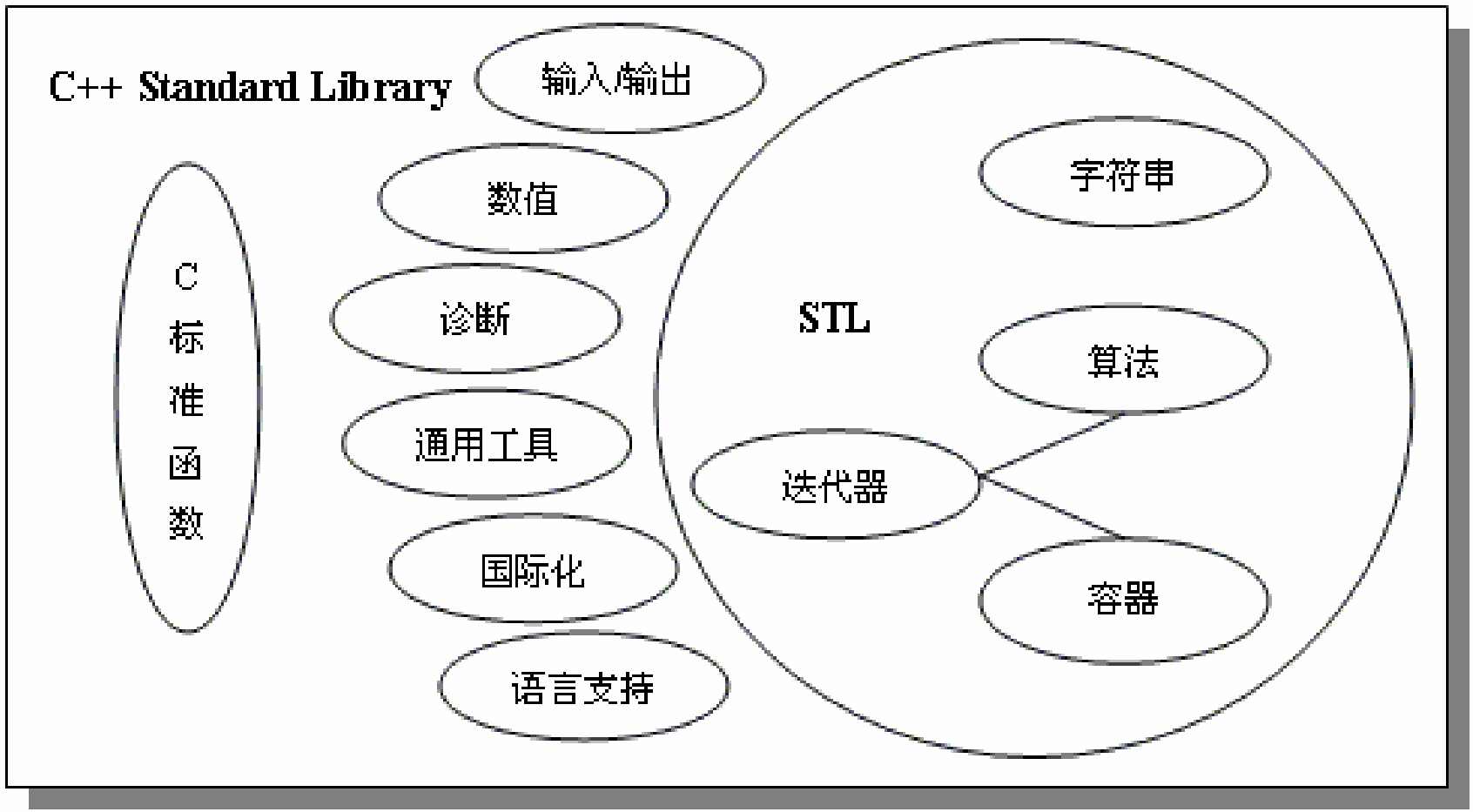
第十六章 标准模板库(STL)简介

Standard Template Library

什么是STL?

- 标准模板库(Standard Template Library)
  - 由惠普实验室开发
  - 产生于C++之前
  - 现在是ANSI/ISO C++标准的一部分
    - 高效的C++程序库
    - 内建在你的编译器之内
  - 几乎所有的代码都采用了模板类和模版函数的方式
    - 这相比于传统的由函数和类组成的库来说提供了更好的代码重用机会

C++标准函数库


C++ Standard Library
输入/输出
数值
诊断
通用工具
国际化
语言支持
C
标
准
函
数
STL
字符串
算法
迭代器
容器

C++标准函数库

- C标准函数库
  - 基本保持了与原有C语言程序库的良好兼容
  - 在C++标准库中存在两套C的函数库，一套是带有.h扩展名的（比如<stdio.h>），而另一套则没有（比如<cstdio>）。它们确实没有太大的不同。
- 语言支持（language support）部分
  - 一些标准类型的定义以及其他特性的定义
- 诊断（diagnostics）部分
  - 提供了用于程序诊断和报错的功能，包含了异常处理（exception handling），断言（assertions），错误代码（error number codes）三种方式。
- 通用工具（general utilities）部分
  - 动态内存管理工具，日期/时间处理工具
- 字符串（string）部分
  - 用来代表和处理文本。它提供了足够丰富的功能。String可以和char*相互转换
- 国际化（internationalization）部分
  - 作为OOP特性之一的封装机制在这里扮演着消除文化和地域差异的角色
- 输入/输出（input/output）部分
  - 就是经过模板化了的原有标准库中的iostream部分，它提供了对C++程序输入输出的基本支持。
- STL

为何有STL

- 可复用性（reusability）
- 程序设计思想
  - 泛型化设计（generic programming）
  - 以一种类型参数化（type parameterized）的方式实现
  - 模板(template)

STL组件

- container（容器）
  - 容器是一种数据结构，以模板类的方法提供
  - <vector>、<list>、 <queue>、 <deque>、 <stack>、<map>、 <set>
- iterator（迭代器）
  - 提供了访问容器中对象的方法，可视为一个指针
  - <iterator> 、<utility> 、<memory>
- algorithm（算法）
  - 用来操作容器中的数据的模板函数
  - <algorithm> 、 <numeric>、 <functional>

算法（Algorithm）

- 函数库对数据类型的选择对其可重用性起着至关重要的作用
- STL提供了大约100个实现算法的模版函数
  - 在一个有效的框架中完成这些算法的
  - 你可以将所有的类型划分为少数的几类，然后就可以在模版的参数中使用一种类型替换掉同一种类中的其他类型
  - 许多代码可以被大大的化简，只需要通过调用一两个算法模板，就可以完成所需要的功能并大大地提升效率
- <algorithm>是所有STL头文件中最大的一个，它是由一大堆模版函数组成的，可以认为每个函数在很大程度上都是独立的，其中常用到的功能范围涉及到比较、交换、查找、遍历操作、复制、修改、移除、反转、排序、合并等等。
- <numeric>体积很小，只包括几个在序列上面进行简单数学运算的模板函数，包括加法和乘法在序列上的一些操作。
- <functional>中则定义了一些模板类，用以声明函数对象。

容器（container）

- 在实际的开发过程中，数据结构本身的重要性不会逊于操作于数据结构的算法的重要性
- 经典的数据结构数量有限，但是我们常常重复着一些为了实现向量、链表等结构而编写的代码，这些代码都十分相似，只是为了适应不同数据的变化而在细节上有所出入
- STL容器就为我们提供了这样的方便，它允许我们重复利用已有的实现构造自己的特定类型下的数据结构，通过设置一些模版类，STL容器对最常用的数据结构提供了支持，这些模板的参数允许我们指定容器中元素的数据类型，可以将我们许多重复而乏味的工作简化

数据结构	描述	实现头文件
向量(vector)	连续存储的元素	<vector>
列表(list)	由节点组成的双向链表，每个结点包含着一个元素	<list>
双队列(deque)	连续存储的指向不同元素的指针所组成的数组	<deque>
集合(set)	由节点组成的红黑树，每个节点都包含着一个元素，节点之间以某种作用于元素对的谓词排列，没有两个不同的元素能够拥有相同的次序	<set>
多重集合(multiset)	允许存在两个次序相等的元素的集合	<set>
栈(stack)	后进先出的值的排列	<stack>
队列(queue)	先进先出的执的排列	<queue>
优先队列(priority_queue)	元素的次序是由作用于所存储的值对上的某种谓词决定的的一种队列	<queue>
映射(map)	由{键，值}对组成的集合，以某种作用于键对上的谓词排列	<map>
多重映射(multimap)	允许键对有相等的次序的映射	<map>

迭代器（iterator）

- 软件设计有一个基本原则，所有的问题都可以通过引进一个间接层来简化，这种简化在STL中就是用迭代器来完成的
- 迭代器在STL中用来将算法和容器联系起来，起着一种黏和剂的作用
  - 每一个容器都定义了其本身所专有的迭代器，用以存取容器中的元素
  - 用户通过迭代器存取容器的元素序列

STL的特点

- 数据结构和算法的分离
  - sort()
- 模板
  - 非面向对象的
  - 能用于任何数据类型和结构
  - STL的组件具有广泛通用性的底层特征
- 可移植性好
  - 稳定

优秀的STL

- 最好不要试图超越STL
  - STL实现使用的是最佳算法。它是几乎完美的。
  - STL实现的设计者通常是相应领域的专家。
  - 各领域的专家致力于提供灵活的、强大和高效的库。这是他们的首要的任务。对于，我们这些其余的人，开发可重用的容器和算法顶多只算第二个目标。我们的首要任务是交付紧扣主题的应用程序。大多数情况下，我们没有时间和专门的技术去和那些专家相比。
- 超越STL不是不可能的
  - 靠牺牲可移植性来提高性能
  - 为了超越STL，我们要付出非常大的努力

STL的头文件

- 为了避免和其他头文件冲突，STL的头文件不再使用常规的.h扩展
  - #include <vector>
  - #include <iterator>
  - #include <algorithm>
  - #include <string>

STL的名字空间

- namespace
  - 名字空间就好像一个信封，将标志符封装在另一个名字中。标志符只在名字空间中存在，因而避免了和其他标志符冲突。
  - 避免名字冲突
  - 例如，可能有其他库和程序模块定义了sort()函数，为了避免和STL地sort()算法冲突，STL的sort()以及其他标志符都封装在名字空间std中。STL的sort()算法编译为std::sort()
- using namespace std;

STL Vector容器介绍

什么是Vector?

- vector是C++标准模板库中的部分内容
- 是一个多功能的，能够操作多种数据结构和算法的模板类和函数库
- vector之所以被认为是一个容器，是因为它能够像容器一样存放各种类型的对象
- 简单地说，vector是一个能够存放任意类型的动态数组，能够增加和压缩数据

使用Vector

- 包含头文件
  - #include <vector>

- 名字空间(namespace)：
- vector属于std命名域的：
  - using std::vector;
- 或者连在一起，使用全名：
  - std::vector<int> vInts;
- 建议使用全局的名字空间
  - using namespace std;

创建一个int型的vector

- vector<int> intarray;

向vector添加一个数据

- vector添加数据的缺省方法是push_back()
  - push_back()函数表示将数据添加到vector的尾部
  - 并按需要来分配内存

```
for(int i= 0;i<10; i++)
    intarray.push_back( i );
```

向vector插入一个数据

- insert()

判断数据个数

- empty()
  - 判断vector是否为空
- size()
  - 返回vector的数据个数

预先分配内存空间

- reserve(int n)
- resize(int n)

vector的元素访问

- begin()
- end()

- front()

访问vector中的数据

- 使用两种方法来访问vector中的数据
  - vector::at(int idx)
  - vector::operator[int idx]
- 区别
  - operator[]主要是为了与C语言进行兼容。它可以像C语言数组一样操作。
  - at()是我们的首选，因为at()进行了边界检查，如果访问超过了vector的范围，将抛出一个例外。
  - 由于operator[]容易造成一些错误，所以尽量少用它。

删除vector中的数据

- pop_back()
- erase(iterator first, iterator last)
- clear()

访问元素的iterator

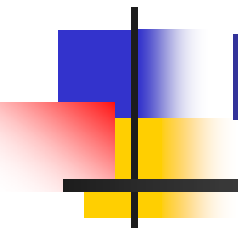
- vector<int>::iterator iter;

- for(iter=intarry.begin(); iter!=intarray.end(); iter++)

Examples

创建一个自定义类型的vector

- 创建一个Widget类型的空的vector对象：
  - vector<Widget> vWidgets;
- 创建一个包含500个Widget类型数据的vector：
  - vector<Widget> vWidgets(500);
- 创建一个包含500个Widget类型数据的vector，并且都初始化为0：
  - vector<Widget> vWidgets(500, Widget(0));
- 创建一个Widget的拷贝：
  - vector<Widget> vWidgetsFromAnother(vWidgets);



list

建立一个list

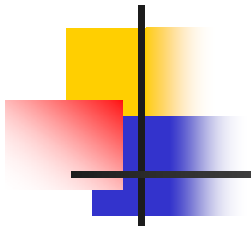
```
#include <string>
#include <list>
int main (void)
 {
     list<string>    Milkshakes;
     return 0;
}
```

添加数据

```
#include <string>
#include <list>

int main (void)
{
  list<string> Milkshakes;
  Milkshakes.push_back("Chocolate");
  Milkshakes.push_back("Strawberry");
  Milkshakes.push_front("Lime");
  Milkshakes.push_front("Vanilla");
  return 0;
}
```

遍历数据

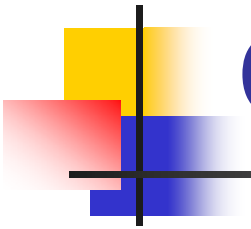
```
/*
|| How to print the contents of a simple STL list. Whew!
*/
#include <iostream.h>
#include <string>
#include <list>

int main (void)
{
  list<string> Milkshakes;
  list<string>::iterator MilkshakeIterator;

  Milkshakes.push_back("Chocolate");
  Milkshakes.push_back("Strawberry");
  Milkshakes.push_front("Lime");
  Milkshakes.push_front("Vanilla");

  // print the milkshakes
  Milkshakes.push_front("The Milkshake Menu");
  Milkshakes.push_back("*** Thats the end ***");
  for (MilkshakeIterator=Milkshakes.begin();
       MilkshakeIterator!=Milkshakes.end();
        ++MilkshakeIterator)
  {
    // dereference the iterator to get the element
    cout << *MilkshakeIterator << endl;
  }
}
```



现在开始用STL!

Learning Resources

- http://stl.winterxy.com/html/000033.html
- STL源码剖析
- My homepage ☺



Q&A