# Item 2:Understand auto type deduction

条款二:理解auto类型推导

如果你已经读过Item1的模板类型推导，那么你几乎已经知道了auto类型推导的大部分内容，至于为什么不是全部是因为这里有一个auto不同于模板类型推导的例外。但这怎么可能，模板类型推导包括模板，函数，形参，但是auto不处理这些东西啊。

你是对的，但没关系。auto类型推导和模板类型推导有一个直接的映射关系。它们之间可以通过一个非常规范非常系统化的转换流程来转换彼此。

在Item1中，模板类型推导使用下面这个函数模板来解释：

```
template<typename T>
void f(ParmaType param);     //使用一些表达式调用f
```

在f的调用中，编译器使用expr推导T和ParamType。当一个变量使用auto进行声明时，auto扮演了模板的角色，变量的类型说明符扮演了ParamType的角色。废话少说，这里便是更直观的代码描述，考虑这个例子：

```
auto x = 27;
```

这里x的类型说明符是auto，另一方面，在这个声明中：

```
const auto cx = x;
```

类型说明符是**const auto**。另一个：

```
const auto & rx=cx;
```

类型说明符是**const auto&**。在这里例子中要推导x rx cx的类型，编译器的行为看起来就像是认为这里每个声明都有一个模板，然后使用合适的初始化表达式进行处理：

```
template<typename T>     //理想化的模板用来推导x的类型
void func_for_x(T param);

func_for_x(27);

template<typename T>     //理想化的模板用来推导cx 的类型
void func_for_cx(const T param);

func_for_cx(x);

template<typename T>     //理想化的模板用来推导rx的类型
void func_for_rx(const T & param);

func_for_rx(x);
```

正如我说的，auto类型推导除了一个例外（我们很快就会讨论），其他情况都和模板类型推导一样。

Item1把模板类型推导分成三个部分来讨论ParamType在不同情况下的类型。在使用**auto**作为类型说明符的变量声明中，类型说明符代替了ParamType，因此Item1描述的三个情景稍作修改就能适用于auto：

- 类型说明符是一个指针或引用但不是通用引用
- 类型说明符一个通用引用
- 类型说明符既不是指针也不是引用

我们早已看过情景一和情景三的例子：

```
auto x = 27;            //情景三
const auto cx = x;      //情景三
const auto & rx=cx;     //情景一
```

Item1讨论并总结了数组和函数如何退化为指针，那些内容也同样适用于auto类型推导

```
const char name[] =     //name的类型是const char[13]
 "R. N. Briggs";

auto arr1 = name;       //arr1的类型是const char*
auto& arr2 = name;      //arr2的类型是const char(&)[13]

void someFunc(int,double);

auto func1=someFunc;    //func1的类型是void(int,double)
auto& func2 = someFunc; //func2的类型是void(&)(int,double)
```

就像你看到的那样auto类型推断和模板类型推导一样几乎一样的工作，它们就像一个硬币的两面。

讨论完相同点接下来就是不同点，前面我们已经说到auto类型推导和模板类型推导有一个例外使得它们的工作方式不同，接下来我们要讨论的就是那个例外。
我们从一个简单的例子开始，如果你想用一个int值27来声明一个变量，C++98提供两种选择：

```
int x1=27;
int x2(27);
```

C++11由于也添加了用于支持统一初始化（**uniform initialization**）的语法：

```
int x3={27};
int x47{27};
```

总之，这四种不同的语法只会产生一个相同的结果：变量类型为int值为27

但是Item5解释了使用auto说明符代替指定类型说明符的好处，所以我们应该很乐意把上面声明中的**int**替换为**auto**，我们会得到这样的代码：

```
auto x1=27;
auto x2(27);
auto x3={27};
auto x4{27};
```

这些声明都能通过编译，但是他们不像替换之前那样有相同的意义。前面两个语句确实声明了一个类型为int值为27的变量，但是后面两个声明了一个存储一个元素27的 **std::initializer_list<int>** 类型的变量。

```
auto x1=27;          //类型是int，值是27
auto x2(27);         //同上
auto x3={27};        //类型是std::initializer_list<int>,值是{27}
auto x4{27};         //同上
```

这就造成了auto类型推导不同于模板类型推导的特殊情况。当用auto声明的变量使用花括号进行初始化，auto类型推导会推导出auto的类型为 **std::initializer_list**。如果这样的一个类型不能被成功推导（比如花括号里面包含的是不同类型的变量），编译器会拒绝这样的代码！

```
auto x5={1,2,3.0};       //错误！auto类型推导不能工作
```

就像注释说的那样，在这种情况下类型推导将会失败，但是对我们来说认识到这里确实发生了两种类型推导是很重要的。一种是由于auto的使用：x5的类型不得不被推导，因为x5使用花括号的方式进行初始化，x5必须被推导为 **std::initializer_list**,但是 **std::initializer_list**是一个模板。**std::initializer_list**会被实例化，所以这里T也会被推导。 另一种推导也就是模板类型推导被归入第二种推导。在这个例子中推导之所以出错是因为在花括号中的值并不是同一种类型。

对于花括号的处理是auto类型推导和模板类型推导唯一不同的地方。当使用auto的变量使用花括号的语法进行初始化的时候，会推导出**std::initializer_list**的实例化，但是对于模板类型推导这样就行不通：

```
auto x={11,23,9};   //x的类型是std::initializer_list<int>

template<typename T>
void  f(T param);

f({11,23,9});       //错误！不能推导出T
```

然而如果指定T是**std::initializer**而留下未知T,模板类型推导就能正常工作：

```
template<typename T>
void f(std::initializer_list<T> initList);

f({11,23,9});       //T被推导为int，initList的类型被推导为std::initializer_list<int>
```

因此auto类型推导和模板类型推导的真正区别在于auto类型推导假定花括号表示**std::initializer_list**而模板类型推导不会这样（确切的说是不知道怎么办）。

你可能想知道为什么auto类型推导对于花括号和模板类型推导有不同的处理方式。我也想知道。哎，我至今没找到一个令人信服的解释。但是规则就是规则，这意味着你必须记住如果你使用auto声明一个变量，并用花括号进行初始化，auto类型推导总会得出std::initializer_list的结果。如果你使用**uniform initialization（花括号的方式进行初始化）**用得很爽你就得记住这个例外以免犯错，在C++11编程中一个典型的错误就是偶然使用了**std::initializer_list**类型的变量,这个陷阱也导致了很多C++程序员抛弃花括号初始化，只有不得不使用的时候再做考虑。

对于C++11故事已经说完了。但是对于C++14故事还在继续，C++14允许auto用于函数返回值并会被推导（参见Item3），而且C++14的lambda函数也允许在形参中使用auto。但是在这些情况下虽然表面上使用的是auto但是实际上是模板类型推导的那一套规则在工作，所以说下面这样的代码不会通过编译：

```
auto createInitList()
{
    return {1,2,3};     //错误！推导失败
}
```

同样在C++14的lambda函数中这样使用auto也不能通过编译：

```
std::vector<int> v;

auto resetV = [&v](const auto & newValue){v=newValue;}; //C++14
...
reset({1,2,3});          //错误！推导失败
```

记住：

- auto类型推导通常和模板类型推导相同，但是auto类型推导假定花括号初始化代表**std::initializer_list**而模板类型推导不这样做
- 在C++14中auto允许出现在函数返回值或者lambda函数形参中，但是它的工作机制是模板类型推导那一套方案。