二、隐喻

三、事前准备很重要 写个设计文档，思考需求如何实现？

架构师做需求 ，设计师取架构，程序员实现设计

好的项目规划， 变量，函数，关系

开发项目必须要计划，文字形式，图片形式，代码形式。从最上层考虑 ，把方方面面考虑清楚

想好**要做什么**，再思考怎么去做 也就是草图和细节图

软件类型：网站，库存管理系统，信息管理系统，工资系统 ，游戏。嵌入式，webservice

生命周期：敏捷开发，极限编程 分阶段交付 渐进交付 螺旋型开发

计划：预先计划 增量式计划 变更控制 随需 QA

需求：形式化 非形式化 需求评审检查

设计

构建

测试和QA 自己测试 测试先行开发 测试小组

部署

迭代开发也需要前期

3、问题定义：产品设想，任务陈述

从客户角度描述，不用专业知识， 需要点网页的连接，输入账号，密码注册

4、 需求， 需求定义

软件应该做什么，应该在需求阶段检测错误

充分详细地描述需求

一个软件系统 ：输入（用户触发）来源精度范围 ，输出目的地，精度

输出格式：string 文件 html?

外部接口 ：（不是内部接口）， 让用户使用的接口，通过按钮或url触发

是否列出了用户的所有行为

每个接口用到的数据，收到的数据

安全性

可靠性

5、架构 程序由哪些模块组成，模块由哪些小模块组成 模块负责哪些工作，和其他模块交流越少越好

类设计， 数据设计（主要针对数据文件，数据库，表，变量应该设计在类中）

业务规则（某些必须的规则会对业务产生影响 ）

Gui设计，用户界面

资源管理

安全性

性能

可伸缩性：未来需求 互用性：和其他硬件软件交互

国际化 界面显示 ： 是把字符串硬编码，还是用某种特定的数据结构来保存界面字符串，或者使用资源文件（安卓）

IO

错误处理:九成代码用来错误处理?

三、构建

1、选择编程语言 2、**约定编码规范** 3、最新技术和工具

4、自我限制，哪里只能读，哪里只能写，让层次清晰，设计上不混杂

不应该受限于语言，语言不支持的功能，要么用限制实现，要么自己想怎么实现

5 进行设计工作 一部分设计可以在编程时同时进行 **模块，输入，输出**

**编码约定，变量函数名称，注释，代码格式 规定一些编程实践，如何处理错误，如何处理安全（非法访问）**

**写单元测试吗？设计测试 单步 集成测试 复审他人代码**

**版本工具 其他工具**

五、设计 伪代码写接口，面向接口编程 画类关系图

1. 设计是会经常修改的，这是开发中常见的
2. 设计阶段是会经常犯错的，需要不断修改
3. 设计的好坏不好取舍 4、设计存在资源限制，时间规定，空间规定，性能规定
4. 设计需要试验

**2、软件的首要使命是管理复杂度**

偶然的难题 本质的

技术原因失败 ：太过复杂 任何人都不可能记得程序的全部，应该要合理组织，让人关注程序的某个方面

分解成多个子系统

降低复杂度 ：作出最明了的设计，不要使用太多技巧，要更直观，更清晰

要好维护 要松耦合 可扩展，不修改原有程序 可重用

层次：软件系统-子系统和包-类-程序-语句 从逻辑上来说应该分成**功能模块**

**常用的子系统： 业务规则（面对现实问题，需要的是什么？）**

**用户界面**

**数据库访问**

**系统的通信**

3、如何设计模块

找出**现实问题**的对象和人造对象 先看用户要干什么， 他要获取什么信息，需要修改什么信息

是员工，是账号？ 是画笔？ 是游戏角色？是键盘？

设计：归纳出对象的属性 确定可以的操作 哪些东西可以让其他对象看到？

设计对象的公开接口：可以是方法，也可以是public属性

4、抽象 抽象是对事物的属性 概念进行统一 某个东西，有窗子，是玻璃，水泥做成的，有宽高，方形，里面可以住人， 那它被抽象为房屋。。。

抽象可以理解为**命名**

抽象是不可能表示出事物的全部属性的，比如上面的例子，我们没说名房屋的质量，房屋的颜色，房屋住了几个人，有几间卧室 不用考虑到元素，原子层面

**忽略无关细节，只用部分属性进行处理**

5、封装是为了不对其他对象暴露属性

6、继承如果能简化设计 或者更加清晰就用继承

继承常用在多态上，或者是留下扩展空间 或者设计了**模板方法**（ 在安卓之类的框架中很多，还有某些游戏框架， 构造器需要使用super() 来继承父方法 ，重写方法中需要使用super.func() ）

**模板方法：父类实现一部分，子类不必再写的方法，这些方法中调用了抽象方法**

**程序员只用重写实现抽象的方法就行了。。。安卓开发中很常见**

继承另外的作用，对方法和数据的扩展， spring/php 控制器类生成不同对象 不同的控制器方法 映射不同的url

7、信息隐藏

把部分接口暴露 部分隐藏， 设计时思考哪些暴露哪些隐藏

接口本身要隐藏内部工作机制 接口调用隐藏的方法（内部接口）

复杂度的根源：数据类型，晦涩的算法， 条件判断，过长代码

数据不应该过于分散 硬编码数字应该作为常量存在，全局数据

循环依赖：A调用B的方法，B又调用A的方法 这样双向的不好通信

多使用类内数据，不要用全局

8、找出容易变化的区域 （对可能发生变化的需求进行预测）

界面的可能变化

数据的可能变化

功能的可能变化

容易变化的地方 分离，然后提升可扩展，易修改性 继承，多态对象，重写方法

业务的变化（修改或扩展） 硬件的依赖变化（不支持，或者扩展）

界面输入输出的值（文件，string等） 状态变量

9、松散耦合 **类和程序要尽可能小 直接 清晰**

**少依赖其他模块，关联模块的数量， 关联数据的复杂度（单参数，多参数） ，关联的形式（参数）**

减少参数

1、耦合规模 模块的连接数 只有一个参数比较好 公共方法少比较好

2、可见性， 通过参数表比较明显， 全局数据不太好

3、灵活性 容易调用的最灵活 调用另一个对象时，接口的用意是否清晰， 能从函数看出读取了对象什么属性吗？ 不要传入太多对象参数， 减少内部语句的复杂度。。。拆分小函数

耦合 简单数据耦合 模块间通过参数传递数据，传递的是数字等数据

简单对象耦合，实例化另一个对象 普通传入另一个对象参数呢（不是组合）

对象参数耦合 传入第三个对象参数 耦合比较强

语义耦合：

传入控制标志 if(){} 可以传入true/false 或者数字来控制另一个对象方法内部如何运行。。。这不好

第一个模块修改全局数据，第二个模块使用全局数据 要求合理修改，顺序执行

方法执行有时间顺序要求

传一个对象参数给第二模块，只使用部分数据？

传入一个转型过的父对象，在第二个模块中被重新转型回来

一个模块同一时间不应该关注多件事的，要最方便使用，理所当然地使用

1. 使用设计模式
2. 其他 高内聚 类中的方法都在支持一个**中心目标**， 目标越集中，功能关系越紧密，内聚越高
3. 分层结构 ，分层非常重要 大纲目录才能增加可读性，**层次越多，同时需要关注的就越少**
4. 契约 设计类接口 需要的**外部输入参数（内部数据由属性提供）** 决定的输出
5. 分配对象职责 10、 设计测试 11、考虑失败的可能
6. 画图 **12 设计模块化，黑盒子，输入 输出**

如何理解问题：未知量，已知量 条件是什么，能满足条件吗？

可能会用到什么知识，解决相关问题？更一般 更特殊

13、设计的循环迭代 需要在高低层面切换

14、分治 一个模块地设计

15、自顶而下 从高层模块 到类 到方法 到实现

**不断分解问题，直到觉得粒度足够小，实现很容易**

自底而上 ： 思考问题的实现，该用什么方法和工具实现，要完成什么职责 思考对象 思考模块

**16、实验原型**

**用最小化的代码，进行简单测试**

17、**合作设计**（寻求意见） 征求想法，白纸画方案问怎么样 开会讨论，论坛提问

18、要做多少设计 理论上越深入细致， 最后设计的越低越好

到函数内部语句都要有一定的想法 ，类似于模板方法，写成伪代码，伪功能

19、记录设计结果 设计图 设计文档 设计注释

**设计的核心是管理复杂度，最终的软件越简单越好 ，要拆分出清晰的模块**

六、类

早期基于语句 思考问题 后来基于函数思考问题 现在是基于类思考问题

**把问题抽象出对象，思考对象之间的联系 对象联系需要发送消息，发送消息就需要调用方法，调用自己的方法是对自己发送消息，调用别人的方法是对别人发送消息**

**数据和程序都有自己的职责**

1、抽象数据类型 ADTs 是指一些数据和对数据的操作 比如 Int String,array数组，vector集合 都是adt **类型**

**ADT的核心是对于数据的操控，这些数据可能是被初始化的 可能是从外面引入的（全局，或者传参）**

2、如何设计

1、方法应该明确说明自己的功能

2、设计对数据正确性，有无验证的方法 ，isArray hasFuncton()

3、隐藏细节 减少不必要的传参 ，使用内部数据 除掉不必要的参数

如果需要修改方法流程 可以不用传参控制 而使用方法接口控制

比如修改display()的字体大小 ，可以使用useBolderFont()

4、直接在外部修改属性不算什么好实现，要直接修改属性就在内部进行

5、get set方法，返回外部可控制的数据，或者数据集合 .setColor

**需要的属性 可能需要的操作 操作和属性都是可扩展的**

**把常用数据结构定义为ADT类 ，String,Interator ,Array 对象化控制**

文件类 ：文件内容 文件名 打开文件 关闭文件 交给输出流 -（读取文件行，读取文件字符）

3、创建好的类接口 设计了哪些属性

然后是属性需要的操纵方法 但是这些方法没必要全部交给外面使用 有些方法的意图也不清晰，也不知道该怎么用

太多太混

需要隐藏不需要暴露的方法 ，**提供清晰，好用的方法作为接口**

类实现为ADT，是有意义的， 如果发现其中的数据没有统一性，方法也混杂，就应该考虑分解了

不相关的数据和功能转移到其他类

抽象要清晰，意义明白

接口的名字功能要符合类的定义，如果存在不太符合定义的**辅助功能**，比如子语句封装成函数。。。这种类中只是用来辅助的方法，就完全不能暴露在外

比如People.countNumber() 被people.getNumberOftheWorld()调用 前者是不能暴露的

不要暴露太多接口

成对的功能 get set open close execute undo

修改扩展时不要破坏接口的含义，必须和类抽象相关

不要添加和类含义不同的属性

抽象性就是内聚性了。。。

4、良好封装

可访问性

不暴露属性数据

隐藏实现细节应该连private代码都藏在下面？ 甚至把实现的方法放在指针指的地方？

5、不假定使用者 设计时按照规定的范围来，如果别人超范围就报错，不执行

If(>0)return

6、让阅读更容易

7、语义上破坏封装性 ——简单来说使用接口不应该依赖内部实现，如果一个方法的文档和名字没告诉你他有什么功能，就不应该依赖它代码内部的功能

比如openfile() 内部代码如果没有文件就会创建一个新的，但是文档没说，你就还是需要createFile()

或者一个接口会调用另一个函数，文档里没说 而你需要用到另一个函数 ，于是只用这个接口

读源码是为了学习，不是为了使用

8、设计和实现

包含关系 has a 类拥有一个数据，或者一个**其他对象**

**有一个——文件夹类里面有一个文件对象**

如果属性不方便是对象，万不得已 可以用private继承实现，

类属性超过七个，考虑拆分类

继承 is a 一个类是另一个类的特殊化

方法、属性对子类可见吗？ 可以覆盖吗？还是必须重写？实现？

**里氏代换原则：必须能调用继承的所有子程序 接口**

**只继承可继承的部分**

**不”覆盖”不应覆盖的部分 ，没继承的方法，不要重写**

**共用的接口，数据 ，操作放在最顶端**

**思考只有一个实例的对象是有必要的吗？ 大部分时候还是必要的，有时候可以用另一个对象的数据属性和方法代替。。。。但是如果有专门的意义，还是应该封装为对象**

只有一个子类的父类有必要继承吗？ 仔细考虑，看情况

用多态代替类型检查，条件判断

9、方法和属性

方法越少越好

减少对其他对象方法的调用

可以禁止构造函数， private 单例模式

减少实例化对象属性的种类，减少其他对象方法调用，减少调用其他对象方法的方法

1. 构造函数

构造函数中初始化

单例模式：禁止构造函数 getInstance **设置一个静态成员变量，指向自己的实例**

**Instance 然后每次调用getInstace 检查该实例是否存在，存在则返回**

实现深拷贝 分别叫引用对象和值对象？

1. 类的建模

对现实对象建模

对抽象对象建模，如形状Shape 从现实存在的事物向上抽取共同点，如奔驰车和车，车和交通工具，交通工具和 物品，

Event 事件就是抽象概念

降低复杂度 隔离复杂度 隐藏实现 限制变化影响范围 隐藏全局变量

参数传递更流畅 创建中心控制点（一个接口调用多个私有方法）

易重用

封装一系列相关的操作和数据

1. 包： package

类的名字，中心目标，接口是否清晰

七、子程序 （方法） 函数 过程

程序名字：这个程序要干什么

文档

代码布局风格

读写了全局变量

应该只有**单一的目的 ，如果一个程序有多个目的，应该拆成多个函数，如果一个程序的目的很复杂，应该拆出多个子程序内部调用**

**使用了魔法数字，没赋给变量**

**防止输入参数的边界错误 /0 等**

**不应有未使用参数 （除非有默认值）**

**参数应该少**

2、使用函数的原因

降低复杂度 （拆分） 函数名方便理解 减少重复代码，复用

方法越短，分得越清楚，子类重写越容易

隐藏代码顺序，实现 隐藏指针操作 可移植性

**把布尔条件判断放进函数中 isTrue() isHitTest()**

**函数应该写多长：**

3、 简短的表达式 就算是一行或者多行，也应该设成函数

比如使用三个变量的表达式，谁都能看懂，但是还是不如一个函数名好懂

需要个好名字，可以自我注解

简单操作有时候也需要对边界条件进行判断， a/0 ，这时候可以在函数中写入判断代码

4、设计

内聚性 函数代码操作的联系 一个函数应该只为了一个目标，里面的数据也是为了这个目标 一个函数应该只有一个功能

顺序上的内聚性，代码是按照顺序执行的？

逻辑内聚???

5、好的程序名字

描述程序做的所有事情 需要完整详细，不要在乎长度 downloadHTMLFromTheWebservice() 描述输出结果，描述

但是也不要长出天际，好几个单词， 还是需要易读性，最好是两三个单词

避免意义模糊，表述不清的动词 动词越精准越好get 和download的区别

程序名不要用序号

要描述返回值

使用动宾语法： 动作+动作施加的物体

使用成对的命名 add/remove insert/detete begin/end destory/create show/hide

Min/max old/new lock/unlock next/previous first/last

设计api文档

6、函数应该多长？

最长应该1-200行 但是需要提高内聚，提高嵌套层次（拆分成多个子函数调用），减少局部变量，**提升代码易理解度**

7、如何使用参数

用途： 输入-修改-输出

被不同函数使用的类似参数，应该按照一样的顺序出现

函数中定义的参数必须被用到，没用的参数不需要

传入状态参数时，应该在最后，true,false

**传入的参数不应该作为中间值，参数本身甚至不应该被修改，它应该提供数值，如果要返回，应该使用另一个变量作为返回值和中间量**

**但是如果修改的是对象属性，那是另一回事。。。。建议无论如何不要修改传入的参数本身**

**对参数的命名 ：应该考虑其意义， 输入还是输出，范围 可以考虑加上前缀，**

传递对象参数：是直接传对象的属性，还是把整个对象传进去？ 如果用不了那么多属性呢？ 应该考虑程序的意义， 是只需要数据， 需要普通数据，还是需要某种有意义的数据比如员工工资， 还是想对对象进行操作？

简单来说，看函数需不需要知道对象。

尽可能让形参和实参的名字一样**，就是调用时不要传入简单值，而是传入变量名**

**类型必须匹配**

6、函数只有一个返回值 永远应该用返回值命名

7、函数返回值 可能有多个返回路径，需要梳理无措 不要返回局部变量的指针

8、宏程序 ：宏会把所有同名变量，替换成他指向的语句，所以宏可以当成函数，但是不安全 展开是简单的文本替换

用const代替宏常数 用内联函数代替展开内嵌函数 模板实现泛型 emun枚举

八、防御性编程

1、避免非法外来数据 非法url等 检查值在合法范围 检查函数输入参数的值 检测到错误值如何进行错误处理？

2、断言 断言在没错时不会执行 如果发生了就报错 产品生成后删除

自己写个断言 if不满足条件 报错，并显示行号等信息，退出程序

错误检测代码是检测可能发生的 断言是防止绝不应该发生的 ，代码不要放在断言中 断言检查前条件和后条件 调用前必须为真的属性 调用后返回必须为真的行为

错误处理是发生了就处理

断言如果发生了必须修改代码

3、错误处理

返回无效值 比如false,0,””等（根据返回值类型）

文件io中发现数据错误，可以跳过

返回之前的结果

把错误值改成接近的合法值

**错误存入日志**

**返回错误码**

关闭程序

**显示出错信息**

4、正确性：不返回错误的结果，中断运行

健壮性：不断尝试其他方式解决，显示一半图像，一半文字

5、在高层次设计 设计统一的错误处理方法，适用于所有的状况

**一定要检查函数的返回值**

6、异常 try catch throw: if() throw

7、隔离程序 针对某些类，认为是不安全的 他们和核心类交流必须通过安全检查， 而这些类自身和相互通信不用安全检查

安全区域和不安全区域的设置

9、进攻式编程

不跳过错误，所有错误都应该终止程序

填充所有分配内存，比如数组全部装满

填充文件和流，检查文件错误

Case和if的错误强行让程序终止

删除对象之前填满数据

10、发布后保留哪些错误处理代码？ 检查严重错误的代码要保留

细微错误检查的代码删除

程序崩溃的代码还是算了

打入日志文件 **log**

**11、过多的防御代码也不好， 应该妥善选择防御的地方，另外防御代码本身也可能有问题**

九、伪代码编程

创建类 ：总体设计 设计方法

伪代码用中文写比较好，美国人喜欢用英文写

不必遵循任何语法规则 只用平实的语言表述你打算怎么实现程序就行了

可以使用某些变量的符号，也可以纯文字表述，或者用某个词代表变量

不必把要用的所有变量，中间变量列出来，只要能明白意思就行

伪代码描述一个语句，或者多个语句

描述要解决什么问题，怎么去解决

增加细节

不要和编程语言混杂，要用普通的语言表述

就当是写文章

1. 设计函数 要解决的问题 要传达的输入（包括参数，和传入的全局，以及属性）

要传出的输出（或者影响的外部变量）

前条件，程序可以执行

后条件，程序执行无错

2命名 3、如何测试 4、思考程序需要用到哪些库 需要用哪些听说过的功能来解决问题 查下相关库 **花一定实践查询**

5、考虑效率

6、看看算法书有什么办法？

7、考虑要用到的数据 **如何从输入数据获取输出数据 ，需要什么中间量，设计局部数据，中间量**

8、检查伪代码 9、尝试一些新想法

10、如何把伪代码转为代码？

**把伪代码变成注释，然后每句下面写上真正的代码**

**最后清除注释**

**伪代码代表的逻辑很长，可以拆为子函数，或者是拆分伪代码**

**11、代码错误基本是自己的原因**

**12、使用lint之类的工具**

**13、单步测试、 单元测试**

1. 替代伪代码的方案

测试驱动开发

重构 契约式设计

十、变量

1、数据类型

ADT string bitmap 位图 数组 指针 float b树 堆 树 结构

具名常量 布尔 栈

1. 变量定义

隐式声明，使用未声明变量 很危险 比如把变量名写错，赋值错误

Javascript可以给变量隐式赋值

所有引入的变量都先声明，提前就设计好声明

命名规则

**用ide检查变量名**

1. 初始化

一直不赋值，那值就是刚分配的内存的值

值过期 部分赋值（数组**）**

声明时初始化， 第一次使用时初始化 在第一次使用的地方声明定义该变量

Final const

**计数器 i j k 累加器sum total**

构造中初始化

编译器警告

内存访问工具检查指针？

**检查参数合法性**

1. 作用域

把用到变量的引用点和变量 集中起来 ，修改代码顺序 不要多个跨度使用到一个变量，集中化

减少变量“存活时间”：总共使用代码的语句跨度，要很快结束一个变量的使用（从开始到结束）

5、 减少作用域 在循环开始前初始化 **变量要使用再赋值**

**相关语句放在一起**

**相关语句提炼成函数**

6、持续性 生命周期 块变量 函数局部变量 对象内的变量

持久化数据

7、绑定时间

编写时 值等于硬编码

编译时 让值等于中间变量

加载时 从外部文件

实例化时 即时（交互赋值）

8、数据类型和语句顺序

比如数组，集合

序列型数据，顺序处理 循环处理

选择型数据，多选一 if

9．变量只能有单一用途 temp保存x又保存y？ 可以，如果用途是从xy中选一个的话

避免变量有不同含义 比如同时表示 数值和布尔，不存在 如果等于0，-1则不存在，这种返回值其实很常见 但是可以设置一系列状态码，0，-1作为状态码之一

10、所有的变量都应该被使用

十一、变量名

可读 易记 明了 对变量代表的事物应该做完整描述

长度不要太长，一到两个词 用直白的词

表达到底是什么，而不是怎么 名词

短变量名尽量在局部用，最好是不用

2、限定词 total sum average max min record string pointer

都放在后面，意义在前面

Num Index count

3、对仗词 begin end up down source target 看函数名部分

4、为某些特殊变量命名

循环变量 状态变量 临时变量 布尔变量 枚举 常量

下标 i j k 如果变量在循环外使用 recordCount

尽量不用i j k更好

状态变量 具体的状态 不用flag这种名字 systemInfo= osType

临时变量 保存中间值 最好不用temp 而应该命名，知道意义

布尔变量名 done error found找到 success ok 使用表示肯定的变量名，不要用否定,也不要用双重否定

枚举组 使用组前缀 常量：按照意义命名，不要用数字值命名

1. 建立命名标准

按照规则办事 容易理解多个项目 方便给相似变量命名

变量下划线？ 匈牙利？

标识全局 标注类型？ 标识成员？ 标识只读？

C ch ij k p s 全大写 小写？ 下划用来 分隔 表示私有变量？

内容，种类，作用域

语义前缀，后缀？

1. 可读的短名字 缩写的方法

去无用的词

标准缩写单词，不要自己乱缩成首字母这种

去部分音调 scrp

使用前几个字母

去无用后缀 ed

**使用能猜得出来的缩写**

写缩写对照表

7、避免的名字 令人误解，含义不明确 含义相似 名字相似 发音相似 不用数字

不用完全无关的名字

十二、数据类型

1、不用魔法数值 0 和1 可以用 for循环中用array.length

2、不要除0

3、**隐式类型转换都改成强制类型转换**

**4、不要进行混合类型的比较，两个比较数应该同类型**

**5**、整数 检查除法，接近零时取哪个值 四舍五入

整数最大值范围， 防止溢出 最大整数

中间值溢出

6浮点数 数量级巨大的计算 有可能得到不希望的值 **浮点数的小数和整数都是占最大位数的，如果整数过大， 加减可能就不准确， 称为精度丢失**

避免判断相等，浮点数和很多数不相等

四舍五入 有误差： 换精度 使用库

1. 字符和字符串

尽量不要用字面量字符串 “”（作为参数等） 应该赋个变量

国际化 存在资源文件中也不错

读取字符串要小心超范围 unicode

**在程序设计的开始就应该考虑怎么国际化了，应该使用unicode**

**C中的字符串指针，字符数组**

1. 布尔变量

把表达式的结果给一个布尔变量

isRight=

简化条件式（ ）把判断表达式改名为**有意义**的变量

9、枚举类型 可以用全局变量模拟

10、具名常量

11、数组

确定数组下标不超界

容器代替数组 数组应该顺序访问，不应该随机访问？ 关于查找算法

用栈，链表？

小心数组边界

多维数组访问，下标正确吗？

12、类型别名 （Windowapi中很常见）

十三、其他类型

1. 结构体 类似类，但是没有封装
2. 指针 容易出错 **缓冲溢出**

内存位置 内存内容的长度 （如何解释）

存入不该存的位置，内存破坏

使用函数封装指针操作，不要在外面操作指针

指针创建就立刻赋值

同一作用域释放指针

使用前检查指针指向是否 正确，确定指针的位置是合理的

如何测试指针破坏 标记字段 冗余重复数据测试（特意做的标记）

简化指针使用

指针用一个数组保存

显式类型转换

1. 全局

阻碍复用 无意修改 名称冲突 破坏模块的分离程度

万不得已才用全局

十四、直线型代码

就近原则，相关代码放一起

顺序执行，前后数据**依赖**

顺序无关

**让代码易于从上而下阅读**

不要让思维跳上跳下，看过了还想这跟前面有什么关系

十五、条件语句

十六、循环控制

十七、其他

多返回多出口

递归

Goto

十八、表驱动法 从表中查找而不用逻辑语句 关联数组 ，哈希 对象属性

这是针对判断返回查询结果语句

日期名对照表 费率对照表

十九、控制问题

二十二、测试技巧

二十三、调试

二十四、重构

二十五、代码调整（效率）

二十七、程序规模

二十八、管理

二十九、集成

**三十、工具：工具很重要**

三十一、布局与风格

三十二、自说明代码

三十三、个人性格

三十四、软件工艺

三十五、更多信息