C++typename的由来和用法

良知犹存
+ 关注他

前言

在C++模板函数的使用过程中，我们经常可以看到一个typename的使用，例如这样的操作

```
template <typename T>
DataPred\ BuildData(const T &proto);
*
typename ClockType::time_point wall_time;
```


但是除此之外，我们也会经常看到这样的用法

```
template <class T>
auto GetTime(const T& t, int) -> decltype(t.time());
*
typedef std::vector<Eigen::Array2i> DiscreteScan2D;
```


那么这里就要问大家，这C++类似的用法下有什么区别呢，且听我细细道来。

作者：良知犹存

转载授权以及围观：欢迎添加微信公众号：羽林君

由来分析

> "typename"是一个C++程序设计语言中的关键字。当用于泛型编程时是另一术语"class"的同义词。这个关键字用于指出模板声明（或定义）中的非独立名称（dependent names）是类型名，而非变量名。

我们经常会这么用 typename，这是一项C++编程语言的泛型编程（或曰“模板编程”）的功能，typename关键字用于引入一个模板参数。

```
template <typename T>
const T& max(const T& x, const T& y)
{
  if (y < x) {
    return x;
  }
  return y;
}
```

在模板定义语法中关键字 class 与 typename 的作用完全一样

```
template<class T>
const T& max(const T& x, const T& y)
{
  if (y < x) {
    return x;
  }
  return y;
}
```

这里 class 关键字表明T是一个类型，后来为了避免 class 在这两个地方的使用可能给人带来混淆，所以引入了 typename 这个关键字，它的作用同 class 一样表明后面的符号为一个类型。

那class使用就够了，为什么又引入了新的关键词 typename ，关于这个问题，Stan Lippman 曾在其博客中表示，最早 Stroustrup 使用 class 来声明模板参数列表中的类型是为了避免增加不必要的关键字；后来委员会认为这样混用可能造成概念上的混淆才加上了 typename 关键字。

而使用 typename 的作用就是告诉 c++ 编译器，typename 后面的字符串为一个类型名称，而不是成员函数或者成员变量，这个时候如果前面没有 typename，编译器没有任何办法知道 T::LengthType 是一个类型还是一个成员名称(静态数据成员或者静态函数)，所以编译不能够通过。

问题浮现

那么问题来了，什么情况下，class定义之后，编译不能通过呢?

```
template<typename T>
void fun(const T& proto){


      T::const_iterator it(proto.begin());
}
```

发生编译错误是因为编译器不知道T::const_iterator是个类型。万一它是个变量呢?
T::const_iterator的解析有着逻辑上的矛盾： 直到确定了T是什么东西，编译器才会知道 T::const_iterator是不是一个类型； 然而当模板被解析时，T还是不确定的。这时我们声明它为一个类型才能通过编译：

而且在模板实例化之前，完全没有办法来区分它们，这绝对是滋生各种bug的温床。这时C++标准委员会再也忍不住了，与其到实例化时才貌似知道到底选择哪种方式来解释以上代码，委员会决定引入一个新的关键字，这就是typename。

千呼万唤始出来，我们来看看C++标准：

对于用于模板定义的依赖于模板参数的名称，只有在实例化的参数中存在这个类型名，或者这个名称前使用了 typename 关键字来修饰，编译器才会将该名称当成是类型。除了以上这两种情况，绝不会被当成是类型。

因此，如果你想直接告诉编译器 T::const_iterator 是类型而不是变量，只需用 typename修饰：

```
typename    T::const_iterator it(proto.begin());
```

这样编译器就可以确定T::const_iterator是一个类型，而不再需要等到实例化时期才能确定，因此消除了前面提到的歧义。

嵌套从属类型

事实上类型T::const_iterator依赖于模板参数T， 模板中依赖于模板参数的名称称为**从属名称**（dependent name），当一个从属名称嵌套在一个类里面时，称为**嵌套从属名称**（nested dependent name）。其实T::const_iterator还是一个**嵌套从属类型名称**（nested dependent type name）。

嵌套从属名称是需要用typename声明的，其他的名称是不可以用typename声明的。比如下面是一个合法的声明：

```
template<typename T>
void fun(const T& proto ,typename  T::const_iterator it);
```

使用

在定义类模板或者函数模板时，typename 和 class 关键字都可以用于指定模板参数中的类型。也就是说，以下两种用法是完全等价的。

```
template<typename T> /* ... */;
template<class T>    /* ... */;
```

既然typename关键字已经存在，而且它也可以用于最常见的指定模板参数，那么为什么不废除class这一用法呢？答案其实也很明显，因为在最终的标准出来之前，所有已存在的书、文章、教学、代码中都是使用的是class，可以想像，如果标准不再支持class，会出现什么情况。

使用关键字typename代替关键字class指定模板类型形参更为直观，毕竟，可以使用内置类型（非类类型）作为实际的类型形参，而且，typename更清楚地指明后面的名字是一个类型名。但是，关键字typename是作为标准C++的组成部分加入到C++中的，因此旧的程序更有可能只用关键字class。

这就是我分享的c++的typename，此外如果大家有什么更好的思路，也欢迎分享交流哈。

—END—

参考链接：

liam.page/2018/03/16/ke...

harttle.land/2015/09/09...

feihu.me/blog/2014/the-...

发布于 2020-12-10 19:42

模板（C++） C++ 编程语言

文章被以下专栏收录

技术人生
嵌入式Linux之路