DELPHI的原子世界(2)计算机等级考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/643/2021_2022_DELPHI_E7_9A_84_E5_c97_643724.htm TClass原子 在System.pas单元中，TClass是这样定义的： TClass = class of TObject. 它的意思是说，TClass是TObject的类。因为TObject本身就是一个类，所以TClass就是所谓的类的类。 从概念上说，TClass是类的类型，即，类之类。但是，我们知道DELPHI的一个类，代表着一项VMT数据。因此，类之类可以认为是为VMT数据项定义的类型，其实，它就是一个指向VMT数据的指针类型！ 在以前传统的C 语言中，是不能定义类的类型的。对象一旦编译就固定下来，类的结构信息已经转化为绝对的机器代码，在内存中将不存在完整的类信息。一些较高级的面向对象语言才可支持对类信息的动态访问和调用，但往往需要一套复杂的内部解释机制和较多的系统资源。而DELPHI的Object Pascal语言吸收了一些高级面向对象语言的优秀特征，又保留可将程序直接编译成机器代码的传统优点，比较完美地解决了高级功能与程序效率的问题。 正是由于DELPHI在应用程序中保留了完整的类信息，才能提供诸如as和is等在运行时刻转换和判别类的高级面向对象功能，而类的VMT数据在其中起了关键性的核心作用。有兴趣的朋友可以读一读System单元的AsClass和IsClass两个汇编过程，他们是as和is操作符的实现代码，以加深对类和VMT数据的理解。 有了`类的类型，就可以将类作为变量来使用。可以将类的变量理解为一种特殊的对象，你可以象访问对象那样访问类变量的方法。例如：我们来看看下面的程序片段： type TSampleClass = class of

TSampleObject. TSampleObject = class(TObject) public constructor Create. destructor Destroy. override. class function GetSampleObjectCount:Integer. procedure GetObjectIndex:Integer. end. var aSampleClass : TSampleClass. aClass : TClass. 在这段代码中，我们定义了一个类TSampleObject及其相关的类类型TSampleClass，还包括两个类变量aSampleClass和 aClass。此外，我们还为TSampleObject类定义了构造函数、析构函数、一个类方法GetSampleObjectCount和一个对象方法GetObjectIndex。 首先，我们来理解一下类变量aSampleClass和aClass的含义。 显然，你可以将TSampleObject和TObject当作常量值，并可将它们赋值给aClass变量，就好象将123常量值赋值给整数变量i一样。所以，类类型、类和类变量的关系就是类型、常量和变量的关系，只不过是在类的这个层次上而不是对象层次上的关系。当然，直接将TObject赋值给 aSampleClass是不合法的，因为aSampleClass是TObject派生类TSampleObject的类变量，而TObject并不包含与TSampleClass类型兼容的所有定义。相反，将TSampleObject赋值给aClass变量却是合法的，因为TSampleObject是TObject的派生类，是和TClass类型兼容的。这与对象变量的赋值和类型匹配关系完全相似。 然后，我们再来看看什么是类方法。 所谓类方法，就是指在类的层次上调用的方法，如上面所定义的GetSampleObjectCount方法,它是用保留字class声明的方法。类方法是不同于在对象层次上调用的对象方法的，对象方法已经为我们所熟悉，而类方法总是在访问和控制所有类对象的共同特性和集中管理对象这一个层次上使用的。在 TObject的定义中，我们可以发现大量

的类方法，如ClassName、ClassInfo和NewInstance等等。其中，NewInstance还被定义为virtual的，即虚的类方法。这意味作你可以在派生的子类中重新编写NewInstance的实现方法，以便用特殊的方式构造该类的对象实例。 在类方法中你也可使用self这一标识符，不过其所代表的含义与对象方法中的self是不同的。类方法中的 self表示的是自身的类，即指向VMT的指针，而对象方法中的self表示的是对象本身，即指向对象数据空间的指针。虽然，类方法只能在类层次上使用，但你仍可通过一个对象去调用类方法。例如，可以通过语句aObject.ClassName调用对象TObject的类方法ClassName，因为对象指针所指向的对象数据空间中的头4个字节又是指向类VMT的指针。相反，你不可能在类层次上调用对象方法，象TObject.Free的语句一定是非法的。 值得注意的是，构造函数是类方法，而析构函数是对象方法！ 什么？构造函数是类方法，析构函数是对象方法！有没有搞错？ 你看看，当你创建对象时分明使用的是类似于下面的语句： aObject := TObject.Create. 分明是调用类TObject的Create方法。而删除对象时却用的下面的语句： aObject.Destroy. 即使使用Free方法释放对象，也是间接调用了对象的Destroy方法。 原因很简单，在构造对象之前，对象还不存在，只存在类，创建对象只能用类方法。相反，删除对象一定是删除已经存在的对象，是对象被释放，而不是类被释放。 最后，顺便讨论一下虚构造函数的问题。 在传统的C 语言中，可以实现虚析构函数，但实现虚构造函数却是一个难题。因为，在传统的C 语言中，没有类的类型。全局对象的实例是在编译时就存在于全局数据空间中，函数的局部对象也是编译时就在堆栈空间中映

射的实例，即使是动态创建的对象，也是用new操作符按固定的类结构在堆空间中分配的实例，而构造函数只是一个对已产生的对象实例进行初始化的对象方法而已。传统C 语言没有真正的类方法，即使可以定义所谓静态的基于类的方法，其最终也被实现为一种特殊的全局函数，更不用说虚拟的类方法，虚方法只能针对具体的对象实例有效。因此，传统的C 语言认为，在具体的对象实例产生之前，却要根据即将产生的对象构造对象本身，这是不可能的。的确不可能，因为这会在逻辑上产生自相矛盾的悖论！然而，正是由于在DELPHI中有动态的类的类型信息，有真正虚拟的类方法，以及构造函数是基于类实现的等等这些关键概念，才可实现虚拟的构造函数。对象是由类产生的，对象就好象成长中的婴儿，而类就是它的母亲，婴儿自己的确不知道自己将来会成为什么样的人，可是母亲们却用各自的教育方法培养出不同的人，道理是相通的。 正是在TComponent类的定义中，构造函数Create被定义为虚拟的，才能使不同类型的控件实现各自的构造方法。这就是TClass创造的类之类概念的伟大，也是DELPHI的伟大。 WIN32的时空观 我的老父亲看着地上玩玩具的小孙子，然后对我说：“ 这孩子和小时的你一样，喜欢把东西拆开，看过究竟才罢手 ”。想想我小时侯，经常将玩具车、小闹钟、音乐盒，等等，拆得一塌糊涂，常常被母亲训斥。 我第一次理解计算机的基本原理，与我拆过的音乐盒有关。那是在念高中时的一本漫画书上，一位白胡子老头在讲解智能机的理论，一位留八字胡的叔叔在说计算机和音乐盒。他们说，计算机的中央处理器就是音乐盒中用来发音的那一排音乐簧片，计算机程序就是音乐盒中那个小圆筒上

密布的凸点，小圆筒的转动相当于中央处理器的指令指针的自然移动，而小圆筒上代表音乐的凸点控制音乐簧片振动发音相当于中央处理器执行程序的指令。音乐盒发出美妙的旋律，是按工匠早已刻在小圆筒上的音乐谱演奏的，计算机完成复杂的处理，是根据程序员预先编制好的程序实现的。上大学之后，我才知道那个白胡子老头就是科学巨匠图灵，他的有限自动机理论推动了整个信息革命的发展，而那个留八字胡的叔叔就是计算机之父冯.诺依曼，冯氏计算机体系结构至今仍然是计算机的主要体系机构。音乐盒没白拆，母亲可以宽心。 有深入浅出的理解，才能有高深而又简洁的创造。这一章我们将讨论Windows的32位操作系统中与我们编程有关的基本概念，建立WIN32中正确的时空观。希望阅读完本章之后，我们能更加深入地理解程序、进程和线程，理解执行文件、动态连接库和运行包的原理，看清全局数据、局部数据和参数在内存中的真相。