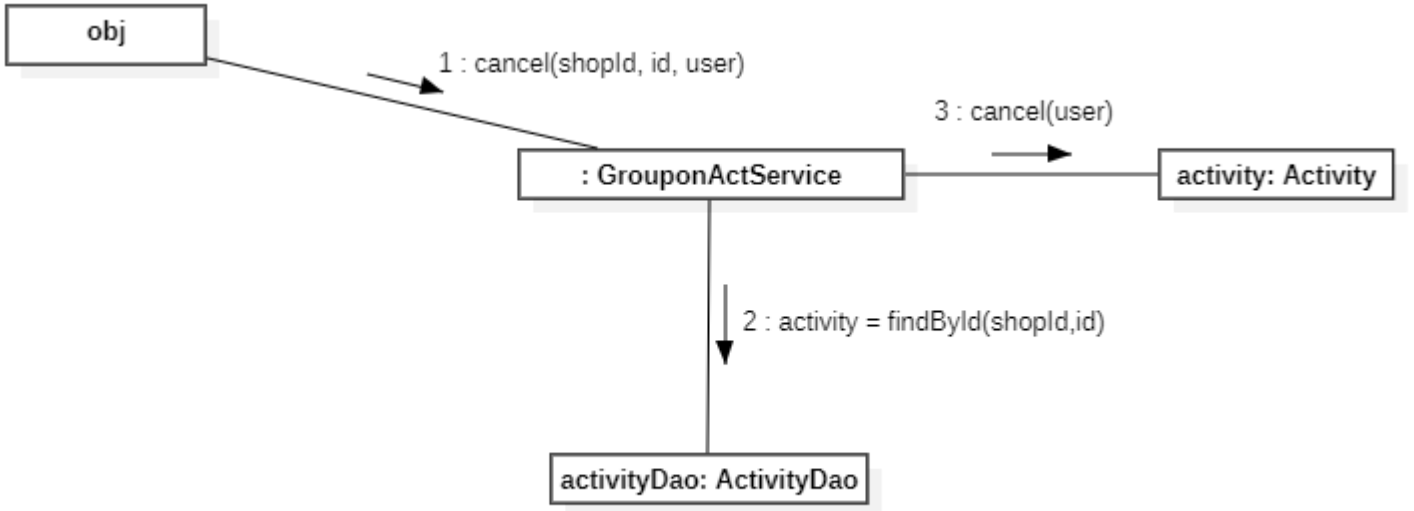
- 顺序图（Sequence Diagram）
  - 用于藩篱格式描述对象之间的交互




```
@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()]));
    }
}
```

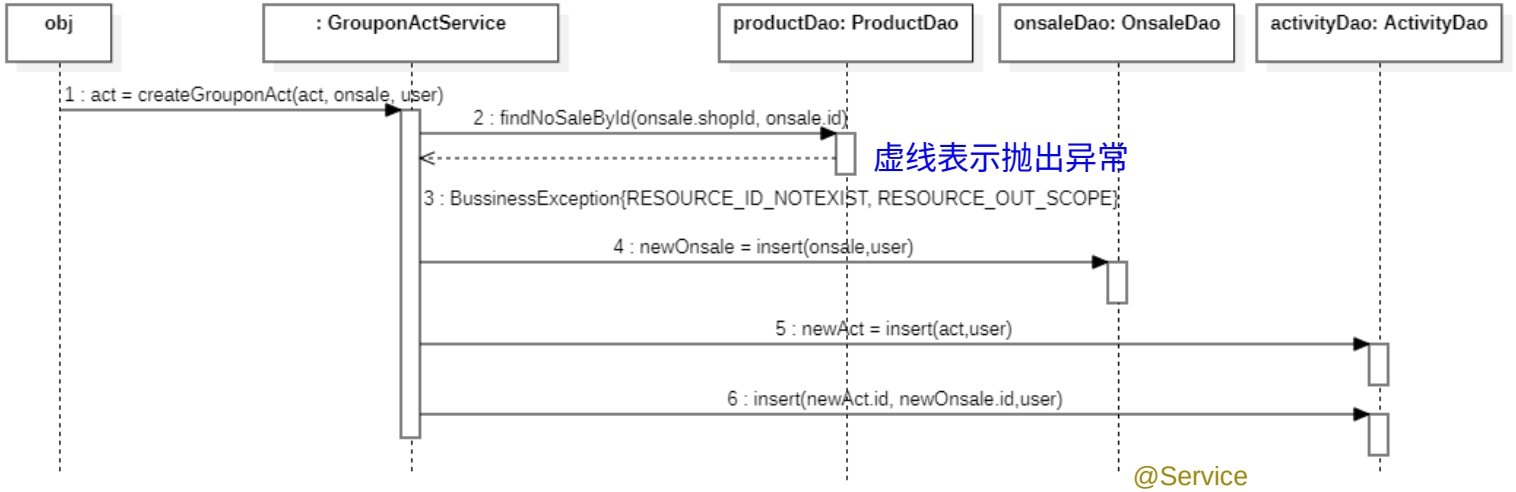
2.1顺序图和通信图
Sequence Diagram and Communication Diagram

- 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

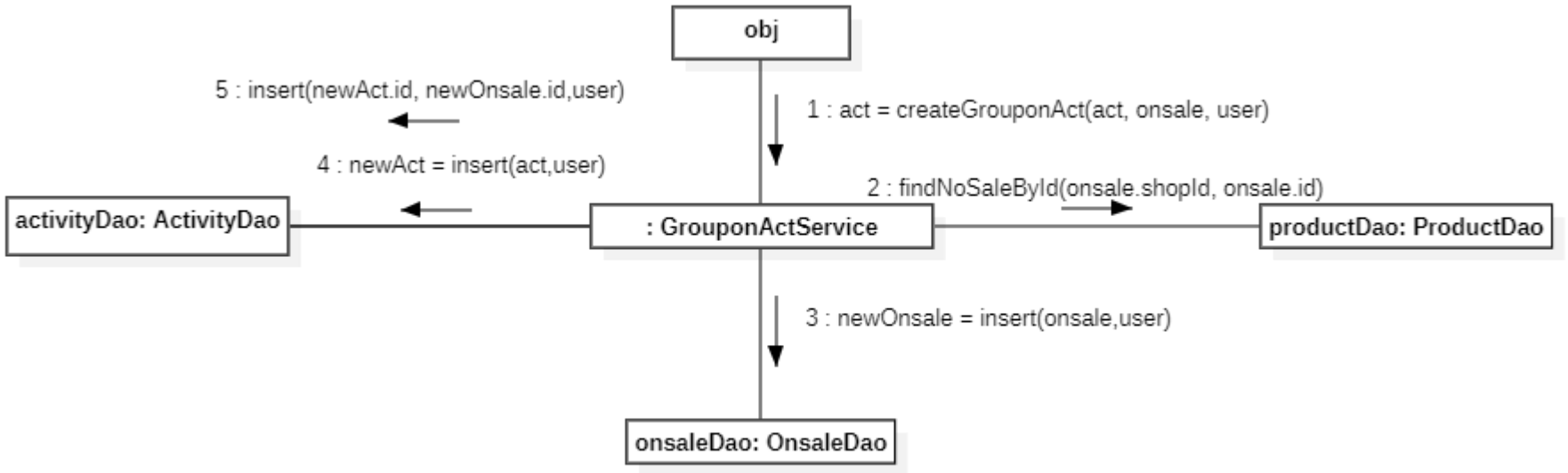
- 顺序图（Sequence Diagram）




```
@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）



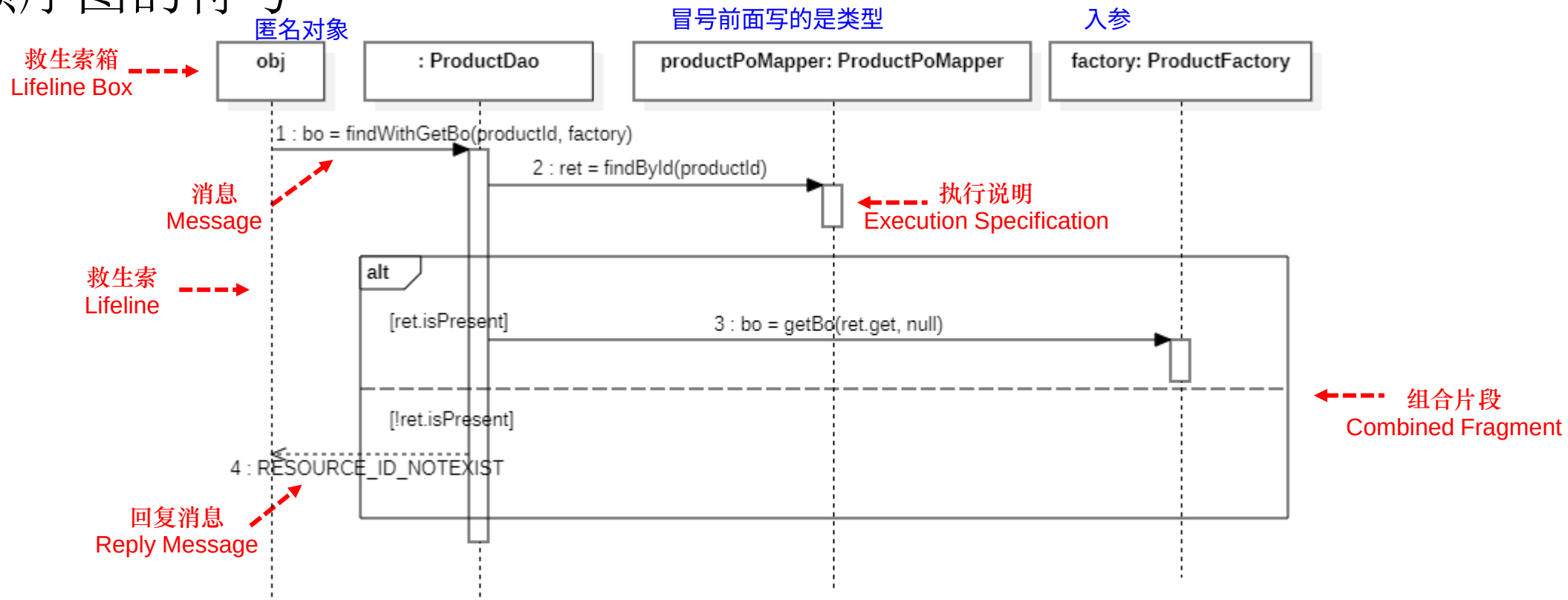
2.1顺序图和通信图
Sequence Diagram and Communication Diagram

- 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

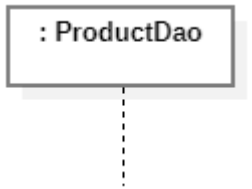
2.2 顺序图
Sequence Diagram

- 顺序图的符号

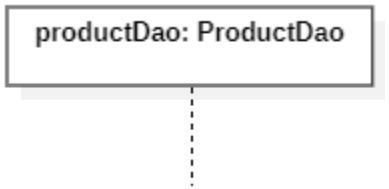


2.2 顺序图
Sequence Diagram

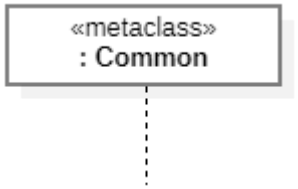
- 用Lifeline Box描述对象



匿名ProductDao对象



变量名为productDao的ProductDao对象



static也行

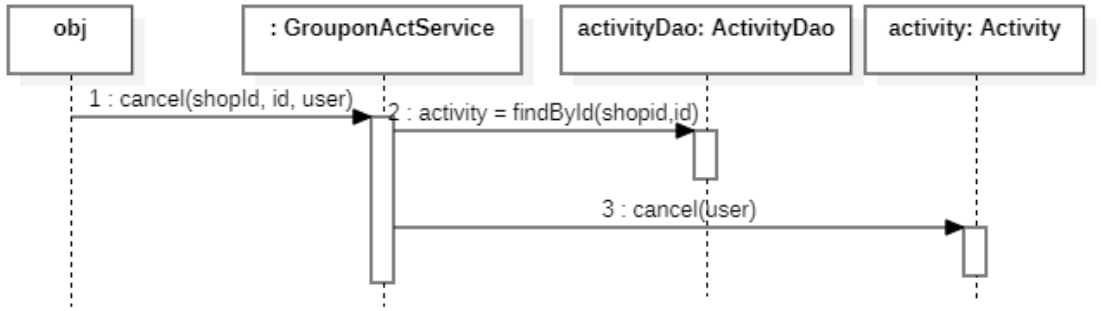
Common的静态类

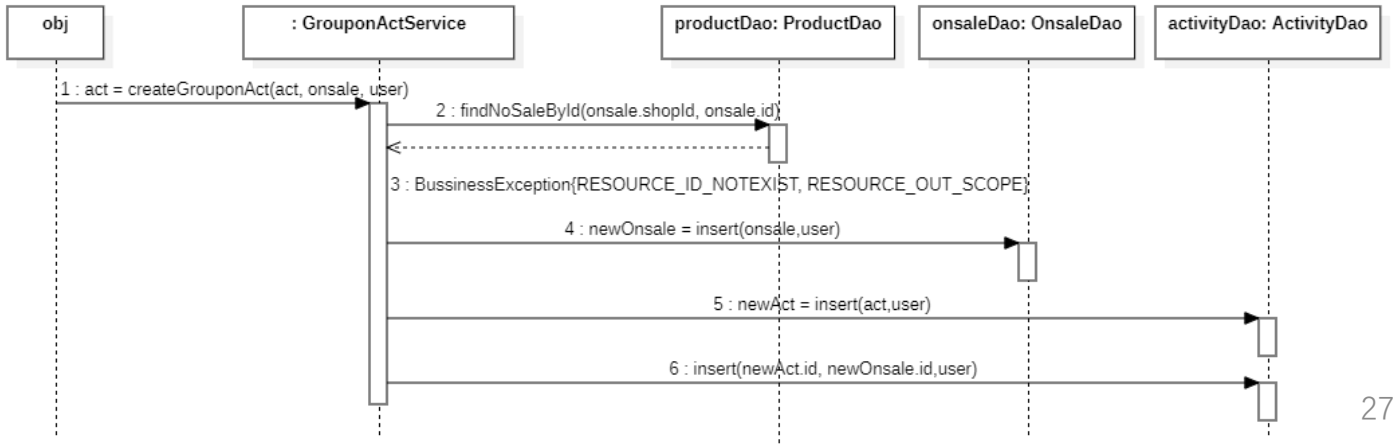
类名调用不是实例化

actList集合中的一个Activity对象

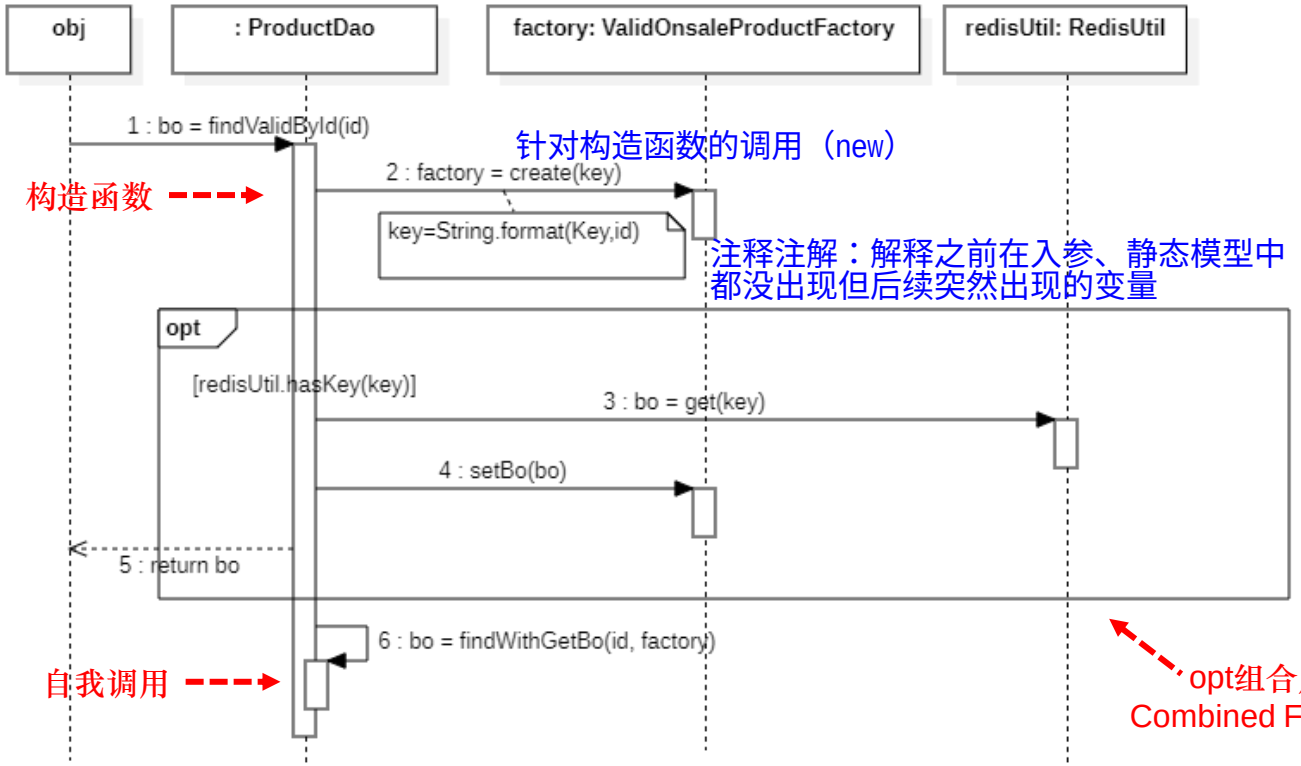
2.2 顺序图
Sequence Diagram

- 消息格式
  - return = message(parameter : parameterType) : returnType





2.2 顺序图
Sequence Diagram




```java
@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);
    }
}
```

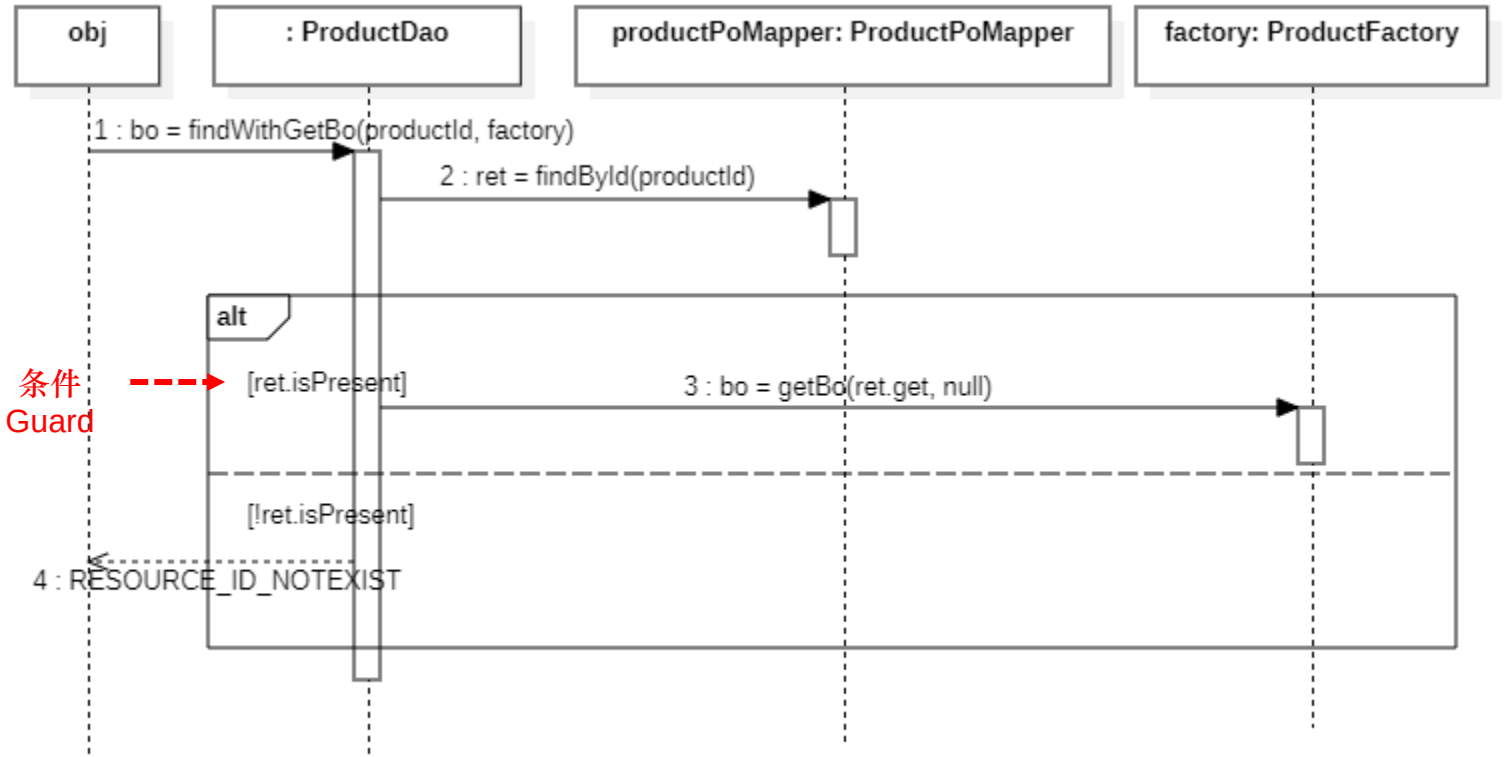
2.2 顺序图
Sequence Diagram

- 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

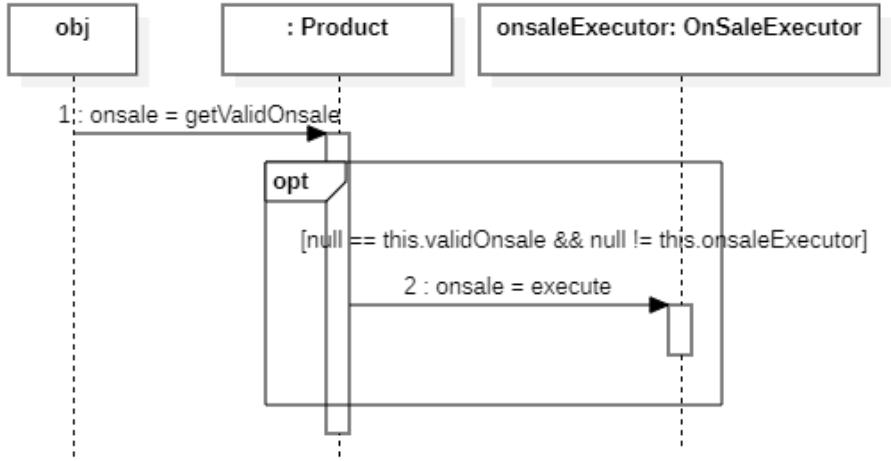
2.2 顺序图
Sequence Diagram

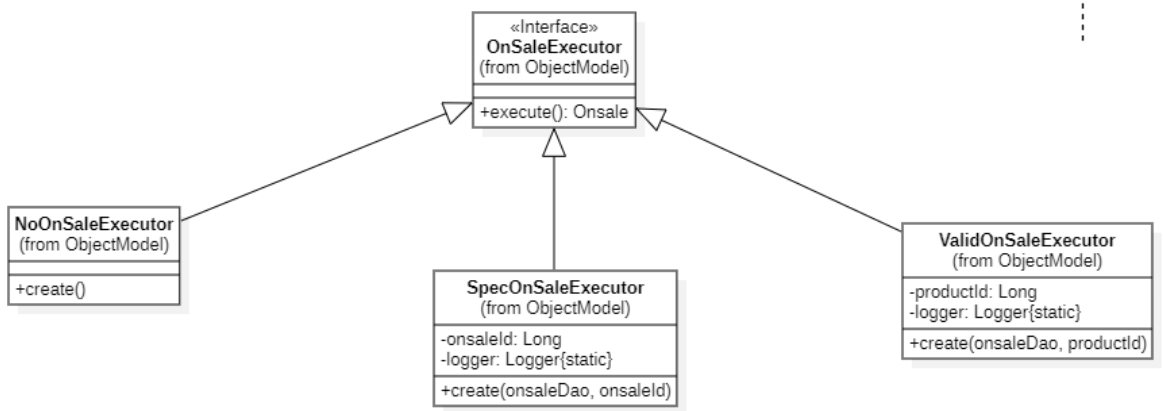
- 组合片段

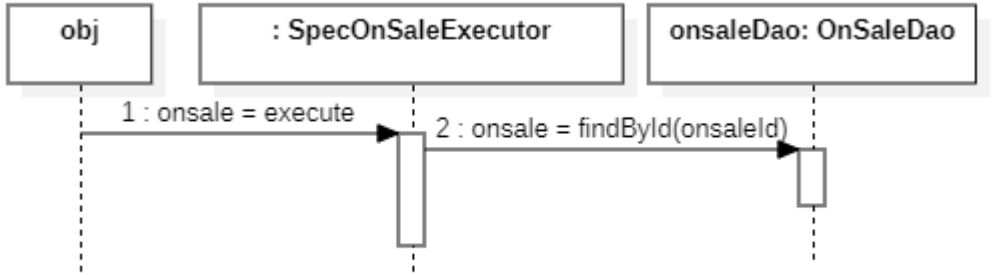


2.2 顺序图
Sequence Diagram

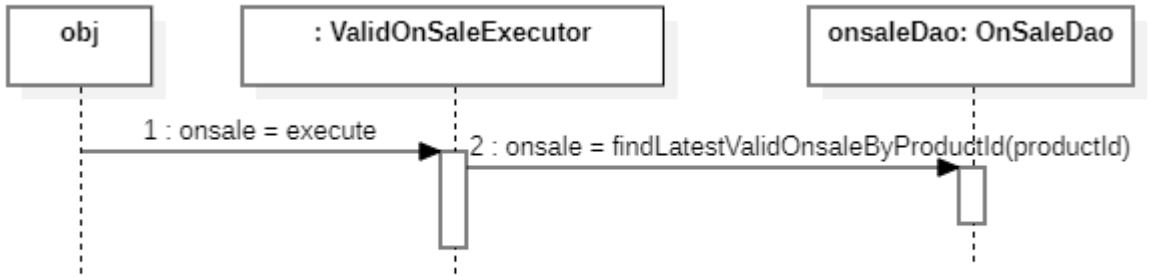
- 多态

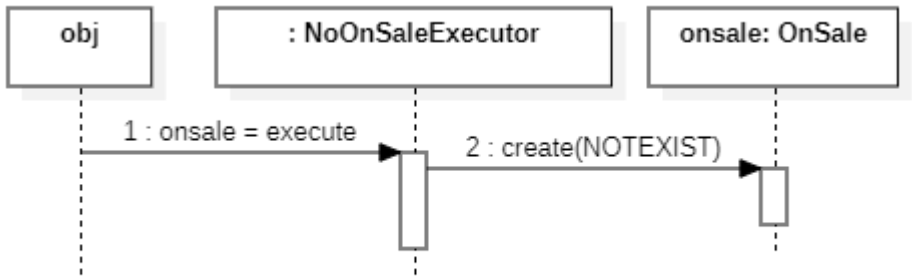






多态画多张





3. UML 状态图

UML State Machine Diagram

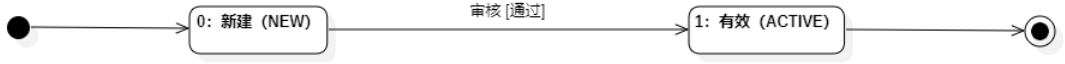
3.1 对象状态建模
Object State Modeling

- 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

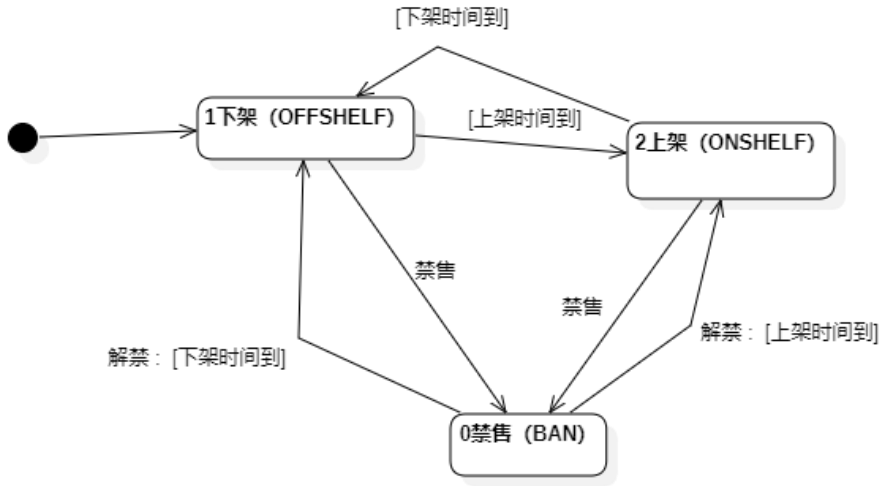
3.2 状态机图
State Machine Diagram

- 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

3.1 职责驱动的设计
Response-Driven Design

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