C++ bitset 快速上手指南

一、bitset 简介

bitset 是一个固定大小的位集合，可以高效地进行位操作，常用于：

- 状态压缩
- 位标志管理
- 集合运算
- 布尔数组优化

二、创建 bitset 的三种方式

```cpp
#include <bitset>
#include <iostream>
using namespace std;

int main() {
    // 方式1: 指定大小，所有位初始化为0
    bitset<8> bs1;           // 00000000

    // 方式2: 从整数创建
    bitset<8> bs2(42);       // 00101010 (42的二进制)

    // 方式3: 从字符串创建
    bitset<8> bs3("10101010"); // 10101010
    bitset<8> bs4("11001100", 4); // 从字符串前4位: 1100

    return 0;
}
```

三、bitset 常用函数速查表

表格：bitset 核心函数一览

函数/操作	示例	说明	时间复杂度
访问与修改			
`[]` 操作符	`bs[3] = 1`	访问或修改第3位（从0开始）	O(1)
`test(pos)`	`bs.test(2)`	检查第2位是否为1，会检查越界	O(1)
`set()`	`bs.set()`	所有位置1	O(N)
`set(pos)`	`bs.set(3)`	第3位置1	O(1)
`reset()`	`bs.reset()`	所有位置0	O(N)
`reset(pos)`	`bs.reset(2)`	第2位置0	O(1)
`flip()`	`bs.flip()`	所有位取反	O(N)
`flip(pos)`	`bs.flip(1)`	第1位取反	O(1)
查询信息			
`size()`	`bs.size()`	返回bitset大小	O(1)
`count()`	`bs.count()`	返回1的个数	O(N/字长)
`any()`	`bs.any()`	是否有任意位为1	O(N/字长)
`none()`	`bs.none()`	是否所有位为0	O(N/字长)
`all()`	`bs.all()`	是否所有位为1	O(N/字长)
转换操作			
`to_ulong()`	`bs.to_ulong()`	转为unsigned long	O(N)
`to_ullong()`	`bs.to_ullong()`	转为unsigned long long	O(N)
`to_string()`	`bs.to_string()`	转为字符串	O(N)
位运算			
`&`, `` ` ``	`,` `^`, `~`	`bs1 & bs2`	与、或、异或、取反

函数/操作	示例	说明	时间复杂度
<< , >>	bs << 2	左移、右移	O(N)

四、完整示例代码

```cpp
#include <bitset>
#include <iostream>
#include <string>
using namespace std;

int main() {
    // 1. 创建和初始化
    bitset<8> bs1;                  // 00000000
    bitset<8> bs2(25);              // 00011001 (25的二进制)
    bitset<8> bs3("11110000");      // 11110000

    cout << "bs1: " << bs1 << endl;
    cout << "bs2: " << bs2 << endl;
    cout << "bs3: " << bs3 << endl;

    // 2. 基本操作
    bs1.set(0);                     // 第0位置1: 00000001
    bs1.set(2);                     // 第2位置1: 00000101
    bs1.reset(0);                   // 第0位置0: 00000100

    cout << "bs1修改后: " << bs1 << endl;

    // 3. 查询信息
    cout << "bs2中1的个数: " << bs2.count() << endl;
    cout << "bs2大小: " << bs2.size() << endl;
    cout << "bs2第3位: " << bs2.test(3) << endl;
    cout << "bs2是否有1: " << bs2.any() << endl;
    cout << "bs1是否全0: " << bs1.none() << endl;

    // 4. 位运算
    bitset<8> a("11001100");
    bitset<8> b("10101010");

    cout << "\n位运算示例:" << endl;
    cout << "a & b: " << (a & b) << endl;  // 与: 10001000
    cout << "a | b: " << (a | b) << endl;  // 或: 11101110
    cout << "a ^ b: " << (a ^ b) << endl;  // 异或: 01100110
    cout << "~a: " << (~a) << endl;        // 取反: 00110011
    cout << "a << 2: " << (a << 2) << endl;// 左移: 00110000
    cout << "a >> 2: " << (a >> 2) << endl;// 右移: 00110011
```

```
    // 5. 转换操作
    cout << "\n转换操作:" << endl;
    cout << "bs2转unsigned long: " << bs2.to_ulong() << endl;
    cout << "bs3转字符串: " << bs3.to_string() << endl;

    // 6. 实际应用：状态压缩
    cout << "\n状态压缩示例(权限管理):" << endl;
    enum Permissions {
        READ = 0,    // 第0位: 读权限
        WRITE = 1,   // 第1位: 写权限
        EXECUTE = 2  // 第2位: 执行权限
    };

    bitset<8> user1_perms(0);
    user1_perms.set(READ);
    user1_perms.set(WRITE);

    bitset<8> user2_perms(0);
    user2_perms.set(EXECUTE);

    cout << "用户1权限: " << user1_perms << " (读+写)" << endl;
    cout << "用户2权限: " << user2_perms << " (执行)" << endl;

    // 检查用户1是否有写权限
    if (user1_perms.test(WRITE)) {
        cout << "用户1有写权限" << endl;
    }

    // 给用户2添加读权限
    user2_perms.set(READ);
    cout << "用户2新权限: " << user2_perms << " (读+执行)" << endl;

    return 0;
}
```

五、运行结果示例

```
bs1: 00000000
bs2: 00011001
bs3: 11110000
bs1修改后: 00000100
bs2中1的个数: 3
bs2大小: 8
bs2第3位: 1
bs2是否有1: 1
bs1是否全0: 0

位运算示例:
a & b: 10001000
a | b: 11101110
a ^ b: 01100110
~a: 00110011
a << 2: 00110000
a >> 2: 00110011

转换操作:
bs2转unsigned long: 25
bs3转字符串: 11110000

状态压缩示例(权限管理):
用户1权限: 00000011 (读+写)
用户2权限: 00000100 (执行)
用户1有写权限
用户2新权限: 00000101 (读+执行)
```

六、重要注意事项

1. **大小固定**：bitset的大小在编译时确定，不能动态改变
2. **越界检查**：使用 `test()` 会检查越界，而 `[]` 操作符不会
3. **性能优势**：bitset比 `vector<bool>` 在大多数情况下更快
4. **存储效率**：通常每个元素只占1位（8个元素占1字节）