

56 The 'auto' keyword in C++

有一种方法可以让C++自动推导出数据的类型，不管是在创建、初始化变量数据时，还是在将一个变量对另一个变量进行赋值时。

1. auto 关键字

可以自动推出b也是 `int` 类型。

```
int a = 5;
auto b:int = a;
```

甚至a的类型也可以用 `auto`：

```
auto a = 5.0f;
auto b (local variable) float a
```

那问题来了，有了 `auto` 我还用写类型吗？我可以到处都用它吗？

这也得归结到编程风格问题上。同时你也需要去思考这种编程风格的优点和缺点。

2. 使用例子

```
std::string GetName()
{
    return "Cherno";
}

int main()
{
    std::string name = GetName();

    char a = 'a'; // 如果只是简单赋值，写上类型会更清楚

    std::cin.get();
}
```

这里我可以把name的类型设为 `auto`：

```
auto name = GetName();
```

这样如果api发生改变时，比如GetName的返回类型改为了 `char*`，客户端不需要任何改动。但是坏处是我也不知道api已经改变了，它可能会破坏依赖于特定类型的代码。

3. 什么时候适合用auto?

C++

```
int main()
{
    std::vector<std::string> strings;
    strings.push_back("Apple");
    strings.push_back("Orange");

    for (std::vector<std::string>::iterator it = strings.begin();
         it != strings.end(); it++)
    {
        std::cout << *it << std::endl;
    }

    std::cin.get();
}
```

代码中`iterator` (迭代器) 的类型太长了, 可以直接用 `auto` 替换以获得更好的可读性:

```
for (auto it = strings.begin();
     it != s (local variable) std::vector<std::string>::iterator it
{
    std::cout << *it << std::endl;
}
```

另外一个例子:

C++

```
class Device{};

class DeviceManager
{
private:
    std::unordered_map<std::string, std::vector<Device*>> m_Devices;
public:
    const std::unordered_map<std::string, std::vector<Device*>>& GetDevices() const
    {
        return m_Devices;
    }
};

int main()
{
    DeviceManager dm;
    const std::unordered_map<std::string, std::vector<Device*>>& devices =
dm.GetDevices();
}
```

这里的类型相当大, 这里可以做的是用`alias` (取别名) :

C++

```
using DeviceMap = std::unordered_map<std::string, std::vector<Device*>>;
    DeviceManager dm;

// 或者

typedef std::unordered_map<std::string, std::vector<Device*>> DeviceMap;
```

然后可以直接调用了，甚至可以把 `using` 部分挪到 `DeviceManager` 类中：

C++

```
const DeviceMap& devices = dm.GetDevices();
```

这样我就得到了一个简洁且更有意义的类型。

但如果我不想用这些方法的话，就可以用 `auto` 帮助自己了；

```
const auto& devices:const unordered_map<string, vector<Device*>>& = dm.GetDevices();
    std::unordered_map<std::string, std::vector<Device*>>
std::cin.get();
```

不过要注意的是，`auto` 不处理引用，所以不要漏掉 `&` 而造成一次复制。

这就是真实世界中两个可能比较适合使用 `auto` 的场合，比如你的变量类型非常长的话，如果变量类型只是个 `int` 你还要用 `auto`，那只会降低代码的可读性，在我看来没什么好处。

当进入到更加复杂的代码集，包含模板等，那种情况相当复杂，你不得不使用 `auto`，因为你不知道类型是什么。