Technology
Tarena
达内科技

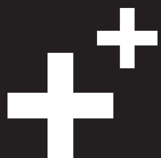
模板和STL

TEMPLATE & STL

DAY03

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	容器
	10:30 ~ 11:20	迭代器与泛型算法
	11:30 ~ 12:20	
下午	14:00 ~ 14:50	向量
	15:00 ~ 15:50	
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑



容器

容器

双向线性链表容器

- 基础设施
- 构造、析构、深拷贝
- 获取首元素
- 向首部压入
- 从首部弹出
- 删除所有匹配元素
- 清空、判空和大小
- 针对char const*类型的特化

Technology Tarena 达内科技

基础设施

Tarena Technology
达内科技

知识讲解

begin
insert
erase
end
--
++
iterator
iterator
iterator
*/->
list
m_head
m_tail
node
...
NULL
...
node
m_data
m_prev
m_next
...
...
...
node
...
...
NULL
push_front
remove
push_back

基础设施（续1）

Tarena Technology 达内科技

知识讲解

- 节点及其指针
  - template<typename T> class List {

```
template<typename T> class List {
private:
    class Node {
    public:
        Node (T const& data, Node* prev = NULL,
            Node* next = NULL) : m_data (data),
            m_prev (prev), m_next (next) {}
        T m_data;
        Node* m_prev,
        Node* m_next;
    };
    Node* m_head,
    Node* m_tail;
};
```

构造、析构、深拷贝

Tarena Technology 达内科技

知识讲解

- 构造函数初始化空链表，析构函数销毁剩余节点，拷贝构造函数和拷贝赋值运算符函数支持深拷贝
  - template<typename T> class List {

```
template<typename T> class List {
    List (void) : m_head (NULL), m_tail (NULL) {}
    ~List (void) { clear (); }
    List (List const& that) : m_head (NULL), m_tail (NULL) {
        for (Node* node = that.m_head; node;
            node = node->m_next) push_back (node->m_data); }
    List& operator= (List const& rhs) {
        if (&rhs != this) {
            List list = that;
            swap (m_head, list.m_head); swap (m_tail, list.m_tail); }
        return *this;
    }
};
```

Technology Tarena 达内科技

获取首元素

- 通过普通容器获取其首元素的左值引用，通过常容器获取其首元素的常左值引用，后者应能够复用前者的实现

```
template<typename T> class List {
public:
    T& front (void) {
        if (empty ())
            throw underflow_error ("链表下溢！");
        return m_head->m_data;
    }
    T const& front (void) const {
        return const_cast<List*> (this)->front ();
    }
};
```

知识讲解

向首部压入

Tarena Technology 达内科技

知识讲解

- 在首节点的前面增加新节点，使新节点成为新的首节点

```
template<typename T> class List {
public:
    void push_front (T const& data) {
        m_head = new Node (data, NULL, m_head);
        if (m_head->m_next)
            m_head->m_next->m_prev = m_head;
        else
            m_tail = m_head;
    }
};
```

从首部弹出

Tarena Technology 达内科技

知识讲解

- 删除首节点，使被删除节点的后节点成为新的首节点

```
template<typename T> class List {
public:
    void pop_front (void) {
        if (empty ())
            throw underflow_error ("链表下溢！");
        Node* next = m_head->m_next;
        delete m_head;
        m_head = next;
        if (m_head) m_head->m_prev = NULL;
        else m_tail = NULL;
    }
};
```

删除所有匹配元素

Tarena Technology 达内科技

知识讲解

- 遍历的同时判断是否相等，删除所有满足条件的节点
  -
```
template<typename T> class List {
    void remove (T const& data) {
        for (Node* node = m_head, *next; node; node = next) {
            next = node->m_next;
            if (node->m_data == data) {
                if (node->m_prev)
                    node->m_prev->m_next = node->m_next;
                else m_head = node->m_next;
                if (node->m_next)
                    node->m_next->m_prev = node->m_prev;
                else m_tail = node->m_prev;
                delete node; }
        }
    }
};
```

清空、判空和大小

Tarena Technology 达内科技

知识讲解

- 在遍历中删除所有节点
  - template<typename T> class List {

```
template<typename T> class List {
    void clear (void) {
        for (Node* next; m_head; m_head = next) {
            next = m_head->m_next; delete m_head; }
        m_tail = NULL; }
    bool empty (void) const {
        return ! m_head && ! m_tail; }
     size_t size (void) const {
        size_t nodes = 0;
        for (Node* node = m_head; node;
            node = node->m_next) ++nodes;
        return nodes; }
};
```

针对char const*类型的特化

- 将类型相关的操作与类型无关的操作分开
  - template<typename T> class List {

```
template<typename T> class List {
    void remove (T const& data) {
        for (Node* node = m_head, *next; node; node = next) {
            next = node->m_next;
            if (equal (node->m_data, data)) {
                if (node->m_prev)
                    node->m_prev->m_next = node->m_next;
                else m_head = node->m_next;
                if (node->m_next)
                    node->m_next->m_prev = node->m_prev;
                else m_tail = node->m_prev;
                delete node; }
        }
    }
};
```

Tarena Technology 达内科技

知识讲解

针对char const*类型的特化（续1）

Tarena Technology 达内科技

知识讲解

- 分别实现equal函数的通用版本和特化版本

```
– template<typename T> class List {
  private:
      bool equal (T const& a, T const& b) const {
          return a == b;
      }
  };
– template<>
  bool List<char const*>::equal (
      char const* const& a, char const* const& b) const {
      return ! strcmp (a, b) == 0;
  }
```

Technology Tarena 达内科技

双向线性链表容器模板

【参见：TTS COOKBOOK 】

课堂练习

- 双向线性链表容器模板

迭代器与泛型算法

迭代器与泛型算法

- 正向迭代器
  - 不用迭代器的遍历
  - 正向迭代器内部类
  - 获取起始和终止迭代器
  - 基于迭代器的插入
  - 基于迭代器的删除
- 泛型算法
  - 线性查找

Technology Tarena 达内科技

不用迭代器的遍历

Tarena Technology 达内科技

知识讲解

- 允许用户以类似数组的方式，通过下标访问容器中的元素
```
template<typename T> class List {
    T& operator[] (size_t i) {
        for (Node* node = m_head; node;
            node = node->m_next)
            if (i-- == 0)
                return node->m_data;
        throw out_of_range ("下标越界！");
    }
    T const& operator[] (size_t i) const {
        return const_cast<List&> (*this)[i];
    }
};
```

不用迭代器的遍历（续1）

- 通过下标遍历链表的平均时间复杂度高达 $O(N^2)$ 级
  - List<int> list;
    ```
    ...
    size_t size = list.size ();
    for (size_t i = 0; i < size; ++i)
        cout << list[i] << ' ';
    cout << endl;
    ```
  - T& List<T>::operator[] (size_t i) {
    ```
        ...
        for (Node* node = m_head; node;
            node = node->m_next)
            ...
    }
    ```

$O(N^2)$

知识讲解

Tarena Technology 达内科技

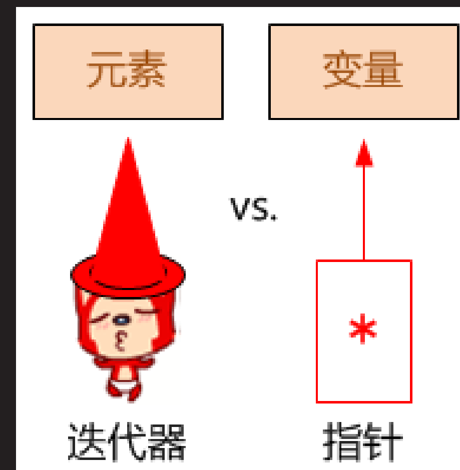
正向迭代器内部类

Tarena Technology 达内科技

知识讲解

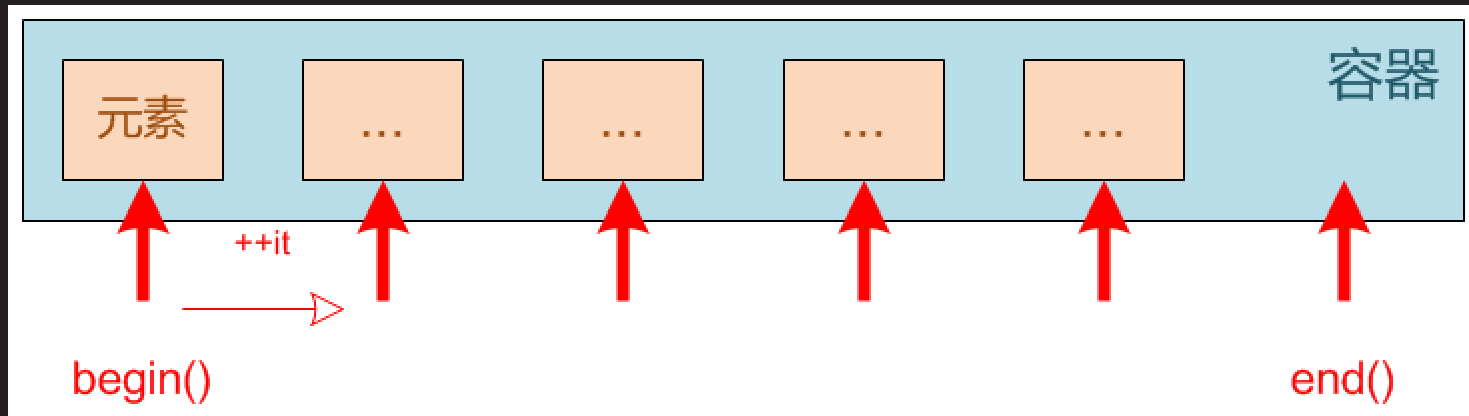
- 封装节点的指针，扮演指向容器中元素的指针的角色
  - template<typename T> class List {

```
public:
    class Iterator {
    public:
        Iterator (Node* head = NULL, Node* tail = NULL,
            Node* node = NULL) : m_head (head), m_tail (tail),
            m_node (node) {}
    private:
        Node* m_head;
        Node* m_tail;
        Node* m_node;
        friend class List;
    };
};
```



正向迭代器内部类（续1）

- 通过任何容器的迭代器访问其中的元素，都与通过指针访问数组中元素无异。允许用户在完全不了解容器内部细节的前提下，以一致且透明的方式，访问其中的元素



正向迭代器内部类（续2）

Tarena Technology 达内科技

- 象指针一样支持“==”和“!=”运算符
  - class Iterator {

```
class Iterator {
public:
    bool operator== (Iterator const& rhs) const {
        return m_node == rhs.m_node;
    }
    bool operator!= (Iterator const& rhs) const {
        return ! (*this == rhs);
    }
};
```

知识讲解

正向迭代器内部类（续3）

Tarena Technology 达内科技

- 象指针一样支持前后“++”运算符
  - class Iterator {

```
class Iterator {
    Iterator& operator++ (void) {
        if (m_node) m_node = m_node->m_next;
        else m_node = m_head;
        return *this;
    }
    Iterator const operator++ (int) {
        Iterator old = *this;
        ++*this;
        return old;
    }
};
```

知识讲解

正向迭代器内部类（续4）

Tarena Technology 达内科技

- 象指针一样支持前后“--”运算符
  - class Iterator {

```
class Iterator {
    Iterator& operator-- (void) {
        if (m_node) m_node = m_node->m_prev;
        else m_node = m_tail;
        return *this;
    }
    Iterator const operator-- (int) {
        Iterator old = *this;
        --*this;
        return old;
    }
};
```

知识讲解

正向迭代器内部类（续5）

Tarena Technology 达内科技

知识讲解

- 象指针一样支持“*”和“->”运算符
  - class Iterator {

```
class Iterator {
public:
    T& operator* (void) const {
        return m_node->m_data;
    }
    T* operator-> (void) const {
        return &**this;
    }
};
```

获取起始和终止迭代器

- 起始正向迭代器即指向容器首元素的迭代器
  - template<typename T> class List {

```
template<typename T> class List {
public:
    Iterator begin (void) {
        return Iterator (m_head, m_tail, m_head); }
};
```

- 终止正向迭代器即指向容器尾元素之后的迭代器

```
template<typename T> class List {
public:
    Iterator end (void) {
        return Iterator (m_head, m_tail); }
};
```

基于迭代器的插入

Tarena Technology 达内科技

- 创建新节点，并使迭代器所指向的节点及其前节点成为新建节点的后节点和前节点，返回指向新节点的迭代器
  - template<typename T> class List {

```
template<typename T> class List {
    Iterator insert (Iterator loc, T const& data) {
        if (loc == end ()) {
            push_back (data);
            return Iterator (m_head, m_tail, m_tail); }
        else {
            Node* node = new Node (data, loc.m_node->m_prev,
                loc.m_node);
            if (node->m_prev) node->m_prev->m_next = node;
            else m_head = node;
            node->m_next->m_prev = node;
            return Iterator (m_head, m_tail, node); }
    }
};
```

知识讲解

基于迭代器的删除

Technology Tarena 达内科技

知识讲解

- 删除迭代器所指向的节点，令该节点前后节点的后前指针指向其后前节点，返回指向被删除节点之后的迭代器
  - ```
    template<typename T> class List {
        Iterator erase (Iterator loc) {
            if (loc == end ()) throw invalid_argument ("无效参数！");
            if (loc.m_node->m_prev)
                loc.m_node->m_prev->m_next = loc.m_node->m_next;
            else m_head = loc.m_node->m_next;
            if (loc.m_node->m_next)
                loc.m_node->m_next->m_prev = loc.m_node->m_prev;
            else m_tail = loc.m_node->m_prev;
            Node* next = loc.m_node->m_next;
            delete loc.m_node;
            return Iterator (m_head, m_tail, next);
        }
    };
    ```

    ```

Technology Tarena 达内科技

# 正向迭代器

【参见：TTS COOKBOOK 】

课堂练习

- 正向迭代器

Technology Tarena
达内科技

# 线性查找

- 基于迭代器的线性查找
  - ```
    template<typename IT, typename T>
    IT find (IT begin, IT end, T const& key) {
        IT it;
        for (it = begin; it != end; ++it)
            if (*it == key) break;
        return it;
    }
    ```
  - ```
    int array[] = {13, 27, 19, 48, 36}, key = 19;
    size_t size = sizeof (array) / sizeof (array[0]);
    int* p = find (array, array + size, key);
    if (p == array + size) cout << "没找到！" << endl;
    else cout << "找到了：" << *p << endl;
    ```

知识讲解

Tarena Technology 达内科技

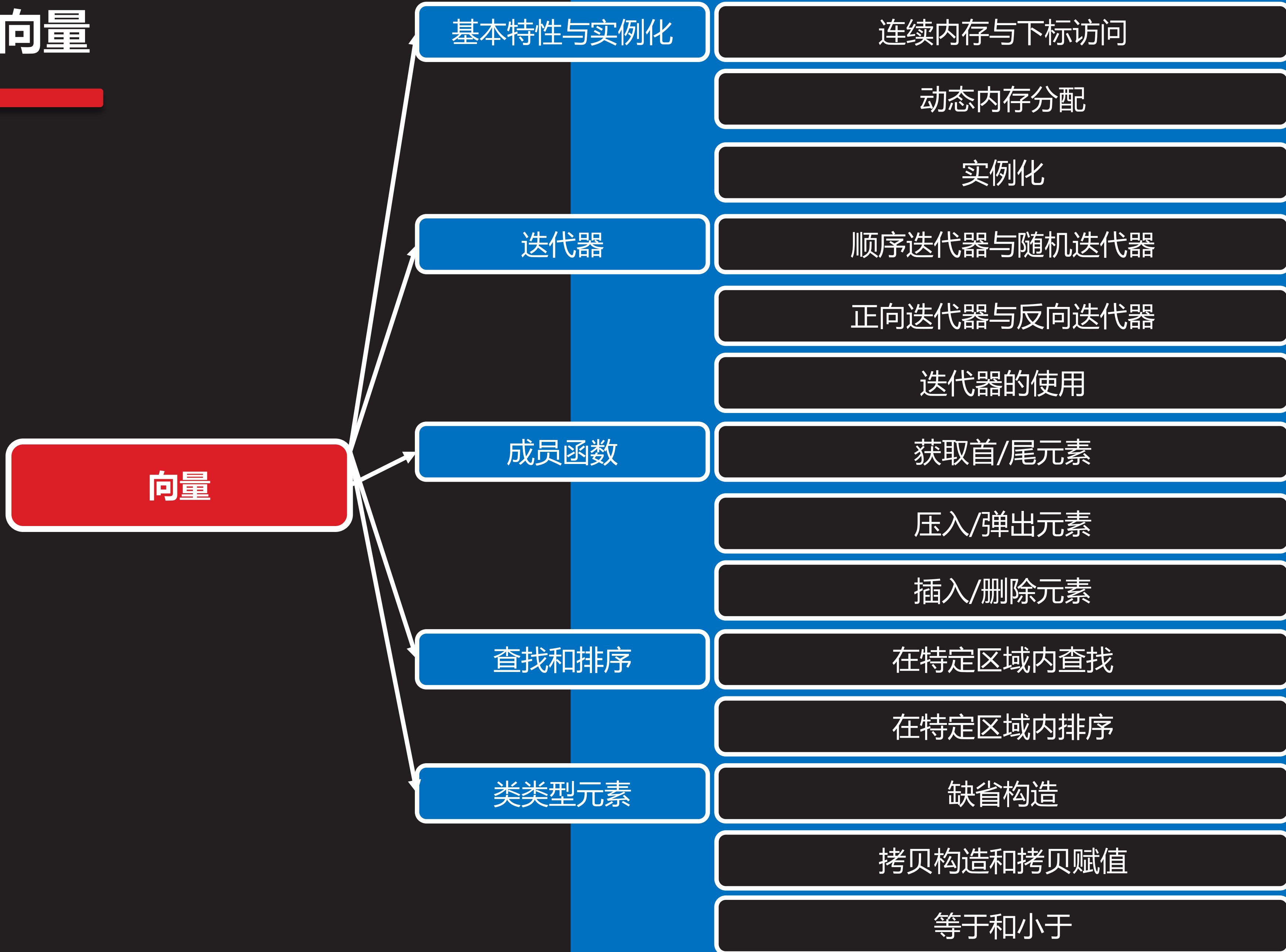
线性查找

【参见：TTS COOKBOOK 】

课堂练习

- 线性查找

向量



Technology Tarena 达内科技

Tarena Technology 达内科技

连续内存与下标访问

- 向量中的元素被存储在一段连续的内存空间中
- 通过下标随机访问向量元素的效率与数组相当

insert()
erase()
push_back()
front()
back()
pop_back()
begin()
end()
vector

知识讲解

动态内存分配

- 向量的内存空间会随着新元素的加入而自动增长
- 内存空间的连续性不会妨碍向量元素的持续增加
- 如果当前内存空间无法满足连续存储的需要，向量会自动开辟新的足够的连续内存空间，并在把原空间中的元素复制到新空间中以后，释放原空间
- 向量的增长往往伴随着内存的分配与释放、元素的复制与销毁等额外开销
- 如果能够在创建向量时合理地为它预分配一些空间，将在很大程度上缓解这些额外开销

实例化

Technology Tarena 达内科技

知识讲解

- vector<元素类型> 向量对象;
  - vector<int> vi;
    空向量不包含任何数据元素，也不占任何用于存放数据元素的内存空间，但向量对象本身的内存空间仍要占用
- vector<元素类型> 向量对象 (初始大小);
  - vector<int> vi (5);
    基本类型元素，用适当类型的零初始化
  - vector<Student> vs (5);
    类类型的元素，用缺省构造函数初始化

实例化（续1）

- vector<元素类型> 向量对象 (初始大小, 元素初值);
  - vector<int> vi (5, 10);
    元素初值用于初始化初始大小范围内所有的数据元素，而不只是初始化第一个元素
  - vector<Student> vs (5, Student ("张飞", 25));
    对于类类型向量而言，初始化的过程其实就是调用元素类型中相应构造函数的过程
- vector<元素类型> 向量对象 (起始迭代器, 终止迭代器);
  - vector<int> v1 (5);
    vector<int> v2 (v1.begin (), v1.end ());
  - int arr[] = {10, 20, 30, 40, 50};
    vector<int> v2 (&arr[0], &arr[5]);

向量的实例化

【参见：TTS COOKBOOK 】

课堂练习

- 向量的实例化

迭代器

顺序迭代器与随机迭代器

Technology
Tarena
达内科技

- 顺序迭代器
  - 一次只能向后或向前迭代一步
  - 只支持“++”和“--”运算
- 随机迭代器
  - 既能一次迭代一步，也能一次迭代多步
  - 除了支持“++”和“--”运算外，还支持对整数的加减运算、迭代器之间的大小比较及相减运算
- 除了向量和双端队列这两个连续内存容器可以提供随机迭代器外，其它STL容器都只提供顺序迭代器

正向迭代器与反向迭代器

- 正向迭代器
  - 起始迭代器指向容器中的第一个元素，终止迭代器指向容器中最后一个元素的下一个位置
  - 增操作向容器的尾端移动，减操作向容器首端移动
- 反向迭代器
  - 起始迭代器指向容器中的最后一个元素，终止迭代器指向容器中第一个元素的前一个位置
  - 增操作向容器的首端移动，减操作向容器尾端移动

迭代器的使用

Tarena Technology 达内科技

知识讲解

- 四个迭代器类型
  - iterator：正向迭代器
  - const_iterator：常正向迭代器
  - reverse_iterator：反向迭代器
  - const_reverse_iterator：常反向迭代器
- 八个迭代器对象
  - begin()：起始正向迭代器
  - begin() const：起始常正向迭代器
  - end()：终止正向迭代器
  - end() const：终止常正向迭代器
  - rbegin()：起始反向迭代器
  - rbegin() const：起始常反向迭代器
  - rend()：终止反向迭代器
  - rend() const：终止常反向迭代器

迭代器的使用（续1）

Tarena Technology 达内科技

知识讲解

- 任何可能导致容器结构发生变化的函数被调用以后，先前获得的迭代器都可能因此失效，重新初始化以后再使用才是安全的

```
vector<int> vi;
vi.push_back (100);
vector<int>::iterator it = vi.begin ();
cout << *it << endl; // 100
vi.push_back (200);
cout << *it << endl; // ?
```

Technology Tarena 达内科技

向量的迭代器

【参见：TTS COOKBOOK 】

课堂练习

- 向量的迭代器

Tarena Technology 达内科技

成员函数

获取首/尾元素

- value_type& front (void);
  value_type const& front (void) const;
  value_type& back (void);
  value_type const& back (void) const;
- 例如
  - ```
    vector<int> vi (5, 10);
    vi.front () += 5;
    vector<int> const& cr = vi; cr.front () -= 5; // 错误
    cout << cr.front () << endl;
    --vi.back ();
    vector<int> const* cp = &vi; ++cp->back (); // 错误
    cout << cp->back () << endl;
    ```


Technology Tarena 达内科技

    ```

# 压入/弹出元素

Tarena Technology 达内科技

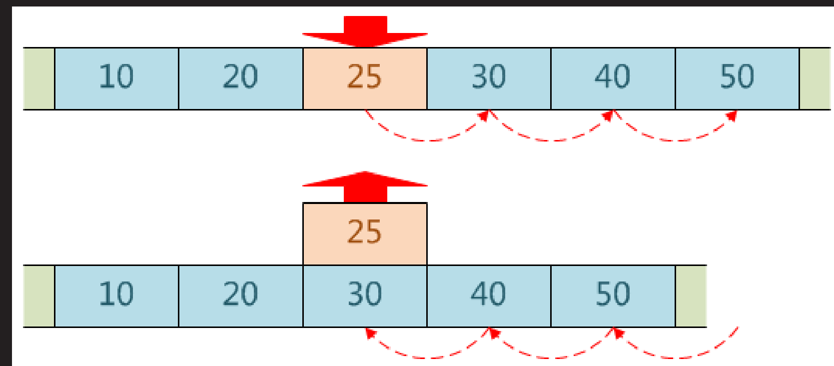
- void push_back (value_type const& val);
  void pop_back (void);
- 例如
  - 
```
vector<string> vs;
vs.push_back ("C++");
vs.push_back ("喜欢");
vs.push_back ("我们");
while (! vs.empty ()) {
    cout << vs.back () << flush;
    vs.pop_back ();
}
cout << endl;
```

知识讲解

插入/删除元素

- iterator insert (iterator loc, value_type const& val);
  iterator erase (iterator loc);
- 例如

```
– vector<int> vi (1, 55);
  vi.insert (vi.insert (vi.insert (vi.insert (
      vi.begin (), 44), 33), 22), 11);
  vector<int>::iterator it = vi.begin ();
  while (it != vi.end ())
      if (*it % 2)
          it = vi.erase (it);
      else
          ++it;
```



知识讲解

Technology
Tarena
达内科技

在特定区域内查找

- #include <algorithm>
  iterator find (iterator begin, iterator end,
  value_type const& key);
  成功返回第一个匹配元素的迭代器，失败返回第二个参数(指向查找范围内最后一个元素的下一个位置的迭代器)
- 例如
  - ```
    template<typename T>
    void remove (vector<T>& vec, T const& key) {
    for (typename vector<T>::iterator it = vec.begin ();
        (it = find (it, vec.end (), key)) != vec.end ();
        it = vec.erase (it));
    }
    ```

    ```

# 泛型查找

【参见：TTS COOKBOOK 】



- 泛型查找

# 在特定区域内排序

Technology Tarena 达内科技

- #include <algorithm>
  void sort (iterator begin, iterator end);
  void sort (iterator begin, iterator end, less cmp);
- 小于比较器
  – 形如：
  ```
  bool cmp (int const& a, int const& b) {
      return a < b; }
  ```
  的函数，或形如：
  ```
  class Less {
      bool operator () (int const& a, int const& b) {
          return a < b; }
  };
  ```
  的函数对象，定义容器元素的小于规则，以供排序之需

Tarena Technology 达内科技

# 泛型排序

【参见：TTS COOKBOOK 】

课堂练习

- 泛型排序

Technology
Tarena
达内科技

# 缺省构造



知识讲解

- 如果一个类类型对象需要被存储在向量中，那么该类至少应支持缺省构造，以确保向量内存的初始化
  - ```
    class Integer {
    public:
        Integer (int i = 0) : m_i (new int (i)) {}
        ~Integer (void) {
            if (m_i) {
                delete m_i; m_i = NULL; }
        }
    private:
        int* m_i;
    };
    ```
  - `vector<Integer> vi (5); vi.resize (10);`

    ```

# 拷贝构造和拷贝赋值

Technology Tarena 达内科技

知识讲解

- 该类还需要支持完整意义上的拷贝构造和拷贝赋值
  - ```
    class Integer {
    public:
        Integer (Integer const& that) :
            m_i (new int (*that.m_i)) {}
        Integer& operator= (Integer const& rhs) {
            *m_i = *rhs.m_i;
            return *this;
        }
    };
    ```
  - ```
    vi.push_back (100);
    vi.erase (vi.begin ());
    ```

等于和小于

Tarena Technology 达内科技

知识讲解

- 该类可能还需要支持“==”和“<”两个关系运算符，用于元素间的比较，其它关系运算可据此推断
  - class Integer {
    public:
        bool operator== (Integer const& rhs) const {
            return *m_i == *rhs.m_i;
        }
        bool operator< (Integer const& rhs) const {
            return *m_i < *rhs.m_i;
        }
    };
  - void greater (Integer const& a, Integer const& b) {
        return ! (a == b) && ! (a < b); }

类类型元素

【参见：TTS COOKBOOK 】

- 类类型元素

Technology
Tarena
达内科技