ACE/C++ Network Programming Volume 2

@于龙

目录

[1 用于网络编程的面向对象框架 2](#_Toc418602731)

[1.1 面向对象框架综述 2](#_Toc418602732)

[1.2 比较各种软件开发和复用技术 2](#_Toc418602733)

[1.2.1 比较框架和类库 2](#_Toc418602734)

[1.2.2 比较框架和组件 2](#_Toc418602735)

[1.2.3 比较框架和模式 2](#_Toc418602736)

[1.3 将框架应用到网络编程 2](#_Toc418602737)

[1.4 ACE框架 3](#_Toc418602738)

[2 服务和配置的设计维度 3](#_Toc418602739)

[2.1 服务和服务器设计维度 3](#_Toc418602740)

[2.1.1 短持续时间服务vs长持续时间服务 3](#_Toc418602741)

[2.1.2 内部服务 vs 外部服务 3](#_Toc418602742)

[2.1.3 有状态服务 vs 无状态服务 4](#_Toc418602743)

[2.1.4 分层式/模块化服务 vs 整体式服务 4](#_Toc418602744)

[2.1.5 单服务服务器 vs 多服务服务器 4](#_Toc418602745)

[2.1.6 一次性服务器 vs 持续式服务器 4](#_Toc418602746)

[2.2 配置设计维度 4](#_Toc418602747)

[2.2.1 静态命名 vs 动态命名 4](#_Toc418602748)

[2.2.2 静态链接 vs 动态链接 4](#_Toc418602749)

[2.2.3 静态配置 vs 动态配置 5](#_Toc418602750)

[3 ACE Reactor框架 5](#_Toc418602751)

# 用于网络编程的面向对象框架

## 面向对象框架综述

在网络编程领域持续的重新发现和发明；

能力复用：

机会主义复用：程序员剪切黏贴来使用自己已有的程度代码，从而创建新的程序.

系统化复用：有意图的、预先确定的，创建并应用多用途的软件架构、模式、框架和组件.

ACE，强大的主机基础设施中间件，其设计专门考虑了系统化复用.

框架提供一组集成的、针对特定领域的结构和功能；

通过回调，框架在运行时展示出控制的反转；

框架是半完成的应用.

## 比较各种软件开发和复用技术

### 比较框架和类库

类库的费用范围是有限的，不具有控制流、协作及可变性；

框架中的类会进行协作，为一组相关应用提供可复用的架构.

类是被动的，框架是主动的，通过多种回调驱动的事件处理模式来引导控制流.

### 比较框架和组件

组件是实现特定的一个或一组服务的软件系统的一个被封装的部分.

第三代开发技术.

框架和组件之间的关系是高度协作的.

### 比较框架和模式

网络化应用的挑战：

连接管理、服务初始化、分布、并发控制、流控制、错误处理、事件循环、可靠性.

ACE可以快速地编写网络化应用，因为ACE中的框架实现了与服务访问、事件处理、并发及同步相关的核心模式.

## 将框架应用到网络编程

主机基础设置中间件框架封装了OS并发和IPC机制：

连接管理和事件处理器初始化；

事件检测、多路分离、事件处理器分派；

字节流上的消息分帧；

网络字节序和参数整解编的表示转换问题；

并发模型和并发操作的同步；

通过动态配置的服务组成网络化应用；

分层的网络化应用和服务的层次结构；

Qos属性的管理，比如调度对处理器、网络和内存的访问.

## ACE框架

Acceptor-Connector

Reactor

Proactor

Service Configurator

Stream

Task

# 服务和配置的设计维度

## 服务和服务器设计维度

服务是提供给客户的某种能力，服务器是提供服务的机制.

### 短持续时间服务vs长持续时间服务

这些时间长度反应了服务持有系统资源的时间长短，主要权衡：

当系统资源可在别处更好地被利用时仍然将其占据；

在需要某服务时对其进行重启的开销.

短持续时间服务通常使用UDP；

长持续时间服务通常使用TCP；

### 内部服务 vs 外部服务

主要权衡：

服务初始化时间、服务间的隔离、简单性；

内部服务指在相同的地址空间；

外部服务指在不同的地址空间.

主要考虑服务的健壮性，但消耗一定的性能.

### 有状态服务 vs 无状态服务

服务在各个请求之间是否维护状态，会影响复杂性和资源消耗.

### 分层式/模块化服务 vs 整体式服务

权衡因素：

服务的可复用性、扩展性和效率；

分层式/模块化服务：每一层处理总体服务的一个自包含部分；

整体式服务：没有进行层次组织的多个紧密耦合的功能块.

### 单服务服务器 vs 多服务服务器

单服务服务器：只提供一个服务；

多服务服务器：把一系列单服务服务器集成进单个管理单元.

### 一次性服务器 vs 持续式服务器

权衡服务器运行和使用系统资源的时间长短，考虑服务的预期使用频率，启动速度和配置灵活性需求.

## 配置设计维度

### 静态命名 vs 动态命名

静态命名服务：在编译或链接时把服务的名字与存在的对象代码关联在一起.

动态命名服务：推迟了对服务名和实现此服务的对象代码的关联，运行时绑定.

### 静态链接 vs 动态链接

操作系统通常支持两种动态链接：

隐式动态链接：直至方法第一次被引用为止，DLL；

显示动态链接：允许应用获取、使用及移除在DLL中定义的、与特定函数或数据相关的符号的运行时地址绑定.

### 静态配置 vs 动态配置

# ACE Reactor框架

Reactor反应器.

允许事件驱动的应用对许多不同事件源的事件做出反应，比如IO句柄，定时器以及信号.

应用定义框架所定义的挂钩方法，框架随机对其进行分派.

## 综述

ACE Reactor简化了事件驱动程序的开发.

ACE Reactor框架负责：

检测来自各种事件源的事件的发生；

将事件多路分离给其预先登记的事件处理器；

分派给处理器所定义的挂钩方法.

ACE\_Time\_Value：提供时间的可移植、规范化表示；

ACE\_Event\_Handler：抽象类，其接口定义的挂钩方法是ACE\_Reactor回调的目标；

ACE\_Timer\_Queue：抽象类，定义定时器队列的能力和接口.ACE含有多种派生自ACE\_Timer\_Queue的类，为不同的定时机制提供了灵活的支持.

ACE\_Reactor：提供一个接口，用来在Reactor框架中管理事件处理器登记，并执行事件循环来驱动事件监测、多路分离和分派.

ACE Reactor框架的力量来自它对各种操作系统上可用的事件多路分离机制之间的差异的封装，以及对框架类与应用类之间的事务分离的维持.

使不依赖于应用的事件多路分离和分派机制与依赖于应用的事件处理策略分离.

优势：

广泛的可移植性；

使事件监测、多路分离以及分派自动化；

透明的可扩展性；

增加复用并使错误减少到最小；

高效的事件多路分离；

## ACE\_Time\_Value

类能力：

它提供了一种标准化的时间表示，可在各种OS平台上使用；

可以在不同的平台时间表示之间转换；

提供运算符重载来简化基于时间的比较；

可以表示持续的时间或绝对时间；

## ACE\_Event\_Handler

ACE\_Event\_Handler是ACE中的所有反应式事件处理器的基类：

定义了输入事件、输出事件、异常事件、定时器事件以及信号事件的挂钩方法；

持有一个指针，指向管理它的ACE\_Reactor，从而使事件处理器对其事件登记和解除的正确管理变得很简单.

事件类型和事件处理器挂钩方法；

注册一个或多个事件类型：READ\_MASK、WRITE\_MASK、EXCEPT\_MASK、ACCEPT\_MASK、CONNECT\_MASK

事件处理器挂钩方法返回值：

返回0：反应器继续为此事件处理器检测和分派；

返回大于0：反应器继续为此事件处理器检测和分派，放弃控制后再次获得控制；

返回-1：停止为此事件处理器检测已登记的事件，它调用处理器的handle\_close()挂钩方法.

事件处理器惯用手法：

handler\_\*挂钩方法的执行时间保持简短，如果挂钩方法可能会长时间允许来处理请求，可考虑在ACE\_Message\_Queue中对请求进行排队；

将事件处理器的清理活动合并进handle\_close挂钩方法；

只在事件处理器的handle\_close方法中，并且只在已从反应器移除了处理器的最后一个进行了登记的事件之后调用delete this.

清理事件处理器：

ACE Reactor 只在挂钩方法返回负值，或是处理器被显示地从反应器移除时调用handle\_close.

## ACE定时器队列类

ACE定时器队列类允许应用登记派生自ACE\_Event\_Handler的时间驱动的事件处理器：

允许应用调度事件处理器，其handler\_timeout挂钩方法将在调用者指定的时间被高效而可伸缩地分派；

一次性或周期性；

可以取消某个定时器；

允许应用配置定时器队列的时间源；

可选择的定时器队列的实现：

ACE\_Timer\_Heap；

ACE\_Timer\_Wheel；

ACE\_Timer\_Hash；

ACE\_Timer\_List；

## ACE\_Reactor类

反应器初始化和析构方法；

事件处理器管理方法；

事件循环管理方法；

定时器管理方法；

通知方法；

# ACE Service Configurator框架

## 综述

配置管理层类：安装、初始化、控制以及关闭服务对象.

ACE\_Service\_Config、ACE\_Service\_Repository、ACE\_Service\_Repository\_Iterator

应用层类：该类实现具体的服务来执行应用的处理.

是ACE\_Service\_Object的后台，后者又继承自ACE\_Event\_Handler.

## ACE\_Service\_Object类

服务配置和生命周期管理涉及到以下几个方面：

初始化；

执行控制；

报告；

终止；

提供了初始化服务和关闭服务的挂钩方法；

提供了暂时挂起服务执行和恢复被挂起服务的执行的挂钩方法；

提供了一个挂钩方法来报告关键的服务信息.

## ACE\_Service\_Repository类

追踪配置进应用的所有服务；

提供插入、管理、移除服务的机制；

提供终止所有服务的机制，其顺序与它们的初始化顺序相反；

允许通过名称来定位服务；

## ACE\_Service\_Config类

它解释一种脚本语言，在运行时定位和初始化服务的实现；

可以管理静态服务和动态服务；

在运行时进行服务重配置；

# ACE Task框架

ACE Task框架提供了强大而可扩展的面向对象并发能力，比如在对象的上下文中派生线程，以及在执行在不同线程中的对象之间传送消息和对消息进行排队.

可用于实现关键的并发模式：

Active Object模式

Half-Sync/Half-Async模式

## ACE\_Message\_Queue类

生产者和消费者在同一进程中时，任务通过进程内的消息队列来交换消息.

ACE\_Message\_Queue是一种可移植的轻量级进程内消息排队机制：

允许将消息放入队列的前部、后部或基于优先级；

可从前部或后部取出；

使用ACE\_Message\_Block提供高效的消息缓冲机制，减少内存分配和数据复制；

既可以多线程化配置，也可以单线程化配置；

多线程化配置中，可以支持流控制；

允许对入队和出队指定超时时间；

可以与ACE Reactor框架集成；

可定制内存分配器；

## ACE\_Task类