STL: последовательные контейнеры

Александр Смаль

CS центр
15 февраля 2017
Санкт-Петербург

STL: введение

- STL = Standard Template Library
- STL описан в стандарте C++, но не упоминается там явно.
- Авторы: Александр Степанов, Дэвид Муссер и Менг Ли (сначала для HP, а потом для SGI).
- Основан на разработках для языка Ада.
- Основные составляющие:
  - контейнеры (хранение объектов в памяти),
  - итераторы (доступ к элементам контейнера),
  - алгоритм (для работы с последовательностями),
  - адаптеры (обёртки над контейнерами)
  - функциональные объекты, функторы (обобщение функций).
  - потоки ввода/вывода.
- Всё определено в пространстве имён `std`.

Общие сведения о контейнерах

Контейнеры библиотеки STL можно разделить на четыре категории:

- последовательные,
- ассоциативные,
- контейнеры-адаптеры,
- псевдоконтейнеры.

Требования к хранимым объектам:

1. copy-constructable
2. assignable
3. “стандартная семантика”

Итераторы — объекты для доступа к элементам контейнера с синтаксисом указателей.

Общие члены контейнеров

Типы (typedef-ы или вложенные класс):

1. `C::value_type`
2. `C::reference`
3. `C::const_reference`
4. `C::pointer`
5. `C::iterator`
6. `C::const_iterator`
7. `C::size_type`

Методы:

1. Конструктор по умолчанию, конструктор копирования, оператор присваивания, деструктор.
2. `begin()`, `end()`
3. Операторы сравнения: `==`, `!=`, `>`, `>=`, `<`, `<=`.
4. `size()`, `empty()`.
5. `swap(obj2)`

Последовательные контейнеры

Общие члены

1. Конструктор от двух итераторов
2. Конструктор от count и defVal
3. Двух итераторный `erase`
4. `push_back, pop_back, back`
5. `front`
6. `assign` от двух итераторов
7. `assign` от count и val
8. `insert` от итератора и val
9. `insert` от итератора, n и val
10. `insert` от трёх итераторов

vector

C-подобный динамический массив произвольного доступа с автоматическим изменением размера при добавлении элементов.

1. `operator[], at`
2. `resize`
3. `capacity, reserve`

Разработан для работы со старым кодом.

```
#include <vector>

void legacy_function(int * m, int size);

std::vector<int> v;
...
legacy_function(&v[0], v.size());
```

deque

Контейнер похож на vector, но с возможностью быстрой вставки и удаления элементов на обоих концах за $O(1)$.
Реализован как список указателей на массивы фиксированного размера.

1. Конструктор от n
2. `operator[], at`
3. `resize`
4. `push_front, pop_front`

```
#include <deque>


    std::deque<std::string> d;
    d.push_back(", world!");
    d.push_front("Hello");
    std::cout << d[0] << d[1] << std::endl;
```

list

Двусвязный список. В любом месте контейнера вставка и удаление производятся за $O(1)$.

1. `merge, remove, remove_if, sort, unique`
2. `splice`
3. `push_front, pop_front`

```
#include <list>

    std::list<std::string> l;
    l.push_back(", world!");
    l.push_front("Hello");
    std::cout << l.size();
```

Итерация по списку

Нет возможности обратиться к элементу списка по индексу, но есть возможность перебирать элементы с помощью *итераторов*.

```
list<int> l;
...
for(list<int>::iterator i = l.begin(); i != l.end(); ++i)
{
    *i += 10;
    std::cout << *i << std::endl;
}
```

Итератор списка можно перемещать в обоих направлениях:

```
list<int>::iterator end = l.end();
list<int>::iterator last = end;
--last; // последний элемент с конца
```

string, wstring, basic_string

Контейнер для хранения символьных последовательностей.

1. Метод c_str() для совместимости со старым кодом:

```
std::string res = "Hello";
...
printf("%s", res.c_str());
```

2. поддержка работы с строками в стиле C
3. множество алгоритмов вроде substr() (в терминах *индексов*),
4. string = basic_string<char>
5. wstring = basic_string<wchar_t>
6. могут быть реализованы как COW (Copy-On-Write).

Адаптеры и псевдоконтейнеры

Адаптеры:

1. `stack` — реализация интерфейса стека.
2. `queue` — реализация интерфейса очереди.
3. `priority_queue` — очередь с приоритетом на куче.

Псевдо-контейнеры:

1. `vector<bool>`
   1.1 ненастоящий контейнер (не хранит `bool`-ы),
   1.2 использует proxy-объекты.
2. `bitset`
   Служит для хранения битовых масок. Похож на `vector<bool>` фиксированного размера.
3. `valarray`
   Шаблон служит для хранения числовых массивов и оптимизирован для достижения повышенной вычислительной производительности.

Ещё о vector

- vector — самый универсальный последовательный контейнер, во многих случаях самый эффективный.
- Предпочитайте vector другим контейнерам.
- Интерфейс вектора построен на итераторах, а не на индексах.
- Итераторы вектора ведут себя как указатели.

```
std::vector<int> v(100, 0);
...
v.erase(v.begin() + 10);          // удалить элемент с номером 10
v.insert(v.begin() + 10, 5);      // добавить 5 перед элементом N 10
```

vector: идиомы

Сжатие и очистка

```
std::vector<int> v;
...
std::vector<int>(v).swap(v); // compress, C++11 shrink_to_fit

std::vector<int>().swap(v);  // clear
```

Использование reserve и capacity

```
std::vector<int> v;
v.reserve(N); // N - upperbound for size
...
if (v.capacity() == v.size()) // reallocation
    ...
```