

asworld

博客园 首页 新随笔 联系 管理 订阅

随笔- 3 文章- 0 评论- 4 阅读- 12155

< 2022年3月 >

日 一 二 三 四 五 六

27 28 1 2 3 4 5

6 7 8 9 10 11 12

13 14 15 16 17 18 19

20 21 22 23 24 25 26

27 28 29 30 31 1 2

3 4 5 6 7 8 9

昵称: asworld

园龄: 7年11个月

粉丝: 0

关注: 0

+加关注

搜索

我找看

谷歌搜索

常用链接

我的随笔

我的论坛

我的参与

最新评论

我的标签

更多链接

我的标签

C++(2)

OOPI(1)

RTTI(1)

Linux(1)

GDAL(1)

Centos(1)

ImageMagick(1)

JMagick(1)

随笔档案

2014年10月(2)

2014年8月(1)

阅读排行

1. C++动态创建类的实例(315 0)

2. Linux下libroot的卸载使与使用GDAL库的方法(2481)

3. Centos 6 安装 JMagick(52 4)

评论排行

1. C++动态创建类的实例(4)

推荐排行

1. C++动态创建类的实例(2)

最新评论

1. ReC++动态创建类的实例
我自己也研究了一段时间，发现
需要动态创建类时必须要在主
函数存在的项目中编译才有效。
如果在外部的动态库中编译，再
把动态库链接到主程序就没效
果。
--duxin_kata

2. ReC++动态创建类的实例
不错，刚好帮得着
--summing

3. ReC++动态创建类的实例
工厂方法，+ +
--eryar

4. ReC++动态创建类的实例
工厂模式的一种变形
--zangyu00544

C++动态创建类的实例

写在前面： 首先声明，C++实际上是不可以动态创建类的实例的。

下面我来做一个解释，所谓动态创建类的实例是指的程序运行过程中创建并使用一个“未知”的类。而“未知”是指的程序编译时并不知道有那些类是需要动态创建的。对于C++这门语言来说，编译器不知道的类是不可以在运行时使用的。所以我说C++是不可以的。

不过C++可以做到另一件事情，基本可以满足大多数类似的需求。

我描述为通过类名创建类的实例。

进入正题。

首先描述一下需求：

编写一个程序实现在程序运行过程中通过类名称（即字符串变量）创建该类的实例，且在主程序不出现该类的声明。

然后进行分析设计：

1.在主程序不出现类的声明，还需要用这个类，那得自然就会想到用基类指针实现。

2.创建一个指向该类的基类指针，则需要使用new运算符。

3.通过类名调用相应的new运算符，则需要使用map加函数指针。

4.最后，如何在主函数执行之前构造该map，则需要用静态成员或主函数运行前进行创建的技巧。

下面给出代码实现：

首先有一个CObject类作为基类，由它派生出CObjectA和CObjectB两个需要动态创建的子类。

```
Object.h
1 #ifndef __C_OBJECT_H_
2 #define __C_OBJECT_H_
3
4 #include "ObjectFactory.h"
5
6 class CObject
7 {
8 public:
9     CObject(): className("CObject") {}
10    virtual ~CObject(){}
11    virtual const std::string GetClassName()
12    {
13        return className;
14    }
15 private:
16    std::string className;
17 };
18
19 #endif // __C_OBJECT_H_
```

```
ObjectA.h
1 #ifndef __C_OBJECT_A_H_
2 #define __C_OBJECT_A_H_
3
4 #include "Object.h"
5
6 class CObjectA : public CObject
7 {
8 public:
9     ~CObjectA(): className("CObjectA") {}
10    ~CObjectA(){}
11    const std::string GetClassName()
12    {
13        return className;
14    }
15 private:
16    std::string className;
17 };
18
19 REGISTER_CLASS(CObjectA);
20
21 #endif // __C_OBJECT_A_H_
```

```
ObjectB.h
1 #ifndef __C_OBJECT_B_H_
2 #define __C_OBJECT_B_H_
3
4 #include "Object.h"
5
6 class CObjectB : public CObject
7 {
8 public:
9     CObjectB(): className("CObjectB") {}
10    ~CObjectB(){}
11    const std::string GetClassName()
12    {
13        return className;
14    }
15 private:
16    std::string className;
17 };
18
19 REGISTER_CLASS(CObjectB);
20
21 #endif // __C_OBJECT_B_H_
```

然后重点来了，上面两个类里面都有宏定义REGISTER_CLASS，实际上就是声明一个带有静态成员注册的类，通过初始化该静态成员去构造该动态创建map。

```
ObjectFactory.h
1 #ifndef __C_OBJECT_FACTORY_H_
2 #define __C_OBJECT_FACTORY_H_
3
4 #include <map>
5 #include <string>
6
7 typedef void* (*NewInstancePt)();
8
9 class CObjectFactory
10 {
11 public:
12     static void* CreateObject(const char *className)
13     {
14         std::map<std::string, NewInstancePt>::const_iterator it;
15         it = dynCreateMap.find(className);
16         if(it == dynCreateMap.end())
17             return NULL;
18         else
19         {
20             NewInstancePt np = it->second;
21             return np();
22         }
23     }
24
25     static void RegisterClass(const char *className, NewInstancePt np)
26     {
27         dynCreateMap[className] = np;
28     }
29 private:
30     static std::map<std::string, NewInstancePt> dynCreateMap;
31 };
32
33 std::map<std::string, NewInstancePt> CObjectFactory::dynCreateMap;
34
35 class Register
36 {
37 public:
38     Register(const char *className, NewInstancePt np)
39     {
40         CObjectFactory::RegisterClass(className, np);
41     }
42 };
43
44 #define REGISTER_CLASS(class_name) \
45 class class_name##Register \
46 { \
47 public: \
48     static void* NewInstance() \
49     { \
50         return new class_name(); \
51     } \
52 private: \
53     static Register reg; \
54 };
55 Register class_name##Register::reg(class_name, class_name##Register::NewInstance)
56
57 #endif // __C_OBJECT_FACTORY_H_
```

最后，当然还有如何使用的主函数内容。

```
RTTI.cpp
1 #include <stdio.h>
2 #include "ObjectFactory.h"
3 #include "Object.h"
4 #include "ObjectA.h"
5 #include "ObjectB.h"
6
7 int main(int argc, const char *argv[])
8 {
9     CObject *objA = static_cast<CObject *>(CObjectFactory::CreateObject("CObjectA"));
10    std::string className;
11    if(objA == NULL)
12    {
13        printf("ERROR! Can't Create Class ObjectA!\n");
14    }
15    else
16    {
17        className = objA->GetClassName();
18        printf("OK! Create %s \n", className.c_str());
19    }
20
21    CObject *objB = static_cast<CObject *>(CObjectFactory::CreateObject("CObjectB"));
22    if(objB == NULL)
23    {
24        printf("ERROR! Can't Create Class ObjectB!\n");
25    }
26    else
27    {
28        className = objB->GetClassName();
29        printf("OK! Create %s \n", className.c_str());
30    }
31
32    return 0;
33 }
```

这样就完成了所谓的动态创建。这里需要注意的是在主函数所在的文件里面需要#include所有的需要动态创建的类，否则的话在编译的过程中该动态类以及其注册类就不会被编译，也就不存在构建动态创建map，整个机制也就失效了。这也就是说的C++实际上是不可以动态创建类的原理。

不得不提，如果想要能做的动态创建，建议使用Java，简单看看Java的反射机制和ClassLoader就会知道原来动态加载如此容易。...

标签: C++, RTTI, OOP

好文推荐 关注 收藏该文

asworld

关注-0 粉丝-0

+加关注

2 0

点赞 反对

上一篇: Linux下libroot库的卸载与使用GDAL库的方法

posted @ 2014-10-23 11:51 asworld 阅读(9150) 评论(4) 编辑 收藏 举报

 登录后才能查看或发表评论, [立即登录](#) 或 [注册](#) 查看评论

最新评论

网友评论

返回顶部

编辑推荐:

- 浅谈 C# 字符串构造利器 StringBuilder
- 宏项目的部署 —— 性能优化篇
- 2021 .NET Conf China 主题分享之-轻松玩转 .NET 大规模版本升级
- 揭秘 OAuth2.0 协议和授权机制
- Asp.net core IdentityServer4 与传统的基于角色的权限系统的集成

最新新闻:

- 雷峰网独家 | 史无前例，阿里云或将空降 M7 级高管
- 可穿戴设备的机器人见过吗？微软和维3D打印而成 | Science子刊封面
- 区块链技术不是下一个物联网吗？
- 丁磊的文创帝国现在这样好未来
- 制造焦虑！外国小哥爆料拍出美剧《恐怖》网络图，还顺手发了篇论文

» 更多新闻...

Copyright © 2022 asworld

Powered by .NET 6 on Kubernetes