2. C++及标准库简介

2. C++及标准库简介

讨论C++的历史及其不同版本，并介绍 Big-O 标记法，用来具体显示标准库各项操作的效能（performance）和可扩展性（scalability）。

2.1 C++ Standard的历史

C++标准化始于1989年，由国际标准化组织（the International Organization for Standardiza-tion，ISO）推动。

1. C++98 ，于1998年批准，是第一份C++标准规格。
2. C++03 ，这是个所谓“技术勘误”（technical corrigendum，TC），内含不甚严重的C++98 bug修正。论C++98或C++03都被视为“第一份C++标准规格”。
3. TR1 ，内含大幅度的标准库扩充。
4. C++11 ，批准于2011年，是第二份C++标准。C++11内含语言和 标准库两方面皆十分重大的改善和强化，而TR1所做的扩充亦被纳入成 为namespace std的一部分。

2.1.1 C++11 Standard 常见疑问

这份标准有着各式各样的源头。事实上，任何公司或国家，甚至个人，都有可能提出新特性和新扩充，那些提案必须被整个标准化组织接受。理论上没有任何东西的设计是从头做起的。因此整个成果的同质性（homogeneous）并不是很高。你会在不同的组件中发现不同的设计原理。例如string class和STL之间就有以下差异（STL是一个针对数据结构和算法而完成的框架）：

- String class被设计为一个安全而便利的组件。因此它们提供 一份几乎足以自我说明的接口，并检查许多可能发生的差错。
- STL被设计为“将不同的数据结构与算法结合起来，产出最佳效能”。因此STL并不非常便利，也不对许多逻辑错误进行检查。为了获得威力强大的STL框架所带来的好处和效能，你必须知道其所使用的概念，并且谨慎地套用它们。

上述两个组件都是标准库的一部分。它们之间存在一些和谐，但依然遵循各自的基础设计哲学。

2.1.2 C++98 和 C++11 的兼容性

C++11的设计目标之一是，对C++98保持向后兼容。

如果你的代码掺杂着C++各版本的身影，但希望从C++11带来的改善中获利，那么，可以试着使用预定义好的宏__cplusplus。

对C++11而言，当编译一个C++转译单元（translation unit，也就是C ++源码文件）时以下定义是有效的：

```
1  #define __cplusplus 201103L
```

与此相对，在C++98和C++03中则是

```
1 #define __cplusplus 199711
```

请注意，有时候，编译器厂商提供的值与上述不同。

注意，向后兼容仅适用于源码，不保证二进制兼容（binary compatibility），因为那会导致问题，特别是当一个原有操作（existing operation）取得一个新式返回类型（new return type），因为“基于返回类型而形成的重载（overloading）”是不被允许的（这适用于某些STL算法及STL容器的某些成员函数）。所以请以C++11编译器重新编译C++98程序的每一份源码，包括用到的所有程序库——通常都会成功。若以C++11编译器产生的二进制码链接（linking）C++98编译器产生的二进制码，则有可能失败。

2.2 复杂度与 Big-O 标记

C++ 标准库某些部分，特别是 STL，需要认真考虑算法和成员函数的效能，因此需要用到复杂度（complexity）概念。计算机科学记使用特殊符号来表现算法的相对复杂度，如此可以很快依据算法的运行时间加以分类，进行算法之间的定性比较。这种衡量称为 Big-O 标记（notation）。

Big-O标记系将一个算法的运行时间以输入量 n 的函数表示。例如，当运行时间随元素个数呈线性增长时（亦即如果元素个数呈倍数增长，运行时间亦呈倍数增长），复杂度为O(n)；如果运行时间与输入量无关，复杂度为 O(1)。

表2.1列出了典型的复杂度和其Big-O 标记。

表2.1 五种典型的复杂度

种类	标记	意义
常量	O(1)	运行时间与元素个数无关
对数	O(log(n))	运行时间随元素个数的增加呈对数增长

线性	O(n)	运行时间随元素个数的增加呈线性增长
n-log-n	O(nlog(n))	运行时间随元素个数的增加呈 “线性与对数的乘积” 增长
二次方	O(n2)	运行时间随元素个数的增加呈平方增长

C++标准手册中提到的某些复杂度被称为 amortized （分期摊还； 折旧成本），意思是长期而言大量操作将如上述描述般进行，但单一操作可能花费比平均值更长的时间。举例，如果你为一个dynamic array追加元素，运行时间将取决于array是否尚有备用内存。如果内存足够，就属于常量复杂度，因为在尾端加入一个新元素总是花费相同时间。但如果备用内存不足，那么就是线性复杂度，因为你必须分配足够的内存并搬动（复制）它们，实际耗用时间取决于当时的元素个数。内存重新分配动作并不常发生（译注：STL的dynamic array容 器会以某种设计哲学来保持备用内存）