C++11列表初始化（统一了初始化方式）

我们知道，在 C++98/03 中的对象初始化方法有很多种，请看下面的代码：

```
//初始化列表
int i_arr[3] = { 1, 2, 3 };  //普通数组
struct A
{
    int x;
    struct B
    {
        int i;
        int j;
    } b;
} a = { 1, { 2, 3 } };  //POD类型

//拷贝初始化（copy-initialization）
int i = 0;
class Foo
{
    public:
    Foo(int) {}
} foo = 123;  //需要拷贝构造函数

//直接初始化（direct-initialization）
int j(0);
Foo bar(123);
```

这些不同的初始化方法，都有各自的适用范围和作用。最关键的是，这些种类繁多的初始化方法，没有一种可以通用所有情况。

为了统一初始化方式，并且让初始化行为具有确定的效果，C++11 中提出了列表初始化（List-initialization）的概念。

> POD 类型即 plain old data 类型，简单来说，是可以直接使用 memcpy 复制的对象。

统一的初始化

在上面我们已经看到了，对于普通数组和 POD 类型，C++98/03 可以使用初始化列表（initializer list）进行初始化：

```
int i_arr[3] = { 1, 2, 3 };
```

```
long l_arr[] = { 1, 3, 2, 4 };
struct A
{
    int x;
    int y;
} a = { 1, 2 };
```

但是这种初始化方式的适用性非常狭窄，只有上面提到的这两种数据类型可以使用初始化列表。

在 C++11 中，初始化列表的适用性被大大增加了。它现在可以用于任何类型对象的初始化，请看下面的代码。

【实例】通过初始化列表初始化对象。

```
class Foo
{
public:
    Foo(int) {}
private:
    Foo(const Foo &);
};

int main(void)
{
    Foo a1(123);
    Foo a2 = 123;  //error: 'Foo::Foo(const Foo &)' is private
    Foo a3 = { 123 };
    Foo a4 { 123 };
    int a5 = { 3 };
    int a6 { 3 };
    return 0;
}
```

在上例中，a3、a4 使用了新的初始化方式来初始化对象，效果如同 a1 的直接初始化。

a5、a6 则是基本数据类型的列表初始化方式。可以看到，它们的形式都是统一的。

这里需要注意的是，a3 虽然使用了等于号，但它仍然是列表初始化，因此，私有的拷贝构造并不会影响到它。

a4 和 a6 的写法，是 C++98/03 所不具备的。在 C++11 中，可以直接在变量名后面跟上初始化列表，来进行对象的初始化。

这种变量名后面跟上初始化列表方法同样适用于普通数组和 POD 类型的初始化：

```
int i_arr[3] { 1, 2, 3 };  //普通数组
struct A
{
    int x;
    struct B
    {
        int i;
        int j;
    } b;
} a { 1, { 2, 3 } };  //POD类型
```

在初始化时，`{}` 前面的等于号是否书写对初始化行为没有影响。

另外，如同读者所想的那样，new 操作符等可以用圆括号进行初始化的地方，也可以使用初始化列表：

```
int* a = new int { 123 };
double b = double { 12.12 };
int* arr = new int[3] { 1, 2, 3 };
```

指针 a 指向了一个 new 操作符返回的内存，通过初始化列表方式在内存初始化时指定了值为 123。

b 则是对匿名对象使用列表初始化后，再进行拷贝初始化。

这里让人眼前一亮的是 arr 的初始化方式。堆上动态分配的数组终于也可以使用初始化列表进行初始化了。

除了上面所述的内容之外，列表初始化还可以直接使用在函数的返回值上：

```
struct Foo
{
    Foo(int, double) {}
};
Foo func(void)
{
    return { 123, 321.0 };
}
```

这里的 return 语句就如同返回了一个 Foo(123, 321.0)。

由上面的这些例子可以看到，在 C++11 中使用初始化列表是非常便利的。它不仅统一了各种对象的初始化方式，而且还使代码的书写更加简单清晰。