TUXONE 开发指南

(针对 TUXONE 2.0)

目录

第1章、TUXONE 简介

TUXONE 是一款分布式交易中间件，同 ORACLE TUXEDO API 级别兼容。现有的 TUXEDO 应用可以不需要修改任何代码，只要在 TUXONE 环境下重新编译，就可以平滑迁移至 TUXONE 平台。

1.1、高度分布式处理

TUXONE 系统屏蔽了硬件、网络、数据库和操作系统的复杂性，提供了一套简单统一的编程接口，使程序员可以把精力集中在业务逻辑的实现上，而不必再为数据库、操作系统和网络通信协议的复杂性、异构性和可靠性担忧。

1.2、超大规模并发性

采用特别设计的网络程序架构，可支持超大规模的并发请求。经测试，单节点稳定并发数在 10K 以上。

1.3、自动化负载均衡

有别于传统式的集中负载均衡机制，TUXONE 采用分布式负载均衡算法，大大缓解中央节点的压力，可在所有可用资源之间动态均衡请求，自动寻找可用服务，隔离故障节点，确保稳定的高吞吐量。

1.4、稳健的故障管理

为了确保应用的不间断访问，TUXONE 连续监控各种组件，以防应用、交易、网络和硬件发生故障。并可以自动重新启动和停止应用服务，消除了单点故障，无论何时何地，只要客户和合作伙伴需要，各种应用总是处于可用状态。

1.5、完善的灾备机制

TUXONE 各节点之间采用实时互备机制，一旦某一节点发生故障，后备节点立刻接替故障节点，保证系统实时在线。

1.6、强大的集群模式

TUXONE 从设计之初就为集群模式做好了准备，可以说 TUXONE 先天就具备集群方式。TUXONE 的核心组件可以部署在多台机器上，用户的 SERVER 也可以同时部署到不同的机器上，它们互相协作，提高了整个系统的性能，并有效避免了单点故障。

1.7、丰富的数据类型

数据类型缓冲区是一块格式化的内存区域，它是 TUXONE 分布式应用程序之间交换数据的基本渠道。TUXONE 支持 5 种数据类型缓冲区，它们是 CARRAY、STRING、VIEW32、FML32、XML，不同数据类型缓冲区的存取方式和用途不一样，CARRAY 适合于传递图片、声音等二进制数据，STRING 适合于传递文本和字符串，VIEW32 和 FML32 适合于传递记录集，XML 适合于传递 UTF-8 的 XML 文档。

第2章、TUXONE 系统的安装

TUXONE 的核心部件可以运行在包括 LINUX、AIX、HPUX、SOLARIS 等类 UNIX 服务器平台之上，WINDOWS NT 平台暂不支持。

2.1、系统组成部件

TUXONE 系统由若干部件组成，如表格 2-1 所示。

组件	描述
ATMI Server	TUXONE ATMI 服务端运行环境，编程语言为 C/C++
ATMI Client	TUXONE ATMI 客户端运行环境，编程语言为 C/C++
txgate	TUXONE 核心进程，负责名字服务，负载均衡，集群管理，灾备恢复等核心功能
命令行工具	TUXONE 管理控制台，编译 VIEW32、FML32 工具、以及编译 TUXONE 服务器与客户端工具等

表格 2-2TUXONE 的组件构成

2.2、对系统的要求

TUXONE 安装时，磁盘的使用量一般在 15MB 左右，不同平台之间可能会有差异，TUXONE 占用的这点磁盘空间可以完全忽略不计。

TUXONE 运行时对内存的需求也不大，通常在 64MB 到 128MB 之间，每个 SERVER 都是多线程程序，所以 SERVER 数量不需要启动太多，这样对内存的要求也会比传统的多进程程序少。

TUXONE 对 CPU 的数量和主频没有特别要求。

TUXONE 对软件的要求主要就是 C/C++编译器，最好使用操作系统提供的编译器，当然也可以使

用免费的 gcc，还可以在开发机上完成编译，再搬到生产机上来运行，这时生产机上就没有必要安装编译器了。

2.3、对 IPC 的要求

TUXONE 少量使用了操作系统提供的 IPC 资源，一般使用默认配置即可。

2.4、对其他内核参数的要求

TUXONE 大量使用了 Socket，尤其是多线程服务器，会占用大量的 Socket 资源，每一个网络连接都会对应一个 Socket 资源，所以每个进程会要求最多可打开的文件数量，可通过配置 NOFILES 内核参数来改变。

2.5、安装方法

首先，通过 telnet 连接到主机，使用普通账号即可，以 REDHAT ENTERPRISE LINUX 5.0 为例，要执行的命令如下：

```
[jack@haw a]$ gunzip tuxone2.0.0.1188.linux.x86.beta.tar.gz
[jack@haw a]$ tar xvf tuxone2.0.0.1188.linux.x86.beta.tar
[jack@haw a]$ chmod +x install.sh
[jack@haw a]$ ./install.sh

Please input install directory:/home/jack/tuxone2.0

The directory /home/jack/tuxone2.0 does not exist!

Do you want to create it?(Y/N default:Y):y

Begin to install the software...

Congratulations. TUXONE has been successfully installed to:

        /home/jack/tuxone2.0
```

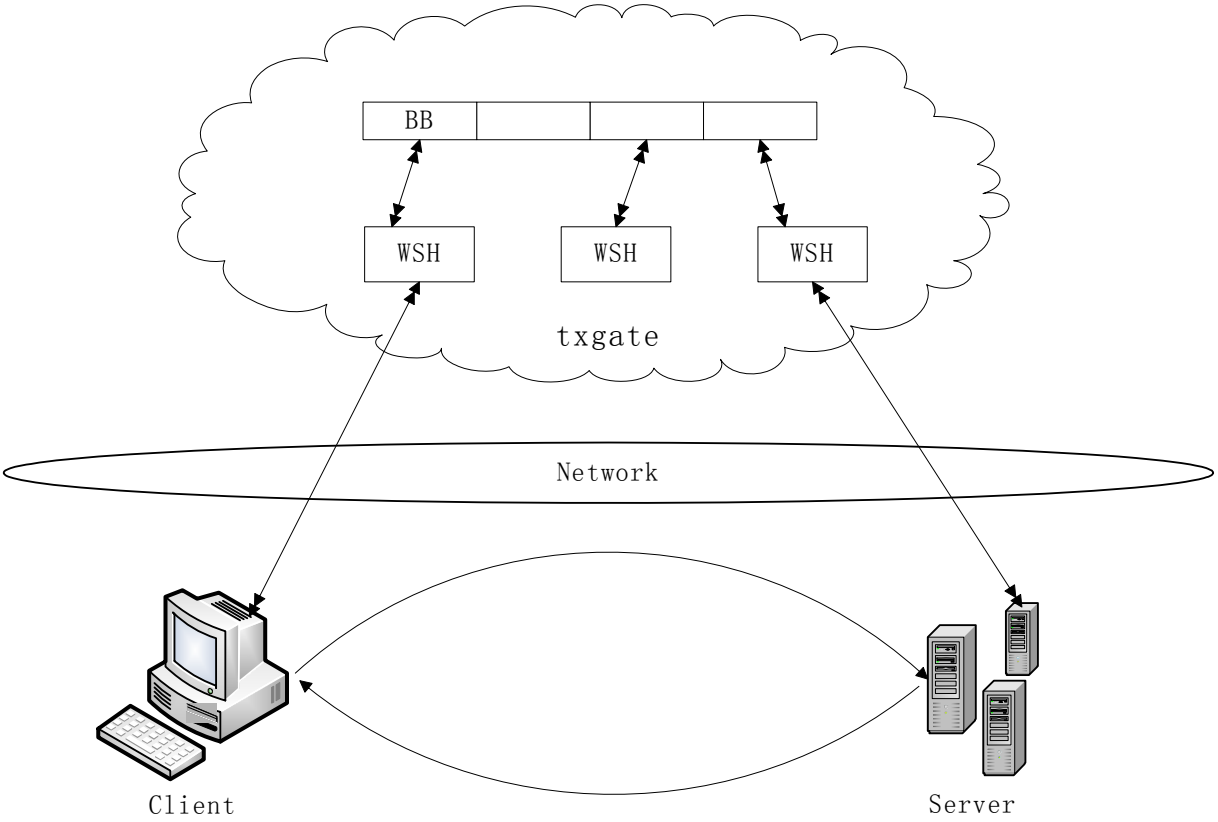
TUXONE 已经成功安装到了/home/jack/tuxone2.0 目录下，接着执行以下命令来配置 TUXONE 运行环境

```
[jack@haw a]$ cd /home/jack/tuxone2.0
[jack@haw tuxone2.0]$ . ./tux.env
```

第3章、TUXONE 系统的体系结构

3.1、整个系统的体系结构

在 TUXONE 系统中，存在三种进程，Gate 进程，Client 进程，Server 进程。其中 Client 与 Server 可以是同一个进程。Gate 进程是整个 TUXONE 系统中的关键进程，它负责名字服务，负载均衡，集群管理等关键功**能，整个**系统如图所示

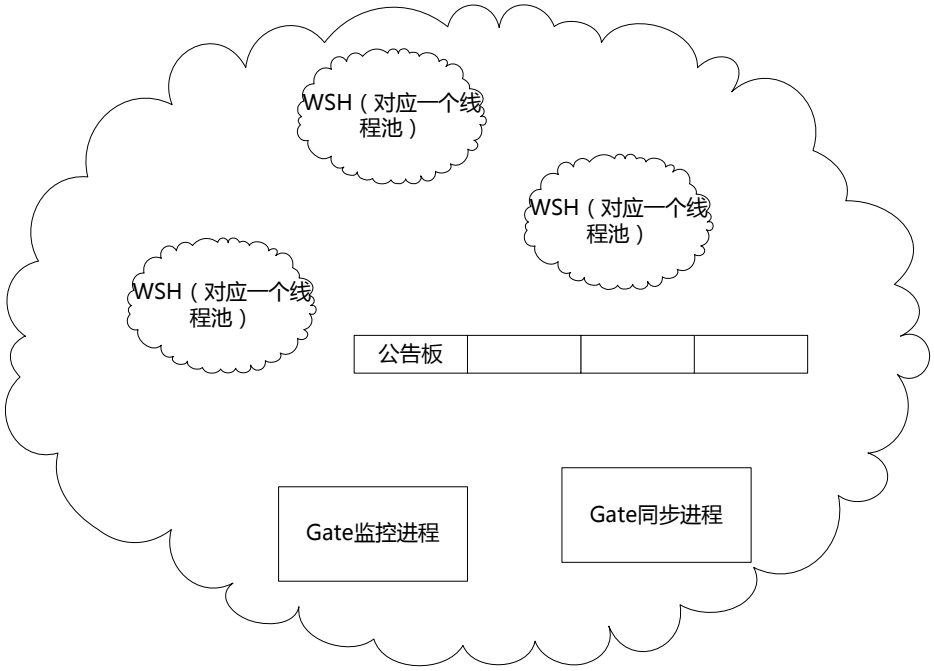


图表 3-1TUXONE 整体架构

当 Server 启动后，会将 Server 自身需要公布的 Service 信息注册到 Gate 上，Client 调用 tpinit 连接服务器时，首先连接 Gate，从 Gate 下载相应的服务信息（服务信息包含相应 Server 的监听地址），然后 Client 调用 tpcall，直接向对应的 Server 发起调用。

3.2、Gate 组件的工作原理

Gate 在整个系统中处于最关键的地位，它负责名字服务，负载均衡，集群管理，灾备处理等功能。Gate 由若干个进程组成，如图所示。



图表 3-2Gate 的组成

Gate 使用公告板来提供命名服务。公告板是一块共享内存区域，它保存着服务进程、服务、运行环境的配置和统计信息。为了便于快速访问，在运行时系统中，它会被复制到 TUXONE 系统中的每一个成员节点上。

Gate 真正对外提供服务的是 WSH 进程池，WSH 进程池由多个 WSH 进程组成，进程数量会随着并发大小自动增减。每个 WSH 进程是由一个网络线程池来对外提供服务，线程池中线程数量会随着当前 WSH 进程的负载大小而自动调整。Gate 中同时使用了进程迟与线程池技术，大大增强了其对外提供网络并发访问的能力。

Gate 监控进程主要负责调整 WSH 进程池中进程数量 监控 Gate 同步进程。当有进程意外退出时，Gate 监控进程负责重启。

Gate 同步进程是用来与其他 Gate 同步公告板信息的，它可以同时监控多个 Gate 的公告板信息变化，当有任何 Gate 公告板发生变化时，Gate 同步进程负责更新本地公告板，使整个 TUXONE 系统中各个节点的公告板信息同步更新。

第4章、TUXONE 应用系统的配置

每个 TUXONE 应用程序都有一个配置文件，它告诉 TUXONE 系统，应该如何配置和部署服务器进

程，应为为服务器进程提供什么样的运行环境。这个配置文件的作用和 J2EE 规范中的部署描述符是一样的，它告诉 TUXONE 系统（相当于 J2EE 规范中的容器），应该如何配置和调度服务进程（相当于 J2EE 规范中的 EJB 组件）。

可以通过环境变量、默认配置文件、命令行指定配置文件、命令行配置项来配置 TUXONE 应用程序，它们的优先级如下：

- 命令行配置项优先级最高；（例如 txgate -D tuxone.gate.listener.addr=0.0.0.0:26000）
- 命令行指定配置文件次之；（例如 txgate -c gate.config）
- 默认配置文件最低；（通过 TUXCONFIG 环境变量指定）

其中环境变量中的配置与其他形式不冲突。

4.1、环境变量

TUXONE 应用程序在建立、启动和运行的过程中，要从环境变量中读取一些配置参数。下面就意表格的形式对这些变量的含义和用法进行说明，如表格所示。

环境变量	含义和用途
TUXDIR	TUXONE 系统的安装路径。只要设置了 TUXDIR ,TUXONE 的 build 系统命令就会自动到$TUXDIR/include 和$TUXDIR/lib 目录下去找头文件和库文件。用法举例（UNIX）:TUXDIR/=/usr/tuxone2.0;eoport TUXDIR
TUXCONFIG	应用程序配置文件名。
LD_LIBRARY_PATH SHLIB_PATH LIBPATH	这些变量的作用都是一样的，用于指定除了$TUXDIR/lib 目录以外的其他公共库路径 LD_LIBRARY_PATH 用于一般 UNIX、LINUX 系统，SHLIB_PATH 用于 HP-UX， LIBPATH 用于 AIX.
PATH	PATH 的作用就不用解释了，它应该包含 TUXONE 的 bin 目录。用法（UNIX 平台）： PATH=$PATH:$TUXDIR/bin
CC、CFLAGS	仅用于 UNIT 平台，指定 C 编译器和编译选项，不是必须的。如果没有使用操作系统默认的 C 编译器（如使用 gcc）就需要指定 CC 变量：CC=gcc;export CC
TUXLOGDIR	指定 TUXONE 应用程序运行过程中产生的日志文件目录

表格 4-1TUXONE 环境变量

4.2、GATE 配置说明

txgate 是 tuxone 系统中的核心进程，负责名字服务，负载均衡等核心功能，其配置参数如表格所示：

配置项	默认值	含义和用途
tuxone.gate.listener.addr	0.0.0.0:26000	Gate 对外公布的监听地址

tuxone.gate.other.addr	NONE	Gate 其他 Gate 地址
tuxone.gate.sync.interval	12	Gate 之间同步公告板间隔
tuxone.gate.concurrent.max	512	Gate 支持的最大并发数
tuxone.gate.concurrent.lowWater	128	Gate 并发低水位
tuxone.gate.concurrent.highWater	2048	Gate 并发高水位
tuxone.gate.housekeeping.interval	10	Gate housekeeping 工作间隔时间
tuxone.gate.bbl.berthMax	128	公告板支持的最大泊位数（存储 Server 信息与 Service 信息）
tuxone.gate.wsh.min	1	WSH 进程最小数
tuxone.gate.wsh.max	32	WSH 进程最大数
tuxone.gate.wsh.accessMax	512	每个 WSH 进程允许被访问的最大并发数
tuxone.gate.wsh.threadPool.threadMin	3	WSH 线程池最小数
tuxone.gate.wsh.threadPool.threadMax	128	WSH 线程池最大数
tuxone.gate.wsh.threadPool.threadMaxIdle	10	WSH 线程池最大空闲数
tuxone.gate.wsh.serverValidTime	30	WSH housekeeping 线程工作间隔时间
tuxone.gate.wsh.housekeeping.interval	10	WSH housekeeping 线程工作间隔时间
tuxone.gate.wsh.app.heart.interval	12	WSH 向 APP 下发的心跳间隔参数
tuxone.gate.log.enable	true	是否开启日志打印
tuxone.gate.log.level	4	日志打印级别
tuxone.gate.log.printFile	true	是否打印至文件
tuxone.gate.log.printConsole	false	是否打印至标准输出

表格 4-2Gate 配置项说明

4.3、应用配置说明

配置项	默认值	含义和用途
tuxone.app.gate.addr	127.0.0.1:26000	Gate 对外公布的监听地址
tuxone.app.server.listener.addr	0.0.0.0:0	SERVER 对外公布的监听地址
tuxone.app.server.concurrentMax	512	SERVER 进程的最大并发数
tuxone.app.server.maxLongConnections	256	SERVER 最大长连接数
tuxone.app.server.threadMode	MT	SERVER 网络线程模型
tuxone.app.server.threadPool.threadMin	3	SERVER 线程池最小数
tuxone.app.server.threadPool.threadMax	128	SERVER 线程池最大数
tuxone.app.server.threadPool.threadMaxIdle	10	SERVER 线程池最大空闲数
tuxone.app.client.cachedConnections	128	CLIENT 缓存的网络连接
tuxone.app.heart.interval	3	APP 心跳间隔（如果连接上 Gate，则从 Gate 上获取此参数）
tuxone.app.buffer.cachedSize	1024000	APP Buffer cached size（=0 表示无限制）
tuxone.app.socket.rwtimeout	90	app socket 读写超时时间
tuxone.app.log.enable	true	是否开启日志打印
tuxone.app.log.level	4	日志打印级别
tuxone.app.log.printFile	true	是否打印至文件
tuxone.app.log.printConsole	false	是否打印至标准输出

表格 4-3 应用配置项说明

4.4、灾备与集群方式配置说明

如何配置两个 Gate 为互备模式，假设 Gate1 监听的地址为 192.168.0.100:26000。Gate2 监听的地址为 192.168.0.100:27000，则通过以下方式启动 Gate，即可成为互备方式。

```
txgate -D tuxone.gate.listener.addr=192.168.0.100:26000 \
       -D tuxone.gate.other.addr=192.168.0.101:27000
txgate -D tuxone.gate.listener.addr=192.168.0.101:27000 \
       -D tuxone.gate.other.addr=192.168.0.100:26000
```

另外，Client 与 Server 的启动也需要配置相应的 Gate 地址，才能起到灾备作用

```
client -D tuxone.app.gate.addr=192.168.0.100:26000;192.168.0.101:27000
server -D tuxone.app.gate.addr=192.168.0.100:26000;192.168.0.101:27000
```

集群方式部署同灾备类似，假设有三台机器需要以集群方式部署 TUXONE 系统，则配置方式如下

```
txgate  -D tuxone.gate.listener.addr=192.168.0.100:26000 \
        -D tuxone.gate.other.addr=192.168.0.101:27000;192.168.0.102:28000
txgate  -D tuxone.gate.listener.addr=192.168.0.101:27000 \
        -D tuxone.gate.other.addr=192.168.0.100:26000;192.168.0.102:28000
txgate  -D tuxone.gate.listener.addr=192.168.0.102:28000 \
        -D tuxone.gate.other.addr=192.168.0.100:26000;192.168.0.101:27000
```

```
client -D tuxone.app.gate.addr=192.168.0.100:26000;192.168.0.101:27000;192.168.0.102:28000
server -D tuxone.app.gate.addr=192.168.0.100:26000;192.168.0.101:27000;192.168.0.102:28000
```

第5章、TUXONE 的缓冲区

跨平台通信应用程序一般使用缓冲区来在两个通信实体之间交换数据。以 UNIX 下 SOCKET 编程为例，客户进程通常使用 char sendBuf[1024]来分配一个大小为 1KB 的缓冲区，然后将要传递的数据复制到 sendBuf 首地址指向的缓冲区中，最后调用 write()函数把缓冲区内容写到发送窗口。运行在服务器上的守护进程检查到 socket 上有数据读取时，就调用 read()函数从接收窗口读取数据。在数据发送和接收的过程中，程序员需要考虑并解决下面一些问题。

- 缓冲区大小。由于在设计时无法预知运行时可能产生的消息的大小，通常定义一个 MAX_BUFSIZE 常量来指定最大缓冲区长度，程序中使用 char sendBuf[MAX_BUFSIZE]来静态分配发送和接收缓冲区。这印出来一个问题就是，MAX_BUFSIZE 定义多大比较合适。

MAX_BUFSIZE 定义得太小会影响性能，定义得太大可能会造成内存浪费。

- 字节序问题。大多数数据通信发生在异构系统之间，这时就不得不考虑字节序的问题，否则通过网络传输整数将会产生混乱。有的操作系统使用小端字节序（LITTLE_ENDIAN，低序字节存储在起始地址），如 AIX，WINDOWS 等；有的操作系统使用大端字节序（BIG_ENDIAN，高序字节存储在起始地址），如 SUN SOLARIS。通常的做法是在发送端将主机字节序转化成网络字节序，在接收端再把网络字节序转化成主机字节序。
- 字符集和编码问题。有的系统只支持单字节字符集（SBCS），它的所有字符都只有一个字节的长度，如 ASCII。有的系统支持多字节字符集（MBCS），它包含的字符中有单字节长的字符，也有多字节长的字符。有的系统支持统一字符集（UNICODE），在 UNICODE 编码标准中的所有字符都是双字节。
- 数据压缩问题。大多数通信系统中交换的数据都不大，因此都不涉及到数据压缩问题。而在有的系统中，通信实体之间常常需要交换大量的数据流，这时使用压缩算法将大大提供性能。
- 加密/解密问题。如果安全对应用来说十分重要，这时就需要考虑采用某种算法来进行数据加密和解密。

对于一个普通的程序员来说，要完全考虑以上这些因素几乎是不可能的。然而在 TUXONE 系统中，这些原本应该由程序员考虑的因素都成为了“平台特性”，这就使程序员可从底层通信细节实现中解放出来，把精力集中在应用逻辑上。TUXONE 就是这样，它以自身实现的复杂性换来了应用实现的简单。

TUXONE 的客户端与服务端之间的数据传送是通过数据缓冲区来进行的。TUXONE 数据缓冲区主要包括 CARRAY、STRING、VIEW32、FML32，XML 数据缓冲区。

在 TUXONE 中客户端与服务端之间进行数据交换的缓冲区(如 :tpcall()中的输入 ,输出缓冲区等），都要用 TUXONE 自己提供的 API 进行操作。不能采用 C 语言的函数（如：mallloc()，free 等分配）分配释放这些缓冲区。同时在程序中要自己管理这些缓冲区，像 C 语言中的缓冲区一样，在用 tpalloc() 分配一块缓冲区之后，在不在需要该缓冲区时，用 tpfree 释放掉。

5.1、CARRAY 缓冲区

CARRAY 是定长的字符串缓冲区，它与 C 语言中的字符缓冲区是等价的。NULL 字符一般被认为是字符串的结束符，但在 CARRAY 缓冲区中，NULL 是有意义的，这就是为什么使用 CARRAY 缓冲区时，需要指定长度的原因。TUXONE 原封不动地将 CARRAY 缓冲区传递给目的主机，而不对其中的数据进行任何解释。也不进行编码/解码处理，所以它比较适合于传输二进制数据，如位图、视频剪辑等。由此可见，CARRAY 缓冲区不支持 DDR 是理所当然的，CARRAY 最大的特点就是速度快，效率高。

CARRAY 缓冲区使用方式，如代码清单所示：

```
char *sendbuf = NULL;

sendbuf = (char *) tpalloc("CARRAY", NULL, 1024);
```

```
    if(sendbuf == NULL) {
        (void) fprintf(stderr, "Error allocating send buffer\n");
        return -1;
    }

    tpfree(sendbuf);
```

5.2、STRING 缓冲区

在 STRING 缓冲区中，NULL 字符被认为是字符串的结尾，这个特征是 STRING 与 CARRAY 的主要区别之一，这也是为什么 STRING 缓冲区不必指定长度的原因，因为从缓冲区的首地址开始，到第一个 NULL 字符之间的数据被认为是缓冲区的内容。STRING 缓冲区是自描述的，因此在字符集不同的主机之间传输时，TUXONE 系统会自动进行数据转换。STRING 缓冲区不支持编码/解码特性，也不支持 DDR。STRING 缓冲区具有使用简单，效率高的特点，适合于跨平台传输大量的文本数据，也是实际编程中用得最多的缓冲区类型。

STRING 缓冲区使用方式，如代码清单所示：

```
    char *sendbuf = NULL;

    sendbuf = (char *) tpalloc("STRING", NULL, 1024);
    if(sendbuf == NULL) {
        (void) fprintf(stderr, "Error allocating send buffer\n");
        return -1;
    }

    tpfree(sendbuf);
```

5.3、VIEW32 缓冲区

VIEW32 缓冲区允许在异构平台之间传输 C 语言的结构(structures)。使用 tpalloc 来分配 VIEW32 缓冲区时，需要使用 subtypes 参数来指定和它关联的 C 语言结构名。VIEW32 支持 DDR，在不同的主机之间传递时，还支持自动编码/解码特性。

使用 VIEW32 缓冲区时，需要先定义一个描述文件，再通过 TUXONE 系统提供的 VIEW32 编译器 viewc32 生成一个.h 文件和一个扩展名为.V 的二进制文件，其中.h 文件定义了 VIEW32 对应的 C 语言结构，.V 文件定义了只有 TUXONE 系统才能识别的运行时描述文件，需要加到客户机和服务器的环境变量 VIEWFILES32 中，另外还需要设置 VIEWDIR32 环境变量来指定.V 文件所在目录。

5.4、FML32 缓冲区

FML（Field Manipulation Language）是一组用于定义和管理字段缓冲区（Fielded Buffers）的 C 语言函数，它提供了一种更为高层的内存访问方法。字段缓冲区是一块格式化的内存区域。

FML32 字段支持的数据类型有：short、long、float、double、char、STRING、CARRAY。对于需要指定长度的字段，FLDID32 后面会自动添加一个 FLDLEN 域来保存字段的长度。

FML32 字段支持 DDR，在异构平台之间传输时支持编码和解码特性。FML32 缓冲区是可变长的，当使用 tpalloc()分配时，如果不指定长度，则默认分配的大小是 1024KB。

另外，程序员只能通过 FML32 函数来存取 FML 缓冲区。分配缓冲区代码如下所示

```
char *sendbuf = NULL;

sendbuf = (char *) tpalloc("FML32", NULL, 1024);
if(sendbuf == NULL) {
      (void) fprintf(stderr, "Error allocating send buffer\n");
      return -1;
}

tpfree(sendbuf);
```

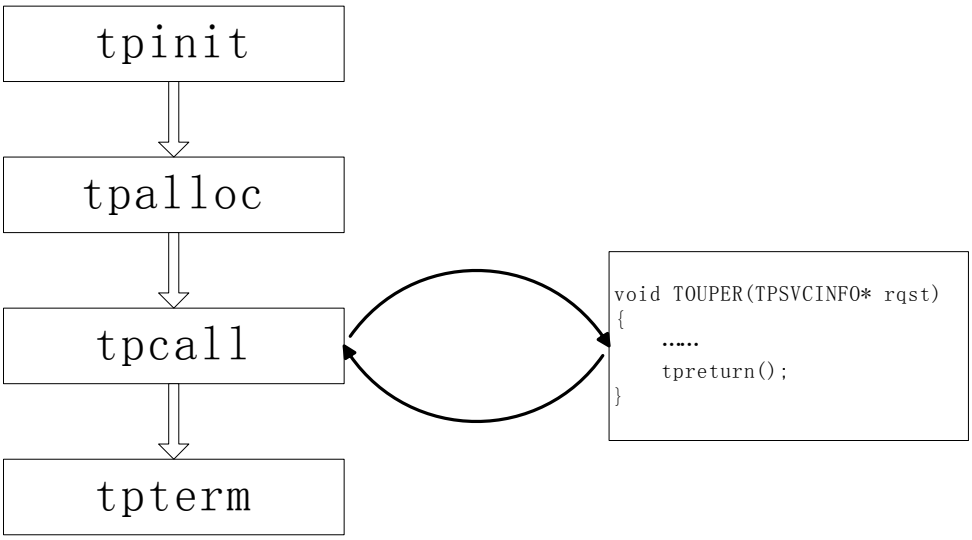
5.5、XML 缓冲区

TUXONE 内部集成了 Apache Xerces C++ Parser 2.8，这样 TUXONE 服务器和客户端就可以直接分析 XML 文档了，分配方式同 CARRAY 类似。

第6章、TUXONE 的通信方式

6.1、同步请求/应答方式

在同步请求/应答方式中，客户端使用 tpcall 向远程服务器(由 TUXONE 系统根据公告板信息确定）发送服务请求，此时客户端将传送请求服务的名字、用于请求服务的输入参数和输出参数。tpcall 发出后，客户的数据被传送至服务器，得到相应的服务处理。在此方式下，服务器处理请求时，客户端将等待，不继续运行，直到服务器返回相应结果。



图表 6-1 同步通信

6.2、异步请求/应答方式

注：此功能将在下一版本增加。

6.3、会话通信方式

注：此功能将在下一版本增加。

第7章、TUXONE 系统的服务器编程

7.1、编译服务器

TUXONE 提供了 buildserver 命令来编译服务器进程。事实上 buildserver 只完成预编译，它会调用当前操作系统中已经安装的默认 C 编译器来完成进一步的编译和连接，最终生成可执行代码，这就是为什么在安装 TUXONE 时，要确保已经安装了 C 编译器的原因。

建议使用与操作系统一起发布的 C 编译器。由于在大多数 UNIX 平台(比如 AIX、SOLARIS、HP-UX 等）上 C 编译器不是免费的，所以在没有配套编译器的情况下，也可以使用免费的 gcc。

buildserver 命令的使用方法如下：

```
buildserver [-s services] [-v] [-o outfile] [-f firstfiles] [-l lastfiles]
```

- -v 表示打开 VERBOSE 模式，即编译过程中打印更多的信息。
- -s services 定义服务与函数的映射关系。通常情况下，函数名与服务名一致。

- -f firstfiles 指定要优先于 TUXONE 系统库之前连接的文件名。则文件名之间应以空格分割，并用引号把整个串引起来（如-f “f1.c f2.c f3.c”），也可以对每个文件使用一个-f 选项（如-f f1.c –f f2.c –f f3.c）
- -l lastfiles 指定要在 TUXONE 系统库之后连接的文件名，用法同-f。
- -o outfile 指定要生成的可执行文件名。

第8章、TUXONE 系统的客户机编程

8.1、编译客户机

TUXONE 系统提供了 buildclient 命令来编译 C 语言客户机程序。与 buildserver 一样 ,buildclient 也只完成预编译，它会调用当前操作系统中已经安装的默认 C 编译器来完成进一步编译和连接，最终生成可执行代码。如果 TUXONE 客户端不是用 C 语言编写的，而是使用 Visual Basic 这样的快速开发工具来编写的，那么编译工作就由 IDE 来完成，与 buildclient 没有什么关系。

buildclient 命令的使用方法如下：

```
buildclient [-v] [-o outfile] [-f firstfiles] [-l lastfiles]
```

buildclient 中选项的含义与 buildserver 完全一致。

第9章、TUXONE 系统的线程管理与连接管理

线程（或一个进程中一小段连续控制流）也称为轻量级进程，它通过与其它线程共享基础部件减少资源开销。线程是轻量级的，因此一个进程中可以包含许多线程。利用多线程可在应用程序中提供并发能力并改善性能。利用多个线程同时执行多个独立的计算，可以使应用程序结构更有效率。例如，数据库系统可以在支持多个用户互动操作的同时执行几个文件和网络操作。虽然可以将软件编写成从一个请求到另一个请求异步移动的一个控制线程，但是，如果将每个请求编写为一个单独序列，而由基本系统处理不同操作的同步交错，就可以简化代码。处理不同操作的同步交错，就可以简化代码。

9.1、关于线程池

网络服务器的线程池模型会定义可分配用于处理客户端请求的最大线程数。每个客户端请求会得到一个工作线程，但是仅用于该特定请求的时段。请求完成后，分配给该请求的工作线程就会放回可用线程池内，准备重新分配，处理以后来自任何客户端的请求。

线程池模型中，线程是根据向服务器发出的请求的通信量分配的。这意味着同时向服务器发出很多

请求的非常活跃的客户端会分配到多个线程（保证请求得到迅速执行），而不活跃的多个客户端可能共享一个线程，但是其请求仍能够立即得到处理。另外，这样可以减少与创建和撤消工作线程相关的资源开销，因为线程得到重新利用而不是撤消了，而且线程可以分配给多个连接。

TUXONE 根据并发客户端请求的数量动态分配线程池中的线程数，从而节约系统资源。如果客户端很活跃，可以分配线程满足其需要。如果线程使用效率不高，TUXONE 会释放它们，只保留足够的线程，满足客户端的当前需要。这样使服务器中的线程数始终保持在最佳状态。线程池的大小根据服务器活动情况变化，而且可以按照特定分布式系统的需要自由配置（无论是在执行前，还是在执行后）。

TUXONE 中的用户应用 SERVER 端使用了线程池模型，用户可以通过以下参数来配置线程池：

- tuxone.app.server.threadPool.threadMin
  线程池中最小线程数
- tuxone.app.server.threadPool.threadMax
  线程池中最大线程数
- tuxone.app.server.threadPool.threadMaxIdle
  线程池中最大空闲线程数

每次收到客户端请求时，都会尝试从线程池分配线程来处理请求。如果这是第一个客户端请求且池为空，则创建一个线程。同样，如果所有线程均忙，就会创建新线程来处理请求。服务器可以定义可分配用于处理客户端请求的最大线程数。如果池中没有线程可用且已经创建最大线程数，请求就会受到阻挡，直到有当前正在使用的线程被释放回池。

9.2、关于网络连接管理

Client 与 Server 之间的网络连接通常采用长连接方式，当并发量超过 Server 配置参数 tuxone.app.server.maxLongConnections 的值后，后续的网络连接请求将自动变为短连接，也就是说 Client 与 Server 之间采用长短结合的方式，并发量小的情况下采用短连接，并发量大的情况下采用长连接。

Client 本地会有一个缓存的网络连接池，连接的最大数量由 tuxone.app.client.cachedConnections 决定。超过配置数量后，新建立的连接自动变为短连接，即调用完成后，自动断开连接。