ACE高级特性

Allen Long
ihuihoo@gmail.com
http://www.huihoo.com
Huihoo - Enterprise Open Source

内容安排

- ACE共享内存(Shared Memory)
- ACE流框架(Streams Framework)
- ACE服务配置框架(Service Configurator Framework)
- ACE命名服务(ACE Naming Service)

ACE内存管理

- ACE含有两组不同的类用于内存管理,能有效管理动态内存(从堆中申请的内存)和共享内存(在进程间共享的内存).
- ACE_Allocator: 这组类使用动态绑定和策略模式来提供灵活性和可扩展性,它们只能用于局部的动态内存分配.
- ACE_Malloc: 这组类使用C++模板和外部多态性(External Polymorphism)来为内存分配机制提供灵活性.在这组类中的类不仅包括了用于局部动态内存管理的类,也包括了管理进程间共享内存的类.这些共享内存类使用底层OS共享内存接口.
- **区别**

ACE_Allocator类更为灵活,这是通过动态绑定（这在C++里需要使用虚函数）来完成的.

ACE_Malloc类有着更好的性能,基于ACE_Malloc的分配器不能在运行时进行配置.

共享内存及持久性

C:\ACE_wrappers\examples\Shared_Malloc\persistence

堆内存分配

-- C语言编写

```
char *c = (char *)malloc (64);
if (c == 0)
exit(1); /* Out of memory! */
```

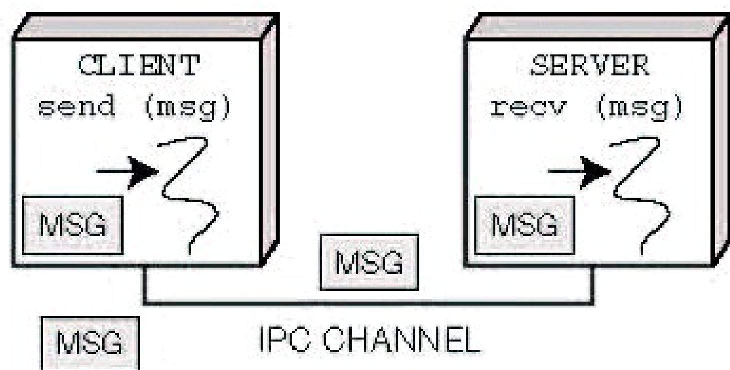
-- C++语言编写

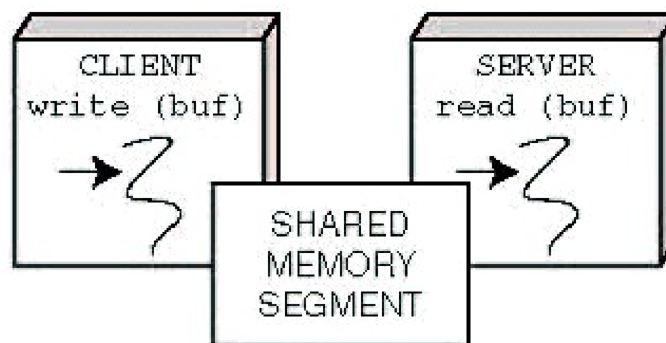
```
char *c = new char[64];
if (c == 0)
exit(1); /* Out of memory! */
```

-- ACE编写

```
char *c;
ACE_NEW_NORETURN (c, char[64]);
if (c == 0)
exit(1); /* Out of memory! */
```

共享内存





它是一种IPC机制. 机器允许相同或不同主机上的多个进程访问、交换数据，就象数据位于每一个进程的本地地址空间一样。

共享内存有本地(local),分布式(distributed)两种

其中,本地内存的常见机制:
。System V UNIX共享内存
。内存映射文件,其内容可以转储至永久存储器

共享内存

内存池类型:

ACE_MMAP_Memory_Pool: 基于内存映射文件的内存池
ACE_Lite_MMAP_Memory_Pool: 基于内存映射文件的内存池的轻量版
ACE_Shared_Memory_Pool: 基于System V 共享内存的内存池
ACE_Local_Memory_Pool: 基于C++ new 操作符的内存池
ACE_Pagefile_Memory_Pool: 基于从Windows页面文件分配的"匿名"内存区的内存池
ACE_Sbrk_Memory_Pool: 基于sbrk(2)的内存池

如: `typedef ACE_Malloc <ACE_MMAP_MEMORY_POOL, ACE_Null_Mutex> Malloc_Allocator;`

ACE含有一个适配器模板类ACE_Allocator_Adapter, 它将ACE_Malloc类适配到ACE_Allocator接口. 也就是说, 在实例化这个模板之后创建的新类可用于替换任何ACE_Allocator:

```
typedef
ACE_Allocator_Adapter<ACE_Malloc<ACE_SHARED_MEMORY_POOL,ACE_Null_Mutex>
Allocator;
```

Stream(流):分层服务的集成

ACE Stream组件简化了那些分层的(layered)或层次的(hierarchic)软件的开发.

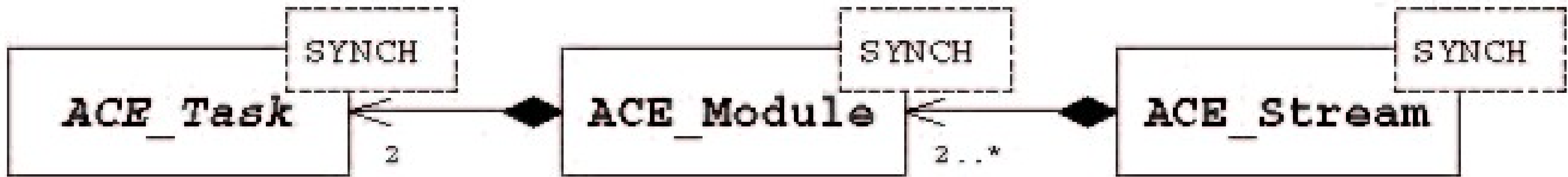
流类属含有自适配服务执行体(ASX)构架,它集成了较低级的OO包装组件(像IPC SAP)和较高级的类属(像Reactor和Service Configurator).

ASX构架合并了来自若干模块化的通信构架的概念,其中包括系统V STREAMS、x-kernel和来自面向对象操作系统Choices的Conduit构架.

ASX构架为通信软件的开发者提供下面两种好处：
1.它嵌入、封装，并实现了通常用于开发通信软件的一些关键设计模式。
2.它严格地区分了关键的开发事务

流类属的主要组件:
 * ACE_Stream
 * ACE_Module
 * ACE_Task

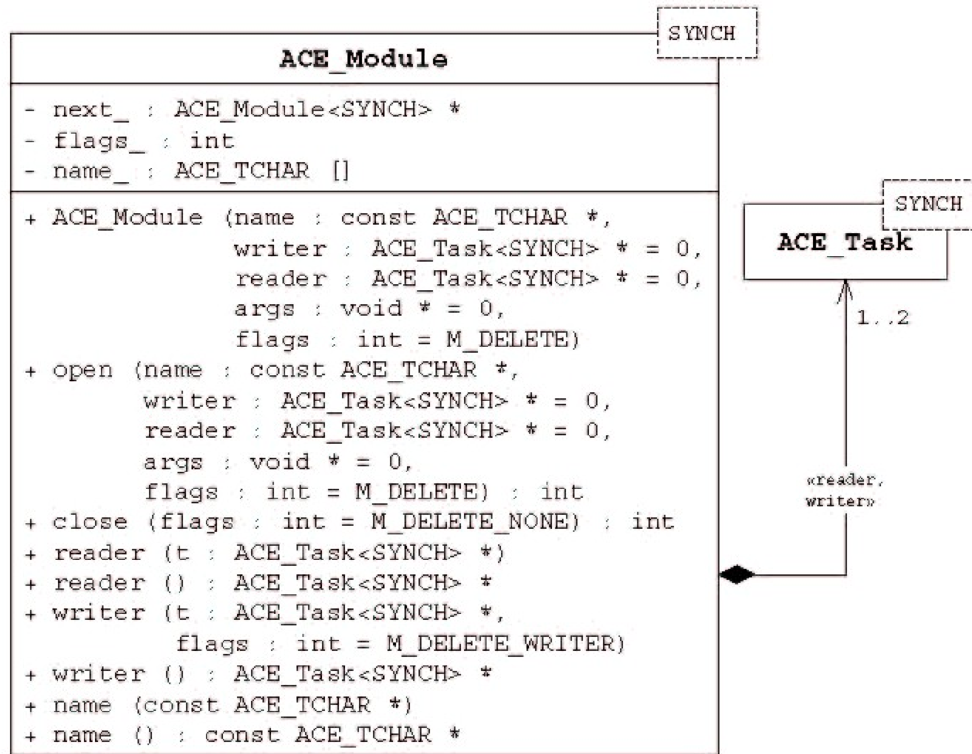
Stream(流)框架



流框架实现了Pipes and Filters(管道和过滤器)模式,这个框架简化了分层式/模块化应用的开发,这些应用可以通过双向处理模块来进行通信

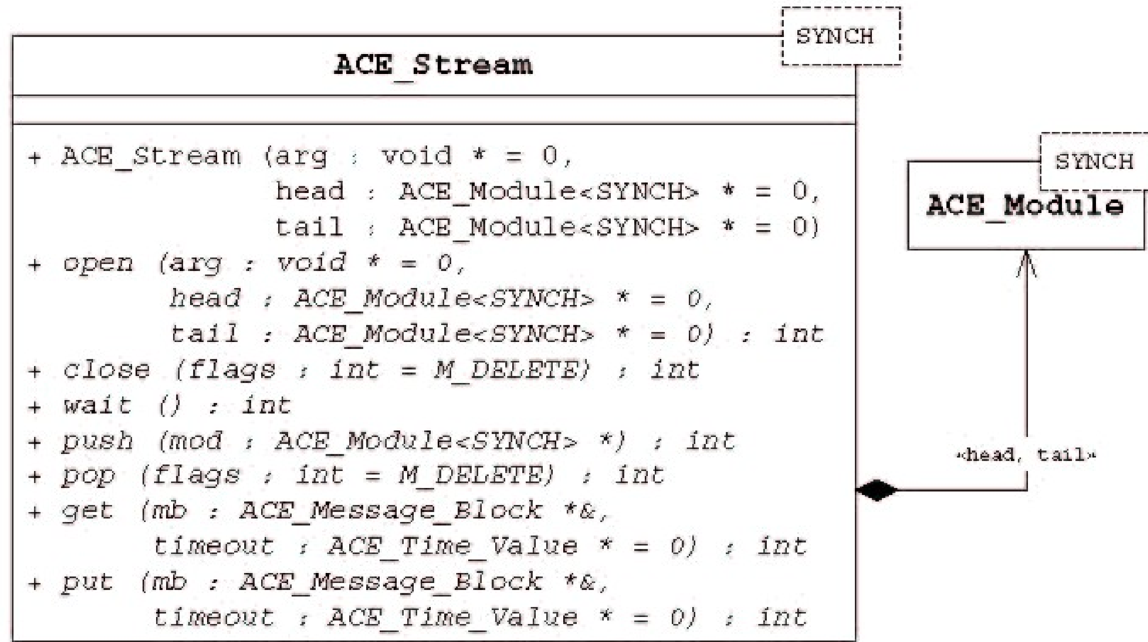
ACE类	描述
ACE_Task	应用所定义功能的内聚单元，它使用消息来交流请求、响应、数据以及控制信息，并且可以顺序地或并发地排队和处理消息
ACE_Module	应用中的一个清晰可辨的双向处理层，其中含有两个ACE_Task对象,一个用于"读",一个用于"写"
ACE_Stream	含有一列有序的、相互连接的ACE_Module对象,可被用于配置和执行分层式应用定义的服务

ACE_Module类



- 每个ACE_Modele都是双向的,应用定义的处理层,其中含有一对派生自ACE_Task的reader和writer任务
- ACE Service Configurator 框架支持ACE_Module对象的动态构造,可以在运行时将这些对象配置进某个ACE_Stream中
- 包含在某个ACE_Module中的reader和writer ACE_Task对象通过使用公共hook方法传递消息来与相邻的ACE_Task对象进行协作，这促进了松耦合，并简化了重新配置
- 可以独立地改变和替换组合进ACE_Module中的对象,从而降低维护和改进的开销

ACE_Stream类



- 提供多个方法，可动态地增加、替换和移除ACE_Module对象,从而形成各种流配置
- 提供多个方法，可发送消息给某个ACE_Stream的方法,或是接收来自某个 ACE_Stream 的消息的方法
- 提供将两个ACE_Stream流连接在一起的机制
- 提供了关闭流中的所有模块,并等待它们停止的途径

Stream使用步骤

- 创建一个或多个ACE_Task派生对象，并实现应用逻辑;
- 把相关任务结对放入顺流和逆流组件;
- 针对每对任务，构造一个模块容纳它们;
- 按照后使用先压入的方式把各个模块压入流中.

使用ACE_Module, ACE_Stream类

```
#include <ace/Task.h>
#include <ace/Module.h>
#include <ace/Stream.h>
#include <ace/streams.h>

typedef ACE_Task<ACE_MT_SYNCH> Task;
typedef ACE_Module<ACE_MT_SYNCH> Module;
typedef ACE_Stream<ACE_MT_SYNCH> Stream;

Task *taskOne;
Task *taskTwo;
taskOne = new Task("Task No. 1", 1);
taskTwo = new Task("Task No. 2", 3);

Module *moduleOne;
Module *moduleTwo;
moduleOne = new Module("Module No. 1", taskOne);
moduleTwo = new Module("Module No. 2", taskTwo);

Stream theStream;

if (theStream.push(moduleTwo) == -1) {
    ACE_ERROR_RETURN ((LM_ERROR, "%p\n", "push"), -1);
}

if (theStream.push(moduleOne) == -1) {
    ACE_ERROR_RETURN ((LM_ERROR, "%p\n", "push"), -1);
}
...
theStream.close();
```

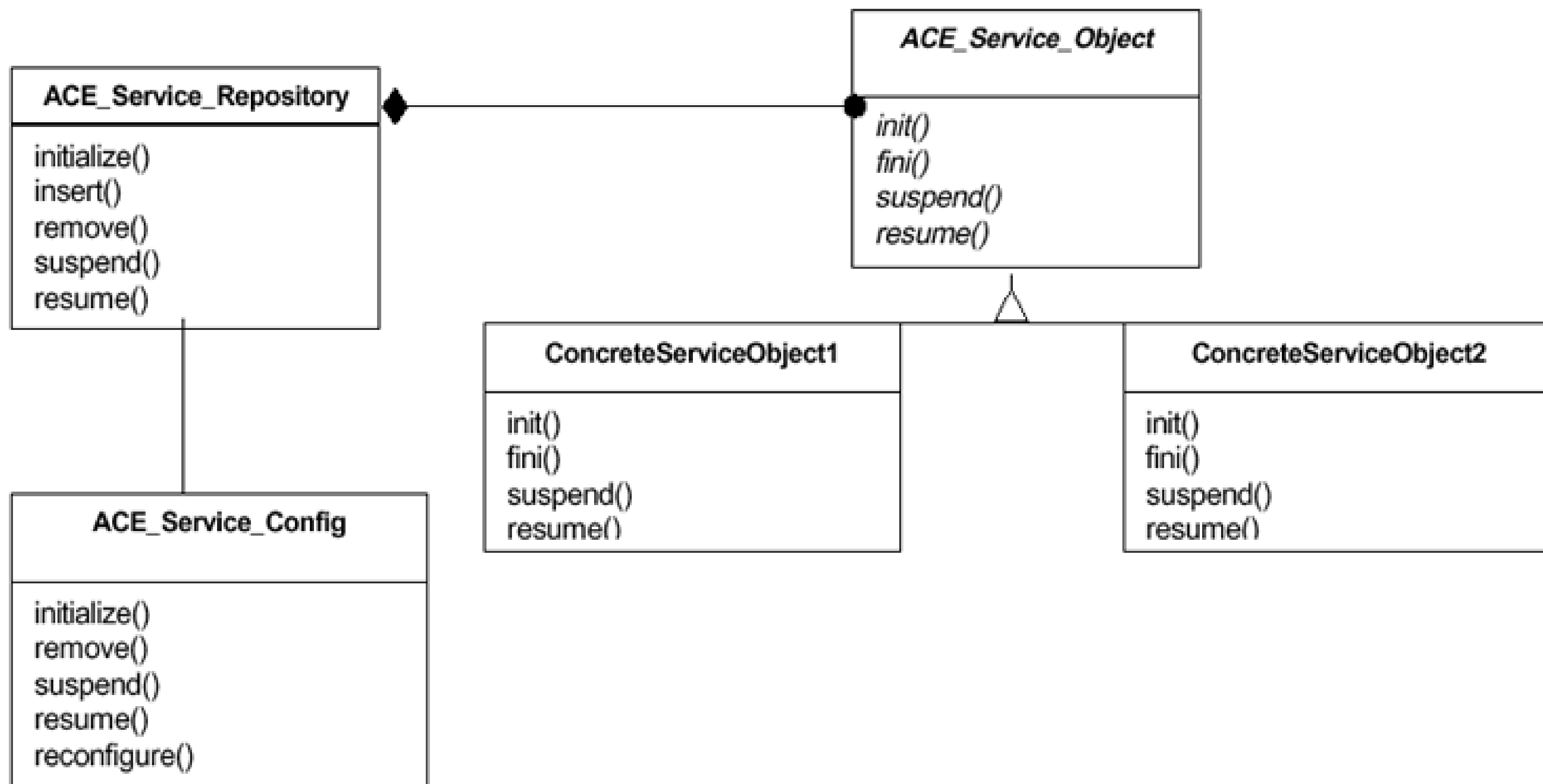
Stream示例分析

实验：

1、查错、修改、编译、运行
2、分析代码

服务配置器(Service Configurator)框架

Service Configurator框架是Component Configurator模式的一种实现



在运行时有选择地激活服务

- 静态服务：被静态链接进应用程序的服务
- 动态服务：它们是从一个或多个共享库(DLL)中链接的服务

服务配置器(Service Configurator)

服务配置器：用于服务动态配置的模式

Service Configurator中的主要配置单元是服务(service)

包含的组件:
--ACE_Service_Object
--ACE_Service_Repository
--ACE_Service_Config (svc.conf)

所有的应用服务都派生自ACE_Service_Object继承层次

ACE_Service_Object继承层次由ACE_Event_Handler和ACE_Shared_Object抽象基类组成。

ACE_Service_Object类

ACE_Service_Object包括了一些由框架调用的方法, 用于服务要启动(init()), 停止(fini()), 挂起(suspend())或是恢复(resume())时. ACE_Service_Object派生自ACE_Shared_Object和ACE_Event_Handler.

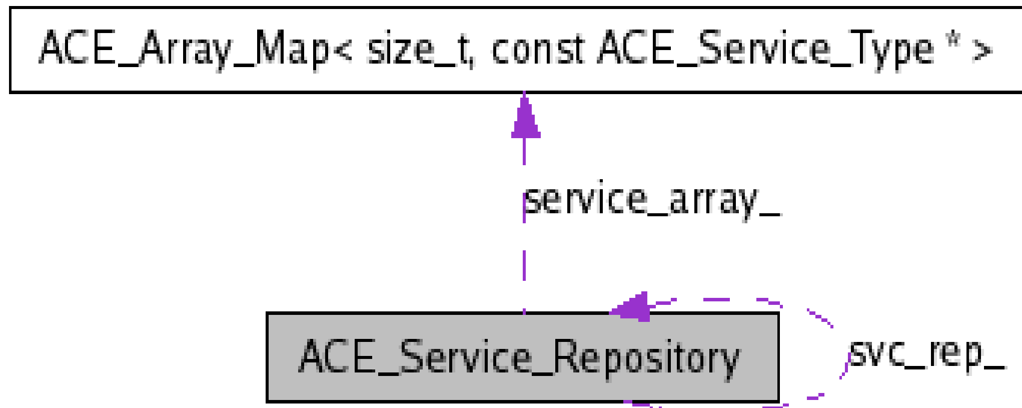
使用ACE_Service_Object

```
#include "ace/Service_Object.h"
class MyDynamicObj : public ACE_Service_Object
{
public:
      MyDynamicObj() {};
      virtual ~MyDynamicObj() {};
      virtual int init (int argc, ACE_TCHAR *argv[])
      {
       printf("MyDynamicObj::init------\n");
       return 0;
      }
   virtual int suspend()
   {
       printf("MyDynamicObj::suspend-----\n");
       return 0;
   }
   virtual int resume()
   {
       printf("MyDynamicObj::resume-----\n");
       return 0;
   }
      virtual int fini()
      {
       printf("MyDynamicObj::fini-----\n");
       return 0;
      }};
MyDynamicObj obj;
```

ACE_Service_Repository

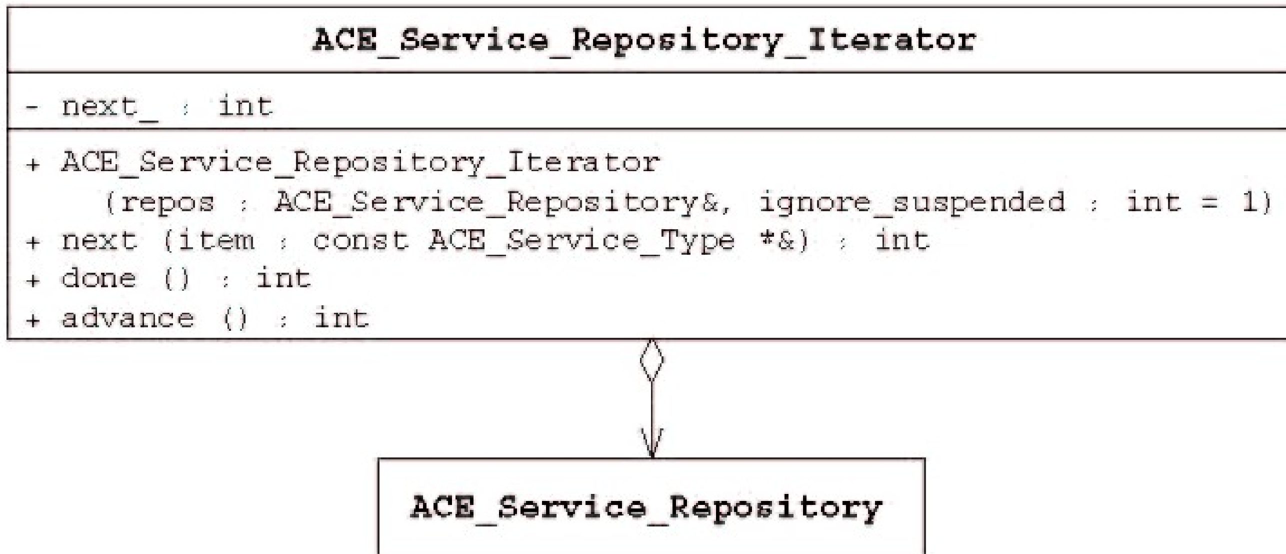
Contains all the services offered by a Service Configurator-based application.

This class contains a vector of ACE_Service_Types *'s and allows an administrative entity to centrally manage and control the behavior of application services. Note that if services are removed from the middle of the repository the order won't necessarily be maintained since the remove method performs compaction. However, the common case is not to remove services, so typically they are deleted in the reverse order that they were added originally.



ACE_Service_Repository_Iterator

- ACE_Service_Repository_Iterator 实现了Iterator模式，为应用提供一种方法,顺序地访问ACE_Service_Repository 中的ACE_Service_Type条目，而不需暴露其内部表示



- 不要在对ACE_Service_Repository 进行迭代时删除其中的条目，因为ACE_Service_Repository_Iterator还不是一种健壮的迭代器

ACE_Service_Config

ACE_Service_Config是整个服务配置器框架的应用开发接口.

当你的应用对ACE_Service_Config发出open()调用时, 服务配置器框架会读取并处理你写在服务配置文件中的所有配置信息, 随后相应地配置应用. 该文件含有所有服务对象的配置信息, 其缺省的名字是svc.conf.

Open()方法是最常用的初始化ACE_Service_Config的方式.它解析在argc和argv中传入的参数,跳过第一个参数(argv[0],它是程序名)

使用ACE_Service_Config

```
#include "ace/OS.h"
#include "ace/Service_Config.h"
#include "ace/Event_Handler.h"
#include "ace/Reactor.h"

class Time_Handler : public ACE_Event_Handler
{
public:
   virtual int handle_timeout (const ACE_Time_Value &tv, const void *arg)
   {
      ACE_Reactor::instance()->schedule_timer (this, NULL, ACE_Time_Value (2));
      ACE_DEBUG((LM_INFO, "regular timeout!!!\n"));
      ACE_Service_Config::reconfigure();
      return 0;
   }
};

int main(int argc, char *argv[])
{
   Time_Handler th;
   if (ACE_Service_Config::open (argc, argv) == -1)
      ACE_ERROR_RETURN ((LM_ERROR,
         "%p\n","ACE_Service_Config::open"),-1);

   ACE_Reactor::instance()->schedule_timer (&th, NULL, ACE_Time_Value (2));
   while(1)
      ACE_Reactor::instance()->handle_events();
}
```

ACE_Service_Config Options

选项	描述
‘-b’	使应用进程变为daemon
‘-d’	在处理指令时显示诊断信息
‘-f’	提供除缺省的svc.conf文件之外的其他含有指令的文件。可以重复这个参数来提供多个配置文件
‘-n’	不要处理static指令,这样就不再需要静态地初始化ACE_Service_Repository
‘-s’	指定用于让ACE_Service_Config重新处理其配置文件的信号。缺省使用SIGHUP
‘-S’	直接提供一条指令给ACE_Service_Config 可以重复这个参数来处理多条指令
‘-y’	处理static指令，需要对ACE_Service_Config 进行静态初始化

Service Config Directives

可由process_directives()和process_directive()处理的服务配置指令

指令	描述
dynamic	动态链接某个服务，并通过调用其init() hook方法进行初始化
static	调用init() hook方法来初始化某个被静态链接的服务
remove	完全移除某服务，调用fini() hook方法
suspend	调用suspend() hook方法使其暂停，但并不移除
resume	调用resume() hook方法继续挂起的服务
stream	初始化一列有序的层次相关的模块

两种指定ACE_Service_Config 的指令

。使用包含一个或多个指令的配置文件(svc.conf 缺省)
。以串的方式传递单个指令：ACE_Service_Config::process_directive()

svc.conf的 BNF语法

Backus/Naur Format (BNF)

```
<svc-conf-entries> ::= <svc-conf-entries> <svc-conf-entry> | NULL
<svc-conf-entry>   ::= <dynamic> | <static> | <suspend> |
                        <resume> | <remove> | <stream>
<dynamic> ::= dynamic <svc-location> <parameters-opt>
<static> ::= static <svc-name> <parameters-opt>
<suspend> ::= suspend <svc-name>
<resume> ::= resume <svc-name>
<remove> ::= remove <svc-name>
<stream> ::= stream <streamdef> '{' <module-list> '}'
<streamdef> ::= <svc-name> | dynamic | static
<module-list> ::= <module-list> <module> | NULL
<module> ::= <dynamic> | <static> | <suspend> |
             <resume> | <remove>
<svc-location> ::= <svc-name> <svc-type> <svc-factory> <status>
<svc-type> ::= Service_Object '*' | Module '*' | Stream '*' | NULL
<svc-factory> ::= PATHNAME ':' FUNCTION '(' ')'
<svc-name> ::= STRING
<status> ::= active | inactive | NULL
<parameters-opt> ::= '"' STRING '"' | NULL
```

动态服务

动态服务可以从DLL中动态加载. 它们不需要链接进程序里.这种动态加载能力允许你在运行时替换服务, 从而提供极大的灵活性.

动态服务需要的唯一与服务有关的宏是 ACE_FACTORY_DEFINE

创建ACE_DLL

。#define ACE_BUILD_SVC_DLL
这样文件会被编译成DLL
或 cl my_dll.cpp /LD -I "C:\ACE_Wrappers" -DWIN32 -link "C:\ACE_Wrappers\lib\ACEd.lib"
调用它的方式：

```
ACE_DLL dll;
int retval = dll.open (“./” ACE_DLL_PREFIX “Today”); // 前缀
Magazine_Creator mc;
mc = (Magazine_Creator) dll.symbol (“create_magazine”); // 调用方法
dll.close ();
```

在另一文件中的方法定义：

```
extern ″C″ ACE_Svc_Export Magazine *create_magazine (void);
Magazine *create_magazine (void)
{ Magazine *mag; ACE_NEW_RETURN (mag,Today,0);
return mag;}
```

演示：C:\ACE_wrappers\examples\DLL

动态服务示例分析

实验：

1、查错、修改、编译、运行
2、分析代码

ACE命名服务(Naming Service)

ACE Naming Context实例的典型应用：

- 把某个键绑定或重绑定到命名上下文中的某个值
- 解除某个条目在上下文中的绑定
- 基于某个键解析或查找某个条目
- 从上下文中获取名字、值或类型的列表
- 从上下文中获取名字/值/类型绑定的列表

属性	描述
TCP 端口	网络模式的命名服务的客户将要连接的TCP端口.
主机名	客户用这个主机名来找到网络模式的命名服务
上下文类型	指示命名上下文使用进程、节点或网络本地数据库
名字空间目录	文件系统中的某个位置, 持久的名字空间数据会存放在那里
进程名	当前进程名. 在 "process-local" 模式下是默认数据库名
数据库	在默认数据库不适用时使用的数据库的名称
基地址	允许你获取/设置底层分配器的地址, 该分配器用于在数据库中创建条目
注册表	Windows环境下的应用可以选择Windows注册表

使用 PROC_LOCAL

```
Naming_Context naming_context;
ACE_Name_Options *name_options = naming_context.name_options();

char *naming_options_argv[] = { argv[0] };
name_options->parse_args
    (sizeof(naming_options_argv) / sizeof(char*),
    naming_options_argv);
name_options->context (ACE_Naming_Context::PROC_LOCAL);
naming_context.open (name_options->context ());

Temperature_Monitor temperature_monitor (opt, naming_context);
temperature_monitor.monitor();

naming_context.close();
Return 0;
```

使用 NODE_LOCAL

```
int ACE_TMAIN (int argc, ACE_TCHAR *argv[])
    { Temperature_Monitor_Options opt (argc, argv);
    Naming_Context process_context;
     {
       ACE_Name_Options *name_options =
       process_context.name_options ();
       name_options->context (ACE_Naming_Context::PROC_LOCAL);
       ACE_TCHAR *nargv[] = { argv[0] };
       name_options->parse_args (sizeof(nargv) / sizeof(ACE_TCHAR*) , nargv);
       process_context.open (name_options->context ());
    }

    Naming_Context shared_context;
    {
       ACE_Name_Options *name_options = shared_context.name_options ();
       name_options->process_name (argv[0]);
       name_options->context (ACE_Naming_Context::NODE_LOCAL);
       shared_context.open (name_options->context ());
    }

    Temperature_Monitor2 temperature_monitor (opt, process_context, shared_context);
    temperature_monitor.monitor ();
    process_context.close ();
    shared_context.close ();
    return 0;
}
```

使用 NET_LOCAL

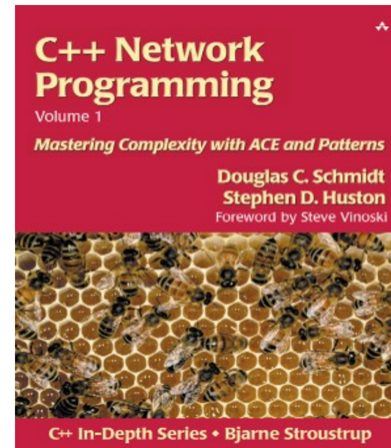
```
int ACE_TMAIN (int argc, ACE_TCHAR *argv[])
{
  Temperature_Monitor_Options opt (argc, argv);

  Naming_Context process_context;
  {
    ACE_Name_Options *name_options =
      process_context.name_options ();
    name_options->context (ACE_Naming_Context::PROC_LOCAL);
    ACE_TCHAR *nargv[] = { argv[0] };
    name_options->parse_args (sizeof(nargv) / sizeof(ACE_TCHAR*),
                              nargv);
    process_context.open (name_options->context ());
  }

  Naming_Context shared_context;
  {
    ACE_Name_Options *name_options =
      shared_context.name_options ();
    name_options->process_name (argv[0]);
    name_options->context (ACE_Naming_Context::NET_LOCAL);
    shared_context.open (name_options->context ());
  }

  Temperature_Monitor2 temperature_monitor (opt,
                                            process_context,
                                            shared_context);
  temperature_monitor.monitor ();
  process_context.close ();
  shared_context.close ();
  return 0;
}
```

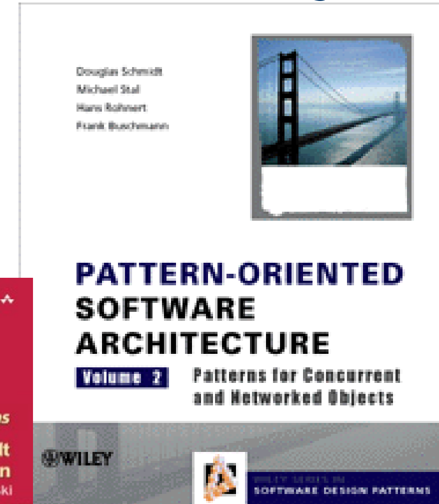
内容回顾

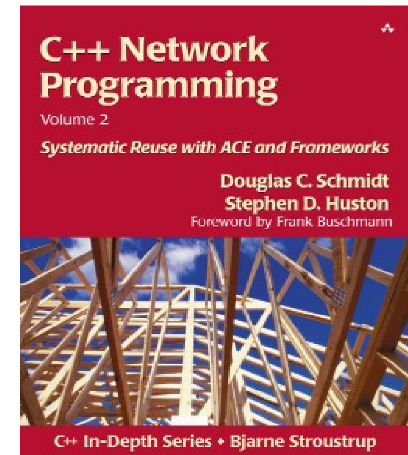
- 共享内存(Shared Memory)
- ACE流框架(Streams Framework)
- ACE服务配置框架(Service Configurator Framework)
- ACE命名服务(ACE Naming Service)

参考资料

- Patterns & frameworks for concurrent & networked objects
  - `www.posa.uci.edu`
- ACE & TAO open-source middleware
  - `www.cs.wustl.edu/~schmidt/ACE.html`
  - `www.cs.wustl.edu/~schmidt/TAO.html`







- ACE research papers
  - `www.cs.wustl.edu/~schmidt/ACE-papers.html`
- Extended ACE & TAO tutorials
  - UCLA extension, January 21-23, 2004
  - `www.cs.wustl.edu/~schmidt/UCLA.html`
- ACE books
  - `www.cs.wustl.edu/~schmidt/ACE/`

谢谢大家！

ihuihoo@gmail.com

新浪微博: http://t.sina.com.cn/huihoo
Twitter: http://twitter.com/huihoo
博客 http//blogs.huihoo.com

http://www.huihoo.com