C语言面向对象(oopc)详细设计说明书

个人使用

180210V1.0

**编 写： 潘小飞 日期： 180211**

**校 对： 日期：**

**审 核： 日期：**

**批 准： 日期：**

目录

[C语言面向对象(oopc)详细设计说明书 i](#_Toc507782266)

[目录 i](#_Toc507782267)

[文档说明 ii](#_Toc507782268)

[修改记录 ii](#_Toc507782269)

[1 引言 - 1 -](#_Toc507782270)

[1.1 编写目的 - 1 -](#_Toc507782271)

[1.2 项目背景 - 1 -](#_Toc507782272)

[1.3 定义 - 1 -](#_Toc507782273)

[1.4 参考资料 - 2 -](#_Toc507782274)

[2 总体设计 - 3 -](#_Toc507782275)

[2.1 需求概述 - 3 -](#_Toc507782276)

[2.2 软件结构 - 3 -](#_Toc507782277)

[3 程序描述 - 6 -](#_Toc507782278)

[3.1 通用宏定义 - 6 -](#_Toc507782279)

[3.2 功能描述 - 6 -](#_Toc507782280)

[3.2.1 接口创建 - 7 -](#_Toc507782281)

[3.2.2 抽象类创建 - 8 -](#_Toc507782282)

[3.2.3 实类创建 - 11 -](#_Toc507782283)

[3.2.4 示例DEMO - 18 -](#_Toc507782284)

[3.3 配置选项 - 18 -](#_Toc507782285)

[3.4 数据类型 - 19 -](#_Toc507782286)

[3.5 总结 - 19 -](#_Toc507782287)

[3.5.1 OOPC优点 - 19 -](#_Toc507782288)

[3.5.2 OOPC缺点 - 19 -](#_Toc507782289)

文档说明

1. 文档普通内容字体使用 宋体 五号。
2. 一级标题使用2号字体。
3. 二级及其他标题使用3号字体。

修改记录

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **修订人** | **修订日期** | **修订描述** |
| V1.0 | 潘小飞 | 180210 | 初次生成当前模板 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. 引言
   1. 编写目的

面向对象是一个很方便的开发方式。在程序设计方面比面向过程简单很多了。而C语言是一个面向过程的语言，不能使用的对象的方式进行程序设计。当前为了解决这个问题，对C代码也可以进行面向对象开发，设计出一套宏来模拟面向对象开发。

只要是能进行C代码编译的开发环境，都能使用当前这套定义的宏。所有C代码都能使用，包括嵌入式行业。

* 1. 项目背景

暂无项目。

目前提供使用例程DEMO。

* 1. 定义

INF:interface接口声明

AC:abstract class抽象类声明

CL:class实类声明

ACC:abstract class constructor抽象类构造函数定义

ACD:abstract class destructor抽象类析构函数定义

CC:class constructor实类构造函数定义

CD:class destructor实类析构函数定义

CACC:call abstract class constructor调用抽象类构造函数

CACD:call abstract class destructor调用抽象类析构函数

CCC:call class constructor调用实类构造函数

CCD:call class destructor调用实类析构函数

IMPL:implement接口或抽象类继承

EXT:extend实类继承

SUB: subordinate接口或抽象类获得继承类指针

SUBC: subordinate class实类获得子继承类指针

FS:function setting函数配置

CN:class new实类实例化

CNNP:class new without(no) param(无参)实类实例化

CF:class free实类资源释放

OPRS: operation result查看操作结果

* 1. 参考资料

1.在sourceforge上存在一个开发出来的宏配置：http://lwoopc.sourceforge.net



2. Object-oriented Programming with ANSI-C



3. Inside\_The\_C++\_Object\_Model中文版.pdf



1. 总体设计
   1. 需求概述

面向过程方式开发的系统，代码复杂，耦合性强，难以维护。随着我们所要解决的问题越来越复杂，代码也变得越来越复杂，越来越难以掌控。

而面向对象改变了程序员的思维方式，以更加符合客观世界的方式来认识世界。通过合理的运用抽象、封装、继承和多态，更好的组织程序，从而很好地应对这种复杂性。

面向对象的过程，就需要像通用的面向对象语言那样能进行对象的创建，继承关系等。需要通过合理的手段来处理对象之间的关系，针对这些功能设计出合适的接口。

* 1. 软件结构

通用的面向对象开发有合理的对象关系，如java/c#/c++等。针对c面向对象设计有如下相关对象关系。



图2.1 类继承接口



图2.2 类继承抽象类再分别继承接口



图2.3 类继承实类再继承抽象类最后分别继承接口

对象之间的关系主要分为以上3类。

* 类继承接口
* 类继承抽象类再分别继承接口
* 类继承实类再继承抽象类最后分别继承接口

类继承关系一般来说第一种和第二种比较常见。深层次的嵌套就必须考虑类是否设计合理，3层嵌套关系一般都能满足常见的需求。

1. 程序描述
   1. 通用宏定义

模块内部定义了以下通用参数宏。

表3.1 通用参数宏

|  |  |  |
| --- | --- | --- |
| 名称 | 数值 | 含义 |
| OOPC\_FALSE | 0 | （操作）失败 |
| OOPC\_TRUE | 1 | （操作）成功 |
| OOPC\_NULL | ((void \*)0) | 空指针 |
|  |  |  |
|  |  |  |

* 1. 功能描述

为实现C代码程序能够像面向对像一样操作，定了义了以下20个功能宏。

表3.2 功能宏定义

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 单词 | 含义 | 其他 |
| INF | interface | 接口声明 | 使用尽量放在.h文件中 |
| AC | abstract class | 抽象类声明 | 使用尽量放在.h文件中 |
| CL | class | 实类声明 | 使用尽量放在.h文件中 |
|  |  |  |  |
| ACC | abstract class constructor | 抽象类构造函数定义 | 使用必须在.c文件中 |
| ACD | abstract class destructor | 抽象类析构函数定义 | 使用必须在.c文件中 |
| CC | class constructor | 实类构造函数定义 | 使用必须在.c文件中 |
| CD | class destructor | 实类析构函数定义 | 使用必须在.c文件中 |
|  |  |  |  |
| CACC | call abstract class constructor | 调用抽象类构造函数 | 实类实例化时构造函数中调用 |
| CACD | call abstract class destructor | 调用抽象类析构函数 | 实类实例化时析构函数中调用 |
| CCC | call class constructor | 调用实类构造函数 | 实类实例化时构造函数中调用 |
| CCD | call class destructor | 调用实类析构函数 | 实类实例化时析构函数中调用 |
|  |  |  |  |
| IMPL | implement | 接口或抽象类继承 |  |
| EXT | extend | 实类继承 |  |
| SUB | subordinate | 接口或抽象类获得继承类指针 |  |
| SUBC | subordinate class | 实类获得子继承类指针 |  |
| FS | function setting | 函数配置 |  |
|  |  |  |  |
| CN | class new | 实类实例化 |  |
| CNNP | class new without(no) param | (无参)实类实例化 | 其作用与CN相同，只不过为了适应在无参情况下，有些编译器不能使用CN进行编译而特别设置的一个宏 |
| CF | class free | 实类资源释放 |  |
| OPRS | operation result | 查看操作结果 | CN/CNNP/CF/CCC操作后的没有返回值，可以通过OPRS查看前一步操作结果 |
|  |  |  |  |
|  |  |  |  |

注意：

当前所有定义的功能宏不能与其他文件中的宏进行重复定义。如有出现重复定义，需要考虑头文件的包含关系。

* + 1. 接口创建

接口说明如下表。

表3.3 接口创建

|  |  |
| --- | --- |
| 内容项 | 详细说明 |
| 功能 | 接口在面向对象的过程中只是作为一个统一形式的作用，不管是哪个类继承了接口，都需要该类定义一个与接口一致的函数。  接口功能与抽象类相似，只不过是少了构造及析构函数。 |
| 要求 | * 接口函数必须有一个指向自身类型的指针作为输入，指针传递可以定义为自身类型也可以定义为void；   一般要求声明为自身接口类型的指针，这样在进行程序编译时，编译器会对数据类型进行检查。   * 接口一般来说只是对函数写法的统一，每个继承接口的实类都应对接口定义一个函数。 * 在实类初始化时，接口是通过实类的init函数进行配置的。 * 指向自身类型指针的形参禁止定义为cthis。 |
| 声明 | INF(type)  {  void (\*inf)(h##type t);  } |
| 定义 | void (\*inf)(h##type t) |
| 构造定义 | -- |
| 析构定义 | -- |
| 继承 | IMPL(type) |
| 继承构造调用 | -- |
| 继承析构调用 | -- |
| 对象实例化 | -- |
| 对象实例释放 | -- |
| 示例 | 声明：  INF(Aaa)  {  // void (\*InfFun)(void);  **void** (\*InfFun2)(hAaa t);  };  定义：  **void** **Ccc\_Aaa\_InfFun2**(hAaa t)  {  SUB(t, Aaa, Ccc);  cthis->ClaParam++;  }  继承关系：  CL(Ccc)  {  hCcc self;  hCcc (\*init)(hCcc t,  **void** (\*AaaInf)(hAaa t2),  **void** (\*BbbFun)(**void**),  **void** (\*BbbFun2)(hBbb t3),  **void** (\*CccFun2)(hCcc t4));  i32 ClaParam;  i32 ClaParam2;  i32 ClaParam3;  **void** (\*ClaFun)(**void**);  **void** (\*ClaFun2)(hCcc t5);  IMPL(Aaa);  IMPL(Bbb);  }; |
|  |  |

* + 1. 抽象类创建

抽象类说明如下表。

表3.4 抽象类创建

|  |  |
| --- | --- |
| 内容项 | 详细说明 |
| 功能 | 抽象类提供一个与实类功能一样的对象，只不过抽象必须依附实类存在。抽象类的存在就有一种天生的被继承关系。 |
| 要求 | * 抽象函数必须有一个指向自身类型的指针作为输入，指针传递可以定义为自身类型也可以定义为void；   一般要求声明为自身类型的指针，这样在进行程序编译时，编译器会对数据类型进行检查。   * 抽象函数一般来说只是对函数写法的统一，每个继承抽象类的实类都应对抽象函数定义一个函数。 * 在实类初始化时，抽象函数是通过实类的init函数进行配置的。 * 指向自身类型指针的形参禁止定义为cthis。 |
| 声明 | AC(type)  {  void (\*fun)(h##type t);  …  } |
| 定义 | void (\*fun)(h##type t) |
| 构造定义 | ACC(type)  {  …  } |
| 析构定义 | ACD(type)  {  …  } |
| 继承 | IMPL(type) |
| 继承构造调用 | CACC(type) |
| 继承析构调用 | CACD(type) |
| 对象实例化 | -- |
| 对象实例释放 | -- |
| 示例 | 声明：  AC(Bbb)  {  i32 AbsParam;  **void** (\*AbsFun)(**void**);  **void** (\*AbsFun2)(hBbb t);  };  定义：  **void** **Ccc\_Aaa\_InfFun2**(hAaa t)  {  SUB(t, Aaa, Ccc);  cthis->ClaParam++;  }  构造定义：  ACC(Bbb)  {  // 如果需要则进行一些特殊的操作，否则直接返回cthis就行  // 抽象类的函数一般在实类初始化时进行初始化，因为函数操作的是地实类  cthis->AbsParam = 0;  **return** cthis;  }  析构定义：  ACD(Bbb)  {  // 如果需要则进行一些特殊的操作，否则直接返回OOPC\_TRUE就行  cthis->AbsParam = 0;  **return** OOPC\_TRUE;  }  继承关系：  CL(Ccc)  {  hCcc self;  hCcc (\*init)(hCcc t,  **void** (\*AaaInf)(hAaa t2),  **void** (\*BbbFun)(**void**),  **void** (\*BbbFun2)(hBbb t3),  **void** (\*CccFun2)(hCcc t4));  i32 ClaParam;  i32 ClaParam2;  i32 ClaParam3;  **void** (\*ClaFun)(**void**);  **void** (\*ClaFun2)(hCcc t5);  IMPL(Aaa);  IMPL(Bbb);  };  继承构造调用：  CC(Ccc)  {  hCcc Rtv = OOPC\_NULL;  **if** (OOPC\_NULL != CACC(Bbb))  {  FS(init, Ccc\_init);  FS(ClaFun, Ccc\_ClaFun);  //FS(ClaFun2, Ccc\_ClaFun2); // 在定义为同一文件下应使用FS设置  cthis->ClaParam = 0;  cthis->ClaParam2 = 0;  cthis->ClaParam3 = 0;  Rtv = cthis;  }  **else**  {  Rtv = OOPC\_NULL;  }  **return** Rtv;  }  继承析构调用：  CD(Ccc)  {  **int** Rtv = OOPC\_FALSE;  **if** (OOPC\_FALSE != CACD(Bbb))  {  Rtv = OOPC\_TRUE;  }  **else**  {  Rtv = OOPC\_FALSE;  }  **return** Rtv;  } |
|  |  |

* + 1. 实类创建

实类说明如下表。

表3.5 实类创建

|  |  |
| --- | --- |
| 内容项 | 详细说明 |
| 功能 | 建立基本的操作类型 |
| 要求 | * 实类声明中必须附带的两个参数self,init； * self - 指向自身的指针，也可以指示当前实例是否可用，self指向自身时实类可用，self为OOPC\_NULL时实类已释放； * init - 配置实类中参数，init函数有个要求在函数参数声明中指向自身实类指针必须声明为cthis；   为了初始化更加统一，接口及抽象类需要通过init传递进行配置，实类自身函数需要通过FS进行设置，除非特别需求可通过init配置；  实类如有继承，则被继承类所有相关函数都得通过init进行配置；   * 指向自身类型指针的形参除init函数外禁止定义为cthis。 |
| 声明 | CL(type)  {  h##type self;  h##type (\*init)(h##type cthis, …);  void (\*fun)(h##type t);  …  } |
| 定义 | void (\*fun)(h##type t)  h##type (\*init)(h##type cthis, …) |
| 构造定义 | CC(type)  {  …  } |
| 析构定义 | CD(type)  {  …  } |
| 继承 | EXT(type) |
| 继承构造调用 | CCC(type) |
| 继承析构调用 | CCD(type) |
| 对象实例化 | CN(type, classptr, …) |
| 对象实例释放 | CF(type, classptr) |
| 示例 | 声明：  CL(Ccc)  {  hCcc self;  hCcc (\*init)(hCcc t,  **void** (\*AaaInf)(hAaa t2),  **void** (\*BbbFun)(**void**),  **void** (\*BbbFun2)(hBbb t3),  **void** (\*CccFun2)(hCcc t4));  i32 ClaParam;  i32 ClaParam2;  i32 ClaParam3;  **void** (\*ClaFun)(**void**);  **void** (\*ClaFun2)(hCcc t5);  IMPL(Aaa);  IMPL(Bbb);  };  定义：  **void** **Ccc\_ClaFun2**(hCcc t)  {  t->ClaParam3++;  }  hCcc **Ccc\_init**(hCcc cthis,  **void** (\*AaaInf)(hAaa t2),  **void** (\*BbbFun)(**void**),  **void** (\*BbbFun2)(hBbb t3),  **void** (\*CccFun2)(hCcc t4))  {  hCcc Rtv = OOPC\_NULL;  **if** (OOPC\_NULL != cthis)  {  cthis->Aaa.InfFun2 = AaaInf;  cthis->Bbb.AbsFun = BbbFun;  cthis->Bbb.AbsFun2 = BbbFun2;  cthis->ClaFun2 = CccFun2;  Rtv = cthis;  }  **else**  {  Rtv = OOPC\_NULL;  }  **return** Rtv;  }  构造定义：  CC(Ccc)  {  hCcc Rtv = OOPC\_NULL;  **if** (OOPC\_NULL != CACC(Bbb))  {  FS(init, Ccc\_init);  FS(ClaFun, Ccc\_ClaFun);  //FS(ClaFun2, Ccc\_ClaFun2); // 在定义为同一文件下应使用FS设置  cthis->ClaParam = 0;  cthis->ClaParam2 = 0;  cthis->ClaParam3 = 0;  Rtv = cthis;  }  **else**  {  Rtv = OOPC\_NULL;  }  **return** Rtv;  }  析构定义：  CD(Ccc)  {  **int** Rtv = OOPC\_FALSE;  **if** (OOPC\_FALSE != CACD(Bbb))  {  Rtv = OOPC\_TRUE;  }  **else**  {  Rtv = OOPC\_FALSE;  }  **return** Rtv;  }  继承关系：  CL(Cla)  {  hCla self;  hCla (\*init)(hCla t,  **void** ClaInfFun(hClaInf t2),  **void** (\*ClaFun)(hCla t3),  hClaFa ClaFaaa,  **void** ClaFaInfFun(hClaFaInf t4),  **void** ClaFaFun(hClaFa t5),  **void** ClaFaFun2(hhClaFa t6));  i32 ClaParam;  i32 ClaParam2;  i32 ClaParam3;  **void** (\*ClaFun)(hCla t7);  IMPL(ClaInf);  EXT(ClaFa);  };  继承构造调用：  hCla **Cla\_init**(hCla cthis,  **void** ClaInfFun(hClaInf t2),  **void** (\*ClaFun)(hCla t3),  hClaFa ClaFaaa,  **void** ClaFaInfFun(hClaFaInf t4),  **void** ClaFaFun(hClaFa t5),  **void** ClaFaFun2(hhClaFa t6))  {  hCla Rtv = OOPC\_NULL;  **if** (OOPC\_NULL != cthis)  {  cthis->ClaInf.CInf = ClaInfFun;  cthis->ClaFun = ClaFun;  // 传参配置父类  cthis->ClaFa = ClaFaaa;  CCC(ClaFa, ClaFaaa, ClaFaInfFun, ClaFaFun, ClaFaFun2);  **if** (OOPC\_NULL == OPRS(\*ClaFaaa))  {  **if** (OOPC\_TRUE == CCD(ClaFa))  {  ;  }  **else**  {  ;  }  Rtv = OOPC\_NULL;  }  **else**  {  Rtv = cthis;  }  }  **else**  {  Rtv = OOPC\_NULL;  }  **return** Rtv;  }  继承析构调用：  CD(Cla)  {  **int** Rtv = OOPC\_FALSE;  **if** (OOPC\_NULL != cthis->ClaFa)  {  **if** (OOPC\_TRUE == CCD(ClaFa))  {  cthis->ClaParam = 0;  cthis->ClaParam2 = 0;  cthis->ClaParam3 = 0;  Rtv = OOPC\_TRUE;  }  **else**  {  Rtv = OOPC\_FALSE;  }  }  **else**  {  Rtv = OOPC\_FALSE;  }  **return** Rtv;  }  对象实例化：  CN(Ccc, &ClassAbs,  Ccc\_Aaa\_InfFun2,  Ccc\_Bbb\_AbsFun, Ccc\_Bbb\_AbsFun2,  Ccc\_ClaFun2);  **if** (OOPC\_NULL != OPRS(ClassAbs))  {  ;  }  **else**  {  ;  }  CN(Cla, &CChild,  ClaInfFun,  ClaFun,  &CFa,  ClaFaInfFun,  ClaFaFun, ClaFaFun2);  **if** ((OOPC\_NULL != OPRS(CChild)) && (OOPC\_NULL != OPRS(CFa)))  {  ;  }  **else**  {  ;  }  对象实例释放：  CF(Ccc, ClassAbs.self);  **if** (OOPC\_NULL != OPRS(ClassAbs))  {  ;  }  **else**  {  ;  }  CF(Cla, CChild.self);  **if** ((OOPC\_NULL != OPRS(CChild)) || (OOPC\_NULL != OPRS(CFa)))  {  ;  }  **else**  {  ;  } |
|  |  |

* + 1. 示例DEMO







* 1. 配置选项

模板代码中配置配置的内容不多，主要有以下3项，目的是为了提供一些关键参数供用户选择。

表3.6 配置参数宏

|  |  |  |
| --- | --- | --- |
| 名称 | 选项 | 说明 |
| OOPC\_USE\_USER\_DEFINED\_OFFSETOF | OOPC\_TRUE/  OOPC\_FALSE | 是否使用自定义offsetof，不选择情况下使用标准库的offsetof，当前选择为OOPC\_TRUE |
| OOPC\_USE\_STATIC\_INLINE\_OPTIMIZE | OOPC\_TRUE/  OOPC\_FALSE | 是否使用static inline对构造及析构函数进行优化，当前选择为OOPC\_TRUE |
| OOPC\_RETURN\_DATATYPE | int | 用户可根据开发平台为析构函数及释放函数的返回数据选择最合适的数据类型，默认选择为int |
|  |  |  |
|  |  |  |

* 1. 数据类型

针对接口、抽象类、实类进行声明时会对应生成各种数据类型声明。具体类型，如下表列出各项。

表3.7 各对象数据类型

|  |  |  |  |
| --- | --- | --- | --- |
| 对象 | 操作宏 | 数据类型 | 说明 |
| 接口 | INF(type) | type | 接口对象结构体 |
|  |  | h##type | 接口对象指针 |
| 抽象类 | AC(type) | type | 抽象类对象结构体 |
|  |  | h##type | 抽象类对象指针 |
| 实类 | CL(type) | type | 实类对象结构体 |
|  |  | h##type | 实类对象指针 |
|  |  | hh##type | 指向实类对象指针的指针 |
|  |  |  |  |

注：h是handle的简写。

* 1. 总结
     1. OOPC优点
* 代码非常轻量化
* 能适应各种平台
* 不存在内存泄漏问题，对象都是编写的结构体变量
  + 1. OOPC缺点
* 无法支持重载（C语言特性）
* 无法支持多态（C语言特性）
* 无法封装（不能区分私有、保护、公有，当前全部都是公有）

1. 应用方法

在拥有方法的基础上是如果快速使用起来，前期类生成都是手写，目前使用脚本生成的方式增加开发速度。

* 1. Python脚本代码生成

使用脚本可以大大简化开发过程，让自己的开发速度和效率上一个台阶。让一些重复性的工作给工具辅助完成。

* + 1. 脚本代码

脚本本身的代码量并不大，使用规则也很简单。Python一般都是指令的方式。

当前脚本指令为: python classTemplate.py path className func\_list

脚本至少需要4个参数，

路径path，当前路径可以使用/或\，其他路径一定要以非/或\开头，否则生成不了

类名称className根据需要选取

func\_list是多个参数，目前配置类时至少需要给定1个功能函数



* + 1. 脚本使用示例

脚本参数：

python classTemplate.py mypath/path My func second third

会生成名为My的类，除去init功能函数，还有func/second/third三个功能函数。

