

面向对象的思想在建筑结构软件开发中的应用

摘要: 以十余种焊缝的计算为例, 介绍了多态在建筑结构软件开发中的应用。

关键词: 多态, 虚函数

The application of Object-Oriented idea in structural software's development

ABSTRACT: how to use Object-Oriented idea in structural software's development is introduced

KYE WORDS: Object-Oriented, polymorphism, virtual function

封装、继承和多态是面向对象(Object-Oriented)的计算机语言的三个最基本特征。封装和继承比较容易理解, 对于有过学习面向过程语言(比如 Fortran 和 c)经历的人来说, 理解多态这个“新”概念似乎需要些时间, 要淳熟地运用起来也似乎不太容易。其实在我们每天使用的语言中, 多态几乎无处不在: 四声是汉语的一个特色。对于“he”这两个简单的字母组合, 如果看成是汉字的拼音, 则加上不同的声调后, 就有不同的发音; 进一步而言, 对应于同是二声的“he”, 就有“河”、“何”、“和”、“合”等几个字。

上述的一音多字, 这就是所谓的多态。其他如一字多音, 一字多义以及汉语中没有显式的时态而需要通过上下文去揣摩等等现象, 也是多态的表现。汉语具有如此众多的多态的特征, 最直接的结果就是汉语的常用字要比英语要少得多, 要掌握具有如此丰富的多态特征的语言, 自然是需要头脑比较灵活(这也是中国人普遍比较聪明的原因之一), 但是一旦掌握, 就会深深体会其带来的便利, 受用无穷。

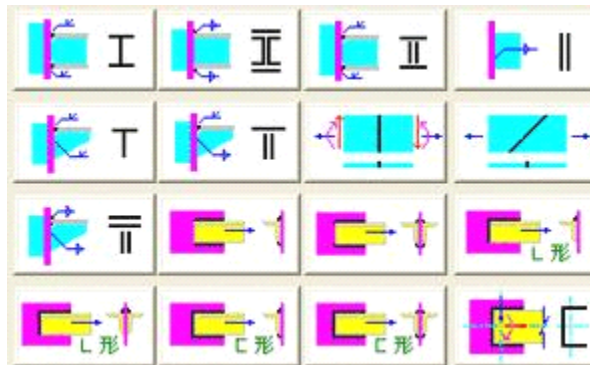
在 c++中, 多态主要包含静态多态和动态多态, 前者主要指基于模板的多态, 这是一种泛型设计(Generic programming)的思维; 后者主要指植根于继承和虚函数机制的技术。而虚函数又是如此重要, 以至于《Thinking in c++》的作者说 “If you don't use virtual functions, you don't understand OOP yet” (如果您不用虚函数, 那您还不懂面向对象)。

本文结合世纪旗云软件的焊缝计算模块, 介绍动态的多态在结构软件开发中的应用。

为什么使用虚函数

国内常用的焊缝形式至少有以下 16 种, 验算这不同形式的焊缝:

- 需要根据用户的选择, 在程序界面上显示不同的焊缝详图;
- 需要根据用户选择的焊缝形式, 在界面上显示相应的外力荷载参数;
- 需要不同的几何参数来描述;
- 每种焊缝的参数也应当赋予不同的初始值;
- 验算所用的公式也不同;
- 最终计算结果输出的内容的格式也不尽相同。



如果我们采用面向过程的编程思路, 一般会需要用类似如下的函数(此处采用伪代码):

<pre> InitParameters(int iWeldType) { Switch(iWeldType) case 1: //第一种焊缝 初始化表格行数、列数; 给各参数赋予各个默认值; Break; Case 2: //第二种焊缝 初始化表格行数、列数; 给各参数赋予各个默认值; Break; } </pre>	<pre> CheckWeld(int iWeldType) { Switch(iWeldType) Case 1: //第一种焊缝 验算工字型翼缘腹板均对接焊缝; Break; Case 2: //第二种焊缝 验算工字形翼缘腹板均为角焊缝; Break; Case 3: //第三种焊缝 验算工字形翼缘对接腹板角焊缝; Break; } </pre>
--	--


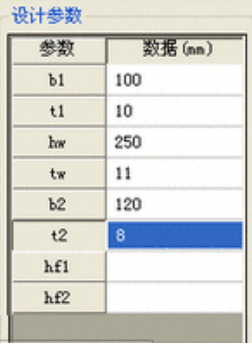

类似地，显示焊缝详图的函数 ShowWeldDetailDrawing、显示各种焊缝对应的外力组合 ShowForces、输出计算结果的函数 OutputCheckResult 也需要用类似的“switch-case”对儿。算起来，至少五个函数需要类似的“switch-case”对儿。对于 OutputCheckResult 这种函数而言，每种焊缝的详细计算书的代码至少需要 40 来行，十几种焊缝的计算书至少需要 600 多行代码，阅读这样长的代码实在不是一件让人感到愉快的事情。而且每种焊缝的计算书中，有很多代码是相似的，如果发现某种焊缝计算书中存在着某种共同错误（比如输出的计算结果的单位量纲给搞错了），那么，每种焊缝中的这种错误都要修改，其工作量显然过于庞大并且容易漏改。

现在还有一个问题：如果新增四种焊缝形式，那么我们需要做些什么？很显然，我们需要在上面的所有函数中(至少五个函数)再增加“case 17: ”、“case 18: ”、“case 19: ”、“case 20:”这四种情况。这非常象社会分工前的状况：那时候，每个人（相当于这里的每个函数）都得是全才：会打猎、会种地、会做饭、上知天文、下知地理。如果新增了一门科学，那么每个人都需要重新学习。无疑，这种状况下，每个人都是身心俱疲的。

看起来，面向过程的编程方式，虽然看起来思路简单，但是很不利于代码的维护和阅读，这是这种架构模式决定的。相比之下，面向对象的编程方式可以使得代码更容易阅读，当然也就更容易维护，其解决思路是：“用子类取代型别码”（Replace Type Code with Subclasses¹⁾。这非常类似社会分工后的状况：社会上存在着很多不同的行业，每个人只需要从事一个行业进行修炼即可——术业有专攻。当出现了一个新行业之后，把这个新行业的规章制度、从业人员等等配置好就可以了，其他行业从业人员丝毫不受影响。

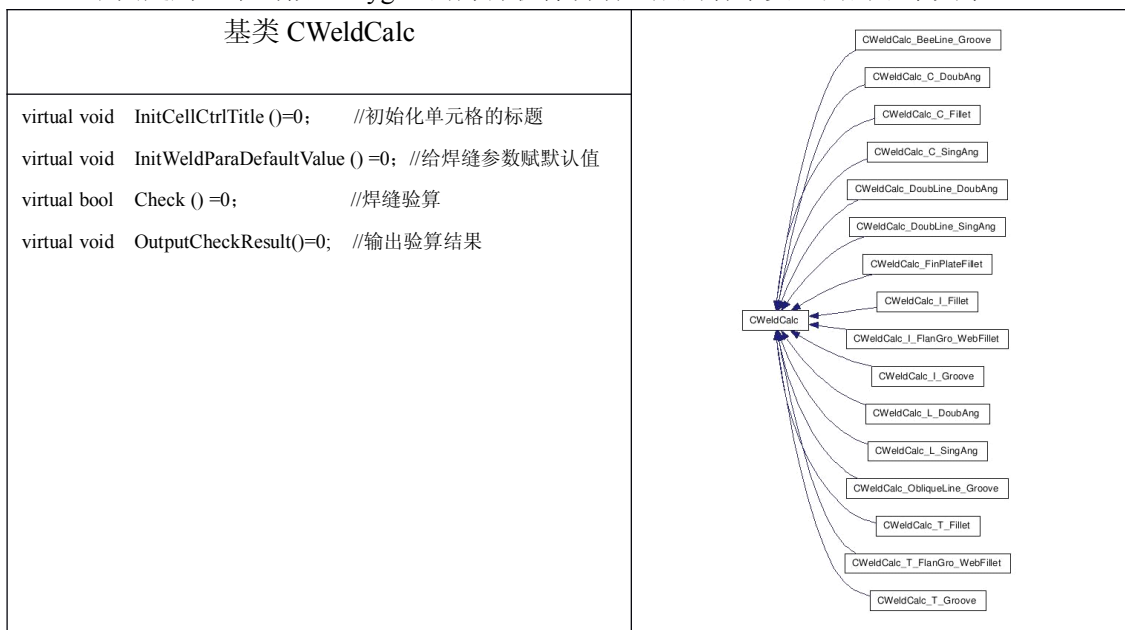
以焊缝的程序为例，我们设计一个基类“CWeldCalc”，其中的函数都是纯虚函数，函数体内没有一句代码；每种焊缝形式各对应一个从 CWeldCalc 派生出来的具体的焊缝类（比如 CWeldCalc_I_Fillet 以及 CWeldCalc_I_Groove 等），显示各种焊缝的详图、对各种焊缝进行验算、输出各种焊缝的计算书等具体工作都在各个具体的焊缝类中实现。

基类 CWeldCalc			
焊缝形式			
			...

所需参数				...
	CWeldCalc_I_Groove	CWeldCalc_I_Fillet	CWeldCalc_FinPlateFillet	...

以类取代型别码的最大优点是条件判断语句被派生类和基类间的继承关系和多态性代替，当我们为该类添加一个新的派生类时，其他派生类不会受到任何影响：即不需要在已有的焊缝类(派生类)代码中做任何修改——这样的手术干净利索。而且诸如 `OutputCheckResult()` 这种函数，在每个派生类里都被赋予了只跟该派生类相关的内容，自然代码也就短小了许多，便于阅读和修改。

下图是用一个叫做 `Doxygen` 的开源软件自动生成的各个类之间的继承关系：



与公司管理类似，公司高层（基类）制定规则和目标，公司的各部门的员工（各派生类）去执行这些规则，实现具体的目标，如果有新的工种出现，增加新的部门就是了，不会影响其他部门，这样的管理者活得非常轻松；而那种不善于划分部门职能、事必躬亲的领导，还是使用面向过程的思路，自然潇洒不起来。

结语

类似地，计算各种截面的参数（比如惯性矩、面积、重心位置等）以及大型的有限元程序也可以用这种思路。采用这种 `Object-Oriented` 的思路，可以大大优化程序的架构，为程序的扩展带来了更多的灵活性。这些优点，是古老的 `Fortran`、`C` 等面向过程的计算机语言所不具备的。

现今的软件越做越大，参与的人数越来越多，正如盖一个平房不需要太多考虑架构，用

砖或泥块垒起来就可以，但是要盖一个高楼大厦时，首要考虑的恐怕是结构的选型。如果在软件设计初期没有好的架构，将来的软件产品是相当难于维护的。融汇了面向对象的设计思路的架构，在近一段时间内，将是建筑结构软件架构师的一种备选方案。

参考文献

1. Fowler M. 侯捷等译. 重构: 改善既有代码的设计. 北京: 中国电力出版社, 2003
2. Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, Mass.: Addison-Wesley, 1995
3. Bruce Eckel, Thinking in c++(Second Edition) , Pearson Education, Inc. , 2000
4. 王咏武, 王咏刚, 道法自然——面向对象实践指南, 电子工业出版社, 2004
5. 位元文化 编著, Visual C++实用编程技术——从c++、面向对象到窗口程序设计, 华中理工大学出版社, 1999