Subject: Template class factory
Posted by mdelfede on Wed, 13 Jan 2010 10:17:46 GMT
View Forum Message <> Reply to Message

Having the need to create some class instances at runtime by class name (String, not type), I developed a small templatized class factory; it's the first step for the polymorphic xmlizer that I'll publish when ready.

Thank to Mirek who helped me with templates !

ClassFactory.h

```
#ifndef _ClassFactory_ClassFactory_h_
#define _ClassFactory_ClassFactory_h_

#include <Core/Core.h>
using namespace Upp;

template<class T> class WithFactory
{
 private:
  typedef One<T> (*CreateFunc)();
  typedef VectorMap<String, CreateFunc>mapType;
  static mapType &classMap() { static mapType cMap; return cMap; }
  template<class D> static One<T> __Create(void) { return One<T>((T *)new D); }
 public:

  template<class D> static void Register(const String &name) { classMap().Add(name,
__Create<D>); }
  static One<T> Create(const String &className) { return classMap().Get(className)(); }
  static Vector<String> const &Classes(void) { return classMap().GetKeys(); }
};

#define REGISTERCLASS(type) \
 INITBLOCK { \
 type::Register<type>(#type); \
}
#endif
```

Sample usage:

```
#include "ClassFactory.h"

class Base : public WithFactory<Base>
{
 public:
```

```
  virtual String WhoAmI(void) { return "Base"; }
};

class Derived : public Base
{
 public:
  virtual String WhoAmI(void) { return "Derived"; }
};

REGISTERCLASS(Base);
REGISTERCLASS(Derived);

CONSOLE_APP_MAIN
{
 One<Base> p1 = Base::Create("Base");
 One<Base> p2 = Base::Create("Derived");

 Cerr() << "p1 is a '" << p1->WhoAmI() << "'\n";
 Cerr() << "p2 is a '" << p2->WhoAmI() << "'\n";

}
```

output :

```
p1 is a 'Base'
p2 is a 'Derived'
```

Ciao

Max

Subject: Re: Template class factory
Posted by mdelfede on Wed, 13 Jan 2010 10:40:59 GMT
View Forum Message <> Reply to Message

A better version with RTTI class identification :

```
#ifndef _ClassFactory_ClassFactory_h_
#define _ClassFactory_ClassFactory_h_

#include <Core/Core.h>
using namespace Upp;
```

```
template<class T> class WithFactory
{
 private:
  typedef One<T> (*CreateFunc)();
  typedef VectorMap<String, CreateFunc>mapType;
  static mapType &classMap() { static mapType cMap; return cMap; }
  template<class D> static One<T> __Create(void) { return One<T>((T *)new D); }
  String myType;
 public:

  template<class D> static void Register(const String &name) { classMap().Add(name,
__Create<D>); }
  static One<T> Create(const String &className)
  {
   One<T> me = classMap().Get(className)();
   me->myType = className;
   return me;
  }
  static Vector<String> const &Classes(void) { return classMap().GetKeys(); }
  String const &IsA(void) { return myType; }
};

#define REGISTERCLASS(type) \
 INITBLOCK { \
 type::Register<type>(#type); \
}
#endif
```

Usage :

```
#include "ClassFactory.h"

class Base : public WithFactory<Base>
{
};

class Derived : public Base
{
};

REGISTERCLASS(Base);
REGISTERCLASS(Derived);

CONSOLE_APP_MAIN
{
```

```
 One<Base> p1 = Base::Create("Base");
 One<Base> p2 = Base::Create("Derived");

 Cerr() << "p1 is a '" << p1->IsA() << "'\n";
 Cerr() << "p2 is a '" << p2->IsA() << "'\n";

}
```

Now classes knows themselves

Ciao

Max

Subject: Re: Template class factory
Posted by mirek on Wed, 13 Jan 2010 13:08:18 GMT
View Forum Message <> Reply to Message

Adds 16 bytes per instance (sizeof(String)). Sometimes that is not a good tradeoff.

Could be solved by mapping typeid.

Mirek

Subject: Re: Template class factory
Posted by mdelfede on Wed, 13 Jan 2010 14:13:30 GMT
View Forum Message <> Reply to Message

Already tried, doesn't work....

```
String const &IsA(void) { Cerr() << "\n" << typeid(*this).name() << "\n"; return myType; }
```

prints
11WithFactoryI4BaseE

which has nothing to do with real type.
What I could do is to store just the char*, it should be static data.

Max

Subject: Re: Template class factory
Posted by mdelfede on Wed, 13 Jan 2010 14:15:52 GMT
View Forum Message <> Reply to Message

And if I do

String const &IsA(void) { T * p = (T *)this; Cerr() << "\n" << typeid(*p).name() << "\n"; return myType; }

It always output '4Base' even if the class is the derived one.
Compiler bug ?

Subject: Re: Template class factory
Posted by mirek on Wed, 13 Jan 2010 14:20:01 GMT
View Forum Message <> Reply to Message

mdelfede wrote on Wed, 13 January 2010 09:15And if I do

String const &IsA(void) { T * p = (T *)this; Cerr() << "\n" << typeid(*p).name() << "\n"; return myType; }

It always output '4Base' even if the class is the derived one.
Compiler bug ?

Do you have any virtual function in Base?

Mirek

Subject: Re: Template class factory
Posted by mdelfede on Wed, 13 Jan 2010 14:20:48 GMT
View Forum Message <> Reply to Message

Nope ! Is that the problem ???

Subject: Re: Template class factory
Posted by mdelfede on Wed, 13 Jan 2010 14:37:28 GMT
View Forum Message <> Reply to Message

Well, quite interesting topic

The definitive factory, thanx to Mirek know-how :

```
#ifndef _ClassFactory_ClassFactory_h_
#define _ClassFactory_ClassFactory_h_

#include <Core/Core.h>
using namespace Upp;

template<class T> class WithFactory
{
 private:
  typedef One<T> (*CreateFunc)();
  typedef VectorMap<String, CreateFunc>mapType;
  static mapType &classMap() { static mapType cMap; return cMap; }
  static VectorMap<String, String> &typeMap() { static VectorMap<String, String> tMap; return
tMap; }
  template<class D> static One<T> __Create(void) { return One<T>((T *)new D); }
 public:

  template<class D> static void Register(const String &name)
  {
   classMap().Add(name, __Create<D>);
   typeMap().Add(typeid(D).name(), name);
  }
  static One<T> Create(const String &className) { return classMap().Get(className)(); }
  static T *CreatePtr(String const &className) { return classMap().Get(className)().Detach(); }
  static Vector<String> const &Classes(void) { return classMap().GetKeys(); }
  String const &IsA(void) { return typeMap().Get(typeid(*this).name()); }
  virtual ~WithFactory() {}
};

#define REGISTERCLASS(type) \
 INITBLOCK { \
 type::Register<type>(#type); \
}
#endif
```

and the sample usage :

```
#include "ClassFactory.h"

class Base : public WithFactory<Base>
{
};
```

```
class Derived : public Base
{
};

REGISTERCLASS(Base);
REGISTERCLASS(Derived);

CONSOLE_APP_MAIN
{
 One<Base> p1 = Base::Create("Base");
 One<Base> p2 = Base::Create("Derived");

 Cerr() << "p1 is a '" << p1->IsA() << "'\n";
 Cerr() << "p2 is a '" << p2->IsA() << "'\n";

 Base *ptr = Base::CreatePtr("Derived");
 Cerr() << "ptr  is a '" << ptr->IsA() << "'\n";
 delete ptr;

 Base *d = new Derived;
 Cerr() << "d  is a '" << d->IsA() << "'\n";
 delete d;

}
```