开发规范

语言编码开发规范

C 语言 C++编码规范

1. 计算机系统基础知识

1. 数据表示：原码，反码，补码，移码
2. 定点数和浮点数格式
   a) 浮点数：
      i. 单精度（32）→→→→→→→→→→→C 语言 float
      ii. 双精度（64）→→→→→→→→→→→C 语言 double
      iii. S E M
         1. S: 尾数符号
         2. M：尾数，用原码表示
         3. E：阶码，用移码表示
3. 校验码：奇偶校验码，海明码，CRC 校验码

2. 基本数据类型

1. 整型，实型，字符型所占的位数 （硬件不同所占字节不同，**以下以 IBM PC 机**为例）
   a) 整形
      i. int/short: 16
      ii. long: 32
      iii. un/signed 的区别在表示范围（e.g. int: 0~65535 / -32768-

32768）

b) 实型：

i. Float:32

ii. double:64

c) 字符型

i. Char/unsigned char: 8

2. **目前32/64位计算中，int所占一般是32，long至少是32，具体看数据模型**
3. 非字符型数据，注意大小端（Endian）模式
4. 大端(Big-endian)：把低位字节序内容放到高地址处，高位字节序内容存放到低地址处
5. 小端（little Endian)：反之，低位字节序内容放到低地址处，高位字节序内容存放到高地址处

3. 数据类型转换原则：

1. 占用内存字节小的类型，相占用多的类型转换，保证精度不降低
2. Char/short → signed int
3. signed int → unsigned int
4. unsigned int → long
5. long → double
6. float →double

4. 数据模型（LP32,ILP32,LP64,LLP64,ILP64）

1. 不同平台，不同数据类型，不同数据模型，所占位数不同
   a) L → long, P→ pointer, LL→long long, I→ int
   b) E.g. LP32: long/pointer →32

5. 数字后缀

1. 不想使用默认类型,可以给数字加上后缀(e.g. l → long, f ➔ float)
2. 小数赋值给整数，只取整数而不是四舍五入

6. C语言整型溢出

1. Unsigned: 溢出后的数会以2^(8*sizeof(type))求模

7. 半角字符、汉字常用编码：

1. 半角字符：

a) ASCII：字母是连续的
b) EBCIDI CCSID： 字母是不连续的
c) Unicode： 与 ASCII 兼容，与 ENCDIC 不兼容

2. 汉字：
   a) GB2312/GBK：两字节/GB18030：包含两字节和四字节
      i. GBK 第二个字节有时会小于 128 的情况，与半角字符冲突
   b) EBCDIC：有些 4 字节 GB18030 没有表示
   c) BIG5：港澳地区的方案，与大陆不兼容
   d) Unicode 编码：对所有语言编码
      i. UTF-8:应用所常用，会有二/三/四字节
      ii. UTF-16：java/windows 内部， 用两字节和四字节表示汉字

8. 常量字符串与指针

1. 对常量字符指针，不能通过指针修改

9. “野”指针：

1. 没有合法指向的指针，会随机指向一块空间，可能是其他程序的数据甚至是系统数据
2. 【规范】用 free 释放了内存以后，立即将指针设置 NULL，防止产生野指针

10. 数组指针和指针数组

1. 加不加括号是数组指针与指针数组的区别
   a) *p[3]→指针数组
   b) （*q）[2]数组指针

11. Void 和 Void*

1. 函数没有返回值，声明为 void 类型
2. 函数无参数，声明其参数为 void
3. 函数参数可以是任意类型指针，声明其参数为 void*
4. Void 指针赋值给其他类型的指针时要进行转换
5. void 指针不能参与指针运算除非进行转换
6. (void*) 0，只想全是 0 的地址，相当于 NULL
7. Void *指针支支持几种有限操作
   a) 与另一指针进行比较
   b) 向函数传递 void*指针或从函数返回 void*指针
   c) 给另一个 void*指针赋值
8. 不允许使用 void*指针操作它所指向的对象

12. Strcpy memcpy memmove 的区别：

1. Strcpy 和 memcpy 的区别
   a) 复制内容：strcpy→字符串，memcpy→任意内容
   b) 复制方法：strcpy 无需指定长度，遇到字符串结束符’\0’结束；memcpy 根据第三个参数决定复制长度
   c) 用途不同：
      i. 字符串→strcpy
      ii. 其他类型→memcpy
      iii. 当源串和目标串有重叠的时候→memmove 可以正确处理，其余 memmove 和 memcpy 近似

13. 安全函数

1. Scanf(), gets(),fgets(),strcpy().strcat()都是 C 语言自带的标准函数，但是不安全。
   a) 会导致数组溢出或者缓冲区溢出
2. Scanf_s(), gets_s (),fgets_s (),strcpy_s ().strcat_s ()是微软 VS 提供的安全函数
   a) 读取/操作字符串时要求指明长度
   b) 过多的字符会被过滤，避免溢出

14. 可重入函数&线程安全函数

1. 线程安全：
   a) 多个线程并发同一段代码，不会出现不同结果==线程是安全的
   b) 线程安全产生的原因：大多是因为对全局变量和静态变量的操作
   c) 常见的线程不安全函数
      i. 不保护共享变量的函数
      ii. 函数状态随着被调用，状态发生变化的函数
      iii. 返回指向静态变量指针的函数
      iv. 调用线程不安全函数的函数
   d) 线程安全的情况：
      i. 每个线程对全局变量或者静态变量只有读取权限,没有写入权限
      ii. 类或者接口对于线程来说都是原子操作
      iii. 多个线程切换不会导致该接口执行结果存在二义性
   e) 线程不安全的类型
      i. 依赖了全局变量，且会修改全局变量。(e.g. rand()的实现每次调用都会修改和读取一个全局的 INT)
      ii. 返回了静态变量（e.g. ctime()）
2. 可重入函数
   a) 概念

i. 重入：同一个函数被不同的执行流调用，当前流程未完，其他进程已经再次调用（执行流之间的互相嵌套执行）
ii. 可重入:多个执行流反复执行一个代码，其结果不会发生改变，通常访问的都是各自的私有栈资源
iii. 不可重入：多个执行流反复执行一段代码时，结果会发生改变
iv. 可重入函数:当一个执行流因为异常被内核切换而中断正在执行的函数而转为另外一个执行流时,后者的执行流对同一个函数的操作并不影响前一个执行流恢复后执行函数产生的结果
v. 不可重入函数： 当正在执行的代码暂时中断转而进去内核。此时另一个信号需要被处理，又需要重新调用中断的函数。如果函数内部有一个全局变量要被操作,那么恢复中断函数的上下文再次继续执行,对同一个全局变量的操作结果可能就会发生改变而并非预期结果。

b) 可重入函数满足条件:
i. 不使用全局变量或者静态变量
ii. 不适用用 malloc 或者 new 开辟出的空间
iii. 不调用不可重入函数
iv. 不返回静态或者全局数据，所有数据都有函数的调用者提供
v. 使用本地数据,或者通过制作全局数据的本地拷贝来保护全局数据

c) 不可重入函数满足条件
i. 调用了 malloc/free 函数，因为 malloc 函数使用全局链表来管理堆的
ii. 调用了标准 I/O 库函数,标准 I/O 库很多实现都以不可重入的方式使用全局数据结构
iii. 可重入体内使用了静态的数据结构

d) 可重入函数的分类
i. 显示：
1. 所有函数参数都是传值传递的（无指针）
2. 所有数据引用都是本地的自动栈变量（没有全局/静态变量）
ii. 隐式
1. 一些参数是引用传递（使用了指针）
2. 调用线程小心地传递指向非共享数据的指针时，才是可重入的

e) 可重入函数特性
i. 可以有多于一个任务并发使用，不担心数据错误。反之，不可重入函数不能由超过一个任务共享（除非能确保函数的互斥/或者在代码关键部分禁用中断）
ii. 可以任意时刻被中断，稍后再运行，不会丢失的数据
iii. 要么使用本地变量，要么在使用全局变量时保护自己的数据

3. 可重入函数与线程安全函数的区别与联系
a) 线程安全函数包含可重入函数,可重入函数一定是线程安全的,反之则不是
b) 如果一个函数中有全局变量，那他既不是线程安全的也不是可重入

的

c) 如果对临界资源的访问上锁，那这个函数时线程安全的。但如果这个重入函数若锁还未释放则会死锁，因此是不可重入的

d) 线程安全可以使不同的线程访问同一块地址空间。可重入函数要求不同的执行流对数据的操作互不影响使结果是相同的

4. C 语言常见的不可重入函数和对应的 unix 线程安全版：+上’_r’

a) 依赖全局变量

i. Rand → rand_r

ii. Strtok→strtok_r

b) 返回静态变量

i. Asctime→asctime_r

ii. Ctime→ctime_r

iii. Gethostbyaddr→gethostbyaddr_r

iv. Gethostbyname→gethostbytname_r

v. Inet_ntoa→暂无

vi. Local_time → localtime_r

15. C++对多态的理解

1. 多态的实现效果：同样的调用语句有多种不同的表现形态
2. 实现的三个条件：
   a) 有继承
   b) 有 virtual 重写
   c) 有父类指针（引用）指向子类对象
3. C++实现：
   a) virtual 关键字，告诉编译器支持多态
   b) 不是根据指针类型判断如何调用
   c) 根据指针指向实际对象类型来判断如何调用
4. 多态的理论基础：动态联编 PK 静态联编。根据实际对象的类型来判断如何调用
5. 多态的重要意义：设计模式的基础是框架的基石
6. 实现多态的理论基础：函数指针做函数参数。 C 函数指针一般有两种用法（正、反）

16. C++对重写、重载的理解

1. 重载
   a) 必须在同一类中进行
   b) 子类无法重载父类函数，父类同名函数将被名称覆盖
   c) 重载是编译期间根据参数类型和个数决定函数的调用
2. 重写
   a) 必须发生于父类与子类之间
   b) 父类与子类中函数必须有完全相同的原型

c) 使用 virtual 声明以后能产生多态，（不使用 virtual 叫重定义）

3. 多态是运行期间根据具体对象的函数决定函数调用

17. C++为什么要定义虚析构函数

1. 什么情况下声明虚函数
   a) 构造函数不能是虚函数，建立一个派生类对象时，必须从类层次的根开始，沿着继承路径诸葛调用基类的构造函数
   b) 析构函数可以是虚的，虚析构函数用于指引 delete 运算符正确析构动态对象

Java 语言编码规范

1. 字节与字符和编码

1. Unicode， UTF-16 一个 char→两个字节，一个中文或者英文都是 2 个字节
2. 其他编码方式则不同

   a) GB2312/GBK，英文字母 1 字节，汉字两字节

   b) UTF-8 英文 1，汉字 3-4 字节

   c) UTF-16,,英文 1,汉字 2/4.(Unicode 扩展区汉字存储要 4)

   d) UTF-32,任何字符都要 4 字节

   e) ISO-8859-1， 单字节编码,向下兼容 ASCII 码,范围 0x00-0xFF.使用了单字节内所有控件,在 ISO-8859-1 的系统中传输/储存任何其他编码的字节流都不会被抛弃

2. 基本数据类型取值范围

1. Byte→1 字节
2. Short→2 字节
3. Int→4
4. Long→8
5. Float→4
6. Double→8

7. Char→2
8. Boolean→1 True or False
9. **注意！Java 的 char 和 long 与 C 语言是不一样的**

3. 自动装箱,拆箱 autoboxing and unboxing

1. 基础数据类型与他们包装进行运算时，编译器会自动转换，转换过程对程序员时透明的→装箱和拆箱
2. 通过 value of 创建的 integer 对象时，数值在[-128，127]之间，返回指向 IntegerCache.cache 中已经存在对象的引用，否则创建新的 integer 对象
   a) Integer i1 = 100
   b) Integer i2 = 100
   c) Integer i3 = 200
   d) Integer i4 = 200
   e) System.out.println(i1 == i2) → True
   f) System.out.println(i3 == i4) → False

4. 大小端和字节序

1. 同 C 语言大小端
2. 操作系统中：
   a) X86 和一般的 CS（windows， freeBSD，Linux）使用小端模式
   b) MacOS 大端模式
3. 不同端模式的处理器进行数据传递时必须考虑端模式的不同。网络上传输数据时，两端对应不同的硬件平台。TCP/IP 协议规定了网上必须采用网络字节顺序，也就是大端模式。
   a) Char 型数据只占一个字节，无所谓大小端
   b) 其他类型数据必须在发送到网络上前转成大端模式
   c) 接收网络数据时按符合接收主机的环境接收
4. JAVA 大小端
   a) 由于虚拟机的关系，屏蔽了大小端问题
   b) 通讯时需要知道的话，可通过 ByteOrder.nativeOrder()查询。操作 ByteBuff 中，也可用 ByteBuffer.order()进行设置。

5. 命名规范

1. Java 可以用汉字定义类和变量名。但是**行内规范禁止使用**。
2. 变量名不要以_或者$开始或结尾，虽然都是合法。
3. 包名：小写
4. 类名：首字母大写、其余组成词一次首字母大写
5. 方法名、参数名、成员变量、局部变量统一使用 lowerCamelCase 风格，

必须遵从驼峰形式

6. 但字符变量名最好不用，除非临时变量，整形临时变量常用 I,j，m,n，字符型常用 c,d,e
7. 常量命名全部大写，单词间用下划线隔开。**语义表达完整清楚，不要嫌名字长**。

6. 与 C 语言中 main()方法/函数区别

1. C 语言：
   a) 一般写成：int main(int argc,char *argv[])此时要求返回 int 整数，argv[0]为程序的路径
   b) 也可写成：void.main(void)此时不返回任何值
2. Java 语言：一般写成 public static void main(String[]args)，不返回任何值。args[0]不是类的名字，时气候第一个命令行参数。

7. Length 属性，length()方法，size()方法的区别

1. 数组提供 length 属性
2. String 提供 length()方法
3. Size 方法是针对泛型集合

8. Final

1. Final 关键字用于三个地方：修饰类，类属性，类方法
   a) 被 final 关键字修饰的类不能被继承，被 final 关键字修饰的类属性和类方法不能被覆盖（重写）
   b) Final 类不能被继承/覆盖。执行速度方面比一般类快
   c) 可以在父类中设置某个方法为 final 方法。虽然类可以被继承，但是子类不能够覆盖 final 方法
2. Final 类可以提高执行速度的主要原因：
   a) 不涉及继承和覆盖
   b) 其地址引用和装载在编译时完成
   c) 在运行时不要求 JVM 执行因覆盖产生的动态地址引用
   d) 用继承链上的一般对象比，垃圾回收器在收回 final 对象所占据的地址空间也相对简单
3. java 常见的 final 类
   a) java.lang 包
      i. 包装类：boolean, character, short, integer, long, float, double, byte, void
      ii. 字符串类：String, stringBuilcer, StringBuffer
      iii. 系统类：Class, System, RuntimePermission, Compiler
      iv. 数字类：Math, StrictMath

v. 其他： Character.UnicodeBlock, Process Builder, StackTraceElement

b) Java.util 包: UUID, Scanner, Optional

c) Java.lang.reflect 包 Array , Constructor , Field , Parameter,ReflectPermission

d) Java.net 包: HttpCookie, Inet4Address, Inet6Address, URL, URI

e) Java.time 包除了 Clock, ZoneId 都是

9. 线程的生命周期

1. 初始状态: 创建线程对象, 没有开始运行
2. 可运行状态: 调用 start()方法, 真正被创建, 等待 cpu 调度
3. 运行状态:拥有 CPU 执行权
4. 阻塞状态:
   a) 正在等待用户输入或者调用了 sleep()和 join()等, 会进去阻塞状态。
   b) 从阻塞状态出来的线程不一定回到运行状态
   c) 回到可运行状态, 等 cpu 再次调度
5. 等待队列状态:
   a) 一个线程调用一个对象的 wait()会自动放弃该对象的锁标记, 进入等待队列状态
   b) 当有另一线程调用临界资源的 notify()或者 notifyAll(),才会释放, 此时进去锁池状态
6. 锁池状态
   a) 每个对象都有互斥锁标记, 防止对临界资源访问造成数据不一致和不完整
   b) 一个线程拥有一个对象的锁标记后, 另一线程像访问对象, 必须在锁池中等待。由系统决定哪个线程拿到锁标记运行
7. 终止状态: 运行结束, 一个进程只有所有线程退出后才会终止

10.继承、重写、重载

1. 继承
   a) 使用 extends 关键字完成继承
   b) 与 C++不同在于只支持单继承不支持多重继承
   c) Extends 和 implements:
      i. entends 继承父类,只要不是声明为 final 或者定义为 abstract 都能继承
      ii. java 不支持多重继承, 但可以用接口来实现, extends 只能继承一个类, implements 可以实现多个接口
      iii. entend 可以继承一个接口, 但仍是一个接口
   d) 继承规律:
      i. 类可以继承(extends)类, 可以继承(extends)抽象类, 可以继承

(implements)接口
ii. 抽象类可以继承(extends)类，可以继承(extends)抽象类，可以继承(implements)接口
iii. 接口只能继承(extends)接口

2. 重写 override，多态：
   a) 父类功能无法满足子类需求需要进行方法重写
   b) 名称参数一样
   c) 前提：存在继承关系
   d) 方法重写：子父类存在同名函数
   e) 重写要求：
      i. 子父类函数名与形参列表必须一致
      ii. 子类权限修饰符必须大于等于父类： public > protected > default > private
      iii. 子类返回值类型范围必须小于等于父类：Object>String
      iv. 子类抛出异常类型必须小于等于父类
3. 重载 overload
   a) 在一个类中存在两个或以上的同名函数，名称一样，参数不同
   b) 重载要求：
      i. 函数名一致
      ii. 形参列表不一致
      iii. 与返回值类型无关

11.JAVA I/O 常用类

1. Input
   a) Byte:
      i. *InputStream*
      ii. ByteArrayInputStream
      iii. StringBufferInputStream
      iv. FileInputStream
      v. PipedInputStream
      vi. SequenceInputStream
      vii. FilterInputStream
      viii. DataInputStream
      ix. BufferedInputStream
      x. LineNumberInputStream
      xi. PushbackInputStream
   b) Char
      i. *Reader*
      ii. CharArrayReader
      iii. StringReader
      iv. FileReader
      v. PipeReader
      vi. BufferReader

vii. LineNumberReader
viii. PushBackReader

2. Output
   a) Byte
      i. *OutputStream*
      ii. ByteArrayOutputStream
      iii. FileOutputStream
      iv. PipedOutputStream
      v. SequenceOutputStream
      vi. FilterOutputStream
      vii. DataIOutputStream
      viii. BufferedOutputStream
      ix. PrintStream
   b) Char
      i. Writer
      ii. CharArrayWriter
      iii. StringWriter
      iv. FileWrite
      v. PipedWrite
      vi. BufferedWrite
      vii. PrintWrite

12.避免和禁止的要求

1. Import 中应该避免使用*
2. 避免定义内部类、匿名类
3. 尽量避免一个类的代码超过 1000 行（除注释）
4. 尽量避免一个方法的代码超过 100 行（除注释）
5. 尽量避免用对象来访问静态成员变量或方法，而以类名替之
6. 尽量避免在一行中将用一个值赋给多个变量
7. 应避免声明抛出 java.lang.Error、Java.lang.RuntimeException 等异常
8. 应避免捕获 java.lang.Exception 、 java.lang.Error 、 java.lang.RuntimeError 等异常
9. 异常信息应记录在日志文件中，禁止使用 System.out 输出

13.其他注意事项

1. 接口中的方法的访问级别都是 public 的,不写访问修饰符默认为 public, private、protected 都错
2. Interface 中定义的变量，编译时会加上 static final，都是静态的常量，访问级别都是 public 的
3. Interface 的方法都是 public 的，声明方法是不能加比 public 低的访

问修饰符

4. 没有真正的多维数组，只有数组的数组，多维数组不一定是规则矩阵形式
5. Static 静态方法
   a) 只能直接调用同类中的其他静态成员（变量和方法）
   b) 不能直接访问类中的非静态成员
   c) 静态方法不能以任何方式引用 this 和 super 关键字
   d) 静态方法使用前不用创建任何对象，非静态要先创建类的实例对象才可用
   e) Main()方法时静态的，JVM 执行 main 时不创建 main 的实例对象。在 main 中，不能直接访问非静态成员，必须创建实例对象才能通过此对象访问。
6. 静态代码块：
   a) 一个类可以使用不包含在任何方法体中的静态代码块
   b) 载入类时，静态代码块被执行且执行一次，常用于类属性的初始化
   c) 类是在第一次被使用的时候才被装载
7. “==”和 equals（）方法的区别，如何比较两数组是否相等
8. String、StringBuffers、String、StringBuffer 和 Stringbuilder
   a) Stringbuilder 是线程不安全的，StringBuffer 是线程安全的
   b) String：少量字符换
   c) Stringbuilder：适用单线程下在字符缓冲区进行大量操作
   d) StringBuffer：适用多线程下字符缓冲区进行大量操作
9. 基础知识：
   a) HTTP 协议
   b) SSL 基础
   c) 标密算法，国密算法
   d) 随机数生成器
   e) 常用 XML 解析器
   f) 有序排序

Shell 编程方法

1. Linux 系统运行级别的含义

1. 关机模式
2. 单用户模式
3. 无 NFS 多用户模式
4. 文本模式
5. 未使用
6. 图形模式，X11,桌面
7. 重启模式
8. 共 7 中编号 0-6，启动运行级别不能为 0/6

2. 文件名/目录名

1. Linux 允许文件名目录名使用特殊字符开头或者包含，**但我们不建议.** 为了删除特殊字符的文件和目录，需要试用反斜杠转义等特殊手段
2. Windows 严格控制了文件名和目录名不得包含的特殊字符

3. 文件的三个时间

1. 访问时间：对文件读操作一次，就会改变
2. 修改时间：文件内容的最后一次修改，ls -l 显示的就是这个
3. 状态时间：状态被改变时，改变。Chmod、chown 等
4. Windows 则是：创建时间，修改时间，访问时间
5. Linux /win 对比
   a) Linux
      i. Touch 修改时间
      ii. ls -lt 从后到先，ls -lrt 从先到后
      iii. cp 只拷贝内容，修改时间不一致
      iv. -p 参数可以复制拷贝访问权限和修改时间
      v. tar 备份和恢复到另外目录的文件与源文件有相同的修改时间
   b) windows
      i. DIR/OD 先到后； DIR/O-D 后到先
      ii. COPY 就有相同的修改时间

4. PATH 环境变量

1. Linux
   a) 未写绝对/相对路径时执行程序，必须再 PATH 环境变量包含的路径中才能搜索到。
   b) 多个路径用冒号分隔
   c) Root 用户不得把‘.’包括到搜索目录列表里；普通用户只能把‘.’当道列表最后位置
2. Windows
   a) 未写路径,首先检查该程序文件名是否在当前目录,不在再搜索 PATH
   b) 多个路径用分号分隔

5. Linux shell 预定义变量

1. $0 脚本文件名
2. $n 传递给脚本或函数的参数，n 是第几个
3. $# 传递参数个数
4. $* 传递的所有参数

5. $@ 传递的所有参数，可当作数组用被(“”)包含时，与$*稍有不同
6. $$ 当前 shell 进程 ID
7. $! 后台运行的最后一个进程的 ID
8. $? 上个命令的退出状态，或函数返回值
9. 各种括号
    a) 圆括号
        i. 单圆括号()
            1. 命令组
            2. 命令替换
            3. 定义数组
        ii. 双圆括号(())
            1. 证书扩展
            2. 跨进制运算
            3. 重定义变量值
            4. 用于算术运算比较
    b) 中括号
        i. 单中括号[]
            1. Bash 内部命令
            2. 引用数组每个元素编号
        ii. 双中括号[[]]
            1. If [$a-ne1]&&[$a!=2] 可以写成 if[[$a!=1 && $a!=2]]
            2. 双括号比[]更通用
    c) 大括号
            1. 表达变量值
            2. 批量操作
            3. 扩展顺序
            4. 特殊替换
            5. 多命令执行

6. Linux 下执行多条命令的方法与区别

1. 可放在一行，执行情况依赖于分隔符
2. (;)分号分隔，连续执行，结果互不影响，不保证每个都执行成功
3. &&分隔，前面成功才继续执行
4. ||分隔，前面失败才下一个直到成功一条

7. 文件权限修改命令 chmod

1. 改变文件的读写和执行权限，
2. 符号法和八进制数字法

8. 目录命令

1. 创建: mdkir -p BBB/Test
2. 删除文件或目录命令 rm -irf

9. 账户管理文件

1. 用户所有信息，/etc/passwd 中，权限 644
2. 用户密码所有信息，/etc/shadow，权限 400
3. 用户组所有信息， /etc/group，权限 644
4. 上述所有文件的所有者和文件所在组都是 root

10.Tar 命令

1. 将备份与恢复的命令写成一行，不需指定备份输出文件名

11.Find 命令

1. Find 【目录】 【条件】 【动作】
   a) 大小 -size
   b) 目录深度 -maxdepth
   c) 多条件
   d) 修改日期 -mtime +N
2. 与 xargs 命令配合使用
   a) 避免查找文件过多导致-exec 后命令参数过长，避免溢出。只需要一个进程。

12.多命令结合杀指定进程

1. Ps -ef |grep java |grep -v grep |cut -c 9-15 |xargs kill -s 9

总体编码规范

1. 编程规范总则

1. 单个函数、过程、方法的代码函数(不包括注释空行)一般不超 200 行
2. 去掉不必要公共变量/全局变量
3. 【推荐】可读性第一，编码输入效率第二
4. 【强制】使用括号避免二义性

5. 【强制】不允许多个短语写在一行中，一行只写一条语句
6. 【强制】，C/C++/JAVA 除了卫语句以外，if/else/for/do/while/case/switch/default 等自占一行。执行语句部分只有一行代码都要加括号
7. 【推荐】，常量标识：避免使用不易理解的数字，字符串（‘魔法值’），用有意义的标识来替代。
8. 【强制】不要用难懂的高技巧语句。高技巧不等于高效率。效率在于算法
9. 【强制】源程序、头文件注释分以下几类
   a) 文件头部注释：整个文件进行整体说明，必须有
   b) 函数、过程、类的方法、模块等；除类的 getter/setter 方法以外，源程序文件必须有注释说明用途
   c) 重要语句块、特殊算法等：应有注释进行解释
   d) 变量命名、语句上方或右侧的注释:除自注释的代码或者变量命名以外，建议提供必要的简明注释
   e) 临时屏蔽但不想删除的代码/变量、修改前的代码：通过注释方式使其不起作用，应有注释说明
10. 【强制】源程序第 1)类注释应包括功能描述、版权说明、版本号、创建者、修改者姓名及日期、修改原因/修改内容等的注释。有中文姓名者必须使用中文姓名而非拼音或英文缩写。保证“程序员对代码负责”。另外注释建议包括与其他文件关系、模块目的/功能、主要函数及其功能
11. 【强制】对分支语句（条件分支、循环语句等）必须编写注释
12. 【强制】标识符长度符合‘min-length&&max-information’原则。清晰、明了，有明确含义，使用完整单词或者大家基本可以理解的缩写，避免使人误解
13. 【强制】命名试用特殊约定或缩写，要注释说明
   a) 再源文件开始处，对缩写或约定，特别是特殊的缩写，进行必要的注释说明
14. 【强制】自己特有的命名风格，要自始至终保持一致
15. 【强制】变量名，除局部循环外，进制取单个字体(I,j,k)。建议除了具体含义以外还要表明变量类型、数据类型。局部循环变量若用与数组访问，不建议命名过长。
   a) 局部变量以单个字符标识，容易敲错，编译又无法检查出，可能造成大量的差错时间
16. 【强制】常量命名全部大写，单词用下划线隔开，语义表达完整清楚，不要嫌长(不超过 60 字符即可)
17. 【强制】避免混肴的特殊字母组合
   a) ‘Oo’&’0’
   b) ‘l’&’l’ 小写 L 和大写 I 以及’1’
   c) ‘b’& ‘6’
   d) ‘Zz’&’2’
   e) 某些编译器字体下大小写不易区分：’Cc’，‘Kk’,’Ss’,’Vv’,’Ww’,’Xx’
18. 【强制】不得使用所用语言的保留字，即使通过也会造成误解

19. 全局变量要有纤细的注释：变量作用、含义、功能、取值范围、哪些函数或过程存取它以及存取是注意事项
20. 【强制】向公共变量和全局变量传递数据时，要十分小心，防止赋予不合理值或越界
    a) 进行合法性检查，提高可靠性、稳定性
21. 【强制】防止局部变量与公共变量/全局变量同名
22. 【强制】严禁使用未经初始化的变量作为右值
    a) 在C/C++中引用未经赋值的指针，会引起系统崩溃
23. 【强制】结构的预留域明确用途后，必须在定义和使用出加以注释
24. 【强制】留心具体语言及编译器处理不同数据类型的原则及有关细节
    a) 在C语言中，static局部变量将在内存‘数据区‘中生成，而非’堆栈‘中
25. 【强制】要注意数据类型的强制转换
    a) 进行数据类型强制转换时，数据意义，转换后取值都可能变化
26. 【强制】对编译系统默认的数据类型转换，也要有充分的认识
27. 【强制】声明用于分布式环境或不同CPU间通信环境的数据结构时，必须考虑机器的字节顺序、使用的位域及字节对齐等问题
28. 【强制】对所调用函数的错位返回码要仔细、全面的处理
29. 【强制】明确函数功能，精确实现函数设计。空函数要明确是待实现或待删除
30. 【强制】编写可重入函数时，注意局部变量的使用
    a) 编写C/C++的可重入函数时
       i. 应使用auto即缺省态局部变量或寄存器变量
       ii. 不应使用static局部变量，否则必须经过特殊处理才能具有可重入性
31. 【强制】编写可重入函数时，如使用全局变量，则应通过关中断、信号量(PV操作)等手段加以保护
    a) 如不保护，就不具有可重入性，当多个进程调用此函数时，有关全局变量变为不可知状态
32. 【强制】同一项目组应明确规定：对接口函数参数的合法性检查应又函数的调用者负责还是由接口函数本身负责。缺省是由调用者负责
    a) 接口函数合法性检查，两个极端现象：
       i. 均不做合法性检查，问题隐患
       ii. 均做合法性检查，冗余代码
33. 【强制】防止将函数的参数作为工作变量：可能错误的改变参数内容，可以先用局部变量代之
34. 【强制】一个函数仅完成一件功能
35. 【强制】尽量不要编写依赖于其他函数内部实现的函数
36. 【强制】检查函数所有参数输入的有效性
37. 【强制】检查函数所有非参数输入的有效性：数据文件，公共变量
    a) 函数输入有两种：参数输入；全局变量、数据文件输入(非参数输入)
38. 【强制】函数的返回值要清楚、明了。让使用者不容忽视错误情况
    a) 【正例】正数或者0等于成功处理，负数表示各种错误原因
    b) 【反例】对不同错误均返回-1，无任何附加信息

39. 【推荐】不要滥用 goto 语句
    a) 行内 C/C++禁止 goto，其他语言各有标准
    b) 使静态结构和动态结构不一致，难以理解+差错
    c) 可能死循环
    d) 使用时建议只跳到函数末尾/退出多重循环
40. 【强制】所有用户输入，必须进行合法性检查
41. 【强制】不要比较浮点数的相等，不可靠。通过相减差值和约定精度比较
42. 【强制】程序与环境或状态发生关系时,必须主动处理发生的意外事件:文件逻辑锁定、打印机未联机、socket 链路断开
43. 【强制】单元检测也是变成一部分，联测的车工序必须通过单元检测
44. 【强制】注意金额单位，防止 100 倍甚至更高的错误。汇率/百分比/超时时间
45. 【强制】报文和用户输入文件涉及中文的,必须考虑汉字被截断的错误,容错处理或报错处理。1234 字节
46. 【强制】处理文本文件时，应支持 windows/linux 两种格式
47. 【强制】捕获异常是为了处理，不要捕获却不处理而抛弃。
    a) 如不想出来，请将异常抛给调用者
    b) 最外层业务使用者，必须处理异常，转化为用户可理解的内容
48. 【强制】不在 finally 块中使用 return。Finally 中 return 返回后方法结束，不会再执行 try 中的 return
49. 【强制】防止 NPE(NullPointerException,空指针异常)调用需要进行 null 判断防止 NPE 问题。方法的返回值可以为 null，不强制返回空集。必须注释 null 值
50. 【强制】认真处理程序遇到的出错
51. 【强制】系统运行最开始，要对加载到系统的数据进行一致性检查
    a) 不一样的数据，会使系统进入混乱和不可知状态
52. 【强制】严禁随意更改其他模块或系统的有关设置
    a) 不能更改不属于自己模块的有关设置
53. 【强制】不能随意改变与其他模块的接口
54. 【强制】了解系统接口之后再使用系统的功能。注意功能调用顺序，是否允许重复初始化
55. 【推荐】设计高扇入，合理扇出(<7)的函数
    a) 扇出：函数直接调用(控制)其他函数的数目
    b) 扇入：有多少上级函数调用它
56. 【推荐】不使用与硬件或操作系统关系很大的语句。使用建议的标准语句。提高可移植性和可重用性。

2. 质量保证要求

1. 【强制】只引用属于自己的存储空间。C/C++字符串应预留结束符空间
2. 【推荐】多个库函数的细微差异
    a) C/C++的 memmove()函数保证拷贝结果正确 memcpy()不能
3. 【强制】防止对数组、指针、内存地址等的内存操作越界，防止访问第

一个或者最后一个元素内存越界

4. 【强制】特别注意数组的下标是 0/1.
5. 【强制】除非是单独管理内存分配的过程/函数，一般过程/函数分配的内存，再过程/函数推出之前要释放
6. 【强制】防止引用已经释放的内存空间
7. 【强制】除非是单独管理文件打开、socket 建立等过程/函数，一般过程/函数中申请文件、socket 等句柄，在推出之前要关闭
8. 【强制】防止差 1 错误，‘<=’写成‘<’
9. 【强制】时刻注意容易混淆的操作符，防止拼写错误
   a) C/C++可以定义宏来代替判断，e.g.’==’以防混淆（‘=’）
10. 【强制】一个 switch 块内，每个 case 要么通过 break/return 来终止，要么注释说明执行到哪个 case
    a) 必须包含一个 default 语句在最后，即使什么代码都没有
11. 【强制】注意变量的边界值，防止上溢/下溢
12. 【强制】留心程序机器码大小，是否超出系统限制
13. 【强制】除非特殊需求，避免使用嵌入式汇编等对可移植性由较大影响的语句、功能
14. 【强制】unix 下，多线程中的子线程退出必须采用主动退出方式，即子线程 return 出口
15. 【强制】对一些具有危险性的操作代码（写硬盘，删数据）要仔细考虑。
    a) 用户界面的删除等不可恢复的操作应有“确认”机制
16. 【强制】在软件系统中设置与取消有关测试手段，不能对软件实现的功能产生影响
17. 【强制】项目组中，要使用统一的源代码版本管理工具
18. 【强制】注意不同平台编译器差异
19. 【强制】项目组中，统一编译开关选项
20. 【强制】代码走读/走查/审查方式对代码进行检查
21. 【强制】单元测试要求达到语句覆盖
22. 【强制】清理、整理或优化后的代码要经过审查及测试
23. 【强制】代码升级要严格测试
24. 【强制】使用工具软件对代码版本进行维护
25. 【强制】正式版本上软件的任何修改都应有详细的文档记录
26. 【强制】仔细测试代码处理数据、变量的边界情况

3. 程序的效率要求

1. 基本程序优化要求
2. 空间效率优化
3. 循环体优化
4. 算法优化
5. 数据结构优化
6. 程序组织优化

4. 其他要求

1. 【强制】禁止使用不安全的库函数
2. 【强制】多线程程序必须注意库函数是否线程安全
3. 【强制】访问 FTP/数据库等密码部的铭文写死在源程序中
4. 【强制】隶属于用户个人的页面和功能必须进行权限的控制校验
5. 【强制】用户敏感数据禁止直接展示，要数据脱敏
6. 【强制】使用用户输入作为 SQL 参数时，应对参数进行转码，或用预编译声明及存储过程。进制字符串拼接 SQL 访问数据库
7. 【强制】对用户请求传入的任何参数必须做有效性验证，忽略参数校验可能会：
   a) Page size 过大导致内存溢出
   b) 恶意 order by 导致数据库慢查询
   c) 任意重定向
   d) SQL 注入
   e) 反序列化注入
   f) 正则输入源串拒绝服务 ReDoS
8. 【强制】对多个资源、数据库表、对象同时枷锁，要保持一致的枷锁顺序，否则可能锁死
9. 【强制】并发修改同意记录，避免更新丢失
   a) 要么在应用层加锁
   b) 要么在缓存加锁
   c) 要么在数据库层用乐观锁，每次访问冲突概率小于 20%，推荐使用乐观锁，否则悲观锁。乐观锁重试次数不得小于 3 次
10. 【强制】环境变量必须参数化，注意开发、测试、生产环境的而环境变量可能不同
11. 【强制】TCP,UDP 通讯的 server 端口必须可配置，不得谢斯在源程序中
12. 【推荐】TCP,UDP 通讯的 server 端应具备对 client 端 IP 白名单检查功能
13. 【强制】新建系统 ASCII 中文编码必须支持 GB18030.不能仅支持 GBK。第三方只支持 GBK 必须约定生僻字处理方法
14. 【强制】用户密码不得在网络报文明文传输
15. 【强制】保密的算法源码不得通过 JavaScript 等客户端脚本下载到客户端
16. 【强制】client 端做了用户输入检查，server 端也必须做全部用户输入的检查
17. 【强制】网络定长字段报文组包时必须明确 C/C++字符串结束符是否允许包含在字段中，接收方解析是们必须考虑结束符后的垃圾数据可能造成的异常
18. 【强制】网络定长字段报文组包、解析时，必须考虑输入值超长时造成的半个汉字异常

应用开发规范

1. 开放平台系统应用日志管理规范：术语

1. 日志级别： 日志分级，根据场景和目的不同标记日志信息的重要程度原则上，WARN 和 TRACE 只用于数据库、中间件，不用与应用日志
   a) INFO
   b) DEBUG
   c) WARN
   d) ERROR
   e) TRACE
2. 日志类别：根据应用服务和用途对日志分类
   a) 第一层按应用服务来：
      i. 应用日志
      ii. 平台日志
      iii. 等等
   b) 第二层按日志信息特性来划分：
      i. 处理过程
      ii. 通讯
      iii. 异常日志
3. UUID：通用唯一识别码，确保同一时空中所有机器都是唯一的。5 个版本.**本规范中要求版本 1**
4. 全局流水号：一笔交易在交易链路在各个系统中保持一致的流水号
5. 发起方流水号:《中信银行应用系统交易流水号总体方案》，交易的发起方系统，为自身系统的每一笔交易分配一个唯一的顺序编号作为业务流水号。目前惯例，发起方流水号作为全局流水号要求其在这个个交易链路的报文中必须登记

2. 开放平台系统应用日志管理规范：

1. 包含四个阶段
   a) 日志规范
   b) 日志记录
   c) 日志使用
   d) 日志掩护
2. 终端应用不记录通讯日志
3. 终端应用不记录处理过程日志
4. 通讯错误记录在异常日志而非通讯日志中
5. 错误日志信息必须记录组成平台各组件和应用的异常状态。所有异常必须写在日志中，不允许输出到屏幕。
6. 交易日志信息必须确保具备支持对交易性能的分析能力。记录系统内各

组件或者跨系统调用的时间戳和状态。批处理作业也类似

7. 每个交易或批处理作业无论成败或出现异常，应具备完整的“开始”“结束”标识
8. 交易进入每个系统要产生和记录 UUID
9. 日志所在文件系统的硬盘空间，按保留三个月日志分配
10. 日志文件大小原则上不超过 100M
11. 日志默认保留 7 天
12. 日志管理中新索引日志保留 1 月，归档日志保留 1 年
13. 应用日志要用异步方式输出到文件系统
14. 通讯类高频输出但使用不频繁日志，采集品读不必到秒级
15. 应用系统日志文件内容记录格式如下：‘日志内容”左边的个字段称为日志头部：
    a) 时间戳|日志级别|所属系统或子系统|uuid|全局流水号|进程号|线程好|源码文件名：行号|调用嵌套层级 ID|日志内容

3. 开放式平台 B/S 结构与异常处理技术规范

1. Exception 异常
    i. 是应用程序中可能的可预测、可恢复问题。
    ii. 一般大多数异常标识中度到轻度的问题
    iii. 一般在特定环境下产生，通常出现在代码的特定方法和操作中
    b) JAVA 异常分类
    i. Checked Exception 检查型异常（又称非运行时异常）：必须对其处理，否则无法通过编译。凡是继承自 Exception 而不继承自 RuntimeException 的异常都是非运行时异常
    ii. Unchecked Exception 非检查型异常、又称运行时异常，又称 RuntimeException：可以对其处理也可以不处理。推荐不对运行时异常处理
2. Error 错误
    a) 应用程序中比较严重的问题
    b) 大多数与代码编写者执行的操作无关，而表示代码运行时 JVM（java 虚拟机）出现的问题。E.g. JVM 不再又继续执行操作所需的内存资源时，将出现 OutOfMemoryError.
    c) 这些问题发生时，JVM 一般会选择线程终止退出，表示出现了不可恢复的错误，只能终止运行
3. 错误和异常处理的原则：
    a) 不要直接忽略异常或仅仅直接 e.printStackTrace()
    b) 不要用异常处理来处理程序的正常控制流，一场不要用来做流程或条件控制，因为处理效率低
    c) 对于检查型异常，如果不能行之有校的处理，需要转为 RuntimeException 抛出
    d) 在程序异常抛出时，一定要在 finally 中关闭资源引用：如
    i. 数据库连接、打开的文件流
    ii. Socket 句柄

iii. 避免造成潜在的内存泄漏

e) 对于抛出的异常信息应在输出的信息添加注释详细说明。注释在 10 字以上。抛出异常中应包括两部分：

i. 面对用户的友好信息

ii. 后台信息：包含详细的错误信息以帮助以后定位问题

f) 关于资源的关闭

i. 数据库资源和事务提交回滚由开发框架统一处理,业务代码只负责声明事务

ii. 任何情况下不允许业务模块自己维护数据库资源

iii. 如确实需要自己提交事务或回滚事务，应在注释中说明理由

g) 一般情况下

i. 不允许吃掉异常

ii. 不允许将异常直接抛出，

iii. 不允许 catch 后作相应处理再抛出异常

iv. 抛出新异常时时任何情况下不允许中断异常链

v. 异常最后由系统架构表示出的最外层统一处理

vi. 如必须吃掉异常，应在注释中写明理由

vii. 嵌套在 finally 块和 catcah 块的异常处理吃掉一场可以不写注释。

viii. 任何情况下不允许在 finally 块中抛出异常

4. 密码配置文件的规范指南

1. 加密存放的密码和用户，在使用时需要还原成密码和用户的原值。这就要求使用的是对称密钥算法或是能还原出密码原值的算法，不能用单向散列函数。
2. 用户密码配置文件使用独立的配置文件，与一般的系统配置文件分开。
3. 服务端的用户密码支持用户密码管理规范，支持定期更换密码，密码修改不影响应用程序，应用程序不需要修改
4. 尽量不要使用用户密码配置，如减少使用 FTP，用 SFTP 并设置免密调用代替

5. 数据字典的开发规范

1. 数据字典：对数据模型中各对象的描述集合，包括
   a) 数据项
   b) 选项代码
   c) 实体
   d) 属性
   e) 物理表
   f) 字段
2. 本规范涉及的对象涵盖：英文词根、数据项、基础数据项、选项代码、物理表、字段。

a) 建议英文词根的字符数长度是 4，至少要求 3 位，除非英文单词本身字符数小于等于 4。英文缩写中不可包含小写字母、下划线、空格等其他字符
b) 数据项的名称由多个英文词根拼接而成，拼接符是下划线“_”，数据项最长 30 位字符
c) 选项代码的名称同数据项
d) 物理表命名规则：由 T 或者 V 开头，多个英文词根拼接，拼接同‘_’，最长 30
e) 字段命名同数据向

6. 数值域统一处理规范

1. 数值域：0-9 和小数点组成的数字字段，如交易金额、余额、基金净值、利率、汇率等
2. 原值：以自然方式表示的数据值，带格式的数值域
3. 非原值：以数字字符串方式表示的数据，小数点不出现，接口双方约定好小数点的位置
4. P 型：AS400 数据类型中的数字型（压缩），即 AS400 的数值域
5. 非原值系统：
   a) 采用 8583 报文的系统：集中收单系统、POS 前置系统
   b) 信用卡相关系统：SEMA CARD 系统、信用卡中心前置系统、国际借记卡
   c) 主机系统：新核心系统
6. 原则上，行内系统间交易报文格式采用 XML 或 JSON 报文，报文中数值域采用统一数据字典定义的原值表达
7. 存量系统特列：
   a) 无分割符定长报文：核心系统采用无分割符合定长报文。数值域是 AS400 数据类型中的数字型（压缩）即 P 型。第三方存管系统、全国支票影像处理系统、人行国库系统采用无分隔符定长报文，其中数字域采用统一规范，即统一数据字典定义的原值表达。
   b) ISO8583 报文：采用 8583 报文的系统+信用卡系统。报文中数值域表达采用数字字符串、小数点不出现
8. 原则上，行内系统间不同报文格式转换由交换平台系统（含一总、二总、新交换）实现
9. 重要数值域规范说明：
   a) 金额类数值域定义为数字型 17.2，15 位整数，2 位小数。
      i. 交易渠道对客户输入允许整数，带 1 位小数、2 位小数的情况
      ii. 在 XML 或 JSON 报文接口及数据库存储时必须进行原值格式标准化处理：
         1. 有小数点，不足 2 位右补 0 补齐小数位
         2. 复值有负号占位
      iii. 要求各服务方系统（含路由转换系统）能按数据字典定义正确识别 XML 或 JSON 报文中没有做原值格式标准化处理的数据域，而不完全依赖关联数据的标准化处理

b) 汇率类数值域定义：数字型12.6，6位整数，6位小数。
   i. 交易渠道允许输入整数，带1-6位小数
   ii. 接口转发及数据库存储时必须原值标准化处理
      1. 有小数点，不足6位右补0
   iii. 其他同a）
c) 利率类数值域定义：数字型9.7，2位整数，7位小数。其他类似上
d) 费率类数值域定义：数字型9.7，2位整数，7位小数。其他类似上
e) 税率类数值域定义：数字型7.5，2位整数。5位小数

10. 新核心数据字典和各交易接口中，统一各类“率”格式约定：
    a) 利率:%为单位,%不出现在交易接口中::年利率7.2%,接口中送7.2
    b) 汇率：按照实际值，即除过报价单位之后的数值，没有%
    c) 费率：%为单位，%不出现在交易接口中
    d) 税率：%为单位，%不出现在交易接口中

7. 通讯程序开发规范

1. 原则上无持续高并发交易的情况下，优先考虑同步短连接模式
2. 在消息个推场景下，消息推送服务器和个人手机之间时TCP长连接
3. 我行应用系统常用的通讯协议包括TCP/IP；SOCKET；HTTP；HTTPS；FTP；SFTP和微服务使用的RPC协议
4. FTP是属于限制使用级，严禁应用系统擅自启动FTP服务。严禁使用匿名FTP功能（用户名输入anonymous，无需密码)。FTP用户名和密码应符合我行信息系统用户管理要求。
5. 对于外联应用，原则上我行不提供对外的FTP/SFTP服务。对于专线模式如果需要提供必须的信息技术管理部安全审批。即应由外部机构提供FTP服务器供我行下载或上传文件
6. 应用程序应该自行通过压缩技术或附加校验文件检查文件传输的完整性，自行实现大文件的断点续传
7. 通过文件目录或扩展名设计，区分待处理文件和已处理文件，防止重复处理，支持出错重做
8. 对于Web服务,新建系统服务原则上提供RESTful风格的web service，
9. 对于使用MQ等消息中间件通讯的应用程序,设计开发时必须考虑消息的时效性。在主备切换、灾备切换等场景下，必须对收发队列进行检查，对残留消息进行清理后才允许启动应用
10. 常见的报文格式：无分隔符定长、有分隔符不定长、XML、JSON、ISO8583
    a) 有分隔符不定长：不建议采用单个竖线符号，与部分汉字编码冲突
    b) 原则上,服务方所在的开发平台,通讯组件应同时支持有分隔符不定长、XML、JSON三种格式。XML是必选格式
    c) 行内系统间，联机交易通信报文格式采用XML格式或JSON；批量数据文件采用csv或其他有分隔符不定长
    d) 新建系统，与行内系统之间应优先考虑JSON
    e) 与行外机构之间通讯报文格式，推荐使用XML
11. 对于交易服务方系统、应在发布的接口中，体现接口的版本、确保同时支持不同接口版本的消费方

12. 通讯报文中的字符编码规范要符合我行 UTF-8 统一编码大方向，对应编码 GBK，在无计划升级到 UTF-8 之前也必须升级到 GBK18030
13. 无论阻塞或非阻塞模式，通讯都必须设置超时
14. 超时设置应遵循漏斗设计原则，服务方最长超市不超过 30 秒
15. 首先对接入流量控制,保护本系统自身不被冲垮;其次对接出流量控制，避免冲垮后端系统
16. 故障隔离设计原则：
    a) 对于接入的通讯资源，应为不同渠道系统分配不同通讯资源。使系统间接入不会发生互抢资源的情况
    b) 对于接出线程，应按不同系统分离，避免不同通讯方式之间对资源的挂起现象
17. 通讯框架的选择原则
    a) 对于 C 语言：一般情况下用 select 方式或 poll
    b) 对 JAVA：无特殊情况用 NIO 框架，推荐用 netty，或核心下移开发平台的通信组件。通常不允许直接使用 JDK 的 NIO 类库进行开发
    c) 数据库访问必须使用 JDBC 连接池，使用长连接通讯模式
18. 通讯程序连接设计要点：
    a) Server/client 两端通讯程序必须检查请求/回应报文和变长域的最大长度，对超长报文拦截，防止缓冲区溢出
    b) TCP/HTTP/HTTPS 的 C/S 或 B/S 两方一般是客户端主动关闭连接（先关闭），服务器端被动关闭（后关闭）
    c) 客户端出现 TIME_WAIT 是正常状态
    d) 避免通过以下措施，解决 TIME_WAIT 状态保持 2MSL 时间的问题
       i. 开启端口重用，允许将 TIME_WAIT sockets 重新用于新的 TCP 连接。部分防火墙会将短时间内的端口重用连接请求当作攻击行为而拒绝，导致连接超时
       ii. 开启 TCP 连接中 TIME_WAIT sockets 的快速回收
       iii. 开启 SO_LINGER 选项，设置时间为 0
19. 必须明确约定报文中金额等数值的单位，防止 100 倍错误。
20. 程序处理时需要与浮点数进行转换，注意小数精度问题，防止一分钱更大误差
21. 用户/客户密码不得在网络报文中明文传输。
    a) 报文中的 PIN 是加密后的密文
    b) 有条件硬加密的机构不建议采用软加密
    c) 采用软加密时，应禁止应用程序将解密后/加密前的明文 PIN 记录在日志
    d) 根据相关安全要求决定是否对保温进行全报文加密
22. 必须显著区分请求报文、响应报文，防止混淆
23. 网络定长字段报文组包必须名确 C/C++字符串结束符是否余韵包含在字段中。接收方解析时必须考虑结束符后的垃圾数据可能造成的异常。
24. 网络定长字段报文组包、解析时必须考虑输入值超长时造成的半个汉字异常
25. 注意不同语言的各种书类型存储的字节数与存储和网络传输的字节序问题

26. 调用 socket 等通讯 API 出错时，要获取和记录引发错的原因（错误代码+信息）不允许只记录返回-1 作为失败返回码
27. 金融交易：前端读取回应超时，后台交易可能最终是成功的，应设计和提醒用户通过查询交易检查后台金融交易是否成功处理。调用记账返回超市，调用方无法确定记账结果，存在单边账的可能，应设计自动冲正机制且日终应对账。冲正交易要符合以下几点
    a) 冲正交易也可能超时，记账方应支持重复发起的冲正
    b) 多交易链路，可能出现冲正交易比原记账交易先到情况，双方必须明确冲正交易返回码表示冲正成功或失败的含义，避免理解偏差
    c) 用户前端发起的一笔交易，到交换平台、中间业务平台等系统变成多笔组合交易时，应采用谁负责组合谁负责处理超时等异常的冲正原则。
28. TCP 长连接的链路检测使用业务应用层面的心跳机制或者使用 TCP 协议自带的保洁功能。即设置 TCP Keepalive 的属性来实现，或两者配合使用
29. 交易重发设计：
    a) 底层的应用程序通讯模块，在 socket connect()函数连接失败时允许重试，但 send()函数发送失败原则上不允许重发交易报文，防止重复记账
    b) 交易处理的服务端应设计有检测控制重复交易报文机制（通过唯一性的发起方交易流水号），防止应用程序错误重发、黑客重放攻击，以及用户有意无意的重复提交交易。批量文件可以采用文件 MAC 值、HASH 值等方法检测重复文件
30. 双方约定应用层协议，处理 TCP 粘包或数据包不完整处理设计
    a) 广域网传输时，应在报文头加入校验字段，使用 CRC32 进行报文完整性校验。防止网络特殊故障造成报文不完整或损坏
31. 存放在共享内存中的配置参数，在第一个进程启动时初始化，要防止每个进程都错误的初始化
    a) 参数更新：共享内存中的配置参数更新，应通知每一个进程。但在一个交易过程中，不得出现不同步骤使用不同参数的情况
32. Linux 下 C/C++通讯程序若采用异步信号处理，避免使用 localtime 等不可重入函数
33. TCP；UDP；HTTP/HTTPS 通讯的服务端口必须可配置，不得写死在源程序中。TCP；UDP；HTTP/HTTPS 通讯
    a) Server 端应具备对 client 端 ip 白名单的检查功能
    b) Client 的 ip、端口应支持可配置是否 bind 以支持 server 的白名单检查
34. 使用 DNS 的应用系统，遵守 DNS 规定，系统之间连接配置使用 DNS 域名不再采用 IP
35. Rdp 默认端口是 3389

8. 外围系统卡 BIN 处理规范

1. 卡号开头数字

a) 62(6221226~622925):银联借记卡或贷记卡,国际通用,卡号16位,未来支持19位
b) 81: 银联在18年7月1日后启用的8位卡BIN, 卡号19位
c) BIN: (bank identification number)发卡行标识代码, 用于标识发卡机构的代码
   i. 根据ISO/IEC7812文件规定,9字头BIN号由一国国内的标准组织分配, 不适用于全球通用

2. 银联卡的卡号长度以及结构符合ISO7812-1有关规定, 由13-19位数表示:
   i. BIN(6位) + 自定义位（6-12位）+一位校验码
   ii. 自定义位由发卡行自行定义
   iii. 校验位根据校验数前的数字计算得到。计算方法在《银联卡发卡行标识代码及卡号》

3. 卡BIN数据
   a) 我行卡BIN数据两类（两个来源）
      i. 本行卡的卡BIN来自于核心系统
      ii. 他行卡的卡BIN数据来自于银联
   b) 我行未单独区分农民工卡,只是限定只有968807,968808,622690,622691几个卡BIN允许做农民工取款业务
   c) 银联业务卡表也包含我行发行的所有银联借记卡和贷记卡的卡BIN,卡BIN长度2-10
   d) 下游外围系统导入核心卡BIN表或者银联业务卡表文件数据异常时应有快速恢复前一版本的能力。导入数据时应对空值等不规范数据进行有效校验
   e) 卡BIN数据只用在明确是卡号的情况下查找发卡行。对本行卡根据业务需求有时区分借记卡和贷记卡

4. 区分卡号和账号：由于19位卡号以及银联“81”字头卡BIN的启用。我行AS/400新核心的19位“8”字头账号仅从数字看**无法区分**。数字长度或字头判断卡/折或路由本他行等逻辑也**不适用**。**以下逻辑都是错的:**
   i. 仅按16/19位数判断卡/折
   ii. 7/8开头的识别位存折账号
   iii. 转入账号为7/8开头→存折, 路由至核心。否则→卡转卡, 路由至银联
   iv. 根据客户提供的数字串在卡BIN表识别为银联银行卡。不在卡BIN表就送本行核心识别

   对于应用场景现有要素无法区分卡号/账号的, 建议在界面上给用户增加一个选项选择卡/账号

5. 从银行卡卡号判断发卡银行:
   a) 使用银联业务卡表, 遵循最长匹配原则:
      i. 根据卡号长度查找卡表
      ii. 从长至短截取卡号左边对应位数作为BIN判断‘发卡机构代码’
      iii. 银联卡表“卡BIN”号字段最大长度12位, 现有数据表明卡BIN可以是2-10位, 所以应从12位（或目前系统中最大卡BIN长度）循环递减到2位搜索

iv. 找不到相应记录，判断为非银联卡（外卡）或失效的银联卡

b) 一串数字判断为不是本行卡号时，可能是他行卡、本行账号、他行账号、都不是

c) “借贷合一卡”，在银联当贷记卡，在我行核心当借记卡

i. 柜面、收单、对私 BP 按现有规则判断‘借贷合一卡’

ii. 新建系统需要提方案给总工办审核

iii. ‘借贷合一卡’有下列 BIN:

1. ‘借贷合一 IC 卡普卡’：622767， 16 位
2. ‘******金卡’：622768， 16 位
3. ‘******白金卡’：622769， 16 位

d) 他行卡 BIN 不完全是 6 位长度

i. 必须配合卡号长度判断发卡行。

ii. 银联将某些相同的 6 位卡 BIN 分给了不同的银行，卡号长度不同：例如

1. 622302，18 位→工商银行
2. 622302，16 位→广东省农村信用社联合社

这是属于银联的历史问题

e) 外围系统从银联卡表数据中找不到相应记录→判断为非银联卡（外卡）或失效的银联卡 BIN，按业务部门相关要求处理

9. 外围系统响应码及错误信息规范

1. 响应信息相关原则：

a) 应用系统在交易返回响应信息中，

i. 必须包括：响应源系统代码、响应标志、响应码

ii. 建议返回具体响应提示/错误信息/用于拼组提示或错误信息的附加信息

b) 渠道端负责基于交易响应码转换对应响应提示或错误信息

i. 涉及海外分行/国外用户，要考虑信息的国际化，至少支持中英文+当地语言

c) 渠道端对技术类出错信息，原则上不能展示给客户

i. 技术错误信息/无明确业务规则的辅助应用信息,建议给客户“暂时未能处理您的请求”+后续操作指引信息引导客户

d) 涉及行内保密的业务限制规则或客户资金安全的保密信息，不得向用户展示

e) 响应码对应信息进行全行统一管理。通过企业级设计工具分类管理、发布、更新

f) 各系统主管业务部门负责响应提示或错误信息的审核及修订工作

g) 电子银行部负责电子渠道响应提示，或错误信息的审核、维护和定期监控工作

h) 错误信息须分级分类，提示须准确完整，不得误导用户

i) 涉及到行内响应码与银联等机构的特有响应码转换时，要考虑的原则：

i. 因涉及到交易成功率考核，应用系统应该具备响应码转换参数化

调整的功能

j) 原则上同一个不带参数的错误码只能对一个错误原因，不能有多个含义

k) 原则上不同的系统即使错误原因相同，也不能使用同一个错误码

l) 渠道端应能支持处理后台返回的带有参数的错误信息。

i. 返回的错误信息要符合错误信息展示要求

ii. 不符合要求的，渠道端应能将其进行转换

m) 后台系统使用带参数的响应码时，应明确发布响应码的前端错误信息展示的指引。未发布的须由该系统保证交易返回的参数转化后的错误信息适合向客户展示

n) 产品服务系统新增、修改、删除响应须向相关系统提供响应码及其业务确认过的相应信息，并在排期流程中确认

o) 交易链路上的中间系统处理接收到的错误码时不能改变交易状态

2. 响应码的构成规则

响应码应由 7 位字母+数字组成。第三位为必须遵循的规则，其他位建议规则如下：

a) 错误码标识，内容：系统自定义，字段类型：字符，长度：2， 取值：建议为两位字母，或字母＋数字。
目前核心系统的前两位是模块名称，建议其他系统参考

b) 错误码类型：描述错误代码的种类，字符，长度 1，取值

i. R-拒绝错误代码

ii. P-提示错误代码

iii. W-警告错误代码

iv. A-授权错误代码

c) 错误编码：字符，长度 4，取值：建议为 4 位数字

特殊响应码 7 个 A(即‘AAAAAAA’)是上述构成规则的唯一例外取值，表示交易成功

10.外围系统账号处理规范

1. 术语:本行/他行账号中的”账号”表示账号和卡号
2. 我行账户体系分为：客户、客户账户、系统账户、附属账户

a) 存款客户账户：

i. 普通账户（非卡）

1. 存量账户，保持不变，科目代号不再有含义
19 位 = 开户网点（6）+币种（2）+科目代号（3）+序号（6）+校验位（2）
2. 新开账户
存量账户以“7”开头，新账号以”8”开头
3. 支持客户自选号码，校验位放前面
4. 序号 8 位，分行排序，每个分行每个币种，每个业务种类最多支持 1 亿客户
19 位 = ‘8’+ 法人编号（1）+ 开户分行号（3）+币种（2）+业务种类（2）+校验位（2）+序号

ii. 银行卡：按银联标准的卡 BIN
借记卡卡号 16 位，未来支持 19 位
信用卡有
16 位：银联 visa 和 mastercard
15 位： 运通卡

b) 本行账号的识别：通过：位数+起始位+第 9 位可以识别 10 种账号：内部账、存款客户账户、附属账户、贷款账户

3. 他行账号：
a) 原则上业务范围在以下交易接口中：
i. 本外币境内外支付结算
ii. 中间业务类
iii. 速汇金资金清算
iv. 表外资产
v. 国际业务
vi. 单位客户账户业务
vii. 集团现金他行账户业务
viii. 对外转账汇款业务
b) 识别他行账号：
i. 位数超过 19 或小于 15 → 他行账号
ii. 位数等于 15 或 16 或 19，如果明确为卡号，前 6 位或 8 位卡 BIN 信息不在我行范围内 →他行账号
iii. 位数等于 19 的，若明确为账号。按本行账号识别方法未能识别为本行账号的 → 反向识别为他行账号

4. 外围系统：应该严格遵循 各系统服务治理接口规范 或业务系统之间既定的规则中 对账号的处理规范要求。在调用服务和接口赋值前主要甄别本行/他行账号

5. 本行账号处理规范
a) 输入界面按之前规则识别账号，并进行基础合法性校验：
i. 输入允许减号、空格、自动分段
ii. 报文发送前格式化
iii. 接收方自动格式化去掉减号、空格
b) 输入界面应按交易要求，对账号校验位进行校验
i. 检查校验位算法不公开，提供公共函数库
ii. 禁止校验位算法写在 js/vbs 前端代码中
iii. 卡号校验位检查按银联标准
c) 对外围系统，
i. 一般不应以账号中要素进行逻辑控制。
ii. 但对新账号规则已经明确的一些要素：（币种、业务种类）可以
iii. 账号中分行号不做判断（目前有跨分行网点撤并、转移）

6. 他行账号处理规范
a) 同 5（a）
b) 他行账号不验证校验位
c) 他行账号不得当作本行账号送往核心查询账户状态、余额
d) 他行账号应不小于 32 位，允许本行和他行账号的字段长度不小于 32

e) 他行卡号 BIN 长不是固定 6 位

11.应用对账技术规范

1. 对账差异时，以哪一方的交易、账务信息为准，则哪一方称为标准方。
2. 常见标准方：
   a) 以我行为标准方
   b) 账户记账方
   c) 交易发起方
   d) 第三方
   e) 按业务特点约定的标准方
3. 对账差异调整方冲补账时采用的手段方式：
   a) 自动冲补账：程序依据对账结果与既定对账标准，自动执行冲补账
   b) 半自动冲补账：程序生成对账结果文件，相关业务人员审阅后，启动专门的冲补账流程/重发交易/批量或逐笔冲补账
   c) 手工冲补账：根据对账结果由相关业务人员设计冲补账会计分录来冲补账
4. 实时对账自动冲补账：
   a) 支持实时对账的交易，一般是同步交易
   b) 若无交易成功返回响应，应支持自动查询交易是否成功
   c) 并决定自动冲账或者重做交易
5. 完整的登记交易登记簿
   a) 对于每一笔交易，成功与否都要登记
   b) 因失败而冲销的交易也要登记而非删除
   c) 以供后续对账时信心的完整性，和将来使用
6. 批量对账报表设计：两份对账报表
   a) 一份是全量对账报表，数量太大时，可考虑设计汇总对账报表
   b) 一份是对账差异报表
7. 批量对账差异处理
   a) 自动处理：须提供完整的报表供业务人员次日查看
   b) 半自动：业务人员确认后，可发起专用交易批量/逐笔冲补账。支持当日/隔日冲账方式
   c) 减少手工方式处理对账差异

代码评审

1. 代码走查规范

1. 代码走查形式：
   a) 会议走查：组织评审会议，在会上开发人员讲解，评审人员提出缺陷与问题

b） 工位走查：不组织会议，在工位上讲解与评审
c） 交叉走查：组织开发人员两两交叉检查
d） 单人检查：不讲解，专家静态阅读代码来检查代码质量
建议以会议走查

2. 代码走查作用
a） 发现缺陷
b） 传播知识
c） 提升编码水平

3. 对代码走查活动的要求
a） 开发项目必须在上线前完成代码走查活动
b） 代码走查至少要覆盖核心模块
c） 逻辑复杂代码和新人负责的代码
d） 走查完成以后，必须解决发现的缺陷并编写《代码走查报告》。附在bufferfly 工具的移交测试和上线申请流程

4. 代码走查参与角色
a） 项目经理
b） 程序开发人员
c） 检察人员
d） 记录人员