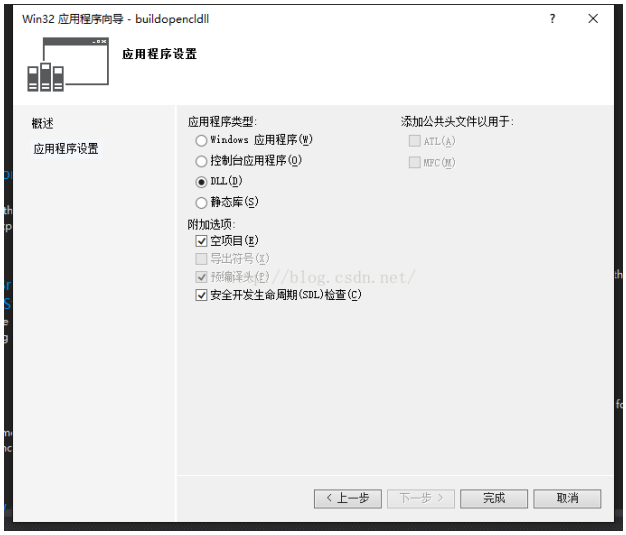
如何将算法的代码封装成动态链接库

注: dll建立和调用的两个example project在

http://192.168.200.100:8000/svn/2017-Pipline-Detection/doc/Algorithm Encapsulation

1. 创建dll工程

新建一个Win32控制台应用程序，项目名称为test_dll，选择DLL和空项目。



2. 在原有代码之外封装接口类(ikperson.h)

```
#ifndef _IKPERSON_H_
#define _IKPERSON_H_
#ifdef DLL_EXPORT
#define DLL_API extern "C" __declspec(dllexport)
#else
#define DLL_API extern "C" __declspec(dllimport)
#endif
/*核心代码，表明当前头文件为dll接口文件*/
/*
设计这个接口类的作用：
能采用动态调用方式使用这个类
*/
class IKPerson
{
public:
 virtual ~IKPerson(void) //对于基类，显示定义虚析构函数是个好习惯（注意，为什么请google）
 {
 }
 virtual int GetOld(void) const = 0;
 virtual void SetOld(int nOld) = 0;
 virtual const char* GetName(void) const = 0;
 virtual void SetName(const char* szName) = 0;
/*接口函数封装为虚函数*/
};
/* 导出函数声明 */
```

```
DLL_API IKPerson* _cdecl GetIKPerson(void);
typedef IKPerson* (__cdecl *PFNGetIKPerson)(void);
#endif
```

---

3. 更改原有代码接口(KChinese.h/KChinese.cpp)

// KChinese.h

```
#pragma once
#define DLL_EXPORT    //加上DLL_EXPORT的定义
#include "ikperson.h"
class CKChinese :
 public IKPerson
{
public:
 CKChinese(void);
 ~CKChinese(void);
 virtual int GetOld(void) const;
 virtual void SetOld(int nOld);
 virtual const char* GetName(void) const;
 virtual void SetName(const char* szName);
/*接口函数封装为虚函数*/
private:
 int m_nOld;
 char m_szName[64];
};
```

// KChinese.cpp

```
#include <stdio.h>
#include <string.h>
#include "KChinese.h"
CKChinese::CKChinese(void) : m_nOld(0)
{
 memset(m_szName, 0, 64);
}
CKChinese::~CKChinese(void)
{
}
int CKChinese::GetOld(void) const
{
 return m_nOld;
}
void CKChinese::SetOld(int nOld)
{
 this->m_nOld = nOld;
}
const char* CKChinese::GetName(void) const
```

```
{
 return m_szName;
}
void CKChinese::SetName(const char* szName)
{
 strncpy(m_szName, szName, 64);
}
/* 导出函数定义 */
IKPerson* __cdecl GetIKPerson(void)
{
 IKPerson* pIKPerson = new CKChinese();
 return pIKPerson;
}
```

这样封装的好处:

最后算法包的接口只有少数的.h头文件，除了接口头文件外还有一些定义类型的头文件(比如Objective Det会用到typedefdet.h来定义RectDet结构体)，同时，外部调用的时候除了需要接口的函数，其他的成员和函数全部都看不到

4. 动态链接库的调用

上面封装动态链接库的方法是支持隐式和显式两种调用方法的

4.1 隐式调用

隐式调用需要把.lib加到“附加依赖库”中，同时把.dll放到工程目录下(加到环境变量也行)

下面是对我们刚才生成的test_dll.dll中函数的调用

我们给软件端应该采取的是这种方式，把dll/lib/.h都给出去

```
IKPerson* pIKPerson = GetIKPerson();
 if (NULL != pIKPerson)
 {
  pIKPerson->SetOld(103);
  pIKPerson->SetName("liyong");
  cout << pIKPerson->GetOld() << endl;
  cout << pIKPerson->GetName() << endl;
  delete pIKPerson;
 }
```

4.2 显式调用

显式调用不需要.lib文件，只要.dll就可以，

这种方式也叫动态调用，可以在代码上随时加上load dll就开始调用动态库的函数

```
 HMODULE hDll = ::LoadLibrary(_T("test_dll.dll"));
 if (NULL != hDll)
 {
  PFNGetIKPerson pFun = (PFNGetIKPerson)::GetProcAddress(hDll, "GetIKPerson");
  if (NULL != pFun)
  {
   IKPerson* pIKPerson = GetIKPerson();
   if (NULL != pIKPerson)
```

```
    {
     pIKPerson->SetOld(103);
     pIKPerson->SetName("liyong");
     cout << pIKPerson->GetOld() << endl;
     cout << pIKPerson->GetName() << endl;
     delete pIKPerson;
    }
   }
   ::FreeLibrary(hDll);
}
```

相比隐式调用，多了红色部分的代码，其实差别不太大