STL和数据结构

讲师：张嘉星

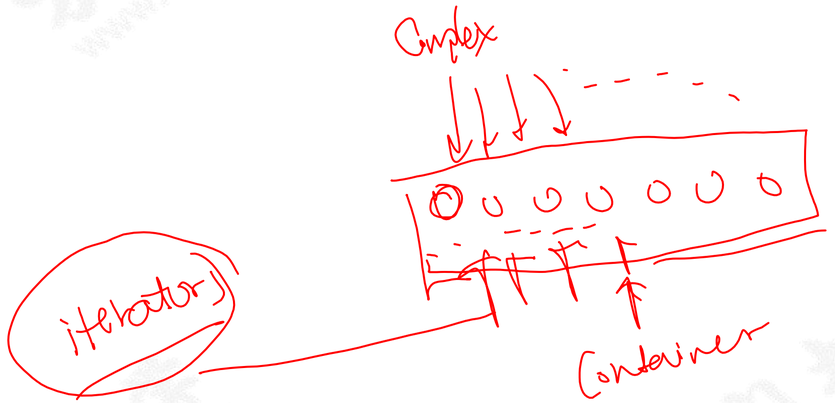
目标

- 为啥STL不是C++灵魂，却胜似灵魂？！
- 程序 = 数据结构 + 算法 (Wirth)
- STL包圆了！

STL简介

- STL（标准模板库）是一套功能强大的 C++ 模板类，提供了通用的模板类和函数，这些模板类和函数可以实现多种流行和常用的算法和数据结构，如向量、链表、队列、栈
- 标准模板库的核心包括以下三个组件：

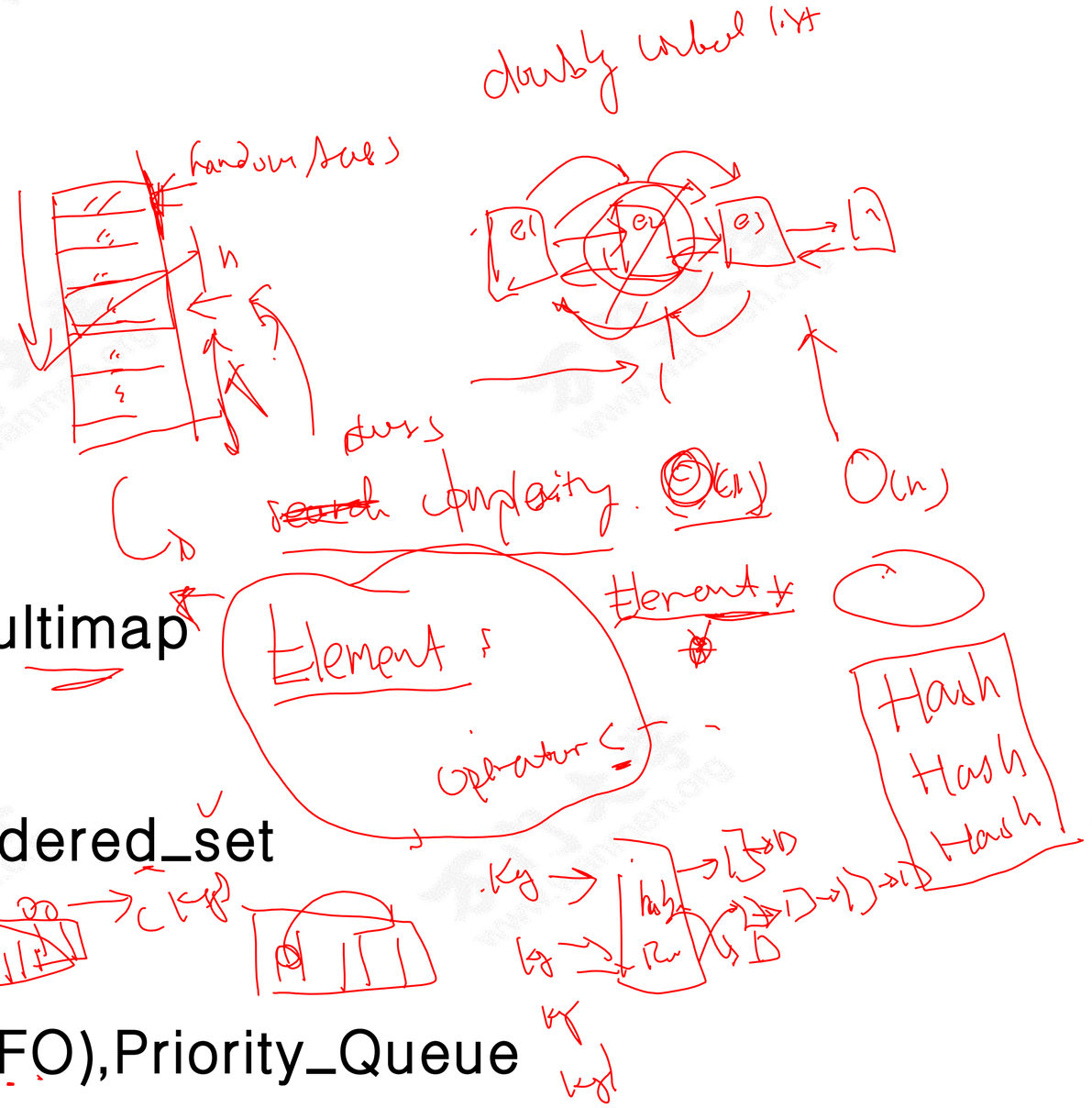
STL组件



组件	描述
容器（Containers）	容器是用来管理某一类对象的集合。C++ 提供了各种不同类型的容器，比如 deque、list、vector、map 等。
算法（Algorithms）	算法作用于容器。它们提供了执行各种操作的方式，包括对容器内容执行初始化、排序、搜索和转换等操作。
迭代器（iterators）	迭代器用于遍历对象集合的元素。这些集合可能是容器，也可能是容器的子集。

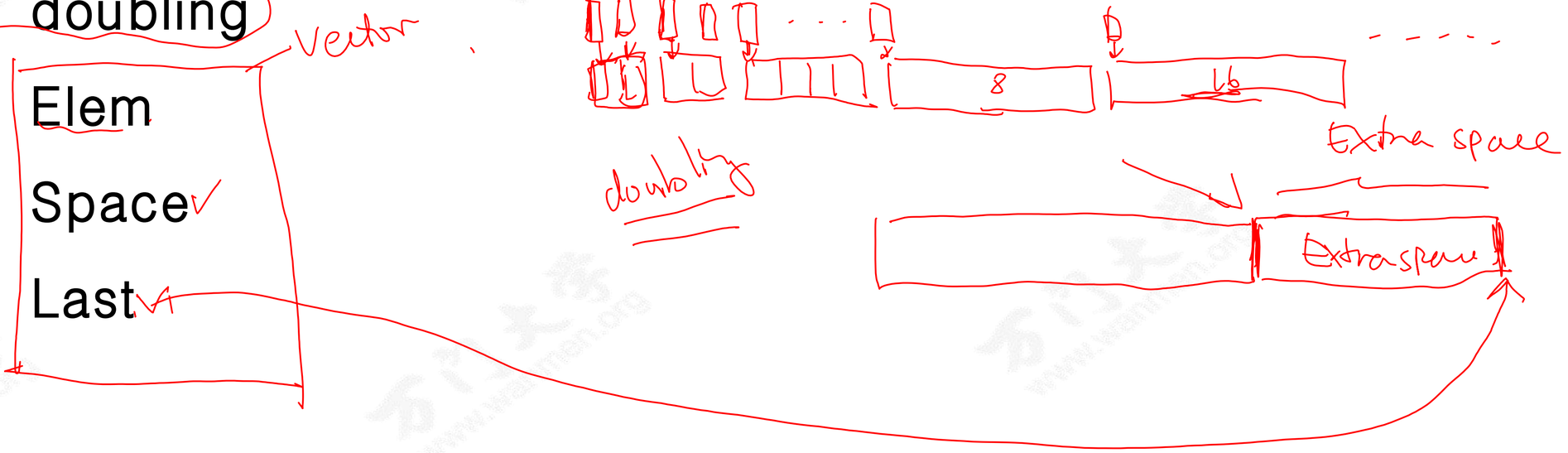
容器（Containers）

- 顺序容器
  - Vector，deque,list
- 关联容器
  - Set，multiset，map，multimap
- 基于Hash table的容器
  - Unordered_map，unordered_set
- 其他：
  - Stack(LIFO)，queue(FIFO),Priority_Queue



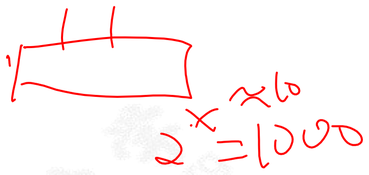
容器（Containers）

- std::vector
- 如何分配内存：
  - doubling
  - Elem
  - Space
  - Last

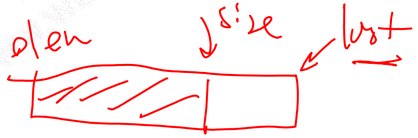


```
vector<Entry>phone_book = {
    {"David Hume",123456},
    {"Karl Popper",234567},
    {"Bertrand Arthur William Russell",345678}
};
```

容器（Containers）

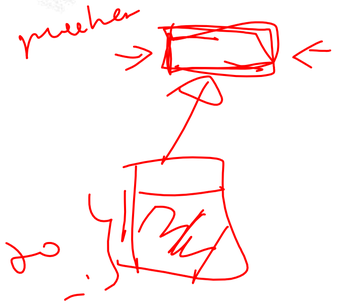


```
template<typename T>
class Vector {
    T* elem;     // pointer to first element
    T* space;    // pointer to first unused (and uninitialized) slot
    T* last;     // pointer to last slot
public:
    // ...
    int size();                 // number of elements (space-elem)
    int capacity();             // number of slots available for
elements (last-elem)
    // ...
    void reserve(int newsz);    // increase capacity() to newsz
    // ...
    void push_back(const T& t);     // copy t into Vector
    void push_back(T&& t);          // move t into Vector
};
```



```
template<typename T>
void Vector<T>::push_back(const T& t)
{
    if (capacity()<size()+1)        // make sure we have space for t
        reserve(size()==0?8:2*size()); // double the capacity
    new(space)T{t};                 // initialize*space to t
    ++space;
}
```

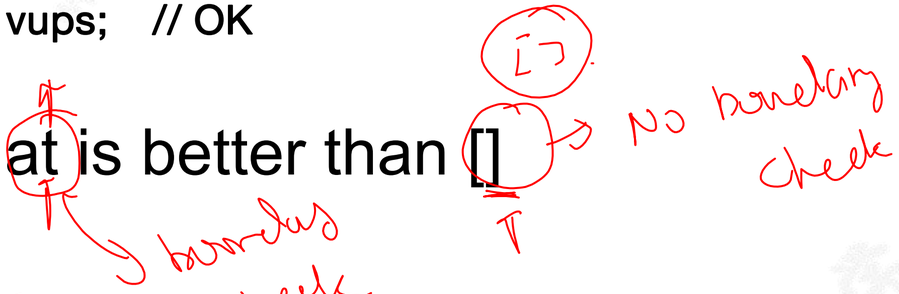
容器（Containers）



- 容器与多态的微妙关系

```
vector<Shape> vs;              // No, don't - there is no room for a Circle or a Smiley
vector<Shape*>vps;             // better
vector<unique_ptr<Shape>> vups;    // OK
```

- 边界：best practice: at is better than []



```
template<typename T>
class Vec : public std::vector<T> {
public:
    using vector<T>::vector;            // use the constructors from vector (under the name Vec)
    T& operator[](int i)                // range check
        { return vector<T>::at(i); }
    const T& operator[](int i) const    // range check const objects
        { return vector<T>::at(i); }
};
```

容器（Containers）

- std::list
- Doubly-linked

```
list<Entry>phone_book = {
    {"David Hume",123456},
    {"Karl Popper",234567},
    {"Bertrand Arthur William Russell",345678}
};
```

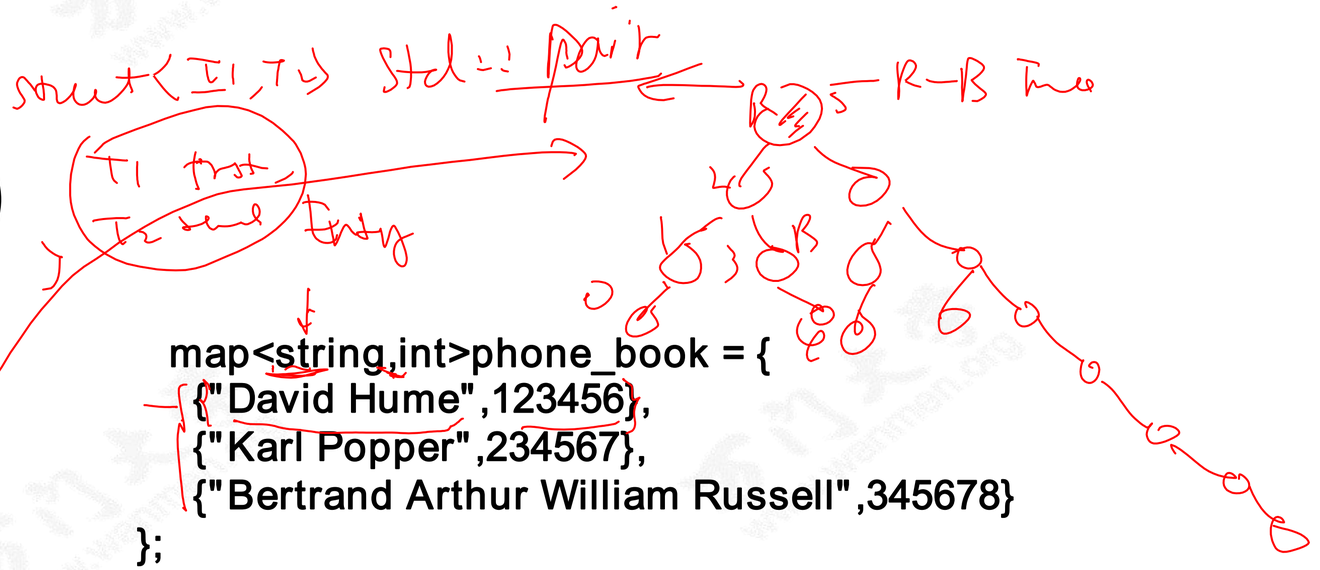
```
int get_number(const string& s)
{
    for (const auto& x : phone_book)
        if (x.name==s)
            return x.number;
    return 0;  // use 0 to represent "number not found"
}
```

```
void f(const Entry& ee, list<Entry>::iterator p, list<Entry>::iterator q)
{
    phone_book.insert(p,ee);      // add ee before the element referred to by p
    phone_book.erase(q);          // remove the element referred to by q
}
```

```
int get_number(const string& s)
{
    for (auto p = phone_book.begin(); p!=phone_book.end(); ++p)
        if (p->name==s)
            return p->number;
    return 0; // use 0 to represent "number not found"
}
```

容器（Containers）

- std::map
  - Key-value pair
  - RB-tree
- std::unordered_map
  - Hash table
- std::set
  - Just key, no value
- std::unordered_set
  - Hash table



```
map<string,int>phone_book = {
    {"David Hume",123456},
    {"Karl Popper",234567},
    {"Bertrand Arthur William Russell",345678}
};
```

```
unordered_  map<string,int>phone_book = {
    {"David Hume",123456},
    {"Karl Popper",234567},
    {"Bertrand Arthur William Russell",345678}
};
```

```
int get_number(const string& s)
{
    return phone_book[s];
}
```

算法（algorithms）

- unique_copy
- back_inserter

```
void f(vector<Entry>& vec, list<Entry>& lst)
{
    sort(vec.begin(),vec.end());           // use < for order
    unique_copy(vec.begin(),vec.end(),lst.begin());   // don't copy adjacent equal elements
}

bool operator<(const Entry& x, const Entry& y)     // less than
{
    return x.name<y.name;          // order Entrys by their names
}

list<Entry> f(vector<Entry>& vec)
{
    list<Entry> res;
    sort(vec.begin(),vec.end());
    unique_copy(vec.begin(),vec.end(),back_inserter(res));    // append to res
    return res;
}
```

算法（algorithms）

- find

```
bool has_c(const string& s, char c)    // does s contain the character c?
{
    auto p = find(s.begin(),s.end(),c);
    if (p!=s.end())
        return true;
    else
        return false;
}
```

```
bool has_c(const string& s, char c)      // does s contain the character c?
{
    return find(s.begin(),s.end(),c)!=s.end();
}
```

算法（algorithms)

- transform
- for_each

作业

- 明白std::map是如何初始化的，以glibc为例

作业

- 编写一个模板函数的find_all，通过以下测试用例

```
  void test()
{
    string m {"Mary had a little lamb"};
     for (auto p : find_all(m,'a'))           // p is a string::iterator
        if (*p!='a')
            cerr << "string bug!\n";
     list<double> ld {1.1, 2.2, 3.3, 1.1};
    for (auto p : find_all(ld,1.1))
        if (*p!=1.1)
            cerr << "list bug!\n";
     vector<string> vs { "red", "blue", "green", "green", "orange", "green" };
    for (auto p : find_all(vs,"red"))
        if (*p!="red")
            cerr << "vector bug!\n";
    for (auto p : find_all(vs,"green"))
        *p = "vert";
}
```

作业

- 编写一个模板函数sort(container)来对任何容器排序

谢谢观看

更多好课，请关注万门大学APP