1. 第一个案例 全局查找，更改变量名（需要一定的筛选，用正则可以保证无错）

提炼出函数exact

Set函数 get式函数 语句是读取型还是修改型

把语句拆分出来 命名成函数，函数名： 是做什么的函数 ，而不是怎么做

拆的代码块 用到的 其他局部变量和参数---作为参数传入新函数 部分只在代码块内使用的临时变量定义直接封装到函数内

委托：函数中使用别的函数，对象委托，函数委托

用查询函数代替临时变量 getA()

即使单语句，如果不好理解，也应该拆成好理解函数

如果是一个很长的调用，一个特殊的转换技巧 比如反转字符串可能一行就行，但是要连续调用方法 document.getElementById().innerText.toStrring().sprit().Reverse().join()

看不明白一个语句和一堆语句在干嘛的时候，就拆成函数

使用多态取代switch条件 把case对应的语句 封装成不同对象的相同方法，然后不同对象调用相同方法

2、准则：小副修改，小副测试

二、重构原则 不改变行为 提高可理解性 调整代码

消除重复代码 更好理解 找bug 提高编码速度（避免bug，不好理解）

重构时机，随时

添加功能时重构

Debug时重构

复审时重构 xp搭档编程

重构增加了间接层： 逻辑共享，**代码可以多处共享**（提取通用代码），分离设计和实现，条件逻辑变成多态消息message, 隔离变化（多处引用，只改一处）

Message：对象的古典意义，调用函数就是对对象发送消息，对自身发送？因为使用了自己的变量 如果方法中访问了其他对象的方法，这就同时对自己和别人发送了消息

和数据库的耦合 系统分层 降低和om的耦合 插入分隔层

对象的接口修改，分开修改接口和实现 就是，修改函数名称 对于已发布的接口，让旧接口调用

何时放弃重构----进行重写 ，应该先拆分组件 然后决定是否重构

2．6 事前设计 设计接口，变量 设计接口调用关系

软件设计比编码重要 预先设计+重构

7、**函数中调用外部函数本身也是一种耦合，如果用高阶函数，函数对象的形式传入可以更灵活 地操纵 ，但是没太多必要， 与其重创建一个函数，传入，还不如直接改调用的函数**

**高阶函数的应用场合不是这个**

**灵活有时候意味着复杂**

8、工厂方法获取单例对象 取得全局的一个对象

9、性能，效率 编写快速软件 时间预算法 每个组件时间资源分配 不能超预算

持续关切，程序员一直注意性能

（3）后期调整性能， 先进行良好的结构分解

性能测量工具 巨大字符串 io 多线程思维

1. 代码和坏味道

1、重复的代码块 同一个类两个函数拥有相同或相似表达式 关联使用的对象方法中有相同表达式 提炼exact 一个中传入另一个对象的提炼函数

如果代码之间相似，分离相似和差异，再提炼

模板方法

不相关的类中重复代码，提炼出单独的对象和方法 在调用处引用该对象 bus对象？

2、过长函数

不断**委托**给其他函数 小型函数 实现的解释代码、共享代码、选择能力（多态和好修改）

**小函数 需要好名字**

**如果你觉得需要注释说明的时候，就把需要说明的代码封装成函数， 函数名称来解释用途，函数名长度无所谓**

一般，直接提取 代码 有时候局部变量声明在前面 ，有时候代码中用到参数，有时候有临时变量 作为提炼的参数传入？

提炼意思不明白的语句

提炼条件句 提炼循环语句 连同if for提炼到一个函数 如果内部仍然不清晰，需要继续提炼

3、过大类 里面很多实例变量 把变量提炼到新class class代码太多，提炼到新类

4、过长参数列

自己的对象属性可以直接用 其他的变量需要用参数传入

可以给函数传入对象参数，数组参数 ，可以考虑把参数列封装在某个结构中

5、发散式变化 需要多处修改

6咸蛋式修改

7、依恋情节 一个对象的大半都是使用另一个对象的特性，自己的比较少 这个函数是否应该转移到另一个对象上？

8、数据泥团 定义的数据 被使用的数据（值域） 差异 把经常使用的数据封装，保留一部分

9 基本型别偏执

使用小对象 class Point {x,y }

可以尽情把需要组织在一起的数据用对象表示，把对数据的处理变成方法 getX()

再小的对象也可以用

用对象代替数据值

用对象代替数组 用对象代替参数 用class代替类型

10、少用switch 修改switch switch的选择 经常根据类型type 或者数值来选择

用多态代替条件

引入空对象null

11、平行继承 12、冗余类 13、没必要的预先设计 抽象类 移除无用参数

14、另人不解的声明 没被使用的变量 15 过度耦合的消息链 a.request(b){ b.request(c) { c.request(d)}} 这种层层耦合的调用链，一但中间有中断就有问题

函数链是坏东西吗？

16、过度委托 方法委托给其他对象方法

17、狎昵关系 彼此探索private

18、不同函数 做同样的事 19、 不完美的库函数（扩展而不是修改）

20、数据类 get set 可以考虑加入其他功能？

21、继承不完美 不愿意用父类接口 却使用父类实现

22、过多注释 可能是代码太糟糕

1. 测试体系
2. 自动测试代码 每个类有一个测试函数 .test .main

.main中对于单元中的函数逐一测试 使用循环？随机数？

整组测试（）

测试代码最好时机实际上是编程之前， **集中在接口上** 频繁测试

每个对象main不太好，建议用独立的测试类 junit suites套件测试用例

Assert() 需要 显示，错误发生在哪个类，哪个函数，哪一行，用的参数是什么，结果是什么

大部分assert 是比较值相等 失败 用循环和assert（）实现测试

Assert一个是函数的实际输出，一个是预期输出 在for中生成或者遍历 预期输出必须是可靠的 应该使用不同的方法生成 也可以自己写个数据结构

多个输出就需要多个assert

比如文件内容读取 ，需要测试文件的信息，文件名，长度，内容等信息 需要一一检查是否正确

**测试 需要寻找边界条件，考虑可能出错的地方，集火**

**测试 是风险驱动的行为，需要自己先预估可能的风险**

**有可能有哪些错误，文件读取不到？文件没读入？**

**考虑预期错误**

**需要各种返回值都测到**

Assert的报错需要自己实现

1. 测试什么，太简单的一般不用测，需要测的是风险驱动

**测试不可能捕捉所有bug**不完善的测试 也比没测试好

捕捉异常exception

1. 单元测试 只对一个组件负责
2. 功能测试
3. 重构名录

重构名称name 概要 场景，做的事 动机 为何用 如何做

Uml图 重构前后

2、如何寻找引用点 一个函数 或者变量（值域） class的引用处

用文本查找工具 或者借助ide

1. 小步进行，频繁测试
2. 函数重组

提炼函数 代码块变函数 内联函数 函数变代码块

临时变量：用查询代替临时变量

1、提炼 粒度小 容易重写 函数名应该准确，而不是看长度

需要有意义命名

**千万不要让提炼的函数内部直接使用原函数的局部变量，这种作用域穿透是有问题的，只能用全局变量，或者传入的参数**

如果引用的全局变量，外部对象，或者函数，那就直接复制

是否引用了局部变量和原参数，作为参数传入（临时变量除外）

是否有临时变量： 只用于提炼代码，就把声明提到提炼函数

如果代码存在对原函数局部变量或者参数的修改（set） 可以处理为查询return 该变量 ，在调用时赋值给原变量

如果不好return 或者返回的参数多

存在修改的时候：如果原函数后面没用到该变量，就不需要管

如果原函数后面还要用，需要返回修改值

如果修改了多个变量：分成多个函数 或者返回一个集合返回值，在外部解析

如果参数被修改

如何减少一个函数的局部变量 ：用查询代替临时变量（简单来说把临时变量最终需要的结果封装成函数过程，返回） 这个查询函数内部怎么组织，看情况

2、inline 在调用处把代码块释放 当非必要的封装的时候 ，当函数需要重构，而调用间接显得不清晰时

太多委托，无用间接层

确定函数没被继承 找到调用点， 释放代码 删除原函数（调用和定义）

3、inline temp 临时变量是简单表达式赋值

S=a.toString() 把s 替换成 a.toString()

一般这种临时的局部变量没坏处， 但是可能影响提炼函数

配合 用查询代替临时变量 把用临时变量表示的表达式结果

变成一个查询函数 这个表达式要简单

然后 ，直接用函数代替局部变量 ，不要把返回值赋给其他变量

Return a.toString() var a = search() + a ;

限制 变量只能赋值一次 用final确认

赋值超过一次 分割成多个变量

1. 复杂表达式 运算结果 赋值给临时局部变量 和上面相反

可以用来表示条件子句 var notbad=a>b||b<c

可以用来表示长运算 的步骤 sum=1 + 2 + 3 + 4 + 5 + 6

这很常见， 不过用提炼函数解释代码意义更好 isok() sum()

什么时候用呢？ 如果提炼函数需要更多功夫时 大量局部变量， 先用新的变量理变量的关系，考虑完代码逻辑后 把局部变量，换成查询函数

1. 剖解临时变量 被赋值超过一次 不是循环变量 集用的变量

那就分成多个变量

7、代码对参数赋值 移动参数到变量

把参数赋给临时变量

参数不作为左值，只作为右值出现

注意，如果对对象参数赋值，会直接修改了引用，指向另一个对象

Java是值传递，修改参数不会对调用端有影响 ， 比如把对象参数指向别处，外面的对象变量的引用是不会变的

Java中不要直接对参数赋值，只应该读取参数，使用对象参数的方法， 或者修改参数对象的属性

把所有对参数赋值改成对局部变量赋值

所以如何提炼函数： 原函数中参数是否修改 提炼函数的参数是否修改

原函数：需要修改参数的时候 移动到局部变量

如果提炼代码使用到了参数，原参数传入到提炼的参数

不需要修改的话,提炼函数内部直接用参数名

如果需要修改，就把参数赋给局部变量 修改后返回改局部，赋给外层的代替参数局部

这里又有一个问题，提炼函数的参数 包括的是原参数 和原局部变量，

这样传进来又成为了参数， 如果要对原局部变量修改，还需要把局部变量代表的参数再赋给新的局部变量？

一直用局部不麻烦吗？ 过于麻烦的修改是无必要的 直接修改参数也没什么太大问题

另一种想法：直接给提炼函数传入参数局部变量，所有的传入值就都是局部变量

统一地处理

8、方法对象 把函数语句单独作为一个对象，所有的局部变量和参数作为属性

9、算法替换 有时就是要替换所有的算法 ，但是在此之前应该分解成小函数

1. 对象间移动属性
2. 移动函数

当大量引用了其他对象的函数时 ，在另一个对象中把这些引用封装成函数，然后原对象的旧函数 调用该函数就行

太多合作形成的耦合

移动时考虑是否移动自己的属性 还是作为调用参数传入 ，如果自己留着没用就搬走

函数最好没有继承性，不然只能重写，不能直接删除

1. 移动属性

如果属性被其他对象大量用到，自己用的不多

在新对象中创建同样的属性，然后自己引用对方的

3、提炼类 类功能太多，应该分拆 应该明确抽象，有明确的责任

不断扩展，增加属性，增加方法，可能导致类太复杂

分离责任 ，给新旧类重新命名 建立两个类的访问关系 比如对象型属性 可以考虑双向连接

1. 内联类 类功能太少，将其移动到另一个类
2. 隐藏委托关系
3. 移除中间人
4. 引入外加函数
5. 重新组织数据

是直接访问 还是通过函数访问

用简单数值表示概念还是用对象？ 哑数据，会说话的数据

关联对象 把数组换成对象 去除魔法数 双向关联对象

型别码

1、自封装属性 只用get set 直接访问好阅读 好用的地方，可以改写行为

在继承上好用

经常有 **继承父类的部分数据属性和方法，同时又有覆盖的需求，同时还需要用父类数据和重写数据，方法交互**

1. 对象代替数据

如果数据需要额外数据或者行为 比如需要格式化，转字符串

包装对象

1. 实值对象变引用对象 需要控制者？ 实值不可变，不在现有的对象上修改

引用对象

1. 引用变实值
2. 对象代替数组 属性代替 元素 针对对象数组，需要命名 给元素加上set get
3. 复制被监视数据 gui控件中的数据 想用其他类来访问 使用观察者模式，同步控制双方数据

Gui和 业务分离，但是数据往往会出现在两边 mvc模式

如果分离gui和业务 ，需要把数据分成两份，出现在两个地方，然后同步

发送通知

Observal 和事件监听器的区别

1. 单向关联改双向 对象需要用到彼此特性，只有一条连揭

反向关联

需要同时更新两者

1. 双向改单向
2. 用命名常量/变量 代替字面量 魔法数

If (a.type==1) 这种完全从表面看不出意思的数字，就是魔法数

又比如 9.81 pi 3.14 替换所有魔法数 为有意义的变量名

Path

1. 属性封装 public 改private get函数 如果只有get set就是哑类，浪费

找到引用属性的地方，换成get set

12、封装群集 集合 collection list get set不应该直接返会list,set， 而是封装add,remove 封装数组 赋值

13、 记录结构 struct{} 或者csv文件等 ，这些外来数据 用哑类取代 get set 用属性列出每一项

14型别码 type code

所谓型别码 就是用数字序号代表”类型” if( a==O) 将之封成对象

用子类代替 用状态对象代替 状态模式

16、用属性代替子类 如果子类只是为了实现常量函数，根据不同类型返回不同常量值。。。。那就没必要存在 return “a”;

九、简化条件表达式 分解条件 相同结果 条件中重复

是否用单一出口？

1. 条件式分解

If ( ) 把条件语句提取成函数 ，条件表达式也提取成函数

**即使条件短，但是如果无法表示意图，也应该提取**

1. 合并条件式 多个if 结果相同时 把 if的条件合并 && or
2. 合并重复的条件片段 后面的语句中有重复的部分，可以抽取成函数，或者是

把代码搬到条件外执行，可以在前，可以在后

条件变化但是行为不变

4、移除控制标记 for() { if() } 用break return代替

5、if else if 多层嵌套改成 单项 if () return if () return

卫语句：如果

多个return 不仅仅只一个 if return if return

条件的情况：罕见条件，单独检查（反向检查） 返回 ; 正常行为

可以用逆反条件实现卫语句

1. 多态实现条件式

Switch( a) 0 1 2 3 根据型别码

If(type=”plane”) 根据型别 字符串

多态，不同对象，同一接口

8、引入null对象 检查对象是否存在 一个自定义的null ，用来取代返回的null

这个null对象有isNull() 方法 标准返回的null没有方法

必须找到所有的 检查对象是否空的地方

1. 引入断言

必须满足某个条件，代码才能运行，一般代码看不出来

如果代码假设了某个条件始终为真 用assertions说明

检查“一定，必须为真” Assert.isTrue(条件) 如果出错就引发异常，没出错就始终没异常

不改变任何行为

十、简化函数调用

修改函数名字 修改变量和类名

增加 减少参数 缩短参数列 创建或者使用对象参数

不可变对象和并发编程

将查询对象状态和函数和修改函数分离 modify

用异常对象代替错误代码

隐藏可隐藏的函数

隐藏转型动作

1. 重新命名函数 需要好名字 必须要非常好的名字
2. 增加参数 当需要修改函数，需要更多信息的时候，可以传入一个对象参数
3. 修改函数时，不再需要的参数删除
4. 查询和修改分离 函数不应该又返回属性 又修改属性

多线程并发时 同时检查赋值

1. 函数工作类似， 使用的局部变量不同 合并同功能函数，用参数代替变化
2. 函数根据参数不同 而 swith 不同语句 每条 建立不同函数，和上面相反，把大函数根据条件拆成小函数
3. 传入整个对象 ，代替传入的多个对象属性
4. 函数返回值作为参数时 ，可以考虑直接在另一个函数中调用该函数，多余参数不一定需要 不过会增加耦合
5. 创建一个参数对象 包装所有参数 range模式，范围对象

10、移除setter 如果对象属性不会改变，只在最开始设置，就不需要setter

11、隐藏函数 private 可能行为增加之后就不用get set了，可以删掉或者private

12、工厂代替构造 create 根据不同的参数，不同的创建行为 单例

可以根据 typecode 数字 或者string字符串创建 创建不同子弹

**可以很方便地制造想要的某种对象 比如不同子弹，不同飞机（只有属性值不同，行为相同？）**

**或者创建不同子类对象，属性行为都不同**

**明确函数创建子类对象**

1. 封装向下转型 移到函数里面

**向下转型，是父类强制转为子类 需要向下转型的时候，往往是因为之前的操作进行了向上转型，导致了接口丢失**

13、异常代替错误码 throws Exception { 可能产生某种异常

异常实际上是种if语句 创建异常对象 非检查还是检查 异常对象的函数重写

1. 测试代替异常 异常不是条件的代替品，只有一些罕见的错误才用异常， 普通条件检查不应该用异常代替

十一、处理概括关系

将函数上下移动于继承体系 上拉属性 上拉方法 下推方法 下推属性 模板方法

提炼子类 提炼父类 提炼接口

继承不一定是好选择，需要的是委托 ，继承改委托 委托改继承

1、上移 多个子类有相同方法

2、下移 父类方法 只和部分子类有关

提炼子类 以下都属于重新创建类

父类提炼

提炼接口

1. 继承折叠，父子没区别 就合并
2. 模板方法 子类类似的函数提升到父类

12、委托代替继承 子类不太需要父类数据 只用了一部分父类接口

去除继承，改为委托

1. 继承代替委托

十二、大型重构

这才是最重要的

1. 梳理 分解继承体系 一个继承体系有两个职责 拆成两个体系（两组类和子类）

混乱的继承是毒草 不断加子类 逻辑越来越混杂

1. 过程化的代码 改成对象类型代码
2. 视图和逻辑分离
3. 提炼继承体系， 用不同对象代替条件

十三、现实 不愿意重构 谁付钱？

短期利益

安全性 安全重构

十四、重构工具

自动化重构工具

程序数据库 ，交叉引用 动态编译，修改就转为bytes grep工具

解析树 parse tree对每个部件分类

十五、重构时

发现错误，先记下来不要改