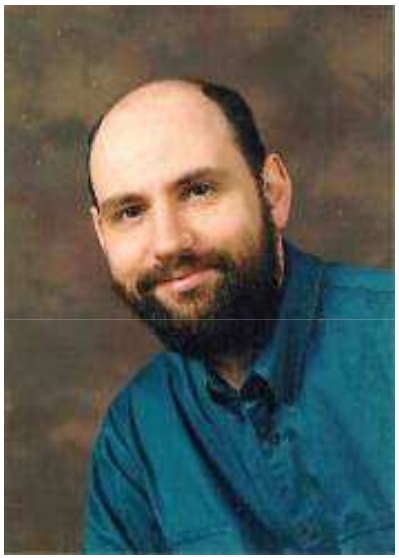
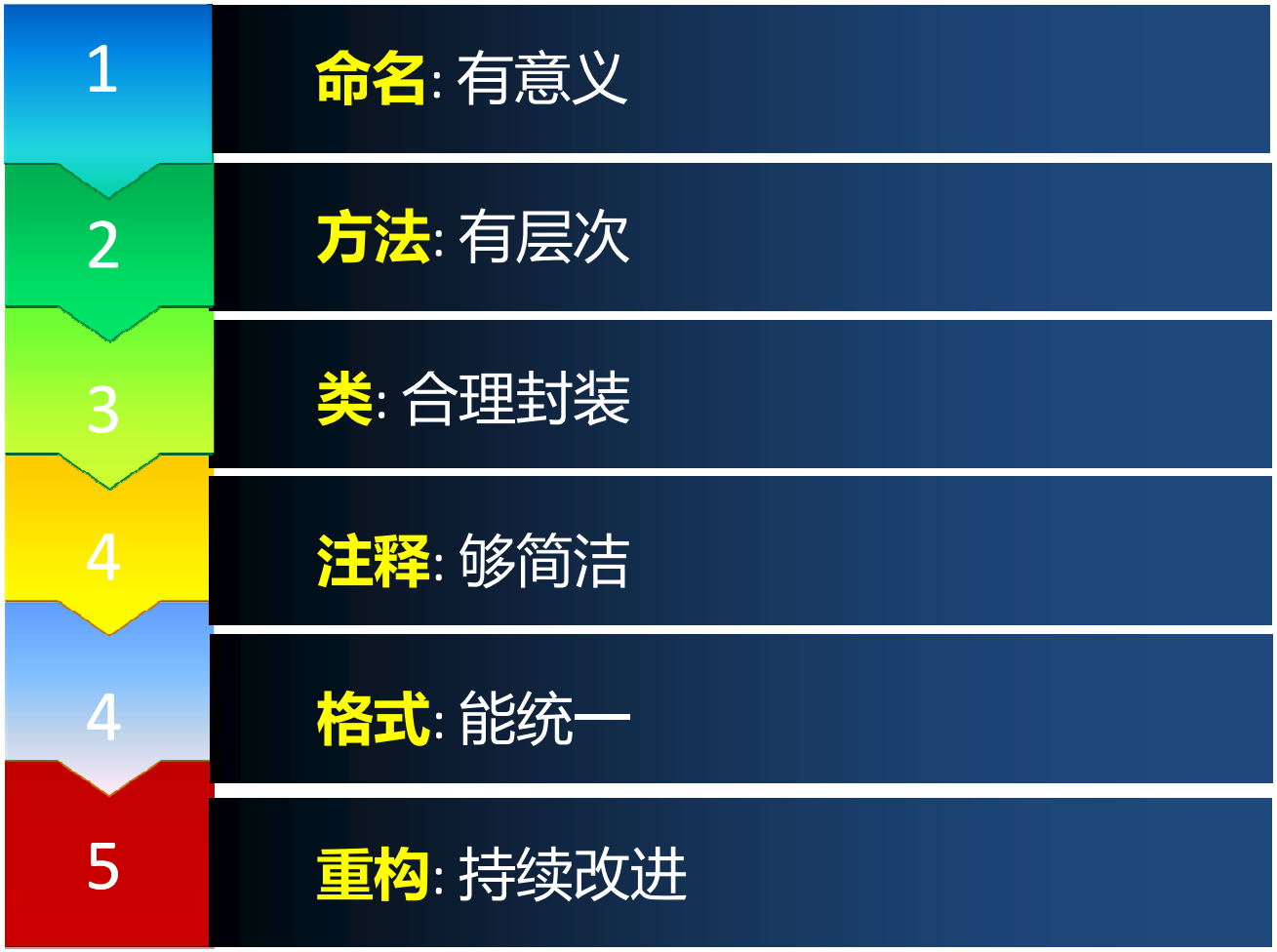

Keep Your Code Clean
——如何编写整洁代码

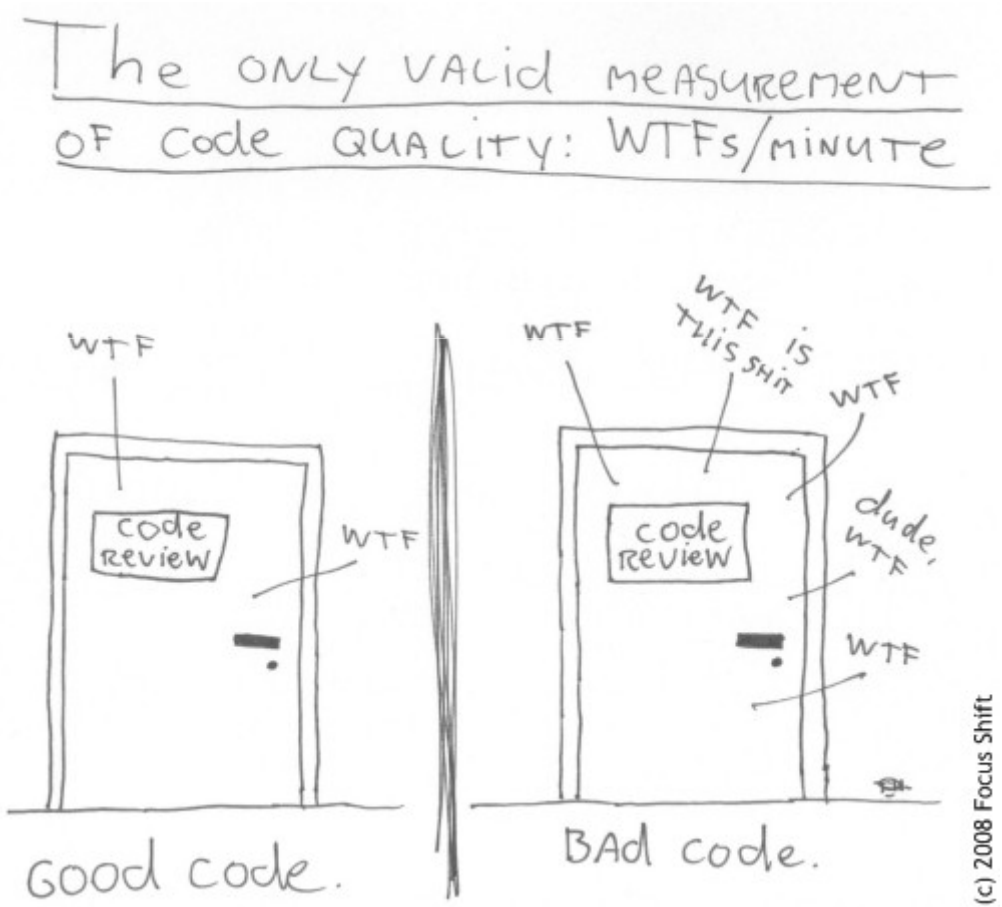
“任何一个傻瓜都可以写出计算机可以理解的代码。唯有写出**人类容易理解**的代码，才是优秀程序员。”

-- Martin Fowler《重构》

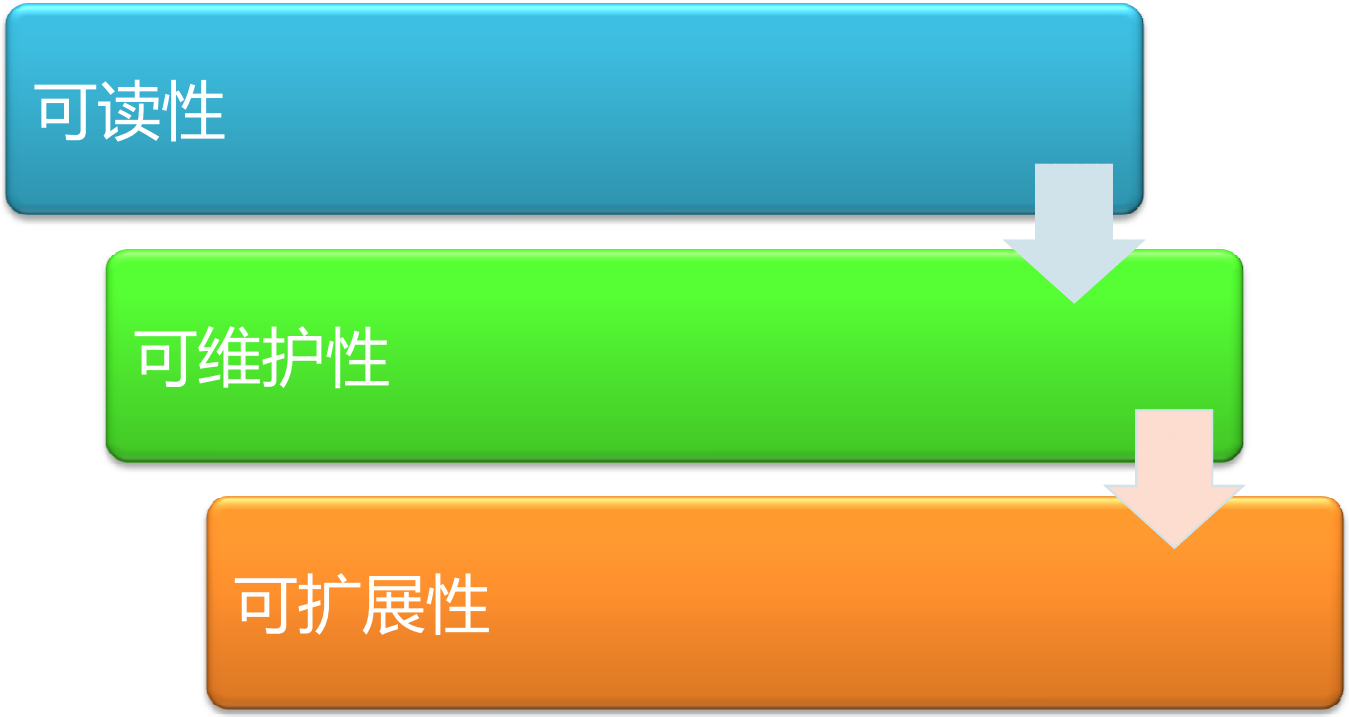



1
命名: 有意义
2
方法: 有层次
3
类: 合理封装
4
注释: 够简洁
4
格式: 能统一
5
重构: 持续改进

整洁代码的标准


The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF is THIS SHIT
WTF
code review
dude, WTF
WTF
Bad code.
(c) 2008 Focus Shift

什么才是整洁代码



1：命名

好命名带来高效率（code once， read more）

名副其实就不需要注释

```
/**
     * 用来处理买家付款之后，将sku从一个换成另外一个，同时减库存，这里会导致商品的库存被多减
     *
     * @param itemId
     * @param quantity 需要减掉的库存
     * @param skuId 要替换成的目标sku
     * @param bizOrderId
     * @param sellerId
     * @return
     * @throws IcException
     */
    public UpdateAuctionQuantityResultDO updateAuctionQuantityAndSku(final long itemId, final int quantity,
                                                                     final long skuId, final long bizOrderId, final long sellerId) throws IcException;
```

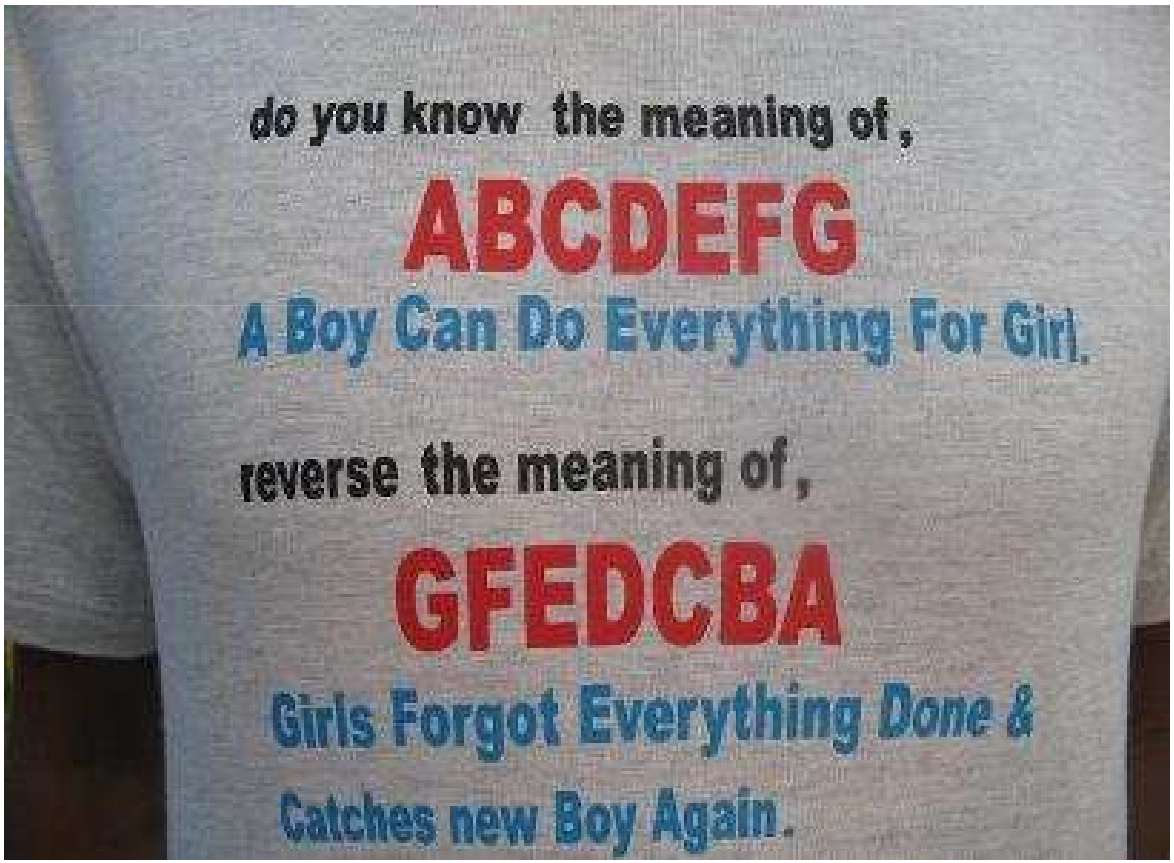
避免命名之间的混淆

我碰到的一些例子

```
StdCategoryDO getCategory(long catID)

StdCategoryDO getCategory(int categoryId);
```

```
StdCategoryDO
getCategoryThrowExceptionIfNull(int
categoryId);
```

请抵制缩写的诱惑

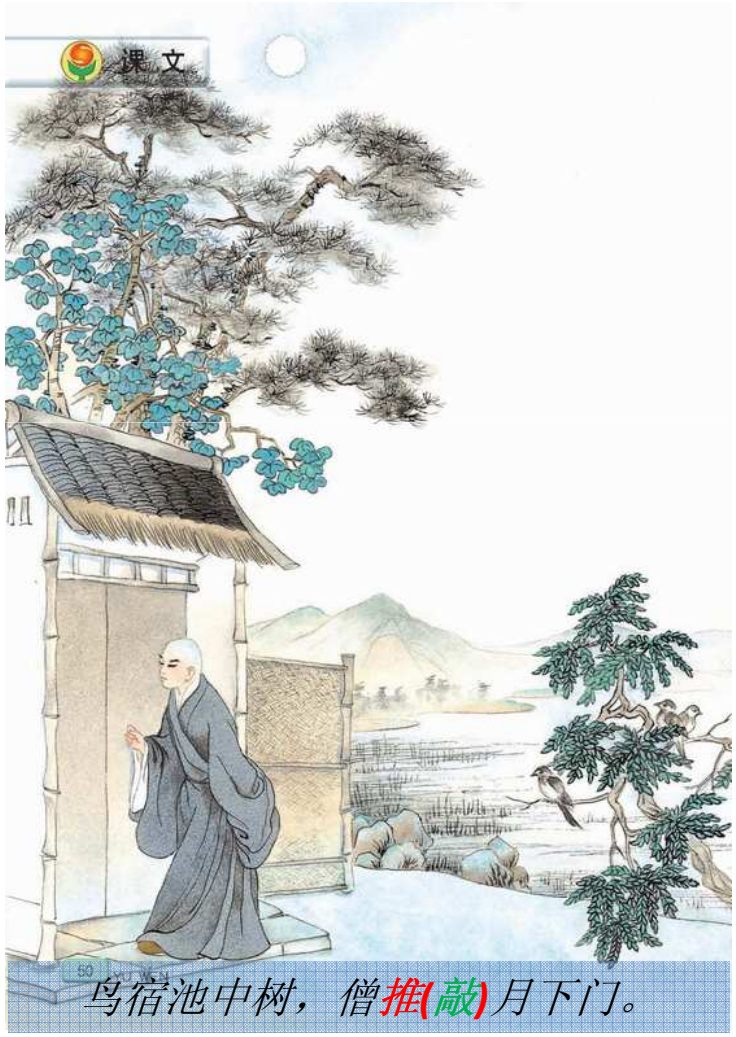

do you know the meaning of,
ABCDEFG
A Boy Can Do Everything For Girl.
reverse the meaning of,
GFEDCBA
Girls Forgot Everything Done &
Catches new Boy Again.

名称长度与作用范围成正比

```
for(int i = 0; i < array.length; i++) {
      //do something
}
```

```
/** 小二删除的状态 */
public static final int DELETED_BY_XIAOER = -4;

/** 小二下架的状态 */
public static final int INSTOCK_BY_XIAOER = -3;
```

取个好名字是一种艺术



一些命名借鉴

add/remove

up/down

send/receive

lock/unlock

first/last

start/stop

open/close

next/previous

show/hide

source/target

old/new

insert/delete

min/max

begin/end

Factory

Template

Composite

Proxy

Visitor

Command

Builder

Adapter

Decorator

Observer

Strategy

如何改善已有代码中的命名

?

一个很常见的例子

```
List<int[]> theList;
public List<int[]> getThem(){
    List<int[]> list1 = new ArrayList<int[]>();
    for(int[] x : theList) {
        if (x[0] == 4) {
            list1.add(x);
        }
    }
    return list1;
}
```

做第一次改进后的结果

```
private static final int FLAGGED = 4;
private static final int STATUS_VALUE = 0;
List<int[]> gameBoard;
public List<int[]> getFlaggedCells(){
    List<int[]> flaggedCells = new ArrayList<int[]>();
    for (int[] cell : gameBoard) {
      if (cell[STATUS_VALUE] == FLAGGED_) {
          flaggedCells.add(cell);
       }
    }
    return flaggedCells;
}
```

再一次重构后的结果

```
class Cell{
    int status;
    private static final int FLAGGED = 4;
    boolean isFlegged() {
        return status == FLAGGED;
    }
}

List<Cell> gameBoard;
public List<Cell> getFlaggedCells(){
    List<Cell> flaggedCells = new ArrayList<Cell>();
    for (Cell cell : gameBoard) {
       if (cell.isFlagged()){
          flaggedCells.add(cell);
       }
    }
    return flaggedCells;
}
```

2：方法

一个方法**只做**一件事情

如何判断”一个方法只做一件事”？

能用”**为了…，需要…”** 来描述这个方法

借鉴Shell脚本编程原则

```
egrep '(Operation|Method)Exception' -B1
/home/admin/itemcenter/logs/hsf.log | fgrep '执
行HSF服务[' | sed 's/\]时出现未知异常：未找到需
要调用的方法：[a-zA-Z]\+；'| sort | uniq  |
fgrep -v '===' | sort | uniq -c
```

保持**同一抽象**层级

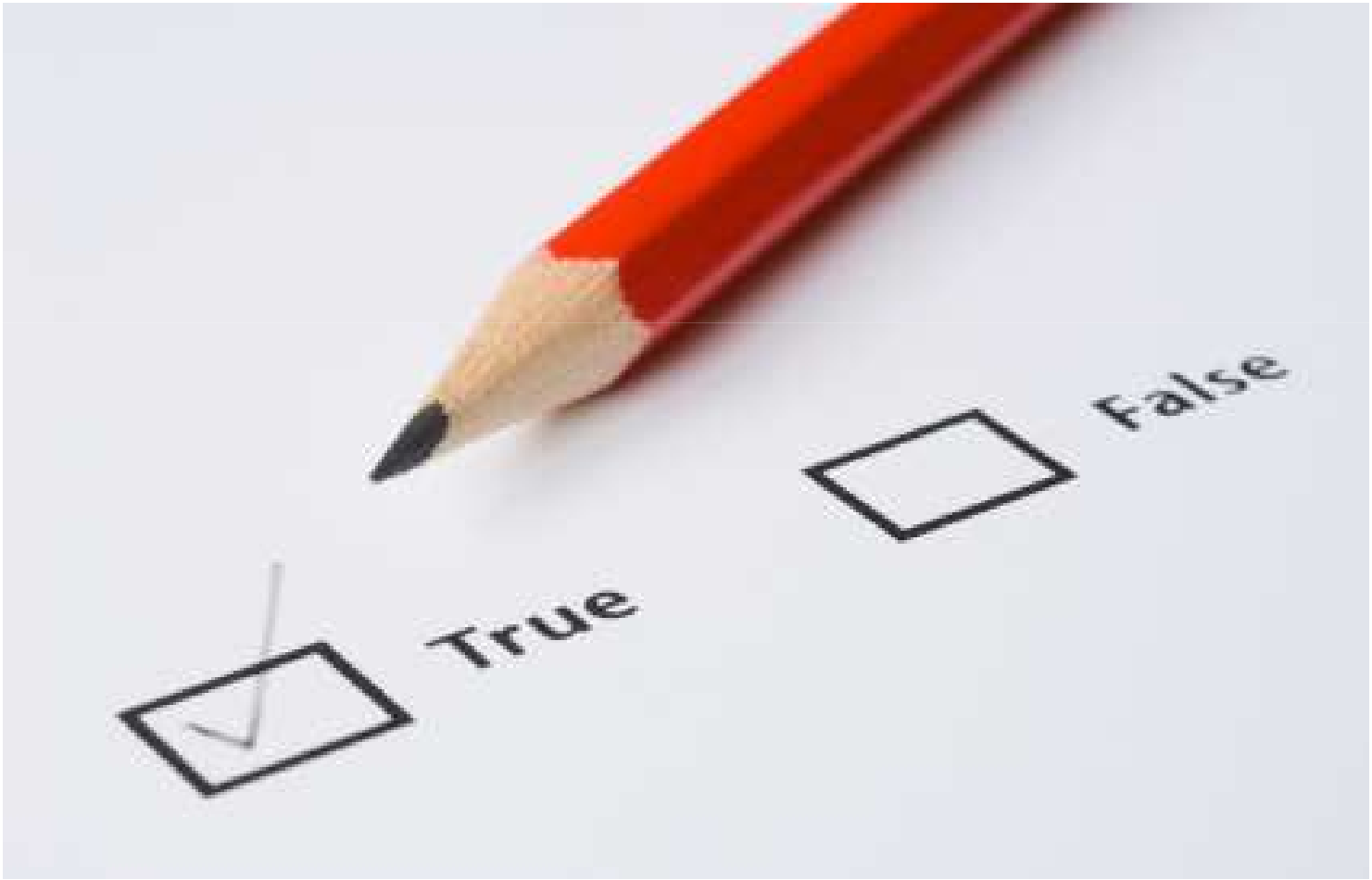
保持同一抽象层级

```
public void increaseItemQuantity(… ) throws IcException {
        …
       // 判断宝贝是否需要被更新
       if (isNotModified(… )) {
           return;
       }

       doIncreaseItemQuantity(… );

       // 如果下架的是拍卖的商品，则更新快照
       updateSnapIfAuction(… );

       // 清tair
       itemTairCacheService.removeItemDO(itemId);

       // 发送notify消息
       sendModifyQuantityNotify(… );

       // 更新实时搜索
       updateModifyQuantityDocument(… );
}
```

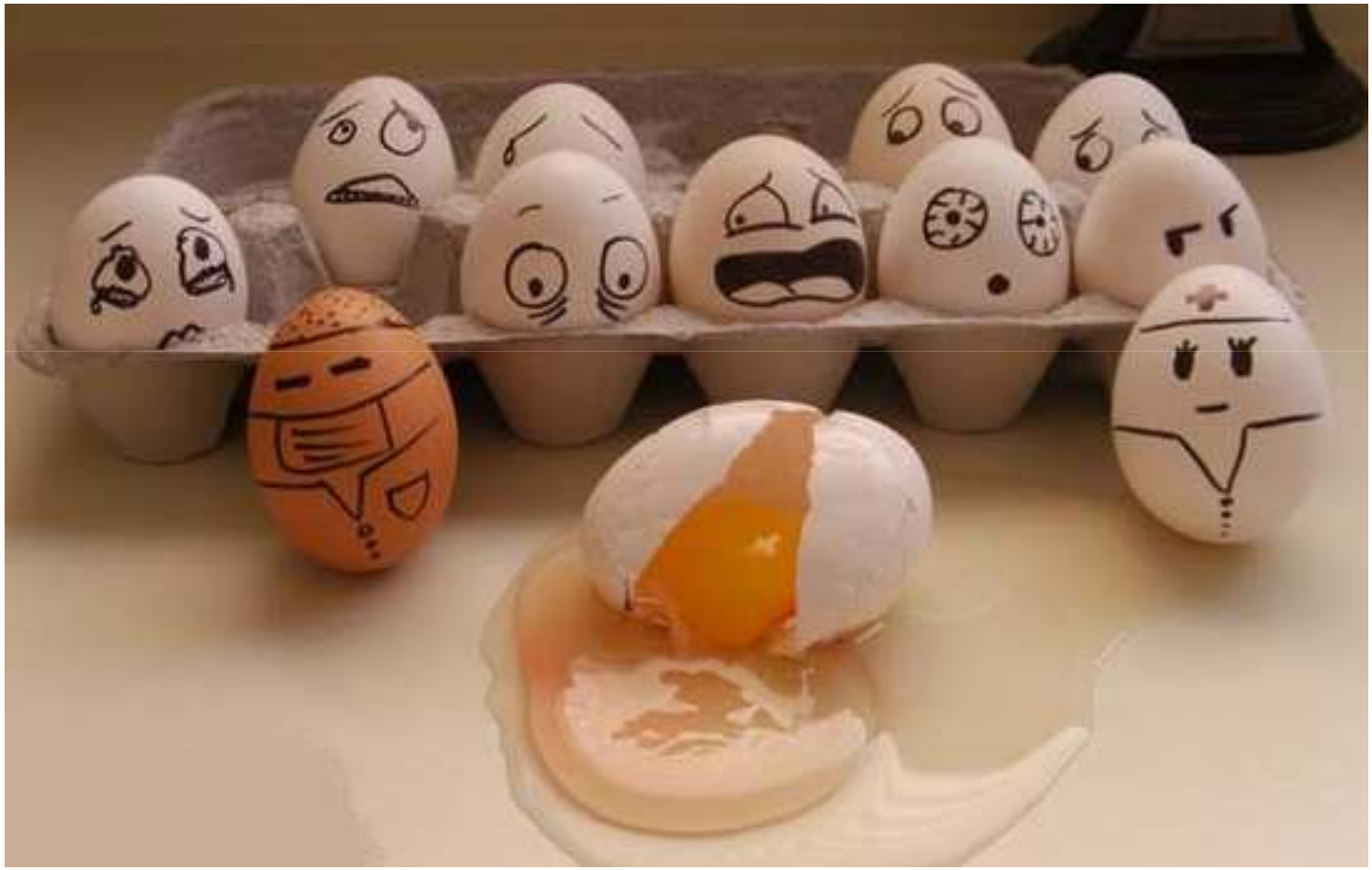
参数越少越好: 0>1>2>3 (参数封装类)

参数的**顺序**最好反应被使用的顺序

标识参数表明函数不止做一件事



尽量将异常处理从主流程中**分离**出来



一个异常的使用例子

```
private ProcessResultDO updateItemAndSkuQuantity(… ) throws IcException {
        // 获取卖家信息
        BaseUserDO seller = getUser(sellerId);
        if (seller == null){
            …
            return result;
        }

        //获取宝贝以及宝贝的SKU
        ItemDO dbItem = getItemWithSku(itemId);
        if (dbItem == null){
            …
            return result;
        }

        //校验是否允许更改库存
        stepCheckModifyQuantity(… );

        //更新宝贝库存
        doModifyItemQuantity(… );
}
```

重构之后

```
private ProcessResultDO updateItemAndSkuQuantity(… ) throws IcException {
        final ProcessResultDO result = new ProcessResultDO();
        try {
            // 获取卖家信息
            BaseUserDO seller = getUserThrowExceptionIfNull(sellerId);

            //获取宝贝以及宝贝的SKU
            ItemDO dbItem = getItemWithSkuThrowExceptionIfNull(itemId);

            //校验是否允许更改库存
            stepCheckModifyQuantity(… );

            //更新宝贝库存
            doModifyItemQuantity(… );
        }catch(ErrorCodeException e) {
            return injectErrorCodeTo(result, e);
        }catch(Exception e) {
            throwIcException(e);
        }
```

3. 类

继承：继承自拖拉机，实现了扫地的接口。

封装：无需知道如何运作，开动即可。

多态：平时扫地，天热当风扇。

重用：没有额外动力，重复利用了发动机能量。

多线程：多个扫把同时工作。

低藕合：扫把可以换成拖把而无需改动。

组件编程：每个配件都是可单独利用的工具。

适配器模式：无需造发动机，继承自拖拉机，只取动力方法。

代码托管：无需管理垃圾，直接扫到路边即可。

包含的内容太多, 就**拆分**它

对外暴露内容太多，封装它

```
public Map<String, String> getFeatures()
{
   return features;
}
```

```
features = item.getFeatures();
 if (features == null) {
          …
 }

features.get( “prop” );
```

```
item.getFeatures().get( “prop” )
```

4：注释

那些令人喷饭的注释

```
//          .==.        .==.
//         //`^\\      //^`\\
//        // ^ ^\(\__/)/^ ^^\\
//       //^ ^^ ^/6  6\ ^^ ^ \\
//      //^ ^^ ^/( .. )\^ ^ ^ \\
//     // ^^ ^/\| v""v |/\^ ^ ^\\
//    // ^^/\/ /  `~~`  \ \/\^ ^\\
//    ----------------------------
/// HERE BE DRAGONS
```

```
//When I wrote this, only God and I understood what I was doing
//Now, God only knows
```

//当我写这段代码的时候，只有老天和我自己知道我在做什么。
//现在，只剩老天知道了。

```
// I am not responsible of this code.
// They made me write it, against my will.
```

//我不对以下代码负责。
//是他们逼我写的，是违背我意愿的。

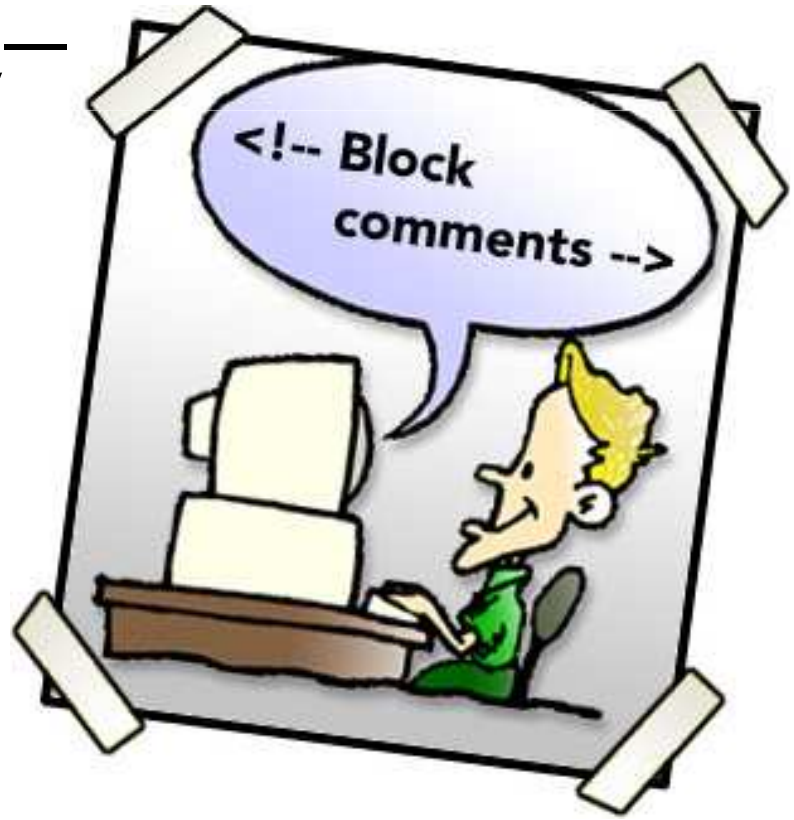
注释尽量做到简洁

绅士的演讲，应该像女人的裙子，越短越好。

--林语堂

最好不写注释

- 写注释说明表达意图的失败
- 糟糕代码是注释存在的动机之一



必须写的注释

- ◆警告他人
- ◆版权协议
- ◆公共API

写注释的**注意事项**

- 无法自注释的情况(**副作用**)
- 注释与代码**同步**
- 注释只说明**Why**

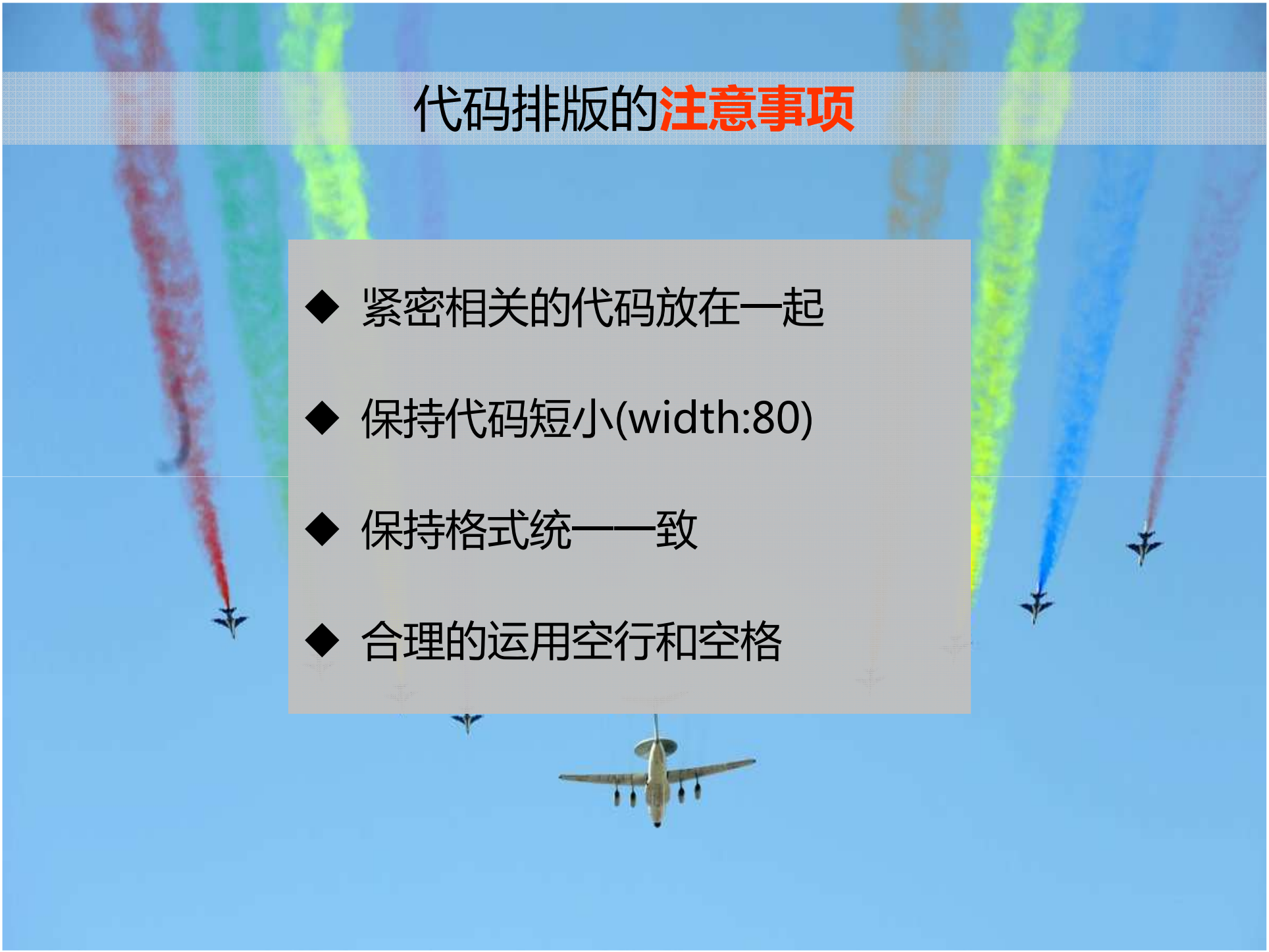
5: 格式

“对代码采用一致的格式，这不是浪费时间的愚蠢修饰，而是一种重要的**交流工具**。”

--Andy Hunt《程序员的思维修炼》

像报纸一样，头版只放重要信息，细节放里面

选择一款赏心悦目的字体



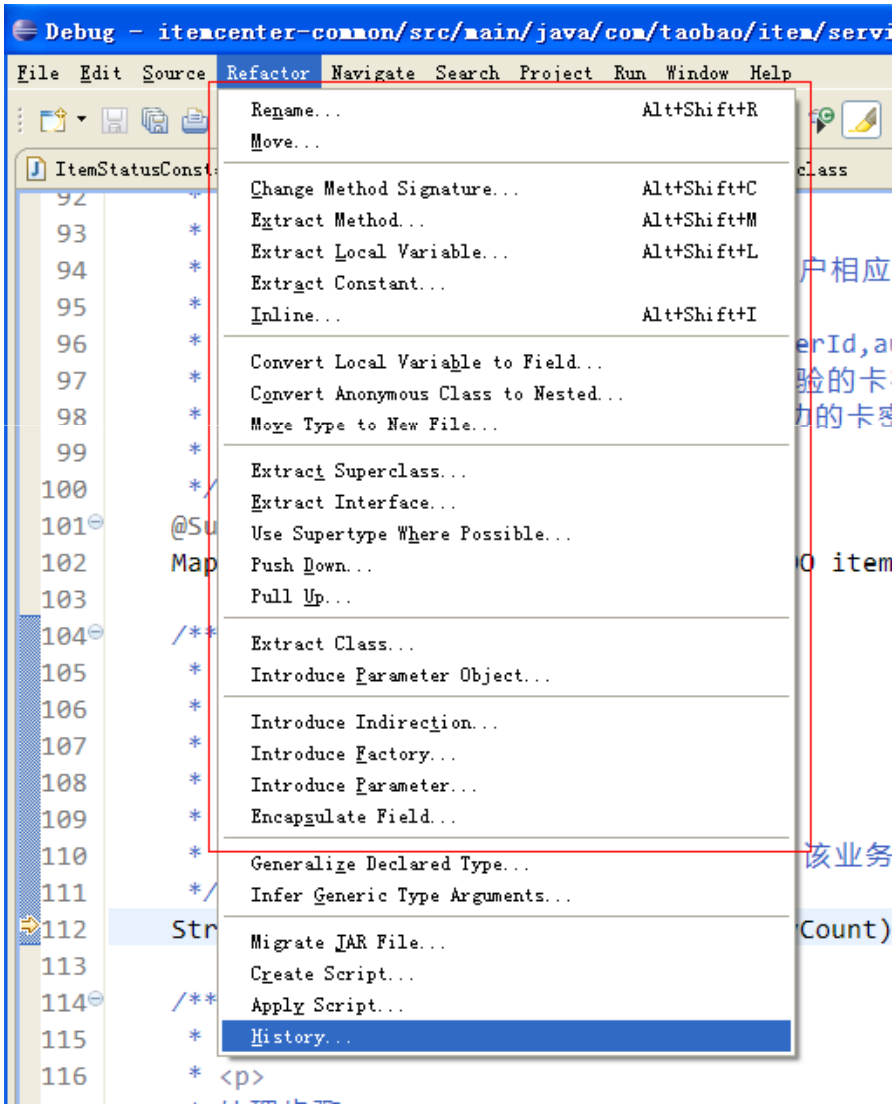
代码排版的**注意事项**

- 紧密相关的代码放在一起
- 保持代码短小(width:80)
- 保持格式统一一致
- 合理的运用空行和空格

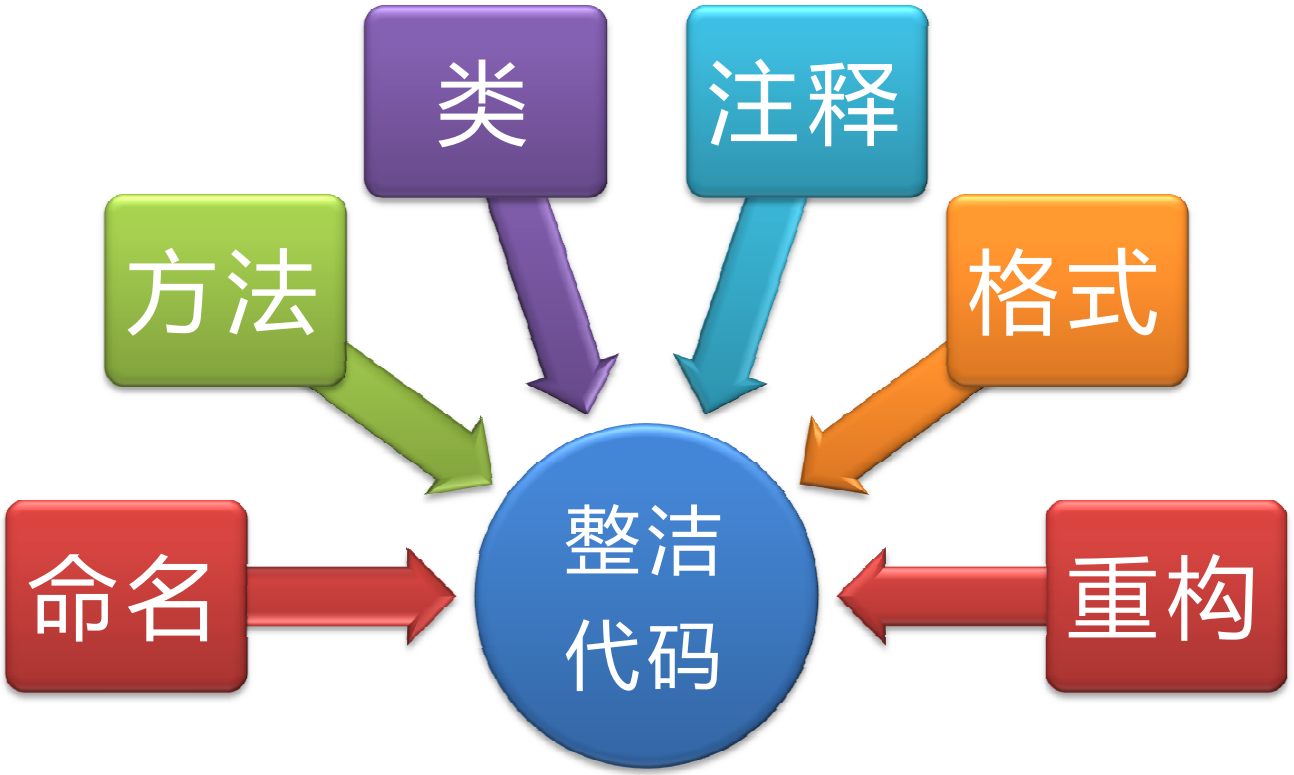
6: 重构

允许先写肮脏的代码, 但必须重构它

尽量借助工具重构，避免人为错误



持续改进, 避免破窗效应


类
注释
方法
格式
命名
整洁
代码
重构

参考资料


Sun 公司 核心技术 丛书
Effective Java中文版
Effective Java
Robert C. Martin Series
Clean Code
A Handbook of Agile Software
The Pragmatic Programmers
The ThoughtWorks Anthology
软件开发沉思录
实现模式
Implementation Patterns
Foreword by James O. Coplien