使用C++11继承控制关键词来防止在类层次结构上的不一致

英文原文：Use C++11 Inheritance Control Keywords to Prevent Inconsistencies in Class Hierarchies

标签： C++11
oschina 推荐于 2年前 （共 9 段，翻译完成于 04-12）（4评）

参与翻译(1人)： jimmyjmh

30多年来，C++一直没有继承控制关键字。最起码这是不容易的。禁用一个类的进一步衍生是可能的但也很棘手。为避免用户在派生类中重载一个虚函数，你不得不向后考虑。但没有更多了：两个新的上下文敏感的关键词让你的工作轻松了许多。下面介绍它们如何工作的。

C++ 11添加了两个继承控制关键字：override和final。override确保在派生类中声明的重载函数跟基类的虚函数有相同的签名。final阻止类的进一步派生和虚函数的进一步重载。接下来让我们看看这些监督者如何消除你在类层次结构的设计和实施中的bug吧。

虚函数重载

一个派生类可以重载在基类中声明的成员函数，这是面向对象设计的基础。然而像重载一个函数这么简单的操作也会出错。关于重载虚函数的两个常见错误如下：

- 无意中重载
- 签名不匹配

首先，我们来分析一下无意中重载的综合症。你可能只是通过声明了一个与基类的某个虚成员函数具有相同的名字和签名的成员函数而无意中重载了这个虚函数。编译器和读代码的人很难发现这个bug因为他们通常以为这个新函数是为了实现对基类函数的重载：

```
1  struct A
2  {
3
```

```
4  virtual void func();
5  };
6  struct B: A{};
7  struct F{};
8  struct D: A, F
9  {
10   void func();//meant to declare a new function but
11
12  //accidentally overrides A::func};
```

阅读以上代码，你不能确定成员函数D::func()是否故意重载了A::func().它也可能是个偶然发生的重载，因为两个函数的参数列表和名字都碰巧一样。

jimmyjmh
于 2年前
顶 翻译的不错哦!

1 人顶

签名不匹配是一个更为常见的情景。这导致意外创建一个新的虚函数（而不是重载一个已存在的虚函数），正如以下例子所示：

```
1  struct G
2  {
3   virtual void func(int);
4  };
5  struct H: G
6  {
7   virtual void func(double); //accidentally creates a new virtual function
8  };
```

这种情况下，程序员本打算在类H中重载G::func()的。然而，由于H::func()拥有不同的签名，结果创建了一个新的虚函数，而非对基类函数的重载：

```
1  H *p=new H;
2  p->func(5); //calls G::f
3  p->func(5.0); // calls H::f
```

碰到这种情况，不是所有的编译器都会给个警告，有时那样做会被设置成抑制这种警告。在C++11中，通过使用新关键字override可以消除这两个bugs。override明确地表示一个函数是对基类中一个虚函数的重载。更重要的是，它会检查基类虚函数和派生类中重载函数的签名不匹配问题。如果签名不匹配，编译器会发出错误信息。

jimmyjmh
于 2年前
顶 翻译的不错哦!

1 人顶

我们来看看override如何消除签名不匹配bug的：

```
1  struct G
2  {
3   virtual void func(int);
4  };
5  struct H: G
6  {
7   virtual void func(double) override; //compilation error
8  };
```

当处理到H::func()声明时，编译器会 在一个基类查找与之匹配的虚函数。回想一下，“匹配”在上下文中的意思是：

- 相同的函数名
- 在声明该函数的第一个基类中的一个虚拟说明符
- 基类函数和派生类重载函数具有相同的参数列表、返回类型（一种情况例外）、cv资格等。

如果这三个条件中任意一个不满足，编译器就会报错。在我们的例子中，两个函数的参数列表不匹配；G::func()需要int型而H::func()要求double的。要是没有关键字override，编译器会简单的认为程序员想要在H中新建一个虚函数。

防止疏忽重载bug是很棘手的。既让这样，缺少关键字override应该引起你的怀疑。如果派生类函数确实是要重载基类的一个函数，它应该包含一个显式的override说明符。否则，假定D::func()是一个新的虚函数（这时，加一个注释是值得赞赏的），或者这也许是个bug。

jimmyjmh
于 2年前
顶 翻译的不错哦!

1 人顶

final函数和类

C++11的关键字final有两个用途。第一，它阻止了从类继承；第二，阻止一个虚函数的重载。我们先来看看final类吧。

某些实现系统服务、基础功能和加密等的类通常是不允许有子类的；实现者不想客户端从这些类派生新类而修改他们。标准库容器，如std:: vector和std:: list的无子类化类型就是另一个很好的例子。这些容器没有虚拟析构函数或者确切地说没有任何虚成员函数。

jimmyjmh
于 2年前
顶 翻译的不错哦!

1 人顶

然而，程序员常常在没有意识到风险的情况下坚持从std::vector派生。在C++11中，无子类类型将被声明为如下所示：

```
1  class TaskManager {/*..*/} final;
2  class PrioritizedTaskManager: public TaskManager {
3  };  //compilation error: base class TaskManager is final
```

同样，你可以通过声明它为final来禁止一个虚函数被进一步重载。如果一个派生类试图重载一个final函数，编译器就会报错：

```
1  struct A
2  {
3    virtual void func() const;
4  };
5  struct B: A
6  {
7    void func() const override final; //OK
8  };
```

jimmyjmh
于 2年前
顶 翻译的不错哦!

1 人顶

```
9  struct C: B
10 {
11  void func()const; //error, B::func is final
12 };
```

C::func()是否声明为override没关系，一旦一个虚函数被声明为final，派生类不能再重载它。

语法和术语

jimmyjmh 于 2年前

顶 翻译的不错哦!

1 人顶

迄今为止，我已经避免了两个有关override和final的次要问题。第一个是它们独特的位置。与virtual、inline、explicit extern以及一些类似的函数说明符不同的是，这两个关键字放在函数参数列表右括号之后，或者（对于无子类的类来说）一个类声明的右大括号之后。

这些关键字的特殊位置是由另一个不同寻常的性质决定的：override和final不是普通关键字。事实上官方地说，它们根本不是关键字。C++11把它们作为只是为了在特定上下文和位置下获取特殊意义的标示符。在任何其他位置或上下文，它们都被当成标示符。因此，一个完全有效的C++11代码如下：

```
1  //valid C++11 code
2  int final=0;
3  bool override=false;
4  if (override==true){
5   cout<<"override is: "<<override<<endl;}
6  struct D{} final;
7  struct A
8  {virtual bool func(); };
9  struct B:A
10 { bool func() override final; };
```

这似乎有点不可思议，final和override酷似PL/ 1的上下文敏感关键词（CSK）。自1972年以来，C和后来的C+ +一直都很抵触CSK坚持保留关键字的做法。

jimmyjmh 于 2年前

顶 翻译的不错哦!

1 人顶

那为什么委员会将final和override另外处理呢？选择CSK只是一种妥协方案。将override和final作为保留关键字可能对现有C++代码造成破坏。如果委员会已经引入了新的保留关键字，他们可能会选择像final_declor_Override这样时髦的，且不太可能与传统C++代码中用户声明的标示符相冲突的字符串等。然而，没有谁喜欢这么丑的关键字（比如，问问C使用者对C99的s_Bool的看法）。这是为什么最终采用CSK方法的原因。

override和final在C++11中作为关键字，但只在特定的上下文使用。不然它们只被当成普通标示符。虽然委员会不愿称override和final为“上下文敏感关键字”（事实上它们就是），作为替代，它们被官方地称为“具有特殊意义的标示符”。的确很特别。

总结

jimmyjmh 于 2年前

顶 翻译的不错哦!

1 人顶

这两个新的上下文敏感的关键字override和final对类层次结构的更严格的控制，让你摆脱一些继承相关的刺激性错误和设计失误。override保证重载虚拟函数匹配它的基类副本。final阻止进一步派生一个类或对一个虚函数进一步重载。关于编译器支持方面,GCC 4.7、英特尔的C++12、MSVC 11和Clang 2.9都支持这些新的关键词。

关于作者

Danny Kalev是一名专注于C + +的，通过认证的系统分析师和软件工程师。Kalev已经写了一些C++的教科书，并定期在各种软件开发网站发布C++知识。他是C++标准委员会的成员，并获得语言学硕士学位。

参考：

- Closer to Perfection: Get to Know C++11 Scoped and Based Enum Types
- Using constexpr to Improve Security, Performance and Encapsulation in C++
- C++11 Tutorial: Lambda Expressions — The Nuts and Bolts of Functional Programming
- The Biggest Changes in C++11 (and Why You Should Care)
- C11: A New C Standard Aiming at Safer Programming

网友评论 共4条

发表评论 回页面顶部

jimmyjmh 发表于 2013-04-12 16:10

也很好。推荐大家学习

C++的这两个新关键字不错，这篇文章

樱散零乱 发表于 2014-07-17 17:19

Eclipse的插件CDT不识别怎办

樱散零乱 发表于 2014-07-17 17:19

Eclipse的插件CDT不识别怎办

自由骑士笃志 发表于 2014-12-12 15:14

确实很不错的关键字~~解决了多人协作编码的一些问题，同时对钻石继承也有辅助作用。

发表评论

回评论顶部 | 回页面顶部