

# 34. 资损防控：搞定交易系统中金额处理规范

## \_V20240519

---

1. 背景
2. 金额处理场景
3. 金额计算常见误区及严重后果
4. 金额处理原则
5. 制定Money类
  - 5.1. 核心属性
  - 5.2. 通过金额数值和币种一个Money类
  - 5.3. 加减乘除
  - 5.4. 比较大小
  - 5.5. 返回元和分单位的数字
  - 5.6. 完整的Money类示例
6. Money类实际应用最佳实践
  - 6.1. 接收入口请求
  - 6.2. 内部应用运算
  - 6.3. 内部数据库存储
  - 6.4. 外发处理
7. 结束语

大家好，我是隐墨星辰，今天和大家聊一个资损防控的课题：交易系统中金额的计算、存储、传输相关的最佳实践。

这篇文章主要讲清楚：交易系统（电商、支付等）金额处理常见资损场景，如何构建一个适合公司业务的Money类，应用Money的最佳实践，包括计算、存储、传输，目的是在金额处理上减少资损风险。

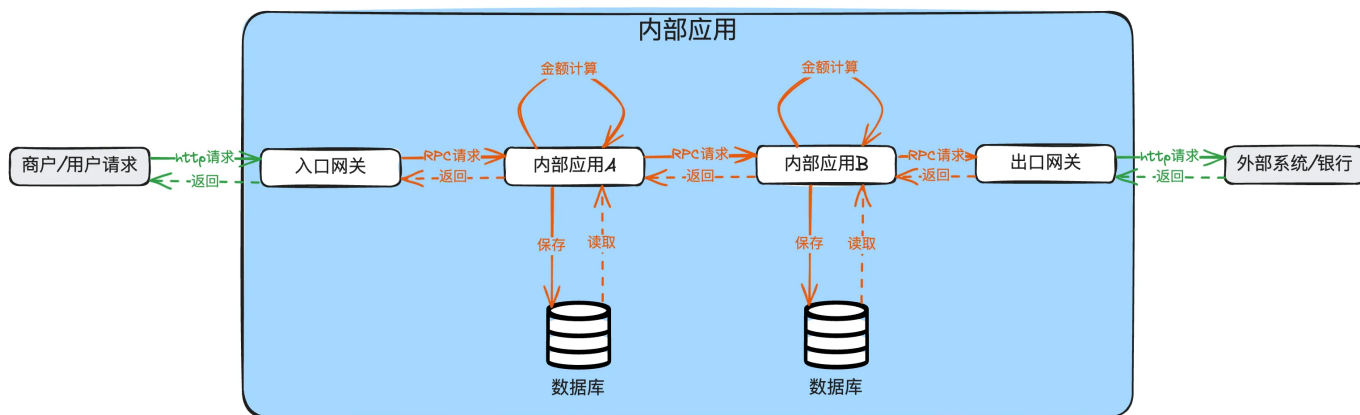
## 1. 背景

前几天有同学私聊我问了几个问题：

1. “做国际支付，不同的币种的最小单位不同，比如人民币是分，日元是元，那数据库里面应该应该保存整数还是小数？”
2. “从哪里获取到这个币种的最小单位是多少？”
3. “我赞成你说的不应该直接对金额进行加减乘除操作，但我还是不知道怎么做，怎样才能落地呢？”

以前在公司时，也有兄弟部门因为金额处理不当，导致了好几万的资损故障，然后过来问我金额处理的最佳实践，当时也给他们做过一次相关分享。

## 2. 金额处理场景



从上图可以看到，对于交易系统而言，一共有下面几种场景需要做金额处理：

1. 接收外部请求。比如商户下单100元，或用户转账1000元。
2. 内部应用处理。比如计算手续费等。
3. 内部应用保存到数据库，从数据库读取。
4. 内部应用之间传输。
5. 发送给外部系统或银行渠道。比如向银行请求扣款100元。

## 3. 金额计算常见误区及严重后果

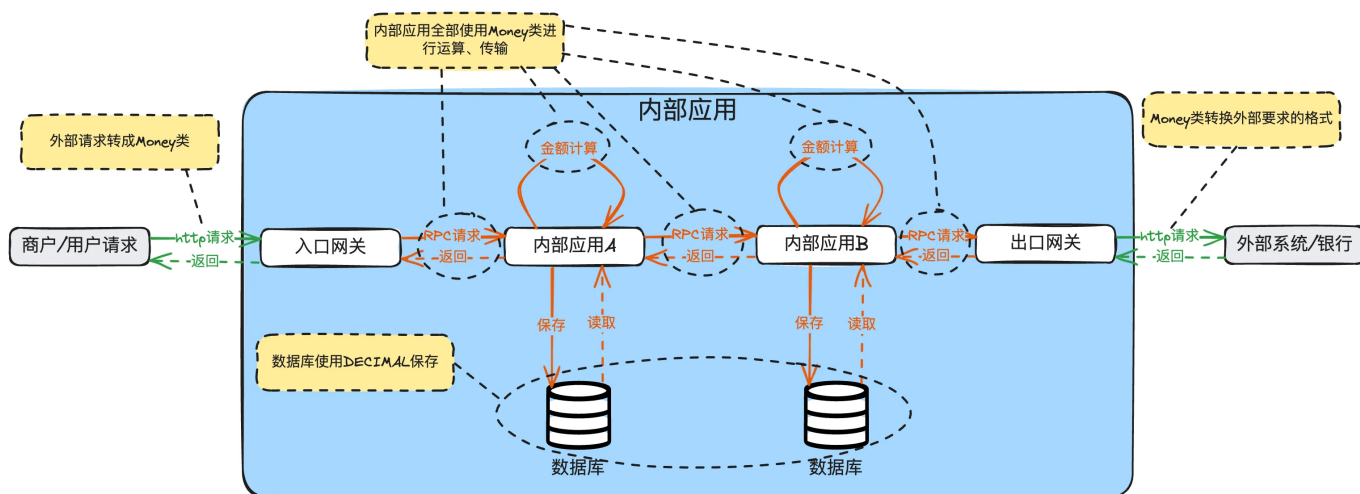
对于研发经验不足的团队而言，经常会犯以下几种错误：

1. 没有定义统一的Money类，各系统间使用BigDecimal、double、long等数据类型进行金额处理及存储。
2. 定义了统一的Money类，但是写代码时不严格遵守，仍然有些代码使用BigDecimal、double、long等数据类型进行金额处理。
3. 手动对金额进行加、减、乘、除运算，单位（元与分）换算。

带来的后果，通常就是资金损失，再细化一下，最常见的情况有下面3种：

1. 手动做单位换算导致金额被放大或缩小100倍。
  - a. 比如大家规定传的是元，但是其中有位同学忘记了，以为传的是分，外部渠道要求传元，就手动乘以100。或者反过来。
  - b. 还有一种情况，部分币种比如日元最小单元就是元，假如系统约定传的是分，外部渠道要求传元，就可能在网关处理时手动乘以100。
2. 1分钱归属问题。比如结算给商家，或计算手续费时，碰到除不尽时，使用四舍五入，还是向零舍入，还是银行家舍入？这取决于财务策略。
3. 精度丢失。在大金额时，double有可能会有精度丢失问题。

## 4. 金额处理原则



直接上答案：

1. 制定适用于公司业务的Money类来统一处理金额。

2. 在入口网关接收到请求后，就**转换为Money类**。
3. 所有内部应用的金额处理，**强制全部使用Money类运算、传输**，禁止自己手动加减乘除、单位换算（比如元到分）。
4. 数据库使用DECIMAL类型保存，保存单位为元。
5. 在出口网关外发时，再根据外部接口文档要求，转换成使用指定的单位。有些是元，有些是分（最小货币单位）

## 5. 制定Money类

JAVA有制定金额处理规范JSR 354（Java Specification Request 354），对应的实现包是Java Money API（javax.money），它提供了一套用于处理货币和货币计算的API。不过我们通常选择实现自己的Money类，主要是方便，可以自由定制，比如小数舍入问题。

一个Money类通常包括以下几个方面：

1. 通过参数生成一个Money类。
2. 加减乘除处理。
3. 比较处理。
4. 获取金额（元）和获取最小单位金额（元或分）。

### 5.1. 核心属性

```
1 public class Money implements Comparable<Money>, Serializable {  
2     /**  
3      * 币种最小单位  
4      */  
5     private long cent;  
6  
7     /**  
8      * 币种  
9      */  
10    private Currency currency;  
11  
12 }
```

## 5.2. 通过金额数值和币种一个Money类

通过币种最小单位创建

```
1  /**
2   * 创建Money对象
3   *
4   * @param minorUnits
5   * @param currencyCode
6   * @return
7   */
8  public static Money of(long minorUnits, String currencyCode) {
9      Currency currency = Currency.getInstance(currencyCode);
10     assertNotNull(currency, "currency can not be null: " + currencyCod
11 e);
12     return of(minorUnits, currency);
13 }
14
15 /**
16 * 创建Money对象
17 *
18 * @param minorUnits
19 * @param currency
20 * @return
21 */
22 public static Money of(long minorUnits, Currency currency) {
23     Money money = new Money(0, currency);
24
25     money.cent = minorUnits;
26
27     return money;
28 }
```

## 通过币种单位元创建

这个方法将自动根据币种计算出最小单位。

比如人民币，传入1，money.cent赋值100，代表是100分（币种最小单位是分）。而日元，传入1，money.cent赋值1，代表是1日元（币种最小单位就是元）。

```
1  /**
2   * 创建Money对象
3   *
4   * @param amount
5   * @param currencyCode
6   * @return
7   */
8  public static Money of(BigDecimal amount, String currencyCode) {
9      Currency currency = Currency.getInstance(currencyCode);
10     assertNotNull(currency, "currency can not be null: " + currencyCod
11 e);
12     return of(amount, currency);
13 }
14
15 /**
16 * 创建Money对象
17 *
18 * @param amount
19 * @param currency
20 * @return
21 */
22 public static Money of(BigDecimal amount, Currency currency) {
23     Money money = new Money(0, currency);
24
25     money.cent = amount.movePointRight(currency.getDefaultFractionDigi
26 ts()).longValue();
27     return money;
28 }
```

## 5.3. 加减乘除

1. 注意除法有除不尽舍入的问题，需要根据业务来指定舍入的模式，建议默认提供四舍五入，但是保留指定模式的能力。具体可以参考：[java.math.RoundingMode](#)。

UP：远零方向舍入。示例：1.6返回2，-1.6返回-2。

DOWN：向零方向舍入。示例：1.6返回1，-1.6返回-1。

**CEILING:** 向上舍入。示例：1.6返回2，-1.6返回-1。

**FLOOR:** 向下舍入。示例：1.6返回1，-1.6返回-2。

**HALF\_UP:** 四舍五入。示例：1.5返回2，-1.5返回-2。

**HALF\_DOWN:** 五舍六入。示例：1.5返回1，-1.5返回-1，1.6返回2，-1.6返回-2。

**HALF\_EVEN:** 银行家算法，尾数小于0.5舍，尾数大于0.5入，尾数等于0.5往最终结果是偶数的方向进。示例：1.51返回2，-1.49返回-1，2.5返回2，3.5返回4（1.5，2.5，3.5，4.5，5.5等这些最终只出现2，4，4，4，6等偶数）。

2. **加和减，需要先判断币种，只有币种相同才能做加减。**



```
1
2  /**
3   * 除法
4   *
5   * @param value
6   * @return
7   */
8  public Money divide(long value) {
9      return divide(value, DEFAULT_ROUNDING);
10 }
11
12 /**
13  * 除法
14  *
15  * @param value
16  * @return
17  */
18 public Money divide(BigDecimal value) {
19     return divide(value, DEFAULT_ROUNDING);
20 }
21
22 /**
23  * 除法
24  *
25  * @param value
26  * @return
27  */
28 public Money divide(BigDecimal value, RoundingMode roundingMode) {
29     BigDecimal newCent = BigDecimal.valueOf(this.cent).divide(value,
30 roundingMode);
31     return Money.of(newCent.longValue(), this.currency);
32 }
33
34 /**
35  * 除法
36  *
37  * @param value
38  * @param roundingMode
39  * @return
40  */
41 public Money divide(long value, RoundingMode roundingMode) {
42     BigDecimal newCent = BigDecimal.valueOf(this.cent).divide(BigDecimal.valueOf(value), roundingMode);
43     return Money.of(newCent.longValue(), this.currency);
44 }
45
```

```

44  /**
45  * 加法
46  *
47  * @param value
48  * @return
49  */
50  public Money add(Money value) {
51      assertSameCurrency(value);
52      return Money.of(this.cent + value.cent, this.currency);
53  }
54
55  /**
56  * 减法
57  *
58  * @param value
59  * @return
60  */
61  public Money subtract(Money value) {
62      assertSameCurrency(value);
63      return Money.of(this.cent - value.cent, this.currency);
64  }
65
66  /**
67  * 乘法
68  *
69  * @param value
70  * @return
71  */
72  public Money multiply(long value) {
73      return Money.of(this.cent * value, this.currency);
74  }
75
76  /**
77  * 乘法
78  *
79  * @param value
80  * @param roundingMode
81  * @return
82  */
83  public Money multiply(BigDecimal value, RoundingMode roundingMode) {
84      long newCent = BigDecimal.valueOf(this.cent).multiply(value).setScale(0, roundingMode).longValue();
85      return Money.of(newCent, this.currency);
86  }
87
88  public Money multiply(BigDecimal value) {
89      long newCent = BigDecimal.valueOf(this.cent).multiply(value).setScale(0, DEFAULT_ROUNDING).longValue();

```

```

90         return Money.of(newCent, this.currency);
91     }
92
93
94     /**
95      * 相同币种比较
96      *
97      * @param value
98      */
99     private void assertSameCurrency(Money value) {
100         if (!Objects.equals(this.currency, value.currency)) {
101             throw new IllegalArgumentException("Money instances must have the same currency");
102         }
103     }

```

## 5.4. 比较大小

Java

```

1
2     /**
3      * 对比
4      *
5      * @param value the object to be compared.
6      * @return
7      */
8     @Override
9     public int compareTo(Money value) {
10         assertSameCurrency(value);
11         return Long.compare(this.cent, value.cent);
12     }

```

## 5.5. 返回元和分单位的数字

所有内部应用全部使用`getAmount()`，不允许使用`getCent()`。保证内部应用大家的语义保持一致。

只有请求外部渠道时，如果渠道要求使用币种最小单位，才使用getCent()。

```
1  /**
2   * 获取金额数，单位为元
3   * 内部系统强制使用getAmount，不能使用getCent。
4   * 比如：人民币 1元，返回1，100分返回1。日元最小单位是1元，返回1
5   *
6   * @return
7   */
8  public BigDecimal getAmount() {
9      return BigDecimal.valueOf(cent, currency.getDefaultFractionDigits());
10 }
11
12 /**
13  * 返回币种最小单位
14  * 内部系统强制使用getAmount，不能使用getCent，除非在和银行渠道对接时，需要使用getCent。
15  *
16  * @return
17  */
18 public long getCent() {
19     return cent;
20 }
```

## 5.6. 完整的Money类示例

完整的Money类还有其它的一些方法，具体可以参

考：<https://gitee.com/yinmosc/gateway/blob/master/src/main/java/com/demo/gateway/common/Money.java>

## 6. Money类实际应用最佳实践

从接收外部请求开始，到内部计算、存储，最后外发到渠道，完整实践说明。

### 6.1. 接收入口请求

在入口网关处，先转换成Money类，再往后请求。

```
1 // 使用外部请求的参数构建Money类
2 Money payAmount = Money.of(BigDecimal.valueOf(outRequest.getPayAmount()),
  outRequest.getCurrency());
3
4 // 构建内部请求
5 PayRequest request = new PayRequest();
6 request.setPayAmount(payAmount);
7 ... ..
8
9 // 发给内部应用
10 payService.pay(request);
```

## 6.2. 内部应用运算

内部所有应用，全部使用Money类流转和计算。

```
1 Money payAmount = request.getPayAmount();
2 Money fee = payAmount.multiply(BigDecimal.valueOf(0.03));
3
4 // 其它处理
```

## 6.3. 内部数据库存储

```
1 Money payAmount = request.getPayAmount();
2 BigDecimal amount = payAmount.getAmount();
3 String currency = payAmount.getCurrency().getCurrencyCode();
4
5 // 构建DO
6 Order order = new Order();
7 order.setAmount(amount);
8 order.setCurrency(currency);
9 ...
10
11 // 保存入库
12 saveToDB(order);
13
```

## 6.4. 外发处理

注意：

1. 渠道要求是元，使用：

```
String amount = payAmount.getAmount();
```

2. 如果要求是分，使用：

```
String amount = payAmount.getCent();
```

```
1 Money payAmount = request.getPayAmount();
2
3 // 渠道要求是元，如果要求是分，使用：String amount = payAmount.getCent();
4 String amount = payAmount.getAmount();
5 String currency = payAmount.getCurrency().getCurrencyCode();
6
7
8 // 外发报文组装
9 ... ..
10
```

## 7. 结束语

金额如果处理得不好，引来的直接后果就是资金损失，哪怕不是今天，早晚也得出事。

如果你是研发同学，发现内部还没有使用Money类处理金额，建议早点对内部系统做改造。如果你是产品经理，建议转给内部研发同学，避免研发同学踩坑。

这是《百图解码支付系统设计与实现》专栏系列文章中的第（34）篇。欢迎和我一起深入解码支付系统的方方面面。