面向对象分析与设计
Object Oriented Analysis and Design

——函数式编程
Functional Programming

邱明 博士
厦门大学信息学院
mingqiu@xmu.edu.cn
2023年秋季学期

- Neal Ford. Functional Thinking. O'Reilly Media, Inc. 978-1-449-36551-6, 2014

1. 编程范式

Programming Paradigm

1.1 起源
origin

- 来自 Robert Floyd 在 1978 年12月4日图灵奖的颁奖演说（The Paradigms of Programming）
- 是指程序的设计方法，即程序应该如何被构建和执行

1.2 主要的编程范式
Three Paradigms

- 结构化编程（Structured Programming）
- 面向对象编程（Object-Oriented Programming）
- 函数式编程（Functional Programming）

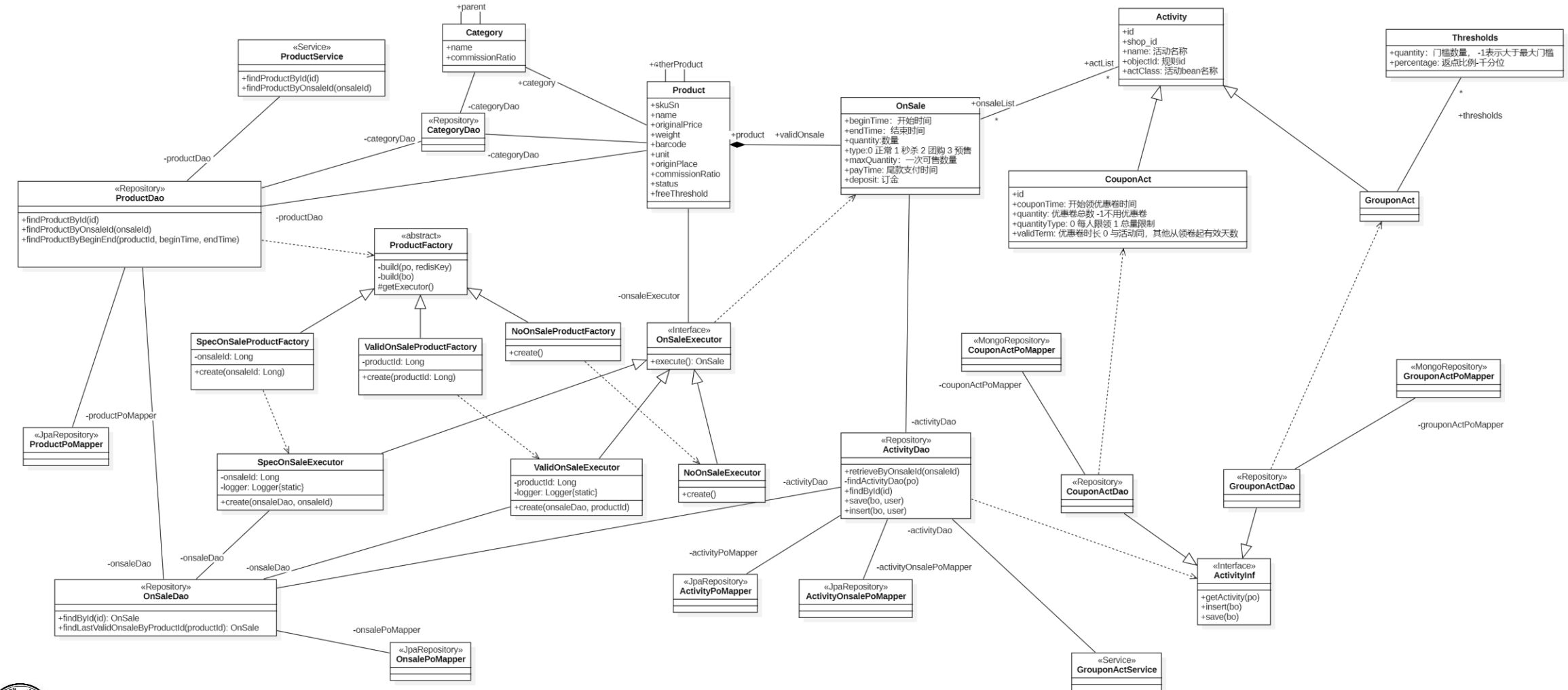
> OO makes code understandable by encapsulating moving parts. FP makes code understandable by minimizing moving parts.
>
> — Michael Feathers

Few Data Structures, Many Operations

1.3 结构化编程
Structured Programming

```
public void updateOnsale(Long shopId, Long id, Integer price, LocalDateTime beginTime, LocalDateTime endTime, Integer quantity, Byte type, UserDto user) {
        Onsale onsale = onsaleDao.findById(id);
        if(Constants.PLATFORM!=onsale.getShopId() && shopId!=onsale.getShopId()){
                throw new BusinessException(ReturnNo.RESOURCE_ID_OUTSCOPE, String.format(ReturnNo.RESOURCE_ID_OUTSCOPE.getMessage(), "价格浮动", id, shopId));
        }
        if(price!=null) onsale.setPrice(Long.valueOf(price));
        if(beginTime!=Constants.BEGIN_TIME) {
        if(beginTime!=Constants.BEGIN_TIME) onsale.setBeginTime(beginTime);
        onsale.setBeginTime(beginTime);
        if(endTime!=Constants.END_TIME) onsale.setEndTime(endTime);
        beginTime=onsale.getBeginTime();
        if(quantity!=null) onsale.setQuantity(quantity);
        if(type!=null) onsale.setType(type);
        if(endTime!=Constants.END_TIME) {
                onsale.setEndTime(endTime);
                endTime=onsale.getEndTime();
        }
        List<Onsale> onsaleByProductId = onsaleDao.findOnsaleByProductId(id);
        //判断商品时间是否冲突
        for(Onsale o:onsaleByProductId){
                LocalDateTime beginTimeExist = o.getBeginTime();
                LocalDateTime endTimeExist = o.getEndTime();
                if(!(endTimeExist.isBefore(beginTime)||beginTimeExist.isAfter(endTime))){
                        throw new BusinessException(ReturnNo.getByCode(299), "商品时间冲突");
                }
        }
        onsaleDao.save(onsale, user);
}
```

1.4 面向对象编程
Object-Oriented Programming



2.函数式编程

Functional Programming

2.1 基本思想
Basic Ideas

- 函数是一个表达式，用以完成计算和转换
  - 结构化编程和面向对象编程中的函数只是一个将一系列顺序执行的指令包裹起来的过程
  - 函数式编程中的函数与数学中的函数定义相同，
    - 表达式在用到时才会计算
    - 不改变输入数据，产生新的输出数据

$$f(x,y) = y\sin(x)$$

$$g(x) = log(x)$$

$$g\big(f(x,y)\big) = \log(ysin(x))$$

$$h(x) = \cos(x)$$

$$g\big(h(x)\big) = \log(\cos(x))$$

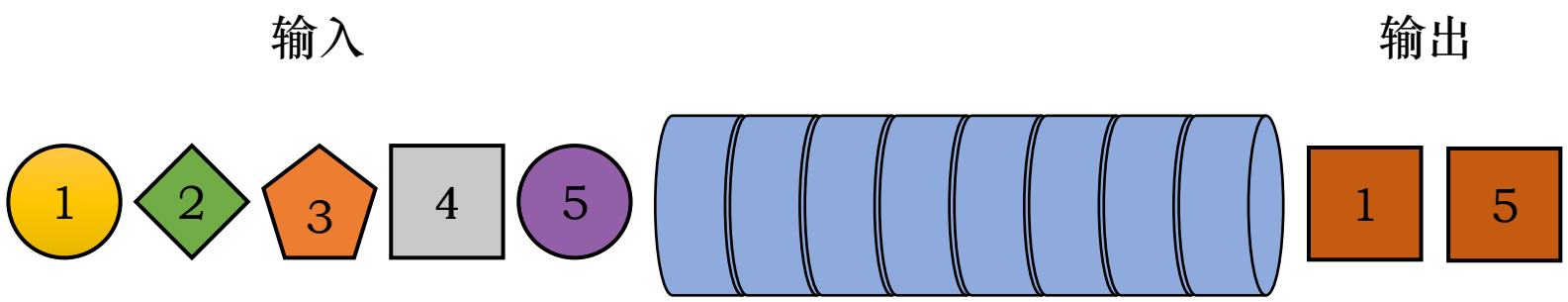
2.2 高阶函数
High Order Operation

- 函数可以作为参数和返回值
  - 纯函数：给定输入返回预定的结果，不修改外部变量的值
  - 不可变的数据：不改变输入数据，产生新的输出数据

```
this.divPayTransDao.retrieveByPayTransId(this.id).ifPresent(o -> this.divTrans = o);

this.divTrans = this.divPayTransDao.retrieveByPayTransId(this.id).orElse(null);

payTrans.getLedger().ifPresent(o -> this.ledger = CloneFactory.copy(new LedgerDto(),o));

this.ledger = payTrans.getLedger().map(o-> CloneFactory.copy(new LedgerDto(),o)).orElse(null);
```

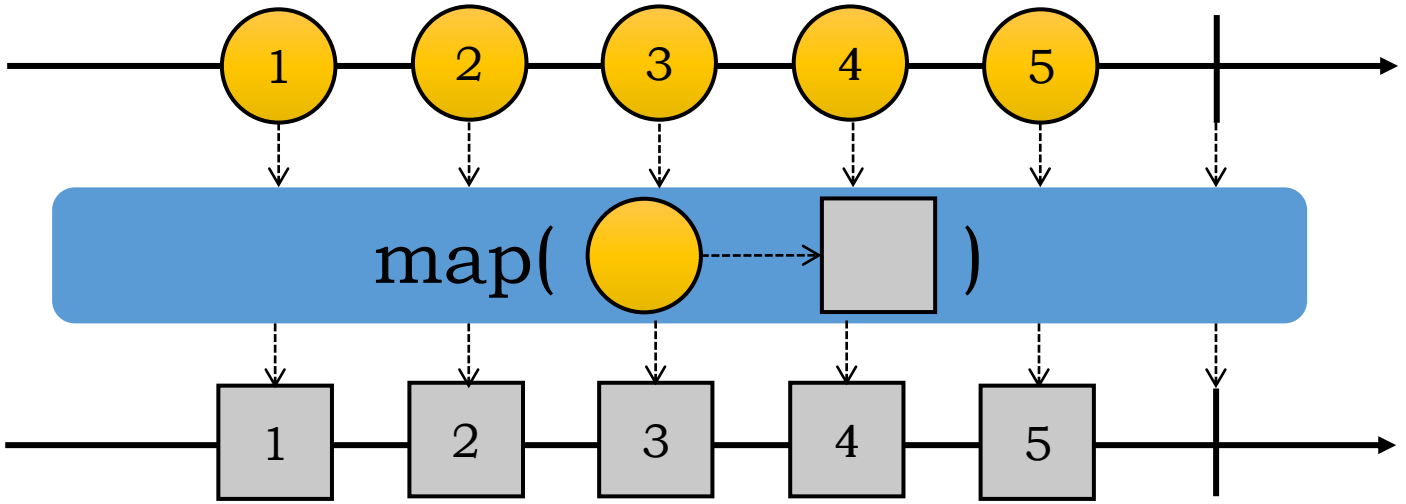
2.2 高阶函数
High Order Operation

- Java中Stream
  - 作为输入和输出之间的管道，不存储值
  - 不修改输入，产生新的输出
  - 惰性计算



2.2 高阶函数
High Order Operation

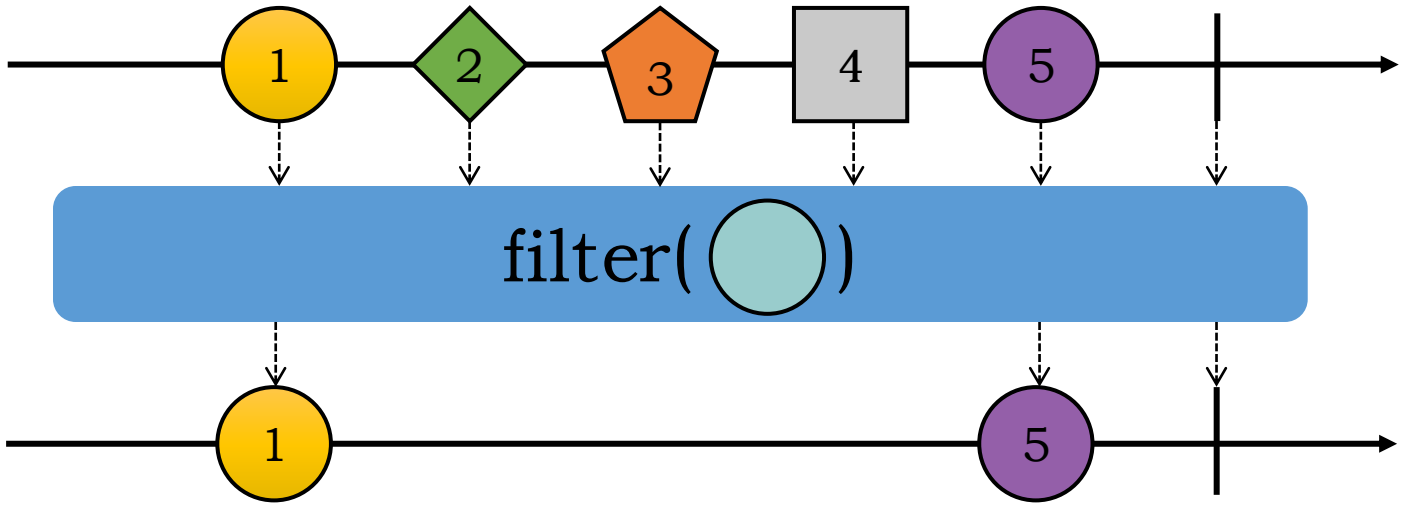
- 转换（map）




```
List<SimpleProductDto> dtos = products.stream().map(obj -> new SimpleProductDto(obj)).collect(Collectors.toList());
```

2.2 高阶函数
High Order Operation

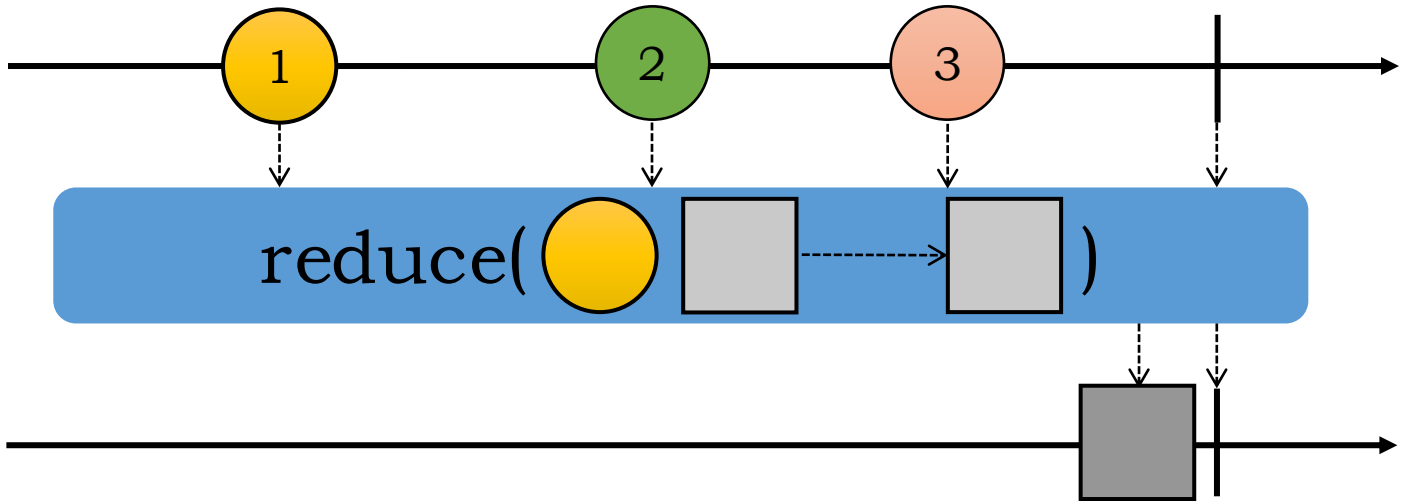
- 过滤（filter）




```
onsaleDto.getActList().stream().filter(activity -> activity.getId() == item.getActId()).count()  > 0
```

2.2 高阶函数
High Order Operation

- 递归（reduce）




```
this.getRefundTransList().stream()
    .filter(trans -> RefundTrans.FAIL != trans.getStatus())
    .map(RefundTrans::getAmount)
    .reduce((x,y)->x + y)
    .orElse(0L);
```

2.3 柯里化
Curry

- 将接受多个参数的函数转换成为一系列接受单个参数的函数。
- 柯里化后的函数不会立刻执行，而是返回一个新的函数，等到后面调用的时候在执行。
- 可以方便的组合和复用，提高代码的重用性

2.3 柯里化
Curry

```
ProductDto.builder().id(bo.getId()).barcode(bo.getBarcode()).beginTime(bo.getBeginTime()).endTime(bo.getEndTime())
        .maxQuantity(bo.getMaxQuantity()).quantity(bo.getQuantity()).name(bo.getName()).weight(bo.getWeight()).unit(bo.getUnit())
        .originalPrice(bo.getOriginalPrice()).price(bo.getPrice()).originPlace(bo.getOriginPlace()).skuSn(bo.getSkuSn()).status(bo.getStatus())
        .otherProduct(other.stream().map(product -> SimpleProductDto.builder().id(product.getId()).status(product.getStatus()).price(product.getPrice())
        .quantity(product.getQuantity()).name(product.getName()).build()).collect(Collectors.toList()))
        .shop(Shop.builder().id(shop.getId()).name(shop.getName()).type(shop.getType()).build()).build();


this.mockMvc.perform(MockMvcRequestBuilders.get(LEDGER, 1)
            .header("authorization", adminToken)
            .contentType(MediaType.APPLICATION_JSON_VALUE)
            .param("beginTime", "2022-11-06T12:00:00")
            .param("endTime", "2022-11-09T12:00:00")
            .param("type", "0")
            .param("channelId", "501"))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.content().contentType("application/json;charset=UTF-8"))
        .andExpect(MockMvcResultMatchers.jsonPath("$.errno", is(ReturnNo.OK.getErrNo())))
        .andExpect(MockMvcResultMatchers.jsonPath("$.data.list[?(@.id == '%d')].transNo", 501).value("1111"));
```

2.3 柯里化
Curry

- 哈斯凯尔·加里，Haskell Brooks Curry，1900年9月12日－1982年9月1日，美国马萨诸塞州米里镇人，数理逻辑学家，专长于组合子逻辑理论。

2.4 函数式编程的优点

- 函数纯净：程序有更少的状态量，编码心智负担更小。
- 高度可组合：函数之间复用方便。
- 副作用隔离：所有的状态量被收敛到一个函数里面处理。
- 代码简洁/流程更清晰
- 惰性计算：被组合的函数只会生成一个更高阶的函数，最后调用时数据才会在函数之间流动。