

30分钟了解C++11新特性

什么是C++11

C++11是曾经被叫做C++0x，是对目前C++语言的扩展和修正，C++11不仅包含核心语言的新机能，而且扩展了C++的标准程序库（STL），并入了大部分的C++ Technical Report 1（TR1）程序库(数学的特殊函数除外)。

C++11包括大量的新特性：包括lambda表达式，类型推导关键字auto、decltype，和模板的大量改进。

本文将对C++11的以上新特性进行简单的讲解，以便大家能够快速了解到C++11对C++的易用性方面祈祷的巨大作用。

新的关键字

auto

C++11中引入auto第一种作用是为了自动类型推导

auto的自动类型推导，用于从初始化表达式中推断出变量的数据类型。通过auto的自动类型推导，可以大大简化我们的编程工作

auto实际上实在编译时对变量进行了类型推导，所以不会对程序的运行效率造成不良影响

另外，似乎auto并不会影响编译速度，因为编译时本来也要右侧推导然后判断与左侧是否匹配。

```
auto a; // 错误，auto是通过初始化表达式进行类型推导，如果没有初始化表达式，就无法确定a的类型
auto i = 1;
auto d = 1.0;
auto str = "Hello World";
auto ch = 'A';
auto func = less<int>();
vector<int> iv;
auto ite = iv.begin();
auto p = new foo() // 对自定义类型进行类型推导
```

auto不光有以上的应用，它在模板中也是大显身手，比如下例这个加工产品的例子中，如果不使用auto就必须声明Product这一模板参数：

```
template <typename Product, typename Creator>
void processProduct(const Creator& creator) {
    Product* val = creator.makeObject();
    // do something with val
}
```

如果使用auto，则可以这样写：

```
template <typename Creator>
void processProduct(const Creator& creator) {
    auto val = creator.makeObject();
    // do something with val
}
```

抛弃了麻烦的模板参数，整个代码变得更加正解了。
返回x的类型，若是表达式，则返回表达式的返回值的类型

decltype

decltype实际上有点像auto的反函数，auto可以让你声明一个变量，而decltype则可以从一个变量或表达式中得到类型，有实例如下：

int x = 10;
decltype((x)) y=20; (错误)

```
int x = 3;
decltype(x) y = x;
```

decltype(x) b; // b类型为int
decltype((x)) c; // c类型为int&,必须初始化

const int x = 10;
decltype(x) y=20 (正确)

有人会问，decltype的实用之处在哪里呢，我们接着上边的例子继续说下去，如果上文中的加工产品的例子中我们想把产品作为返回值该怎么办呢？我们可以这样写：

```
template <typename Creator>
auto processProduct(const Creator& creator) ->
decltype(creator.makeObject()) {
    auto val = creator.makeObject();
    // do something with val
}
```

nullptr

nullptr是为了解决原来C++中NULL的二义性问题而引进的一种新的类型，因为NULL实际上代表的是0，

```

void F(int a){
    cout<<a<<endl;
}

void F(int *p){
    assert(p != NULL);

    cout<< p <<endl;
}

int main(){

    int *p = nullptr;
    int *q = NULL;
    bool equal = ( p == q ); // equal的值为true, 说明p和q都是空指针
    int a = nullptr; // 编译失败, nullptr不能转型为int
    F(0); // 在C++98中编译失败, 有二义性; 在C++11中调用F (int)
    F(nullptr);

    return 0;
}

```

序列for循环

在C++中for循环可以使用类似java的简化的for循环，可以用于遍历数组，容器，string以及由begin和end函数定义的序列（即有Iterator），示例代码如下：

```

map<string, int> m{{"a", 1}, {"b", 2}, {"c", 3}};
for (auto p : m){
    cout<<p.first<<" : "<<p.second<<endl;
}

```

此处元素为int和string类型，最好使用auto，或者auto&

Lambda表达式

lambda表达式类似Javascript中的闭包，它可以用于创建并定义匿名的函数对象，以简化编程工作。Lambda的语法如下：

[函数对象参数]（操作符重载函数参数）->返回值类型{函数体}

```
vector<int> iv{5, 4, 3, 2, 1};
int a = 2, b = 1;

for_each(iv.begin(), iv.end(), [b](int &x){cout<<(x + b)<<endl;}); // (1)

for_each(iv.begin(), iv.end(), [=](int &x){x *= (a + b);}); // (2)

for_each(iv.begin(), iv.end(), [=](int &x)->int{return x * (a + b);}); // (3)
```

- []内的参数指的是Lambda表达式可以取得的全局变量。(1)函数中的b就是指函数可以得到在Lambda表达式外的全局变量，如果在[]中传入=的话，即是可以取得所有的外部变量，如（2）和（3）Lambda表达式
- ()内的参数是每次调用函数时传入的参数。
- ->后加上的是Lambda表达式返回值的类型，如（3）中返回了一个int类型的变量

变长参数的模板

我们在C++中都用过pair，pair可以使用make_pair构造，构造一个包含两种不同类型的数据的容器。比如，如下代码：

```
auto p = make_pair(1, "C++ 11");
```

map等

由于在C++11中引入了变长参数模板，所以发明了新的数据类型：tuple，tuple是一个N元组，可以传入1个，2个甚至多个不同类型的数据

```
auto t1 = make_tuple(1, 2.0, "C++ 11");
auto t2 = make_tuple(1, 2.0, "C++ 11", {1, 0, 2});
```

这样就避免了从前的pair中嵌套pair的丑陋做法，使得代码更加整洁

另一个经常见到的例子是Print函数，在C语言中printf可以传入多个参数，在C++11中，我们可以用变长参数模板实现更简洁的Print

```
template<typename head, typename... tail>
void Print(Head head, typename... tail) {
    cout<< head <<endl;
    Print(tail...);
}
```

Print中可以传入多个不同种类的参数，如下：

```
Print(1, 1.0, "C++11");
```

更加优雅的初始化方法

在引入C++11之前，只有数组能使用初始化列表，其他容器想要使用初始化列表，只能用以下方法：

```
int arr[3] = {1, 2, 3};  
vector<int> v(arr, arr + 3);
```

在C++11中，我们可以使用以下语法来进行替换：**容器初始化，使用{}**

```
int arr[3]{1, 2, 3};  
vector<int> iv{1, 2, 3};  
map<int, string>{{1, "a"}, {2, "b"}};  
string str{"Hello World"};
```

然后呢...

如果你想了解更多C++11令人兴奋的新特性，我会向你推荐这两个博客：

[胡健的C++11系列博文](#)

[ToWriting的C++11系列博文](#)

[C++11的编译器支持列表](#)