Connection 工程技术文档

1. 工程简介

该工程的目的是在学习网络套接字编程知识的同时，利用书中介绍的理论知识对网络套接字进行面向对象的封装，使得套接字编程更加方便快捷。

目前封装的东西包括：基于 IPV4 的 TCP 套接字（ ConnectionIPV4），套接字异常类（ConnectionException），利用 select I/O 模型构建的并发服务器（SelectServer），利用 poll I/O 模型构建的并发服务器（PollServer），以及利用 epoll I/O 模型构建的并发服务器（EPollServer）。从当前的想法来看，以后需要逐步加入的东西包括：UDP 套接字，UNIX 域的流式和数据报式套接字，Windows 端套接字，Windows 端的 select 并发服务器，完成端口并发服务器。

封装的目的在于更加方便的构建应用程序。因此，利用上面封装的类库，在当前工程中加入了几个 demo 程序的编写。这些 demo 中既包括服务器端，也包括客户端。目前有两个类型的 demo，一个是基于字符串回射功能的客户端和服务器，另一个是一个简单的网络聊天室的 demo，同样包括客户端和服务器的实现。

2. 文件目录说明

include 封装库的头文件目录
src 封装库的实现文件目录
lib 编译完成的库文件目录
demo 利用上面的封装库的例子程序目录
bin 编译完成的例子程序的可执行文件目录
doc 文档目录
Makefile 整个工程的 Makefile
Makefile-exe 编译可执行文件需要的 Makefile
Makefile-so 编译动态库文件需要的 Makefile

3. 封装类说明

3.1 ConnectionException

我们在编写一般的函数都是通过函数返回值来反映函数的执行情况，但是 C++鼓励人们充分利用 C++的异常机制来实现编程。C++为了实现异常机制，改造了一般的 C 语言的函数调用方式，在函数调用栈中加入了部分数据以正确的实现异常的抛出和接收。

异常的介绍是一个比较大的工程，这里不再进行详细的描述。以后我会在博文中加入对异常的底层实现，如何设计函数对异常的抛出，如何捕获异常等。

我们这个工程仅仅加入了一个异常类，当网络套接字捕获了无法处理的错误情况时，会抛出这样的一个异常类的引用，供用户获取该错误的字符串表述。注意其中的析构函数和 what 函数的声明方式，这两个虚函数的声明方式是在 std::exception 中定义的。

```
class ConnectionException : public std::exception{
public:
    ConnectionException(const char *exceptStr);
    virtual ~ConnectionException() throw();
    virtual const char* what() const throw();
private:
    std::string mExceptionStr;
};
```

3.2 ConnectionIPV4

该类是对基于 TCP 的 IPV4 套接字的封装，该类封装了对套接字的一般操纵接口。该类的头文件如下：

```
class ConnectionIPV4{
public:
    const char*  GetIP(void);
    int          GetPort(void);
    int          GetSockfd(void);
    const char * GetLastError(void);
    void         SetAddrReuseable(void) throw(ConnectionException&);
    //server&client use
    void InitialSocket(void) throw(ConnectionException&);
    void Close(void) throw(ConnectionException&);
    //server use
    void BindIpPort(const char *ipStr, int port) throw(ConnectionException&);
    void ListenForClient(int backlog) throw(ConnectionException&);
    Bool AcceptClient(ConnectionIPV4& clientConn);
    //client use
    bool ConnectToServer(const char *serverIpStr, int serverPort);
    bool             SendData(void *data, int n);
    bool             RecvData(void *buffer, int n);
    bool         operator<(const ConnectionIPV4& rhs) const;
private:
    struct sockaddr_in mSockAddr;
    bool mClosed;
    int mSockfd;
    std::string mLastErrorStr;
};
```

该类的数据成员较为简单，一个套接字的地址结构成员 mSockAddr，一个辅助变量 mClosed，一个套接字文件描述符 mSockfd，一个字符串 mLastErrorStr，用来反映当错误发生的时候的错误描述字符串。

函数成员则是简单的对套接字功能函数的封装，初始化套接字就是创建一个套接字文件描述符，绑定则是 bind 函数进行绑定等等，发送数据和接收数据使用的是 read 和 write 系统调用，但是功能略有不同，这里会保证全部发送或者接收，或者返回 false 表示没有发送成功，让用户对该客户进行处理。

这里需要讨论的是，这里的函数为什么进行这样的设计，有些函数抛出了异常，而另外一些函数则是没有抛出异常而是返回一个布尔值用来反映当前函数执行是否成功。我在这里的想法也经历了一些变化：当时刚刚在看函数异常的实现，当时一个文档的作者建议读者尽可能的使用异常来返回错误而不是返回值的方式。我轻信了这种方式，于是在所有套接字函数调用不成功的时候就返回一个异常。当时，后来在编写回射服务器的过程当中，我就发现简单一个客户的异常就会导致整个服务器终止，实在不是恰当的实现方式。

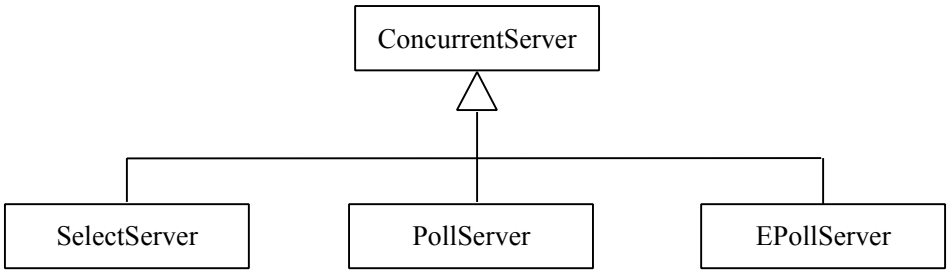
后来，我就考虑哪些函数需要通过返回值来告知用户，于是得出了这样的结论：

1. 知道原因的错误绝不抛出异常，通过返回值供用户进行处理
2. 不知道原因的错误才会返回异常，会导致应用程序退出

网络套接字编程过程中经常用到的一个东西是 errno，当函数调用不成功的时候可以通过查看该数据可以知道到底发生了什么错误。因此，我的处理方法是：连接到服务器的函数，发送和接收数据的函数不需要抛出异常，仅仅通过一个字符串来告知用户发送的错误就好。因此在每个 ConnectionIPV4 中加入了一个 string 对象，用于反映刚刚发生错误的 errno 值。这些具体的错误可以参见 ConnectionIPV4 的代码，同时也可以参考 UNP V1，都是在这里获取的并加以总结。

另一个注意的地方时，这里对小于号操作符进行了重载，目的是需要将 Connection 类加入到 map 容器中去。上次面试官问过我一次，关联容器需要重载哪些运算符，当时我答了小于号运算符。这次的实践表明，对于 map 容器来说，只需要实现了小于号运算符就可以正确的实现关联容器的功能。

3.3 并发服务器类的层次结构



构建这样的一个服务器模型的目的主要在于学习而不是功能，若纯粹为了功能考虑，则使用 EPOLL 就可以了。

并发服务器的思想还是较为“单纯”的：创建一个监听套接字，并维护一个客户端的 Connection 的一个数组。用户使用一个阻塞的调用 WaitForClient 获取当前需要获取服务的客户的连接。从客户数组中删除客户连接的任务有用户自己来调用，一般的实现是读写失败的时候就从当前的客户数组中删除该客户。

并发服务器端主要实现是对于操作系统特定的 I/O 模型的使用。I/O 模型又是一个比较大的话题，需要专门进行详细的了解。这里只是对三种较为常见的 I/O 模型的一个封装：select 模型，poll 模型和 epoll 模型。

select 和 poll 模型的有点是，当大多数的连接都是非活跃的连接是，由于遍历非常耗时，因此代价较大。Select 模型的另一个缺点是收到系统的 FD_SETSIZE 的限制，这个宏在 windows 系统中是 64，在 linux 系统中是 1024。

Epoll 模型特别适合大多数的连接都是非活跃的连接的客户，如 QQ 这种应用。对于 WEB 服务器这种客户都是活跃用户的情形，使用 epoll 非但没有什么优势，反而会因为内核检测的代价而降低而模型的性能，此种情形 poll 模型可能更加适合。