C++ 中的模板和类型名关键字

C++ 中的 `template` 和 `typename` 关键字有两种用途。其中之一在程序员中是众所周知的：定义模板。另一种用法更加晦涩：指定表达式引用模板函数或类型。这会经常让使用 Eigen 库的程序员绊倒，通常会导致编译器发出难以理解的错误消息，例如“预期表达式”或“运算符 < 不匹配”。

使用 template 和 typename 关键字来定义模板

在 `template` 和 `typename` 关键字通常用于定义模板。这不是本页的主题，因为我们假设读者知道这一点（否则请参阅 C++ 书籍）。以下示例应说明 `template` 关键字的这种用法。

```
template <typename T>
bool isPositive(T x)
{
    return x > 0;
}
```

我们也可以这样写 `template <class T>`；关键字 `typename` 和 `class` 在这种情况下具有相同的含义。

显示模板关键字的第二次使用的示例

让我们 `template` 用一个例子来说明关键字的第二种用法。假设我们要编写一个函数，将矩阵上三角部分的所有条目复制到另一个矩阵中，同时保持下三角部分不变。一个简单的实现如下：

例子：

```
#include <Eigen/Dense>
#include <iostream>

using namespace Eigen;

void copyUpperTriangularPart(MatrixXf& dst, const MatrixXf& src)
{
    dst.triangularView<Upper>() = src.triangularView<Upper>();
}

int main()
{
    MatrixXf m1 = MatrixXf::Ones(4,4);
    MatrixXf m2 = MatrixXf::Random(4,4);
    std::cout << "m2 before copy:" << std::endl;
    std::cout << m2 << std::endl << std::endl;
    copyUpperTriangularPart(m2, m1);
    std::cout << "m2 after copy:" << std::endl;
    std::cout << m2 << std::endl << std::endl;
}
```

输出：

```
m2 before copy:
   0.68   0.823  -0.444   -0.27
 -0.211  -0.605   0.108  0.0268
  0.566   -0.33 -0.0452   0.904
  0.597   0.536   0.258   0.832

m2 after copy:
     1      1      1      1
-0.211      1      1      1
 0.566  -0.33      1      1
 0.597  0.536  0.258      1
```

这工作正常，但它不是很灵活。首先，它只适用于单精度浮点数的动态大小矩阵；该函数`copyUpperTriangularPart()`不接受静态大小的矩阵或具有双精度数字的矩阵。其次，如果您使用诸如`mat.topLeftCorner(3,3)`参数之类的表达式`src`，则将其复制到 MatrixXf 类型的临时变量中；这个副本是可以避免的。

正如编写以特征类型作为参数的函数中所述，这两个问题都可以通过`copyUpperTriangularPart()`接受任何MatrixBase类型的对象来解决。这导致以下代码：

例子：

```
#include <Eigen/Dense>
#include <iostream>

using namespace Eigen;

template <typename Derived1, typename Derived2>
void copyUpperTriangularPart(MatrixBase<Derived1>& dst, const
MatrixBase<Derived2>& src)
{
    /* Note the 'template' keywords in the following line! */
    dst.template triangularView<Upper>() = src.template
triangularView<Upper>();
}

int main()
{
    MatrixXi m1 = MatrixXi::Ones(5,5);
    MatrixXi m2 = MatrixXi::Random(4,4);
    std::cout << "m2 before copy:" << std::endl;
    std::cout << m2 << std::endl << std::endl;
    copyUpperTriangularPart(m2, m1.topLeftCorner(4,4));
    std::cout << "m2 after copy:" << std::endl;
    std::cout << m2 << std::endl << std::endl;
}
```

输出：

```
m2 before copy:
 7  9 -5 -3
-2 -6  1  0
 6 -3  0  9
 6  6  3  9

m2 after copy:
 1  1  1  1
-2  1  1  1
 6 -3  1  1
 6  6  3  1
```

函数体中的一行`copyUpperTriangularPart()`显示了`template` C++ 中关键字的第二种更模糊的用法。尽管看起来很奇怪，但`template`根据标准，关键字是必需的。没有它，编译器可能会拒绝代码并显示错误消息，例如“没有匹配运算符<”。

解释

`template`在最后一个例子中需要关键字的原因与模板应该如何在 C++ 中编译的规则有关。编译器具有其中模板定义的点以检查正确的语法的代码，在不知道的模板参数的实际值（`Derived1`和`Derived2`在本例中）。这意味着编译器无法知道这`dst.triangularView`是一个成员模板，并且后面的 < 符号是模板参数的分隔符的一部分。另一种可能性是它`dst.triangularView`是一个带有 < 符号的成员变量，它指的是`operator<()`函数。事实上，根据标准，编译器应该选择第二种可能性。如果`dst.triangularView`是一个成员模板（就像我们的例子一样），程序员应该用`template`关键字和write明确指定它`dst.template triangularView`。

确切的规则相当复杂，但忽略一些微妙之处，我们可以将它们总结如下：

- 甲*从属名称*是名称依赖（直接或间接）上的模板的参数。在示例中，`dst`是一个依赖名称，因为它的类型`MatrixBase<Derived1>`取决于模板参数`Derived1`。
- 如果代码包含其中一个结构`xxx.yyy`or`xxx->yyy`并且`xxx`是一个从属名称并`yyy`引用成员模板，则`template`必须在 之前使用关键字`yyy`，导致`xxx.template yyy`或`xxx->template yyy`。
- 如果代码包含构造`xxx::yyy`并且`xxx`是从属名称并`yyy`引用成员 typedef，则`typename`必须在整个构造之前使用关键字，从而导致`typename xxx::yyy`.

作为`typename`需要关键字的示例，请考虑稀疏矩阵操作中的以下代码，用于迭代稀疏矩阵类型的非零条目：

```
SparseMatrixType mat(rows,cols);
for (int k=0; k<mat.outerSize(); ++k)
    for (SparseMatrixType::InnerIterator it(mat,k); it; ++it)
    {
        /* ... */
    }
```

如果`SparseMatrixType`依赖于模板参数，则`typename`需要关键字：

```cpp
template <typename T>
void iterateOverSparseMatrix(const SparseMatrix<T>& mat;
{
    for (int k=0; k<m1.outerSize(); ++k)
        for (typename SparseMatrix<T>::InnerIterator it(mat,k); it; ++it)
        {
        /* ... */
        }
}
```

进一步阅读的资源

有关此主题的更多信息和更全面的解释，读者可以参考以下来源：

- David Abrahams 和 Aleksey Gurtovoy 所著的“C++ 模板元编程”一书在附录 B（“类型名称和模板关键字”）中包含了非常好的解释，它构成了本页的基础。
- http://pages.cs.wisc.edu/~driscoll/typename.html
- http://www.parashift.com/c++-faq-lite/templates.html#faq-35.18
- http://www.comeaucomputing.com/techtalk/templates/#templateprefix
- http://www.comeaucomputing.com/techtalk/templates/#typename