Simi Docs / CPlusPlus

Source: https://github.com/saidake/simi-docs/tree/release/simi-docs-1.6.2

Edited by: Craig Brown
Version: 1.0.1
Date: May 6, 2025

CPlusPlus / Concept

基础

```
#include <iostream>
using namespace std;
int main()
{
    cout<<"a"<<endl;
    return 0;
}
```

编译过程

预处理　将头文件写入包含的文件中　(test.**i**　)
编译　生成汇编代码　(test.**s**　)
汇编　生成二进制代码　(test.**o**　)
链接　链接库文件生成可执行文件　(test.**exe**)

标识符

由字母 下划线 数字组成　不能数字开头

头文件

afx.h　将各种 MFC 头文件包含在内
afxwin.h　包含了各种 MFC 窗口类。包含了 afx.h 和 windows.h。
afxext.h　提供了扩展窗口类的支持，例如工具栏，状态栏等。

CPlusPlus / Core

Constants and Variables

Integer (int)

Size:
    4 bytes (usually)
Range:
    -2,147,483,648 to 2,147,483,647
Example:

```
int a=999;          // Decimal (Scientific notation is Invalid for int)
int a=0b11001;      // Binary (prefix 0b, C++14+)
int a=-017;         // Octal (prefix 0)
int a=-0xD8;        // Hexadecimal (prefix 0x)
```

Short Integer (short)

Size:
    2 bytes
Range:
    -32,768 to 32,767
Example:

```
short a=999;          // Decimal (Scientific notation is Invalid for int)
short a=0b11001;      // Binary (prefix 0b, C++14+)
short a=-017;         // Octal (prefix 0)
short a=-0xD8;        // Hexadecimal (prefix 0x)
```

Long Integer (long)

Size:

4 or 8 bytes (platform-dependent)

Range:

4 bytes: Same as int

8 bytes: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Example:

```
long a=999;              // Decimal (Scientific notation is Invalid for int)
long a=999L;
long a=0b11001L;      // Binary (prefix 0b, C++14+)
long a=-017l;            // Octal (prefix 0)
long a=-0xD8L;          // Hexadecimal (prefix 0x)
```

Long Long Integer (long long)

Size:

8 bytes (guaranteed)

Range:

-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Example:

```
long long a = 9'223'372'036'854'775'807LL;   // Max value (C++14 digit separators)
```

Single Precision (float)

Size:

4 bytes

Range:

~±3.4e±38 (6-7 significant digits)

Example:

```
float a=3;
float a =.75f;
float a=3.88f;
float a=4.2F;
float a=2e-5f;
float a=4.4E-10F;
```

Double Precision (double)

Size:

8 bytes

Range:

~±1.7e±308 (15-16 significant digits)

Example:

```
double a=3;
double a=3.88;
double a=4.2d;
double a=2e-5D;
double a=4.4E-10;
```

Extended Precision (long double)

Size:
    10-16 bytes (platform-dependent)
Range:
    ~±1.1e±4932 (18-19 significant digits)
Example:
    long double a = 3.14159265358979323 85L;

Character (char)

Size:
    1 byte (typically)
Storage:
    ASCII values (0-127) or extended character sets
    ASCII Code: 48-56 (0-9) 65-90 (A-Z) 97-122 (a-z)
Example:

```
char a='x';              // Character literal
char a=68;               // ASCII value ('A')
char a= '\x41';          // Hexadecimal escape
char a='\u005d';         // Unicode escape (UTF-16)
```

Boolean Type (bool)

Size:
    Typically 1 byte
Values:
    true (1) or false (0)
Example:

```
bool a=false;
bool b=true;
```

Standard String (std::string)

Required Header:
    #include <string>
Size:
    Dynamic — grows as needed.
Example:

```
std::string a="John"
std::string a = "John\n"
                "Alice\n";    // Multiline strings, use backslash \ at line end for continuation or use \n for line breaks.
```

Array

Array names act as pointers to the first element.
Arrays cannot store mixed types.
Uninitialized elements may contain garbage values.

Example:

```
int a[] = {1, 2, 3};             // One-dimensional array with initial values, default value is '\0'
char a[3] = "fa";                // Character array with null terminator
char a[10], b[] = {"COPY"};   // Multiple assignment
int a[][3] = {{2, 3, 4}, {4, 6, 7}};   // 2D array
```

```
char b[2][3] = {{"fa"}, {"do"}};       // 2D char array
```

Calculation:

```
strcpy(a, b);   // Copies "COPY" into a
a[1] == &a[1][0]    // in 2D arrays — points to start of row 1 (i.e., a 1D array).
size_t length = sizeof(arr) / sizeof(arr[0]);   // Get array length
```

Constants

const variables must be initialized — no assignment allowed afterward.

Size:

Pointers: 4 or 8 bytes (platform-dependent)

References: Same size as the data they refer to

Example:

```
const double pi = 3.14;          // Constant: can only be assigned once
const int *p;                    // Pointer to const int (modifiable address, value read-only)
int *const p;                    // Const pointer to int (fixed address, value modifiable)
const int *const p;              // Const pointer to const int (fixed address, value read-only)
const int &ra = a;               // Const reference
```

References

References must be initialized at declaration.

Cannot be reseated to refer to another object.

Used to alias variables or arrays directly.

Example:

```
int &b = a;                      // Reference to variable a
const int &t = 12;               // Reference to a temporary value (read-only)
int (&p2)[3][3] = arr2;          // Reference to a 2D array
int *(&x) = po;                  // Reference to a pointer
```

Function Parameters with const and References

Const reference avoids copy but disallows modification.

Useful for large objects or temporary values.

Example:

```
void func(int &x);               // Modifiable reference
void func(const int a);          // Value passed by copy, not modifiable
void func(const int &a);         // Const reference, original cannot be changed
```

Temporary variable

Temporary variable, destroyed after the full expression.

Example:

```
int(999)
```

类型转换

1. 自动转换（低到高） short , char → int → long → double ← float (signed → unsigned)
2. 强制转换 (int)x 不影响原变量

运算符

一级 :: 作用域运算符

一级 [ ]数组下标 ()圆括号 . -> 获取成员 ++ --后缀自增自减

二级 -负号 !非 ~位取反 ++ --前缀自增自减 * & 地址取变量 取地址符 (int) 强制转换 sizeof new delete【同级右到左】

算术运算符 * / % > +-

左移右移 << >>

关系运算符 > >= < <= > == != 结果为 01 （0 为假，非 0 为真）

位运算符 & > ^ > |

逻辑运算符 && > || 结果为 01

条件运算符 a>b?x:y 【同级右到左】

赋值运算符 = *= /= %= += -= <<= >>= &= ^= |= 【同级右到左，左侧只能为变量】

逗号 ,（最后一个表达式的值为逗号表达式的值）

语句

if() if(){}elseif(){}else{} switch(){case value: code; break;} while(){} do{}while(); for(a;b;c;){} continue break

for(int i=0;;) 只能在 for 循环内使用的局部变量

```
for( int a = 10; a < 20; a = a + 1 )      // for  循环执行
{
      cout << "a  的值：" << a << endl;
}
```

CPlusPlus / Features

作用域

文件作用域 > 大括号内 大括号外

extern int x; （提前激活 本文件或其他 c 文件的变量/函数） 【extern void fa(); 等效于 void fa(); 】

激活变量需严格对应变量声明格式 char a[]="dog" extern char a[];

static int x; 存储在静态区（初值默认为 0） 声明和定义同时进行

全局静态变量只能在当前.c 文件上使用（激活也无效）

函数

```
void fa(int a,int b){          函数定义（不允许在函数中定义函数，不能在头文件中定义函数，只能为其他文件声明）
    struct  WriteFileContent {
        void operator()(int val);
    };
    WriteFileContent writeFileContent;
    writeFileContent(9);          //  函数内函数
}

fa( );                   函数调用
```

函数使用——

作用：提前激活 本文件或其他 c 文件的函数，主函数内外都可以，接受参数不能计算，只能传入参数计算

变量 int c 【 int int a 】

普通地址 int *p 【 int * int *a int a[] int [] int a[999] int [888] 】

指针数组地址 char *a[10] 【 int **a int *a[] int

*a[999] 】 一维数组形参大小随意

二维数组地址 int (*p)[10] 【 int (*)[10] int (*p)[10] int a[][10] int [][10] int a[999][10] int [888][10] 】

函数指针 int (*fun)(int, int) 【 int (*fun)(int, int) int (*fun)(int x, int y) 】 形式必须和函数完全相同 p=fun fun(a, b); (*fun)(a, b); p(a, b); (*p)(a, b);

引用 int &a 【 int & 】 形参为 const 或引用时 由实参初始化

指针引用 int *&a,int *&b 【 int *& 】 能通过指针修改原本值（引用地址赋值给其他地址，同样可修改原本值）

数组引用 **int (&a)[3]** 【 **int (&)[3]** 】 传入数组地址可改变数列值

void main(int argc, char **argv) argc（命令行输入参数 个数） argv（命令行输入参数 字符串） 从控制台得到值

argument count 变量个数 argument values 变量值

lambda 表达式——

[] 捕获列表为空。在函数内无法使用外部变量。

[a] 捕获列表为按值传递形式。在函数内仅能使用传递的变量值，无法改变变量。值在匿名函数生成时便已经确定，后续修改不会影响函数内的变量值。

[&a] 按应用传递。可改变变量。

&（以引用隐式捕获被使用的自动变量）

=（以复制隐式捕获被使用的自动变量）

```
#include<iostream>
using namespace std;
int main(){
    int x=1,y=2,z=0;
    auto add = [&z](auto x,auto y){z=x+y;return z;};
    auto res = add(x,y);
    cout<<res<<z<<endl;
}
```

可变参数使用——

```
void func(const initializer_list<CString> & currentList)          //C++11 引入了 initializer_list
{

    for(auto item = list.begin(); item != list.end(); ++it) {
        cout << *item << endl;
    }
    for(CString item : currentList)
    {
        cout << *item << endl;
```

```
    }
}
func({1,2,6,54,3,2,5});



int sum( int count , ....)
{
    if ( count <= 0){
        return 0 ;
    }
    va_list arg_ptr ;
    va_start(arg_ptr , count) ;        //初始化 arg_ptr，后面的参数是 函数定义的前方固定参数

    int CountSum = 0 ;
    for (int i = 0 ; i < count ; ++ i){
        CountSum += va_arg(arg_ptr , int) ;  //执行一次就取下一个参数
    }
    va_end(arg_ptr) ;                  //将 va_list 类型的指针复位成空值，就是清空可变参数列表
    return CountSum ;
```

类和对象

定义——

class Person : public Animal {　　定义类（派生类会析构 继承来的成员。 继承时自动调用一个 不用传参的基类构造函数，有参数会报错）

　　public　　全继承。　三种形式保持原样

　　private　　私有继承。全变为 private（子类内部不可调用）

　　protected　　保护继承。public 和 protected 变为 protected（子类内部可调用）

private:　　private 可省略 默认最上方为 private

public:

　　static int stc;　静态成员

　　int b=0;

　　Person(int c):stc(10), b(9) { }　构造函数（没有返回值，要为 public 对外可见，可以调用任意成员 ）

　　初始化列表会覆盖之前成员初始化的值

　　可重载 系统有默认一个无形参构造函数

　　~func( ){ }　　析构函数（不能重载　不能有形参　无返回值）

　　堆区的对象只有主动释放时才会调用析构函数

　　临时对象 func(12,2.33f); 作用域就在它所在的那条语句

　　临时变量：int c=int(23);　fuck b=fuck(12,2.33f);

　　virtual ~func(){}　　虚析构函数，实现多态的父类指针可以调用子类析构函数来释放（防止内存泄漏）

protected:

　　Dog lala;　　类成员能被初始化

　　virtual lala(){}　　虚函数（抽象类不能使用 new，子类可以使用 new）　【含虚函数的类为抽象类】

　　virtual lala()=0;　　纯虚函数（必须由子类实现）

　　void lala() override;　　表示重写了父类的虚函数（跳过重写检查）

friend class a;　友元类（友元类中可调用目标类的 private 和 protected 成员）

friend int b();　友元函数

enum{a, b, c}　　枚举

};

创建——

int Person::stc=0;　　public 静态成员可用类名访问 A::x=10

Person::Person(int c){ }　　构造函数写在类外

Person::~func(){ }　　析构函数写在类外

Person()　　创建临时对象，作用域就在它所在的那条语句

Person a(1);　　创建对象，栈中分配

Person a = Person(1);　　创建对象，栈中分配（临时变量，接收临时对象）

Person* a = new Person(1);　　创建对象，堆中分配

delete a;　　释放对象，并调用对象的析构函数

如果基类的析构函数是虚函数

如果 创建的对象是派生类的对象，会调用派生类的析构函数（赋值执行顺序：基类的构造函数 --> 派生类的构造函数 --> 派生类的析构函数 --> 基类的析构函数）

如果 创建的对象是基类的对象，　会调用基类的析构函数，这样就不会造成内存泄露（赋值执行顺序：基类的构造函数 --> 基类的析构函数）

如果基类的析构函数不是虚函数，根据指针的类型调用析构函数，这样就会造成内存泄露。

结构体

struct xx{};　　union xx{};

模板类型

template < class T >　　下方函数，类都可用此模板限制内部类型

CPlusPlus / Others

枚举

enum color { red, black, white }a,b,c;　　color x = red;（枚举值以序号数字存储）

类内 public 声明枚举　　类外访问枚举值 className::red;

内存申请

1. 申请　　int *p=new int(133);　　释放　delete p;　（可用括号初始化）

class 申请　　name *p=new name()　　释放　delete p; 【不可以释放多个】

数组申请　　int *p=new int[10];　　释放　delete[] p;

3. new delete 可以触发析构函数而 malloc calloc 不可以

预处理

#include<name.h>　直接去系统配置的库中寻找

#include"name.h"　先在项目当前目录寻找头文件　再去系统配置的库中寻找

#pragma once　　头文件只编译一次

#define　VV　　#define N 100　　标识符宏名，用 N 表示 100（替换过程中不计算，字符串中不包含宏）

#define　M(a, b, c)　a*b+c　　标识符表达式，先替换整个式子 再替换参数 abc

#undef　M　　清除宏名（不带参数）

命名空间

1.功能　区分同名变量或者函数　（C++标准库标识符都在 std 中）

2.定义　namespace lala{ 变量/函数/类/枚举　}

3. 用法　A.using namespace lala;　成员全部开放

B. lala::变量/函数　　成员未开放情况下 作用域运算符直接调用

C.using lala::成员　　开放特定成员

4.nemespace 的名字不能相同 （开放的成员名相同时可用 :: 区分成员）

位运算

数用补码存储，补码运算（有符号时，正数首位 0 负数首位 1）

1. 位与& a&b 全真则真
2. 位或| a|b 一真则真
3. 位异或^ a^b 不同为真
4. 位取反~ ~a 补码 0 变 1 1 变 0
5. 左移<< 溢出位移除 移入位 0
   右移>> 溢出位移除 【逻辑移位（移入位 0） 算术移位（移入原符号位）】

CPlusPlus / build-in libraries

输入输出

输出 cout<< 可以自动识别变量类型 cout<<"hello"<<' '<<12.33;
cout << 输入内容之间 可以任意空格回车 tab
endl 换行并且清空缓存区

输入 cin>> cin 可以自动识别变量类型 cin>>c>>a>>b; 可连续输入
cin 和> >和输入内容之间 可以任意空格回车 tab
cin.get(a 字符数组地址，20 接收字符个数) 接收一行字符串，可空格
cin.getline(a 字符数组地址，20 接收字符个数) 接收一个字符串

字符串

#include<string> 新增数据类型 string char *p= s.c_str() string 变为临时 c 字符串地址 【执行函数后 s 和返回的字符串 p 被释放掉】 【void func(const string& para)】

#include <cstring> C 语言的字符串函数 【可比较 char a[20]数组 】

1. strlen(a 地址) 字符个数不算'\0'
2. strcpy(a 地址, b 地址) b 覆盖 a （连'\0'也拷贝）
3. strcmp(a 地址, b 地址) 依次用 a 减 b（ASCII 字符） 结果不是零就自动结束 返回 1 或 0 或 -1
4. strcat(a 地址, b 地址) 把 b 连接到 a 上（默认覆盖'\0'之后的内容）

读写文件

#include <fstream>

fstream file("D:\\test.txt"，ios::out); 打开文件（file 对象） ios::in 读文件 【文件不存在则创建，传入 file 对象给函数时，要引用&传递】

ios::out 写文件 【文件不存在则创建，若文件已存在则清空原内容】

ios::ate 读文件 【文件打开时，指针在文件最后。可改变指针的位置】

ios::app 写文件，原内容后写入新内容 【文件不存在则创建】

file <<"xxxxx"<<end; 写入文件

getline(file, A) 读取 file 的一行存入 string A 中（文件指针自动下移，配合循环，无数据返回-1，有数据返回 1）

file.close(); 关闭文件

函数

#include <stdlib.h>

srand((unsigned) time(NULL)); 让随机数根据时间种子生成，程序每次运行产生的随机数都不同【#include<ctime> 使用 time() 函数】

rand() 返回一个非负随机数 rand()%(n-m+1)+m m<=x<=n【余数<被取余数】

数学

#include<math.h>

atoi() 将字符串转换为整数

floor(x)　<=x　的最大整数

ceil(x)　>=x　的最大整数

round(x)　四舍五入到最邻近的整数

mod(a,b)　返回 a%b　　// k mod 12==k%12

sqrt(x)　返回根号 x

CPlusPlus / MFC

控件事件

打开文件

```
void CsaidakemanageprojectDlg::OnBnClickedButton5()
{
	selectJavaFile(this, IDC_EDIT14);
}
void CsaidakemanageprojectDlg::selectJavaFile(CsaidakemanageprojectDlg* thisObj,int targetId) {
		// 设置过滤器
	TCHAR szFilter[] = _T("Java Files (*.java)|*.java");
	// 构造打开文件对话框
	CFileDialog fileDlg(TRUE, NULL, NULL, NULL, szFilter, thisObj);
	CString strFilePath;
	// 显示打开文件对话框
	if (IDOK == fileDlg.DoModal())
	{
		// 如果点击了文件对话框上的"打开"按钮，则将选择的文件路径显示到编辑框里
		strFilePath = fileDlg.GetPathName();
		SetDlgItemText(targetId,strFilePath);
	}
}
```

打开文件夹

```
void CsaidakemanageprojectDlg::OnBnClickedButton1()
{
	// set folder dialog
	CFolderPickerDialog folderPickerDialog(NULL, OFN_FILEMUSTEXIST | OFN_ALLOWMULTISELECT | OFN_ENABLESIZING, this,
sizeof(OPENFILENAME));
	CString folderPath;
	if (folderPickerDialog.DoModal() == IDOK)
	{
		folderPath = folderPickerDialog.GetPathName();
		SetDlgItemText(IDC_EDIT1, folderPath);
	}
	// set folder dialog
}
```

改变静态文本文字

```
HBRUSH CYourDlg::OnCtlColor(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
HBRUSH hbr = CDialog::OnCtlColor(pDC, pWnd, nCtlColor);

 // TODO: Change any attributes of the DC here
 if (pWnd->GetDlgCtrlID() == IDC_STATICText)
 {

  pDC->SetBkColor(RGB(0,255,0));//背景色为绿色
  pDC->SetTextColor(RGB(255, 0, 0));//文字为红色
  pDC->SelectObject(&m_font);//文字为 15 号字体，华文行楷
```

```
  return m_brush;
 }

 // TODO: Return a different brush if the default is not desired
 return hbr;

}
```

ado 数据库连接

环境准备 pch.h

```
#ifndef PCH_H
#define PCH_H
// add headers that you want to pre-compile here
#include "framework.h"

#include<odbcinst.h>
#include"afxdb.h"
#import "C:\\Program Files\\Common Files\\System\\ado\\msado15.dll" no_namespace rename("EOF","adoEOF")

#endif //PCH_H
```

连接数据库

```
	//============================================================= connect db
	_ConnectionPtr m_pConnection;
	try {
		// 创建Connection对象
		m_pConnection.CreateInstance("ADODB.Connection");
		// 设置连接字符串，必须是BSTR型或者_bstr_t类型
		_bstr_t strConnect = _bstr_t("Driver=MySQL ODBC 8.0 Unicode
Driver;SERVER=127.0.0.1;UID=root;PWD=root;DATABASE=boss;PORT=3306;");//"Provider=SQLOLEDB;
Server=127.0.0.1;Database=EventLogg; uid=event; pwd=event;";
		m_pConnection->ConnectionTimeout = 8;
		//m_pConnection->Open(strConnect,   _T("root"), _T("root"), adModeUnknown);
		m_pConnection->Open(strConnect, "", "", adModeUnknown);

	}
	catch (_com_error e) {
		AfxMessageBox(L"connection error");
		return;
	}
	//============================================================= search db
	try {
		//添加一个指向Recordset对象的指针:
		_RecordsetPtr m_pRecordset;
		// 创建记录集对象
		m_pRecordset.CreateInstance(__uuidof(Recordset));
		// 取得表中的记录
		m_pRecordset->Open(_bstr_t("SELECT table_name FROM information_schema.TABLES WHERE table_schema='boss';"),
m_pConnection.GetInterfacePtr(), adOpenDynamic, adLockOptimistic, adCmdText);

		wprintf(L"tset%d", m_pRecordset->Fields->GetCount());

		m_pRecordset->Close(); // 关表
```

```
        m_pConnection->Close(); // 关数据库

    }
    catch (_com_error e) {
        AfxMessageBox(L"search error");
        return;
    }
```

操作数据

```
db.m_pRecordset->AddNew();              //添加新纪录
//在表中名为“序号”那一列添加一行数据为“2”的内容
db.m_pRecordset->PutCollect(L"序号", _variant_t(2));
//在同一行的“类型”那一列，插入名为“高度”的内容
db.m_pRecordset->PutCollect(L"类型", _variant_t(“高度”));
 //将更改的内容更新，可理解成将更改写入到文件中
db.m_pRecordset->Update();
//关闭记录集
db.m_pRecordset->Close();


m_pRecordset->Fields->GetCount();




        CString lpDest;
        VARIANT    vt;
        vt = m_pRecordset->GetCollect("user_phone");
        if (vt.vt != VT_NULL)
            lpDest = (LPCSTR)_bstr_t(vt);
        else
            lpDest = "";
```

测试消息

```
                CString testMsg = L"test msg: ";
                int test = 0;
                //testMsg.Format(L"%s", serviceWriteStr.Mid(0, serviceWriteStr.Find(L"fafafafaff2332\n", 0)));
                testMsg.Format(L"%d", next);
                AfxMessageBox(testMsg);
                return;
```

写入宽字节

```
            DWORD dwFileLen = writeServiceFile->GetLength();
            LPCTSTR content = currentWriteContent;
            if (0 == dwFileLen)
            {
                const unsigned char LeadBytes[] = { 0xEF, 0xBB, 0xBF };
                writeServiceFile->Write(LeadBytes, sizeof(LeadBytes));
            }
            int nSrcLen = (int)wcslen(content);//content为要写入的文本
            CStringA utf8String(content);
            int nBufLen = (nSrcLen + 1) * 6;
            LPSTR buffer = utf8String.GetBufferSetLength(nBufLen);
            int nLen = AtlUnicodeToUTF8(content, nSrcLen, buffer, nBufLen);
            //上面的函数AtlUnicodeToUTF8()需头文件：<atlenc.h>
            //功能：将unicode转换成utf-8
            buffer[nLen] = 0;
            utf8String.ReleaseBuffer();
```

```
        writeServiceFile->SeekToEnd();
        //将转码后的内容写入mht文件
        writeServiceFile->Write(utf8String.GetBuffer(), nLen);
```

设定区域

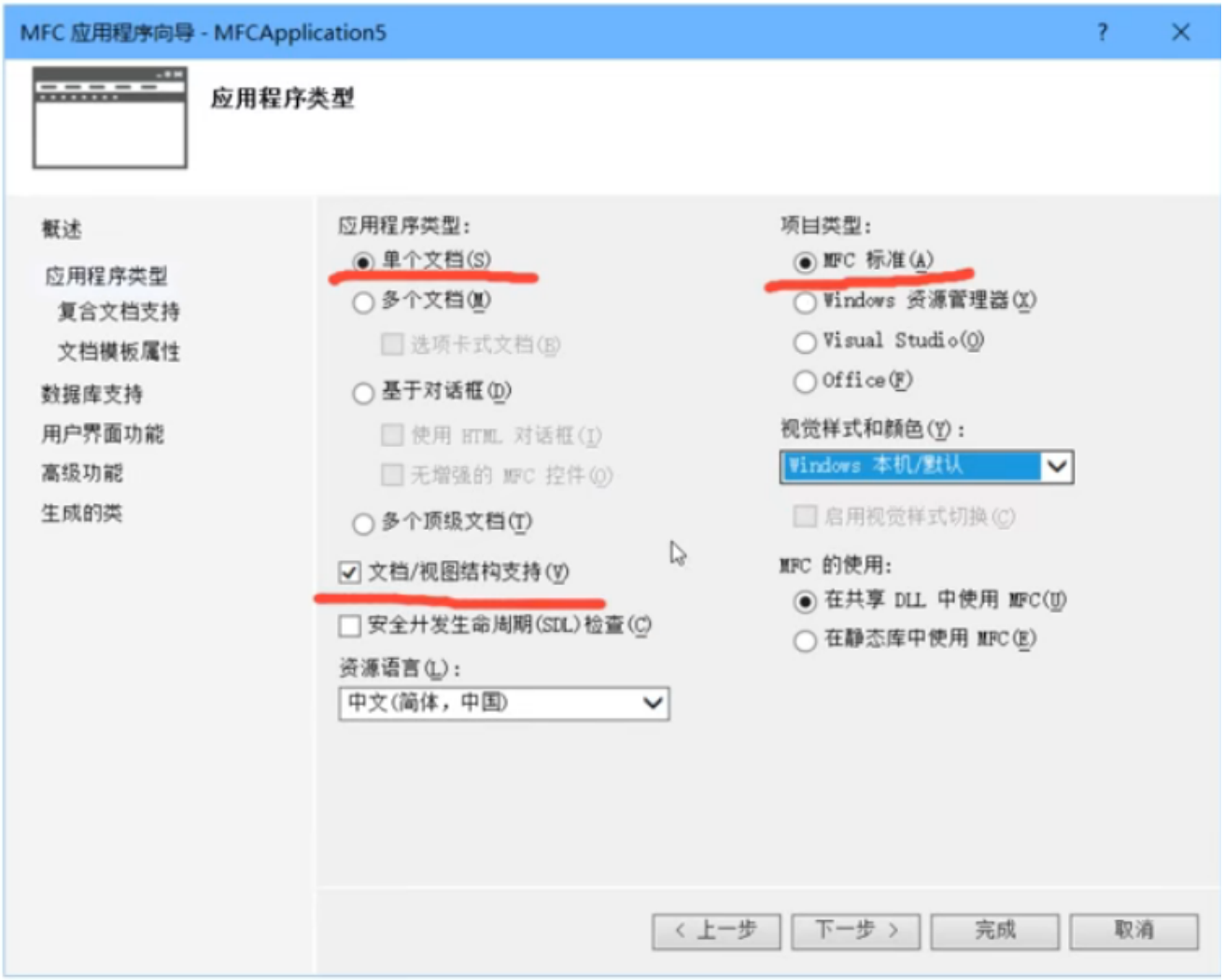
```
char* old_locale = _strdup(setlocale(LC_CTYPE, NULL));
    _tsetlocale(LC_CTYPE, _T("chs"));

setlocale(LC_CTYPE, old_locale);
    free(old_locale);//还原区域设定
```

CPlusPlus / MFC source code

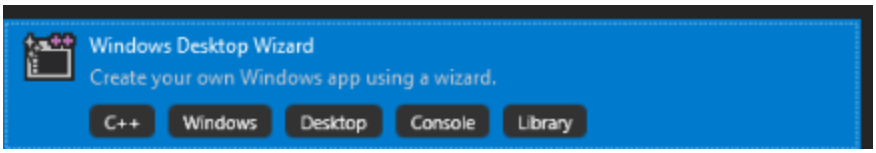
项目开始

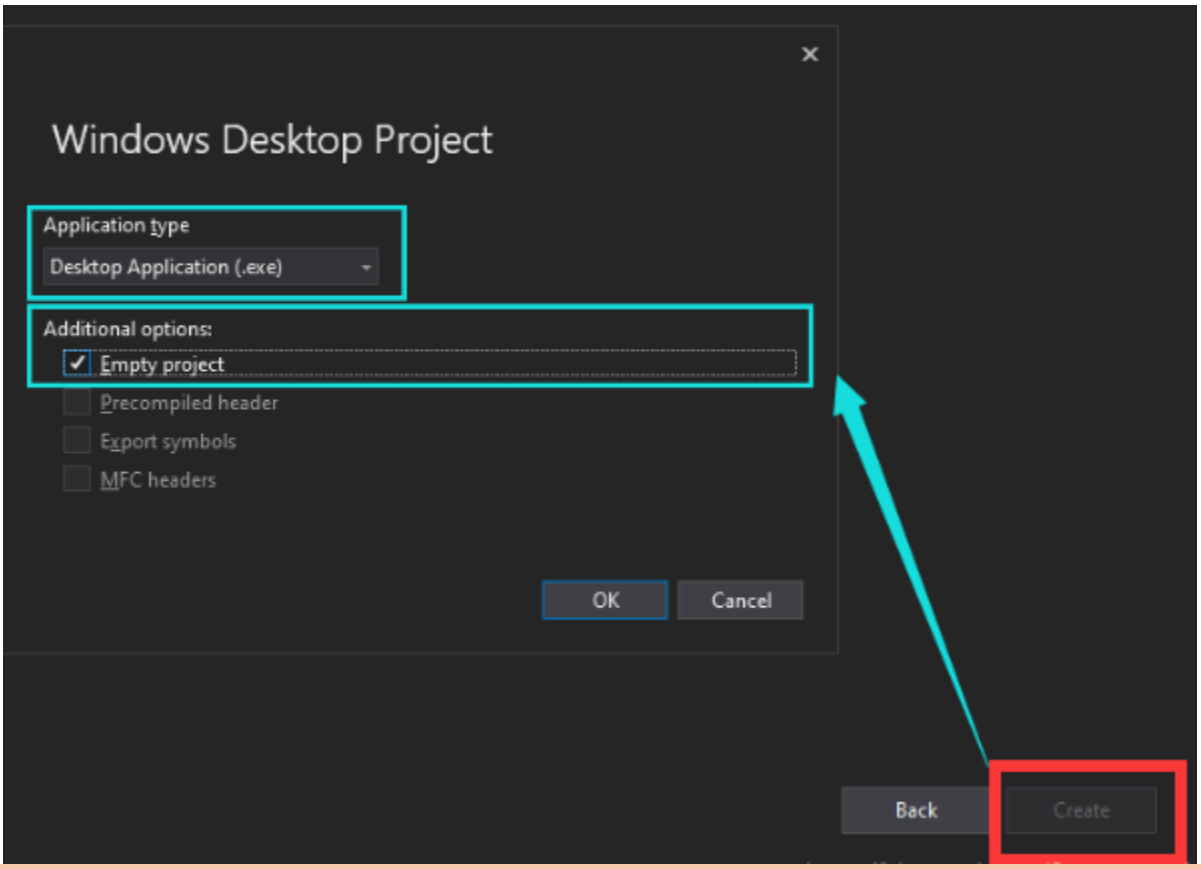
项目创建

MFC 程序：单文档



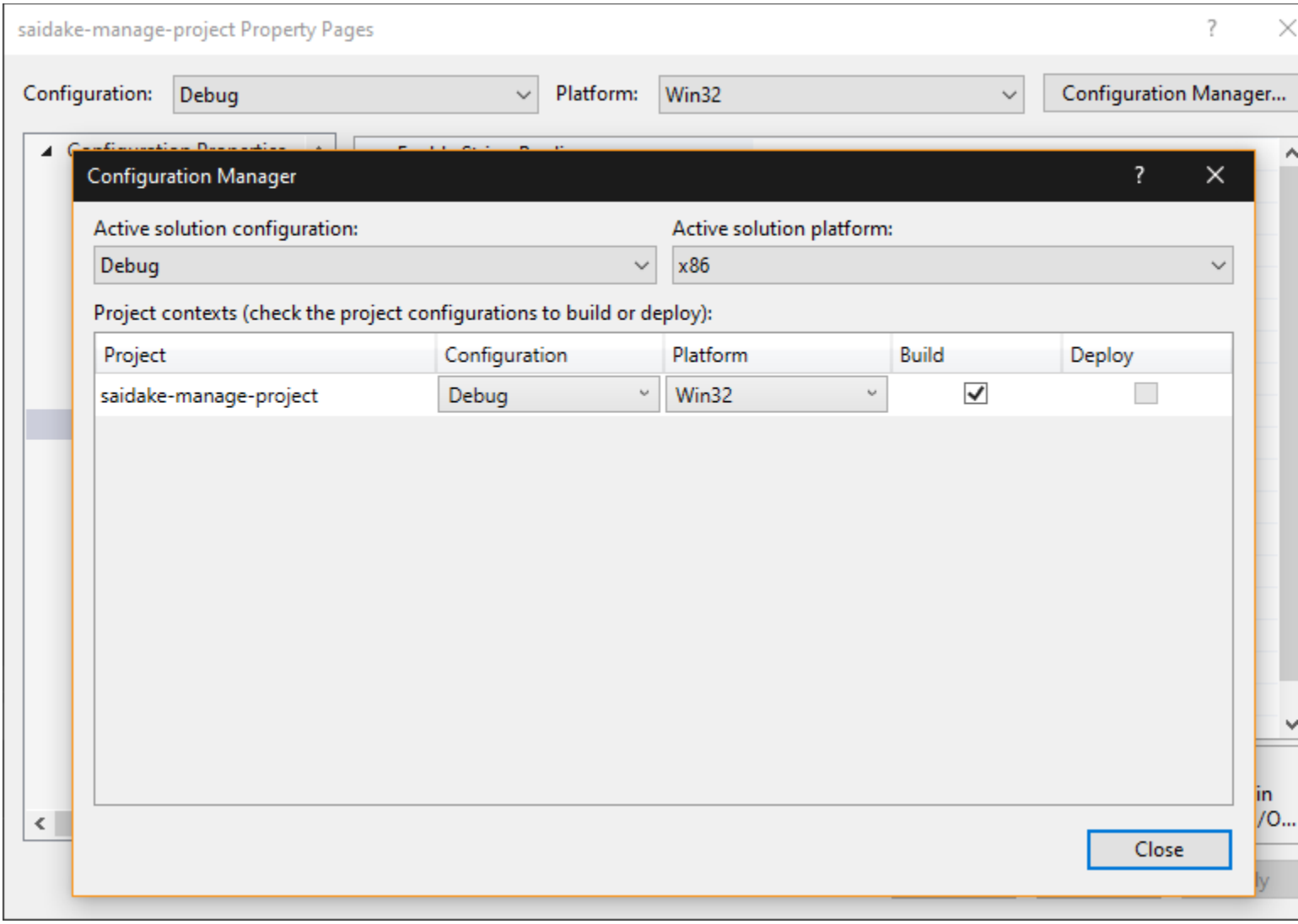
win32 空项目：



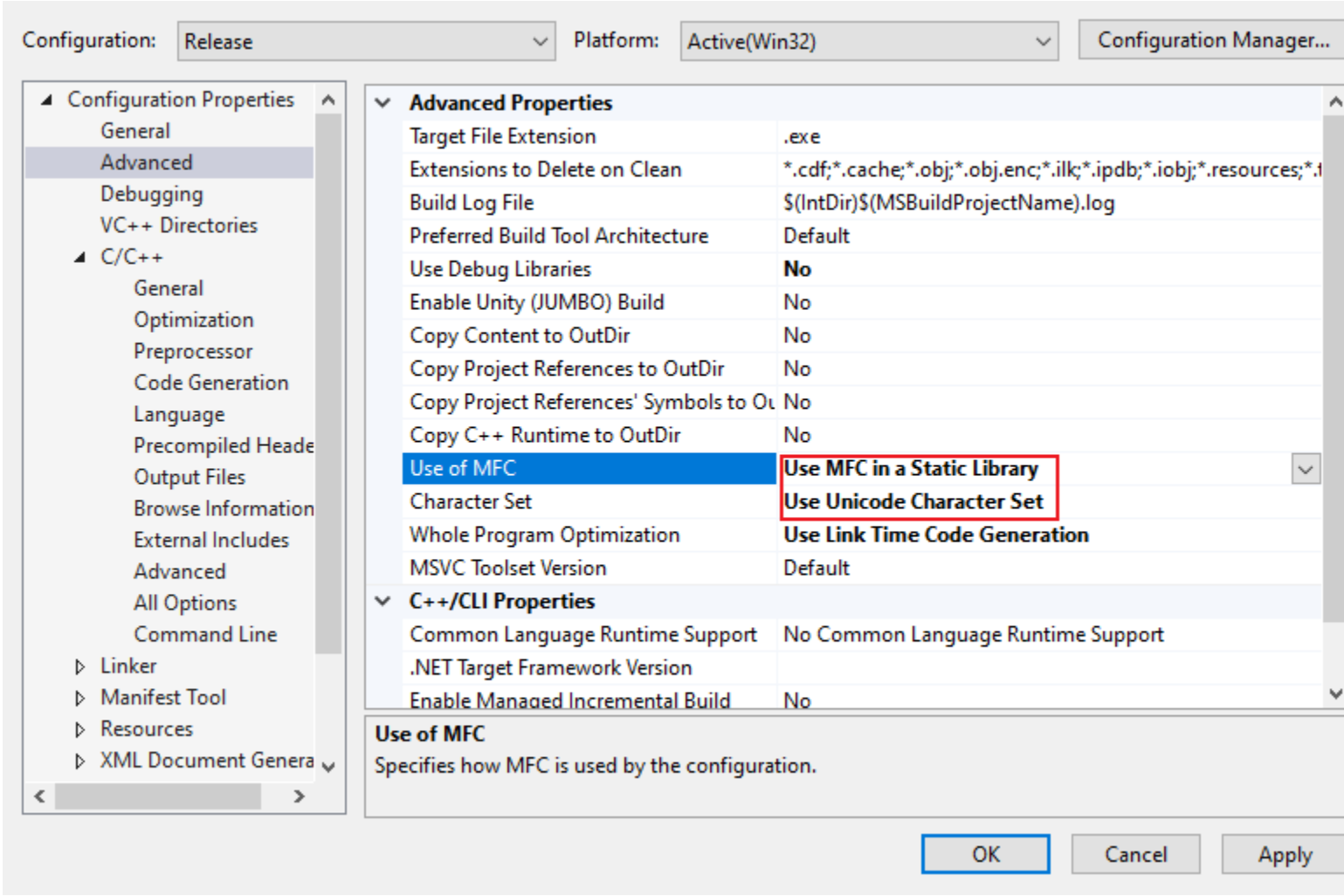

Windows Desktop Project
Application type
Desktop Application (.exe)
Additional options:
Empty project
Precompiled header
Export symbols
MFC headers
OK
Cancel
Back
Create

项目初始化

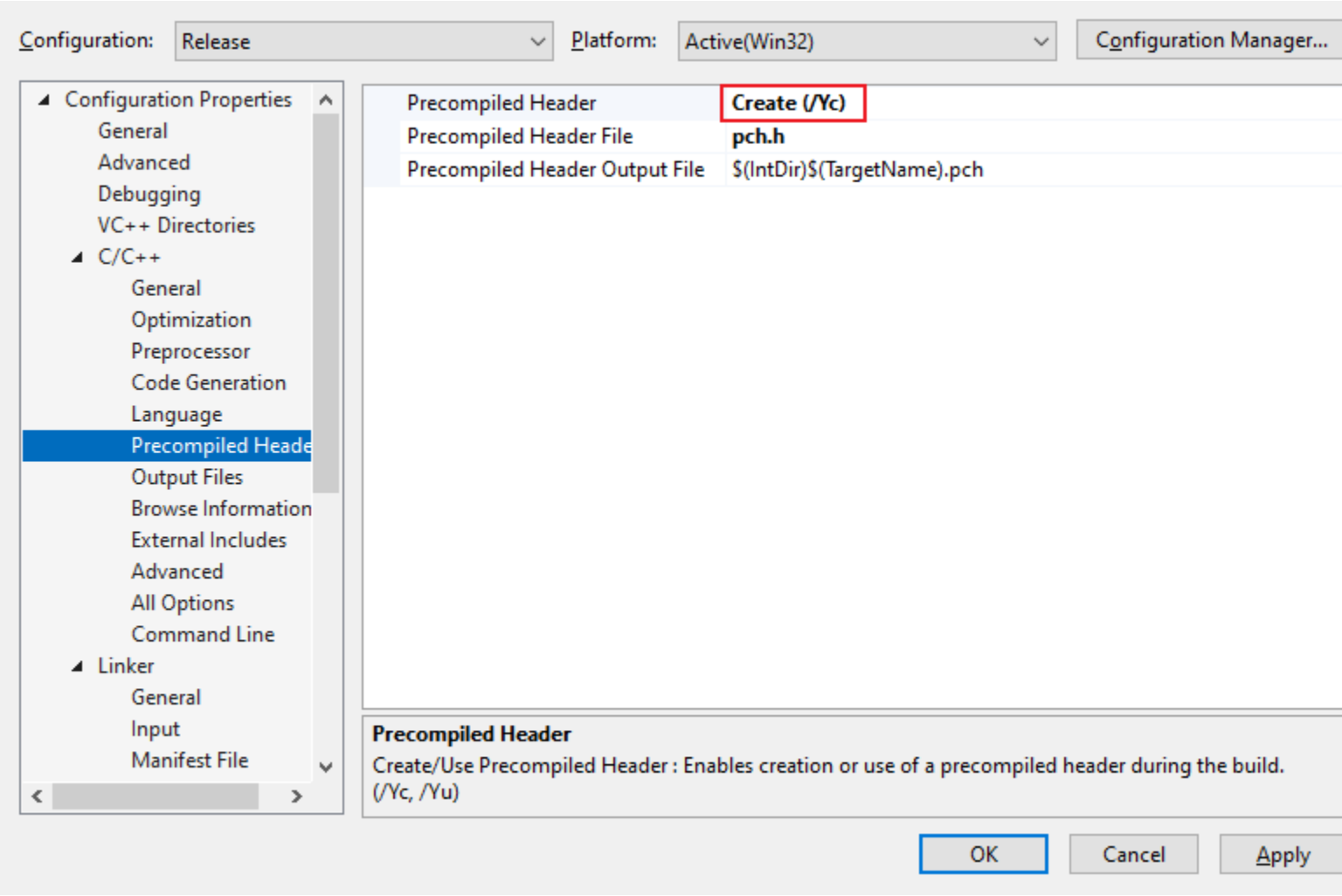
编译配置

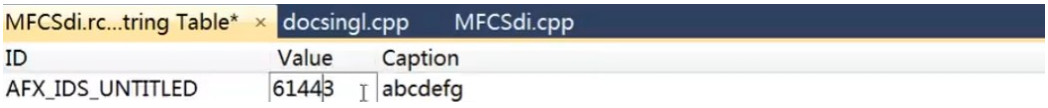

saidake-manage-project Property Pages
Configuration:
Debug
Platform:
Win32
Configuration Manager...
Configuration Manager
Active solution configuration:
Debug
Active solution platform:
x86
Project contexts (check the project configurations to build or deploy):
Project
Configuration
Platform
Build
Deploy
saidake-manage-project
Debug
Win32
Close

使用静态库


Configuration: Release
Platform: Active(Win32)
Configuration Manager...
Configuration Properties
General
Advanced
Debugging
VC++ Directories
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Heade
Output Files
Browse Information
External Includes
Advanced
All Options
Command Line
Linker
Manifest Tool
Resources
XML Document Genera
Advanced Properties
Target File Extension .exe
Extensions to Delete on Clean *.cdf;*.cache;*.obj;*.obj.enc;*.ilk;*.ipdb;*.iobj;*.resources;*.
Build Log File $(IntDir)$(MSBuildProjectName).log
Preferred Build Tool Architecture Default
Use Debug Libraries No
Enable Unity (JUMBO) Build No
Copy Content to OutDir No
Copy Project References to OutDir No
Copy Project References' Symbols to Ou No
Copy C++ Runtime to OutDir No
Use of MFC Use MFC in a Static Library
Character Set Use Unicode Character Set
Whole Program Optimization Use Link Time Code Generation
MSVC Toolset Version Default
C++/CLI Properties
Common Language Runtime Support No Common Language Runtime Support
.NET Target Framework Version
Enable Managed Incremental Build No
Use of MFC
Specifies how MFC is used by the configuration.
OK
Cancel
Apply

预编译设置



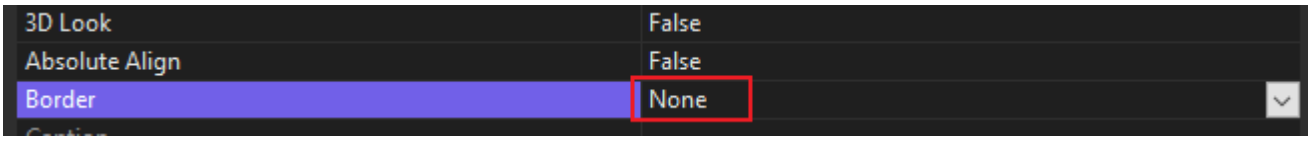


去掉 untitled

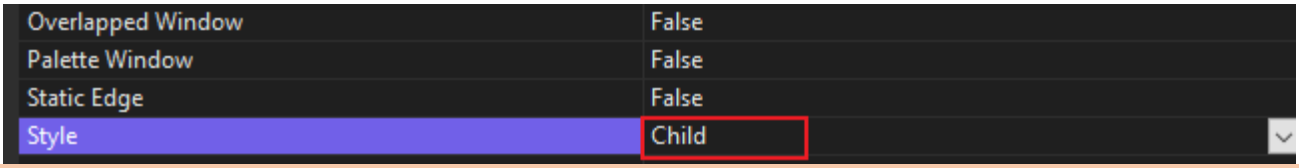
```
void CMainFrame::OnUpdateFrameTitle(BOOL bAddToTitle)
{
	CFrameWnd::OnUpdateFrameTitle(false);
}
void CsaidakemanageprojectDoc::SetTitle(LPCTSTR lpszTitle)	// 去掉untitled
{
	CDocument::SetTitle(_T(""));
}
```

对话框

去掉边框

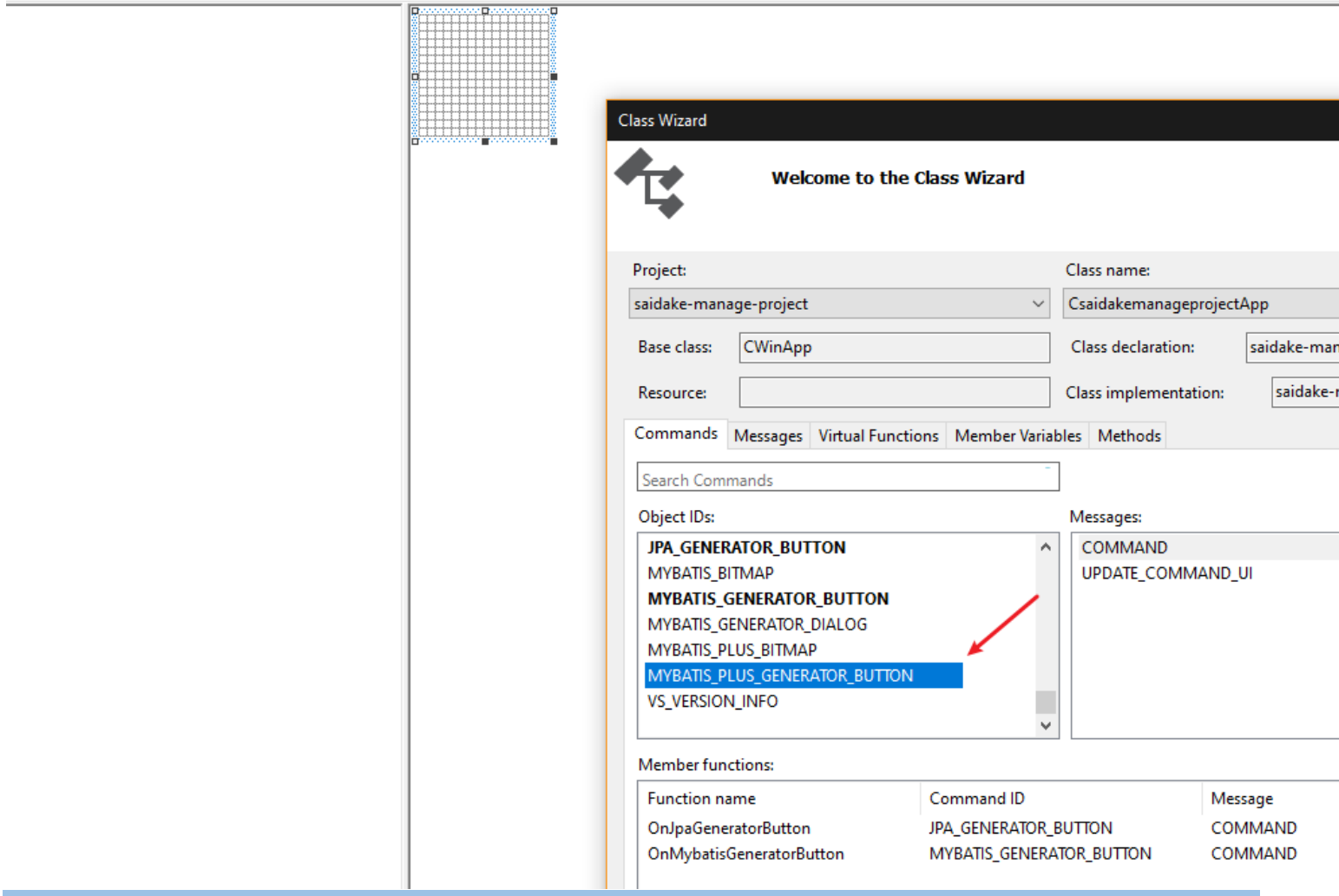


设置子窗口



工具栏

添加事件



全局函数

全局

AfxMessageBox("消息")　　弹框消息

GetClientRect(&rect);　　获取当前窗口大小

CWinApp* AFXAPI AfxGetApp()　　获取 winapp

DECLARE_DYNCREATE(CsaidakemanageprojectFrame);　　动态声明类（类内）

IMPLEMENT_DYNCREATE(MybatisGeneratorView, CView);　动态实现类（类外）

#define WM_MYMESSAGE WM_USER+1001　　自定义事件

BEGIN_MESSAGE_MAP(CsaidakemanageprojectApp, CWinApp)　　开始定义内部消息（类外）

　　ON_COMMAND(ID_FILE_NEW, &CWinApp::OnFileNew)　　定义一个菜单 COMMAND 消息

　　ON_MESSAGE(WM_CREATE,OnCreate)　　通用消息监听【WM_CREATE 事件类型，OnCreate 处理消息函数】

ON_MESSAGE(WM_MYMESSAGE, OnMyMessage)　　自定义消息
ON_WM_INITMENUPOPUP()　　wm 封装消息
int OnCreate(LPCREATESTRUCT pcs);　　专用消息监听 处理函数
END_MESSAGE_MAP()　　结束定义内部消息（类外）

基础类型
vector

vector<int> a(10)　　定义了 10 个整型元素的向量（尖括号中为元素类型名，它可以是任何合法的数据类型），但没有给出初值，其值是不确定的。
vector<int> a(10,1)　　定义了 10 个整型元素的向量,且给出每个元素的初值为 1
vector<int> a(b);　　用 b 向量来创建 a 向量，整体复制性赋值
vector<int> a(b.begin(),b.begin+3)　　定义了 a 值为 b 中第 0 个到第 2 个（共 3 个）元素
int b[7]={1,2,3,4,5,9,8};　　vector<int> a(b,b+7);　　从数组中获得初值

a.assign(b.begin(), b.begin()+3);　　b 为向量，将 b 的 0~2 个元素构成的向量赋给 a
a.assign(4,2);　　是 a 只含 4 个元素，且每个元素为 2　　// vector< vector<CString> > Array(10, vector<CString>(0));
a.back();　　返回 a 的最后一个元素
a.front();　　返回 a 的第一个元素
a[i];　　返回 a 的第 i 个元素，当且仅当 a[i]存在 2013-12-07
a.clear();　　清空 a 中的元素
a.empty();　　判断 a 是否为空，空则返回 ture,不空则返回 false
a.pop_back();　　删除 a 向量的最后一个元素
a.erase(a.begin()+1,a.begin()+3);　　删除 a 中第 1 个（从第 0 个算起）到第 2 个元素，也就是说删除的元素从 a.begin()+1 算起（包括它）一直到 a.begin()+　　3（不包括它）
a.push_back(5);　　在 a 的最后一个向量后插入一个元素，其值为 5
a.insert(a.begin()+1,5);　　在 a 的第 1 个元素（从第 0 个算起）的位置插入数值 5，如 a 为 1,2,3,4，插入元素后为 1,5,2,3,4
a.insert(a.begin()+1,3,5);　　在 a 的第 1 个元素（从第 0 个算起）的位置插入 3 个数，其值都为 5
a.insert(a.begin()+1,b+3,b+6);　　b 为数组，在 a 的第 1 个元素（从第 0 个算起）的位置插入 b 的第 3 个元素到第 5 个元素（不包括 b+6），如 b 为 1,2,3,4,5,9,8　　，插入元素后为 1,4,5,9,2,3,4,5,9,8
a.size();　　返回 a 中元素的个数;
a.capacity();　　返回 a 在内存中总共可以容纳的元素个数
a.resize(10);　　将 a 的现有元素个数调至 10 个，多则删，少则补，其值随机
a.resize(10,2);　　将 a 的现有元素个数调至 10 个，多则删，少则补，其值为 2
a.reserve(100);　　将 a 的容量（capacity）扩充至 100，也就是说现在测试 a.capacity();的时候返回值是 100. 这种操作只有在需要给 a 添加大量数据的时候才显得有意义，
因为这将避免内存多次容量扩充操作（当 a 的容量不足时电脑会自动扩容，当然这必然降低性能）
a.swap(b);　　b 为向量，将 a 中的元素和 b 中的元素进行整体性交换
a==b;　　b 为向量，向量的比较操作还有!=,>=,<=,>,<

```
for(int i=0;i<obj.size();i++)
{
    cout<<obj[i]<<",";
```

```
}


std::vector<std::vector<CString>> tableData;
for(int r = 0; r < oTA.rows; r++)
{
    tableData.push_back(std::vector<CString>());
    for(int c = 0; c < oTA.cols; c++)
        tableData.back().push_back("Test");
}


std::vector<std::vector<CString>> tableData(oTA.rows,std::vector<CString>(oTA.cols));
for(int r = 0; r < oTA.rows; r++)
    for(int c = 0; c < oTA.cols; c++)
        tableData[r][c]="Test";
```

map

```
insert(make_pair(90,"hi"));


map<int,map<int,string> >multiMap; //对于这样的 map 嵌套定义，有两种插入方法：

map<int, string> temp;   //定义一个 map<int, string>变量，对其定义后在插入 multiMap

temp.insert(make_pair(90,"hi"));

temp.insert(pair<int,string>(100,"maxi")); //pair<int,string>()和 make_pair()有相同作用

multiMap.insert(make_pair(10, temp));    //将临时变量插入到 multiMap 中

m_map.erase(m_map.begin(), m_map.end());    清空 map

multiMap[10][80]="xiaoyu"; //可以直接赋值
mulitMap[5][30]="xiaoma";


    map<int, string>::reverse_iterator    iter;
     for(iter = mapStudent.rbegin(); iter != mapStudent.rend(); iter++){
          cout<<iter->first<<" "<<iter->second<<endl;
     }
```

CString

```
CString                                    给 CString 初始时，会调用 MultiByteToWideChar
CString Left( int nCount ) const           表示从左边 1 开始获取前  nCount  个字符
int Find(
          _In_z_ PCXSTR pszSub,                    查找字符串   0 1 2 3 4
          _In_ int iStart = 0) const throw()        开始索引
CStringT Mid(                                截取子串    // Mid(2,3)   "abcdefg"      "cde"    [2, 5)
          _In_ int iFirst,                         开始索引
```

_In_ int nCount) const　　字符个数

BOOL LoadString(_In_ UINT nID)　　加载 stringTable 资源

格式化//Format("my name is %6s","wind");

CStdioFile

virtual ULONGLONG Seek(LONGLONG lOff, UINT nFrom);　向指定位置移动 // Seek(-sizeof(int),CFile::end);　表示从文件末尾向前移动 8 个字节，指向相对于文件开始位置 18 个字节偏移量的位置

CRect

```
inline CRect::CRect(
        _In_ int l,                指定矩形框左上角的 X 坐标
        _In_ int t,                指定矩形框左上角的 y 坐标
        _In_ int r,                指定矩形框右下角的 x 坐标
        _In_ int b) throw()        指定矩形框右下角的 y 坐标
{
        left = l;
        top = t;
        right = r;
        bottom = b;
}
```

CImageList

图片列表

int CImageList::Add(CBitmap* pbmImage, COLORREF crMask)　添加图片

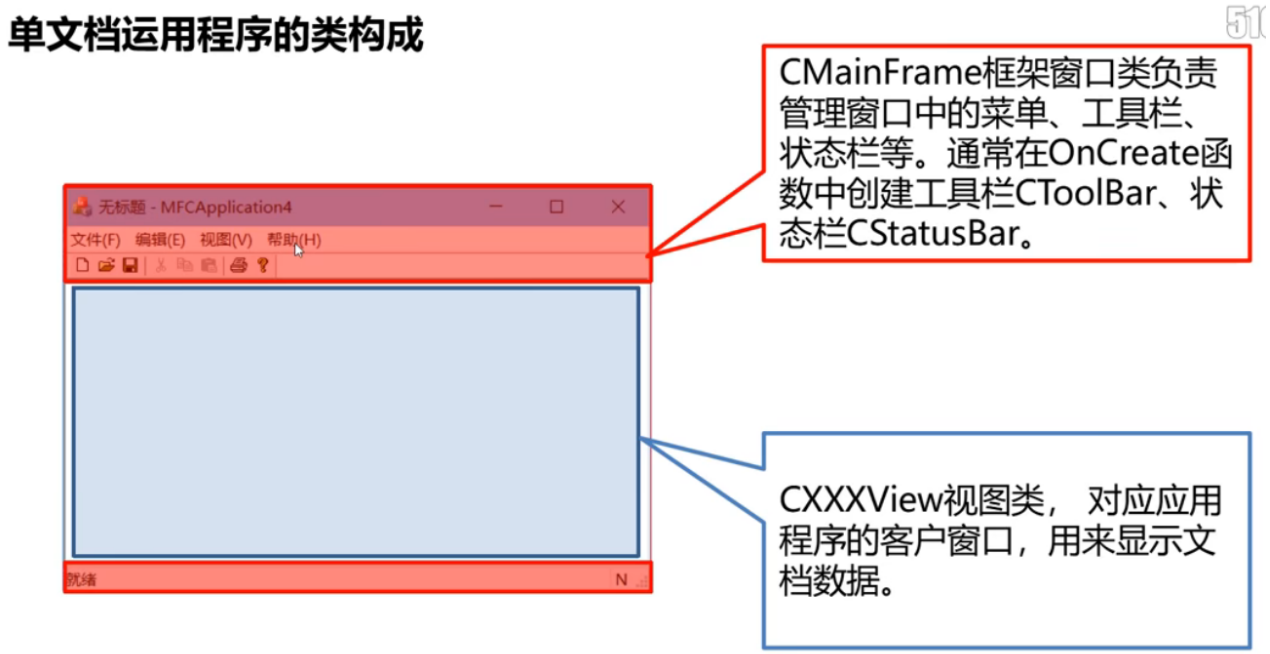
框架类型

CWinApp

HICON LoadIcon(UINT nIDResource) const;　　加载 icon 资源

CFrameWnd

用于 SDI 应用程序的框架窗口，SDI 框架窗口既是应用程序的主框架窗口，也是当前文档对应的视图的边框。



CDocument

文档是 MFC 的 CDocument 类的派生类，他主要负责应用程序数据的保存和装载，实现文档的序列化功能。

一个文档可以对应多个视图，所以文档类 CDocument 使用成员变量 CPtrListm viewList;来保存多个视图。文档类内容被修改后，一般调

用虚函数 SetModifiedFlag()设定文档是否修改标志，调用 UpdateAllViews()函数则刷新和此文档关联的所有视图 lb

CView

视图类

CView:OnInitialUpdate 在 CView 类第一次构造后调用，负责 View 的初始化。

CView:OnUpdate 当框架调用此函数时，表示 Document 的内容已经发生了变化

CView::OnDraw 由 CView::OnPaint()函数调用

CDialog

BOOL CDialog::Create(UINT nIDTemplate, CWnd* pParentWnd) 创建一个对话框

BOOL SetWindowPos(改变窗口的大小、位置和设置子窗口、弹出窗口或顶层窗口的排列顺序

```
    const CWnd* pWndInsertAfter,   //排列顺序的句柄
    int x,          //水平坐标
    int y,           //垂直坐标
    int cx,         //宽
    int cy,         //高
    UINT nFlags)     //窗口定位标识
        SWP_NOMOVE     不发生移动   // SWP_NOMOVE||SWP_NOSIZE
        SWP_NOSIZE     不改变大小
```

BOOL ShowWindow(int nCmdShow) 显示窗口

```
        SW_SHOW        显示
```

ado 数据库操作

创建连接

```
    // 创建Connection对象
    m_pConnection.CreateInstance("ADODB.Connection");
    // 设置连接字符串，必须是BSTR型或者_bstr_t类型
    _bstr_t strConnect = _bstr_t("Driver=MySQL ODBC 8.0 Unicode
Driver;SERVER=127.0.0.1;UID=root;PWD=root;DATABASE=boss;PORT=3306;");
    m_pConnection->ConnectionTimeout = 8;
    //m_pConnection->Open(strConnect,   _T("root"), _T("root"), adModeUnknown);
    m_pConnection->Open(strConnect, "", "", adModeUnknown);
```

开始查询

```
_RecordsetPtr tableNameRecordList;
tableNameRecordList.CreateInstance(__uuidof(Recordset));
tableNameRecordList->Open(_bstr_t("SELECT table_name FROM information_schema.TABLES WHERE table_schema='boss';"),
m_pConnection.GetInterfacePtr(), adOpenDynamic, adLockOptimistic, adCmdText);
int total = tableNameRecordList->Fields->GetCount();
```

windows.h

```
#include <windows.h>

if (CreateDirectory("folder_name", NULL)) {
    // Directory created
}
else if (ERROR_ALREADY_EXISTS == GetLastError()) {
    // Directory already exists
}
else {
    // Failed for some other reason
}
```

```
RemoveDirectory("folder_name");
```