Website: ycyk.brit.com.cn Email: ycyk704@163.com

位控类时间指令的实时处理方法

石 峰
（中国西安卫星测控中心 西安 710043）

摘要：针对火箭遥测实时处理软件中不同位跳变指令处理代码复用性差的问题，通过分析几种典型的位跳变指令处理特点，抽象出位跳变指令处理的一般步骤，并从软件角度给出了位跳变指令的定义。根据 C++语言多态性原理，将位跳变指令的软件实现划分为抽象基类和实现子类两个部分，规范了位跳变指令的处理逻辑、体现了不同种类位跳变指令处理特点。应用实践表明，经过重构的处理方法代码量显著减少，程序处理流程更加清晰，软件复用率有很大提高。

关键词：位跳变指令；实时处理；遥测
中图分类号：TP311.1 **文献标识码：**A **文章编号：**CN11-1780(2022)01-0082-09
DOI：10.12347/j.ycyk.20210606001
引用格式：石峰. 位跳变时间指令的实时处理方法[J]. 遥测遥控, 2022, 43(1): 82–90.

Real-time processing method of bit control instruction

SHI Feng
（Xi’an Satellite Control Center, Xi’an 710043, China）

Abstract: In aiming at the problem of poor reusing of different bit control instruction processing codes in rocket telemetry real-time processing software, the general steps of bit control instruction processing are abstracted by analyzing the processing characteristics of several typical bit control instructions, and the definition of bit control instruction is given from the point of view of software. According to the principle of C++ language polymorphism, the software implementation of the bit control instruction is divided into abstract base class and implementation subclass, the processing logic of the bit control instruction is standardized, and the processing characteristics of different bit control instructions are presented. The application practice shows that the amount of code is significantly reduced, the process flow is more clearer, and the software reuse rate is greatly improved.

Key words: Bit conversion instruction; Real-time processing; Telemetry
DOI：10.12347/j.ycyk.20210606001
Citation: SHI Feng. Real-time processing method of bit control instruction[J]. Journal of Telemetry, Tracking and Command, 2022, 43(1): 82–90.

引 言

时间指令是反映运载火箭某一动作及控制行为是否发生及发生时刻的关键参数，如火箭起飞触点接通时间、爆炸螺栓起爆时间、发动机点火指令时间、星箭分离指令时间等，是主动段对火箭飞行状态评估的重要信息来源。位跳变时间指令是时间指令应用最为广泛的一种类型，文献[1]对早期的位跳变时间指令的处理方法、实时处理时存在的不足及改进设想进行了一些探讨。近年来，随着新型号火箭相继首飞[2]，新的参数处理方法层出不穷，位跳变时间指令衍生出高精度型、遥测字型、总线型、电压型等多种处理类型。关于此类指令的处理方法虽然设计思路大致相同，但是由于需求差异、人员安排等因素，其实现手段均有所不同，软件复用性较差。

在每次火箭飞行试验任务前，虽然型号研制部门都要发布一份关于飞行试验任务遥测数据处理要求和方法的技术文件，但该文件仅给出了处理上的一些原则性要求，以及参数原码与物理量之间的转换公式，作为数据处理方法是远远不够的。火箭遥测实时处理软件在进行位跳变指令处理时还需要对遥测数

收稿日期：2021-06-06 收修改稿日期：2021-07-08

据的连续性，时标修正的准确性，发生时刻的合法性等诸多方面进行综合处理后，才能计算出时间指令发生的真实时间信息，软件处理过程比较复杂。本文发挥 C++语言“多态性”优势，采用虚函数，研究了各类位跳变时间指令的处理逻辑，提出了位跳变时间指令的通用化处理方法，同时在国产麒麟系统上对现有的处理方法库进行重构，增加了抽象化的处理基类，明确了位跳变时间指令的一般处理步骤，在继承基类处理方法的基础上实现了各类型处理子类，体现了不同类型位跳变指令的处理特点。

1　位跳变指令的种类及特点

1.1　经典的位跳变指令

这类指令是用编码信号中某一位由“0”变“1”（或“1”变“0”）表示指令出现，连续保持 3 帧以上为有效，以其中第一次出现变化的时间为准，计算该指令的发生时刻[3]。由于该指令属于缓变参数，因此通常安排在缓变副帧中进行传输。以目前常见的火箭遥测全帧数据 25 ms 的发送周期计算，该类指令的采样频率为 40 帧/秒。

1.1.1　数据连续性判断

数据的连续性是火箭遥测实时处理软件必须考虑的重要指标。地面遥测设备接收火箭遥测数据时难免出现误码和中断现象，表现在遥测数据中就是数据的不连续。位跳变指令是一类连续变化的参数，其判断逻辑必须依靠缓存的历史数据，如果数据不连续必然导致后续判断过程无法进行或计算出错误的结果。目前，常用的连续性判断方法是利用帧计数的连续计数机制。如某型火箭遥测数据的帧计数用两个字节的无符号整数表示，其值的变化范围是 0~65535，如果接收到的遥测帧的帧计数值是在前一帧的基础上加 1，则认为目前遥测数据是连续的，否则是不连续的。

1.1.2　数据时标的计算

位跳变指令的每一个采样点都对应一个准确的采样时刻，这也是时间指令判断的意义所在。当地面遥测设备接收了一个完整的遥测帧时，将当前时刻作为帧时标编排在遥测数据中进行传输，该时标可以作为时标计算的依据，一般用时标减去火箭起飞时间，即可得到时间指令的发生时间。

1.1.3　数据跳变的判断

数据跳变的判断可以概括为以下 3 个步骤：

步骤 1：获取初值。如某时间指令编码信号某一位为“0”时表示事件没有发生，当变为“1”时表示事件发生，当处理软件连续接收到 3 帧“0”即取到初值，如果没有满足条件则始终处于获取初值状态。

步骤 2：判断跳变。当处理软件连续收到 3 帧“1”时任务跳变发生，取第一帧时间为跳变发生时间，如果没有满足条件则重新判断跳变。

步骤 3：判断时间合法性。当步骤 2 完成跳变判断并计算出发生时间时，对时间合法性进行判断，如果发生时间在理论范围内则判断完成，否则重新进入步骤 1。

另外有一种高精度型位跳变指令，其处理方法与经典型位跳变指令基本一致，只是采样频率更高，得到的时间指令发生时间更精确。

1.2　四位跳变的位跳变指令

这类指令是用编码信号中的四位由“0”变“1”（或“1”变“0”）表示指令出现，主要用来表示时序参数。如某型火箭的关机时间有七次，时序参数对应的四位码每改变一次状态，即从“0H”变为“FH”，或从“FH”变为“0H”，表示一次关机事件的发生。

这类指令除了保持与经典位跳变指令相似的处理逻辑外，还具有自身的处理特点。一是在判断跳变的过程中，为了防止误码的发生，实时处理时仅要求四位编码信号中的三位发生改变就可以判断跳变发生。这需要对四位编码信号分别比对，比经典位跳变指令的跳变判断过程略微复杂一些。二是在判断时间合法性时，这类指令没有固定的初值，并且同一波道表示多条指令，只能依靠理论时间判断指令发生时间合法性；同时在前后两条指令之间需要建立关联，形成依赖指令，当前一条指令正确判出后，后一条指令才能开始判断。

1.3　遥测字型位跳变指令

这类指令通常由箭上的时序控制器采集，采用遥测字的方式传输编码信号。如某型火箭每组遥测字长 32 位，遥测字由 26~31 位组成的特征码及 0~26 位组成的若干参数构成。由不同的特征码表示不同的参数，在一定周期内循环传送。

这类指令与经典的位跳变指令的主要区别在取位方式和数据时标的计算上。当处理软件接收到一组遥测字时，首先要判断遥测字的特征码是否与位跳变指令所对应的特征码一致，如果满足要求才能根据指令对应的位置取出编码信号进行判断。另外，编码信号对应的采样时间也在一组遥测字中传输，当指令判出后，以最近的遥测字传输的时间为指令发生时间。

1.4　总线型位跳变指令

总线是近年来应用比较广泛的数据传输协议。它将各类采编器产生的不同数据采用数据包的形式在总线上传输，可以在有限的波道资源上传递更多的数据。采编器产生的时间指令被封装在各自的数据包中并通过不同的命令字加以区分。

这类指令的处理依赖于对总线协议的解码，这类工作一般由单独的处理软件完成。当时间指令处理软件接收到解码后的包数据，需要对数据包中的命令字进行判断，如果与待处理的时间指令的命令字一致，则取出对应的编码信号进行处理。同时，总线数据在进行传输时会定时发送时间戳，即包含时间的数据包，当指令判出后，如果所在包中不包含时标信息，可以以最近的时间戳作为指令的发送时间。

1.5　电压型位跳变指令

电压型位跳变指令严格意义上说不能称之为位跳变指令，因为它实际上是箭上测量设备输出的一种脉冲信号。但我们仍然可以将经典的位跳变指令定义进行推广，将电压型位跳变指令归入其中。这也是面向对象思想的魅力所在。

这种指令在处理时首先要计算出遥测电压值，然后根据变换器校准表进行差值计算，得出被测电压值。如某型火箭理论输出为 28 V 的宽脉冲信号，取 20 V 作为数据处理门槛，处理信号开始和结束时间。当指令未发生时，遥测电压保持在 0 V 左右，指令处于类似经典位跳变指令的“0”状态，当指令发生时，遥测电压应大于 20 V，指令处于类似经典位跳变指令的“1”状态。

1.6　小结

经典位跳变指令等 6 种位跳变指令的概念和处理特点见表 1，可以看出其判断过程遵循比较一致的处理逻辑，即指令遥测值从一种状态变化为另一种状态表示指令发生。在对处理方法进行独立开发时，由于处理逻辑的相似性必然会导致重复代码的存在；又由于不同的开发人员设计思路不同，各种中间变量、命名方式不能做到集中统一，导致方法间无法进行代码复用。

表 1　位跳变指令的处理特点

Table 1　Bit control instruction processing characteristics

指令类别	类名	主要步骤									
		循环遍历缓冲区	取初值	比较跳变	跳变时间合法性判断	记录帧计数	帧连续性判断	保存时标	计算跳变时间	取位	依赖判断
经典型	CFtmBitCls	※	※	※	※	※	※	※	√	√	○
高精度型	CFtmBitHacu	※	※	※	※	※	※	※	√	√	○
四位跳变型	CFtmBitFour	※	※	※	※	※	※	※	√	√	√
遥测字型	CFtmBitWrd	※	※	※	※	※	※	√	√	√	○
总线型	CFtmBitBus	※	※	※	※	○	○	√	√	√	○
电压型	CFtmBitVolt	※	※	※	※	※	※	※	√	√	○
					※ 复用		√ 特殊处理		○ 不涉及		

2　C++多态性相关技术

多态性是面向对象编程的特征之一。多态性使程序具有“一个接口，多个版本”的特点，同时也体

现了程序由“一般到个别”的特点。C++提供了两种多态性，编译时刻的静态多态性和运行时刻的动态多态性。本文讨论的是运行时刻的动态多态性。动态多态性与继承相关，有两个方面：子类型关系所实现的类型多态性和虚函数所实现的行为多态性[4,5]。

2.1　继承与派生

继承性是类之间的一种关系，表示一个比较抽象的类与一个比较具体的类之间的关系。在继承性关系中，基类表示比较抽象的概念，拥有较少的内涵，而具有较大的外延。派生类表示比较具体的概念，具有较多的内涵，而具有较小的外延。派生类拥有基类描述的属性，派生类的定义是基于基类存在的。派生类的对象也是基类的对象，当需要一个基类对象的地方，而实际提供了一个派生类对象，应该是无条件满足要求的。

2.2　虚函数与抽象类

虚函数是用 vitual 关键字修饰的成员函数。一个类中的虚函数可以在其派生类中重新定义，这就是改写（override）。所谓改写就是派生类中对其基类中的虚函数按相同的基调（函数名和形参表）和返回值，进行重新定义。当通过基类的引用或指针来调用虚函数时，实际执行的是派生类改写后的虚函数，而不是基类定义的虚函数。虚函数的核心思想是将行为规范与具体实现方案分离开来，将函数调用与具体执行分离开来。规范是抽象的，由基类说明并提供默认实现，而实现是具体的，派生类可以继承也能改写。这样的好处是灵活性、适应性、可扩展性而不失规范性。在定义一个基类时，对于一个虚函数能确定其行为规范，但往往还不能提供一种具体实现，其具体实现完全依赖派生类。这时可以把这个虚函数定义为纯虚函数。纯虚函数就是在类中只提供行为规范而没有具体实现的虚函数。这时基类就成为了抽象类。

下面，通过一段伪代码来说明虚函数的调用过程，如图 1 所示。设计位跳变指令的处理基类 CJFtmBitBase，其中包含一个采集时标的纯虚函数 save_time_mark()和一个用来存储时标的浮点型变量 time_mark。经典位跳变指令处理子类 CJFtmBitCls、总线型位跳变指令处理子类 CJFtmBitBus 继承自基类 CJFtmBitBase。在两个子类中分别对纯虚函数 save_time_mark() 进行改写。在主函数 main 中，将子类对象地址分别赋予基类对象指针，并以基类对象指针调用采集时标的纯虚函数 save_time_mark()。运行此程序可得结果：

①经典位跳变指令采集时标；

②总线型位跳变指令采集时标。

```
class CJFtmBitBase {
        public:
        double time _mark ;
        virtual void save _time_mark () =0;
};
class CFtmBitCls  :public CJFtmBitBase {
        public:
        virtual void save _time_mark (){
        cout<<" 经典位控指令采集时标。"<<endl;
        }
};
class CFtmBitBus  :public CJFtmBitBase{
        public:
        virtual void save _time_mark (){
        cout<<" 总线型位控指令采集时标。"<<endl;
        }
};
void main (){
        CJFtmBitBase * base[2];
    CFtmBitCls  son _cls;
        CFtmBitBus son _bus;
    base[0] =&son_cls;
        base[1] =&son_bus;
    base[0]-> save_time_mark ();
    base[1]-> save_time_mark ();

}
```

图 1　伪代码

Fig. 1　Pseudocode

2.3　多态性实现机制与原理

必须使用基类的指针或引用来调用虚函数才能实现动态多态性。当一个类中含有虚函数声明时，编译器就会为该类加上一个成员变量，是一个指向该类虚函数表的指针(VPTR)。而虚函数表是一个存放指向类中虚函数的函数指针的线性表。在子类中如果改写了虚函数，则表中的函数指针就会被改写。这种在运行时根据不同的数据类型匹配正确的处理函数的能力在 C++语言中称之为运行时类型识别（RTTI）。对象的内存布局如图 2 所示。

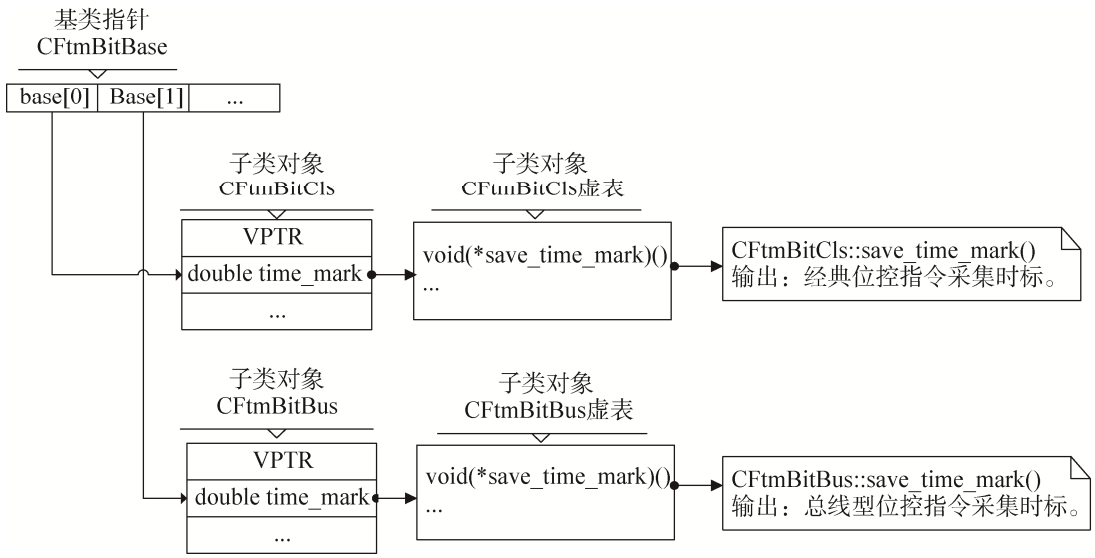

图 2　对象的内存布局

Fig. 2　The memory layout of the objects

每一个对象开始的 4 个字节，存放的是一个虚表指针，这个指针指向一个虚函数指针数组。子类 CJFtmBitCls、CJFtmBitBus 分别复制继承了基类 CJFtmBitBase 的虚表，由于各子类重新定义了虚函数 save_time_mark()，所以在虚表中对应函数的地址也相应地改为子类改写后的虚函数地址。

3　位跳变指令的实时处理方法设计

目前，常见的帧格式有两种：一种是原始跳点数据格式，如图 3 所示，是一种定长数据。该格式将 1 秒内收到的全帧数据根据跳点要求循环跳点并组帧发送。循环数据区长度和循环次数是固定的，每一帧带有时标，循环数据区的采样时间可以根据全帧周期计算出来，循环数据区的帧计数作为数据连续性的判断依据。经典型、四位跳变型、遥测字型、电压型位跳变指令采用这种格式。另一种是总线包数据格式，如图 4 所示，是一种变长数据。该格式将 1 秒内解出的总线包按不同命令字分别组包，每帧数据包含的数据包由包个数表示。该数据帧头不包含时标，时标包含在数据包内。该数据帧也不包含帧计数，因此在处理时对包数据的连续性判断不作要求。

	参数名称	代号	字节
	时标	T	4
第一遍	帧计数	FrCnt	2
	经典型	FrmBit	1
	四位型	BitFor	1
	电压型	BitVolt	1
	遥测字型	BitWrd	4
	...		
第*N*遍	帧计数	FrCnt	2
	经典型	FrmBit	1
	四位型	BitFor	1
	电压型	BitVolt	1
	遥测字型	BitWrd	4

经典型

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	0	0	1

四位跳变型

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	1	1	1	1

遥测字型

特征码	数据区		
B31-B24	B23-B16	B15-B8	B7-B0
110001			0 0 0 0 0 0 0 1
110010	时标		其它参数
110011	其它参数		

电压型

B7-B0
0-255

图 3　原始跳点数据格式

Fig. 3　Raw pick point data format

	参数名称	代号	字节
	命令字	CMD	2
	包个数	N	2
第一遍	数据包	Date	8
	...		
第*N*遍	数据包	Data	8

总线型

<table>
<tr><td colspan="8">数据包</td></tr>
<tr><td>B63-B56</td><td>B55-B48</td><td>B47-B40</td><td>B39-B32</td><td>B31-B24</td><td>B23-B16</td><td>B15-B8</td><td>B7-B0</td></tr>
<tr><td colspan="2">时标</td><td colspan="2">其它参数</td><td colspan="2">0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1</td><td colspan="2">其它参数</td></tr>
</table>

图 4 总线包数据格式

Fig. 4 Bus packet data format

从图 3、图 4 中可以看出，各类位跳变指令均处于所在帧的循环数据区，因此，遍历每个循环数据区，对包含的位跳变指令进行处理，判断出时间指令的发生时间是各类处理方法都要遵循的一般处理步骤。按照这个思路，以经典型位跳变指令为模板，可以设计出位跳变指令的处理方法。将其他位跳变指令处理过程代入其中，对处理方法进行迭代，梳理出适应各类处理方法的一般处理步骤，如图 5、图 6 所示。

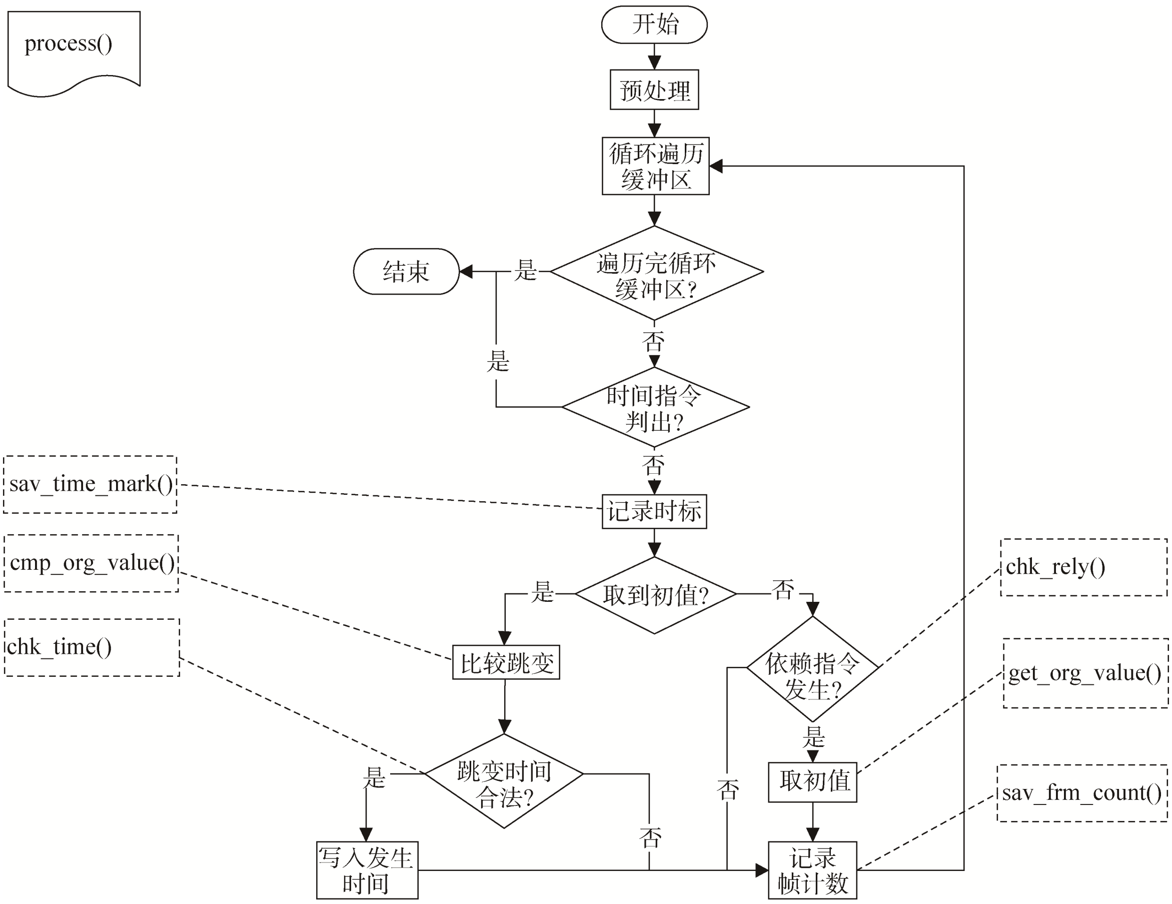

图 5 函数流程图 1

Fig. 5 Function flow chart 1

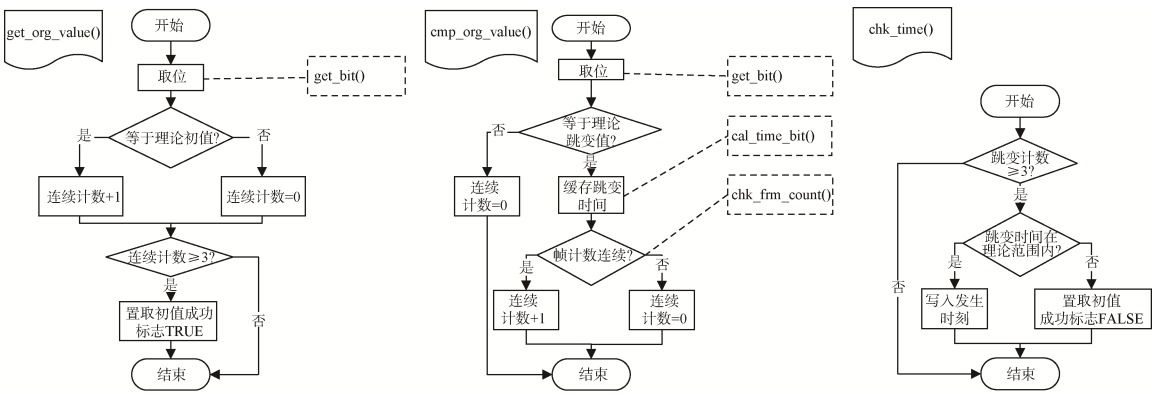

图 6　函数流程图 2

Fig. 6　Function flow chart 2

3.1　定义

根据位跳变指令的处理特点给出一般定义：位跳变指令包含两个状态，当指令未发生时为初态，指令发生时为终态，当指令由初态变为终态时即表示指令发生，取当前时刻为时间指令的发生时间。为了描述方便，对待处理的遥测数据做如下假定：

① 遥测数据频率为 1 帧/秒，其中包含的指令波道按一定步长循环存储；

② 循环数据区的长度和循环次数是已知的或方便得到的；

③ 循环数据区包含帧计数；

④ 遥测数据包含时标。

3.2　抽象基类设计

设计位跳变指令的抽象基类 CJFtmBitBase。该基类包含非虚函数和虚函数两个部分。其中非虚函数 process()设置为 public 属性，对外可见，是位跳变指令处理方法的调用接口。该函数是根据位跳变指令的定义，抽象出的一般处理步骤，各类位跳变指令都遵从相同的处理步骤，从而达到“一个接口”的目的。

为了保证程序代码的逻辑清晰、层次清楚，将图 1 中获取初值、比较跳变、跳变时间合法性判断等三个步骤提炼为非虚函数 get_org_value()、cmp_org_value()、chk_time()，设置为 private 属性，对外不可见。函数中封装了对应步骤的具体实现，流程图如图 6 所示。以上函数封装了位跳变指令的一般处理逻辑，子类中不需要实现。

抽象基类 CJFtmBitBase 包含六个虚函数，分别是：sav_frm_count()，对应于图 5 的记录帧计数步骤；chk_frm_count()，对应于图 6 的帧计数是否连续步骤；sav_time_mark()，对应于图 5 的记录时标步骤；cal_time_bit()，对应于图 6 的缓存跳变时间步骤；get_bit()，对应于图 6 的取位步骤，chk_rely()，对应于图 5 依赖指令是否发生步骤，该步骤仅针对波道复用的指令，如四位跳变型，其他类型直接返回即可。这六个虚函数属性为 protected，仅对子类可见，涵盖了不同位跳变指令之间的差异部分，在基类设计时仅保留调用接口，具体的实现步骤由子类根据具体需求进行代码编写，从而体现了“多个版本”的特点。

3.3　子类设计

子类的设计集中在对基类声明的六个虚函数进行改写上。每种位跳变指令可以根据自身的处理特点，在相应的虚函数中编写处理方法，完成具体的处理任务。由于位跳变指令处理方法的设计是以经典型位跳变指令为设计原型，考虑到大部分火箭实时遥测数据格式基本一致，因此在基类中保留了虚函数的默认实现方式，没有改写的虚函数调用基类提供的默认处理方法。以遥测字型位跳变指令为例，由于它与经典位跳变指令仅在取位方式和数据时标计算上存在不同。因此需要改写 sav_time_mark()、cal_time_bit()、get_bit()三个函数，其余函数继承基类处理方法即可。从图 7 中可以看出，经典型、四位跳变型、电压型的处理方法最为相似，仅在取位方法上有所不同，高精度型由于采样精度更高，因此时

标计算方法与其他的不同。遥测字型和总线型由于输入数据的组合方式不同，与其他处理方法差异较大，需要改写更多的处理方法。

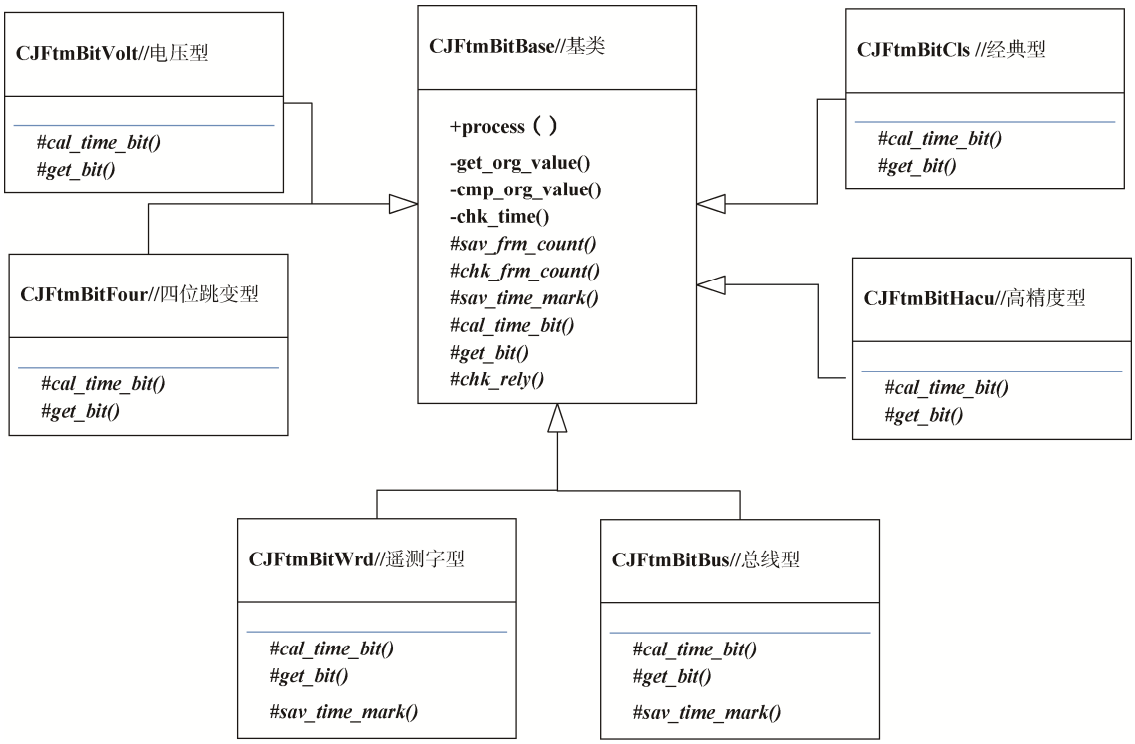

图 7　类图

Fig. 7　Class

4　重构后的代码特点

4.1　代码行数显著减少

在 Linux 服务器上，采用 Magic C++开发环境，根据以上设计原则，对现有的六种位跳变指令方法库重构后的结果列于表 2 中。由于主要的处理方法被封装在基类 CJFtmBitBase 中，各子类的代码行数显著减少。

表 2　重构前后代码统计

Table 2　Code statistics before and after refactoring

指令类别	类名	重构前代码行	重构后代码行	复用率（%）
经典型	CFtmBitCls	361	28	92
高精度型	CFtmBitHacu	371	24	94
四位跳变型	CFtmBitFour	403	46	89
遥测字型	CFtmBitWrd	536	68	87
总线型	CFtmBitBus	540	68	87
电压型	CFtmBitVolt	510	26	95

4.2　开发过程更加优化

代码重构前，对于位跳变时间指令的需求分析采用的是面向过程的编程思路，即将处理过程分为不同的处理步骤，然后对处理步骤进行细化形成不同处理方法，再进行代码编写。软件编写完成后要对所编写的所有函数进行单元测试，最后还要对方法库进行系统测试。整个过程费时、费力。采用面向对象的设计思想后，对需要开发的位跳变指令首先进行类型识别，即判断处理需求是否满足位跳变指令一般定义要求，如果确认此次开发是位跳变指令的一种相似处理方法，则只需识别出处理差异，并针对差

异改写相应的处理方法。采用这种方法，需求分析和代码编写测试阶段工作量大大减轻，开发效率显著提高。

4.3 编码风格更加统一

采用面向对象开发方法后，从软件实现角度对位跳变指令的处理方法提出了统一的处理定义。采用抽象基类的设计，对位跳变指令的处理流程进行了规范，并对不同的处理步骤提供了一致的调用接口，解决了由于开发人员不同，编码风格差异，造成代码维护困难的问题。同时由于采用了子类设计，位跳变指令的处理种类一目了然，方便从宏观角度准确掌握现有处理方法底数，对箭遥处理软件的规范化管理和确定任务的继承基线都有较大的帮助。

5 结束语

本文通过梳理几种位跳变指令的处理方法，总结出位跳变指令的主要处理逻辑，给出了位跳变指令的一般化处理定义，设计了位跳变指令抽象基类及实现子类。采用基类指针作为位跳变指令处理方法的调用接口，保证了处理方法的规范统一，同时使软件具有较好的可扩展性。通过虚函数机制在实现子类中改写指定函数的实现代码，体现了不同指令类型的处理特点。使用该编码方法使处理流程更加清晰，方便不同开发人员熟悉掌握，同时较好的代码复用使开发工作量大幅减少，提高了编码效率和规范性。

参考文献

[1] 茅永兴, 朱伟康, 刘冰, 等. 箭遥时间指令参数实时处理方法研究[J]. 遥测遥控, 2013, 34(3): 72–75.
MAO Yongxing, ZHU Weikang, LIU Bing, et al. A fusion method of real-time and delay telemetry data[J]. Journal of Telemetry,Tracking and Command, 2013, 34(3): 72–75.

[2] 周彬, 郭亮杰, 田鸿瀛, 等. 我国箭载遥测传输设备发展现状及展望[J]. 遥测遥控, 2017, 38(6): 7–9.
ZHOU Bin, GUO Liangjie, TIAN Hongying, et al. Development status and prospect of chinese telemetry transmission equipments on rockets[J]. Journal of Telemetry,Tracking and Command, 2017, 38(6): 7–9.

[3] 陈以恩, 张俊刚, 袁嗣杰, 等 .遥测数据处理[M]. . 北京: 国防工业出版社, 2002 年.

[4] 罗粮. C++中通过 VTABLE 实现多态性机制的分析[J]. 电脑编程技巧与维护, 2014(11): 46–47, 93.
LUO Liang. Analysis of polymorphism mechanism through VTABLE in C++[J]. Computer Programming Skills and Maintenance, 2014(11): 46–47, 93.

[5] 谢云博. 多态性实现机制在 C++与 JAVA 中的比较分析[J]. 软件导刊, 2014, 13(6): 47–49.
XIE Yunbo. Comparative analysis of polymorphism implementation mechanism in C++ and Java[J]. Software Guide, 2014, 13(6): 47–49.

[作者简介]
石 峰 1977 年生，本科，高级工程师，主要研究方向为火箭遥测实时处理。

(本文编辑：潘三英)