## STAGE v C/C++ 11 on Linux

Version 1.0.0

目录

[1. 概述 8](#_Toc8747)

[1.1. 核心组件 8](#_Toc2033)

[1.1.1. ep::Object 8](#_Toc20259)

[1.1.2. ep::Reference 8](#_Toc11564)

[1.1.3. ep::Hosting 8](#_Toc24740)

[1.1.4. ep::threading::Timer 8](#_Toc11139)

[1.1.5. ep::threading::Stranding](#_Toc7109)

[1.1.6. ep::net::Socket 8](#_Toc1003)

[1.1.7. ep::net::SocketListener 8](#_Toc15975)

[1.2. 适配器 8](#_Toc7451)

[1.2.1. Adapter driver model 8](#_Toc25013)

[1.3. 服务器 9](#_Toc29803)

[2. 要点 9](#_Toc2703)

[2.1. 服务器 9](#_Toc2589)

[2.1.1. 虚拟交换机 9](#_Toc26356)

[2.1.3. Control Acceptor 9](#_Toc10191)

[2.1.4. ServerExchanger 10](#_Toc12539)

[2.1.4.1. 概述 10](#_Toc14451)

[2.1.5. Sendback 10](#_Toc12383)

[2.1.5.1. 示意图 10](#_Toc12735)

[2.1.5.2. 概述 11](#_Toc4957)

[2.1.5.3. 服务器 11](#_Toc3286)

[2.1.5.3.1. ep::stage::ServerSendbackSettings 11](#_Toc30830)

[2.1.5.3.1.1. MaxSendQueueSize 11](#_Toc3600)

[2.1.5.3.1.2. MaxSendBufferSize 11](#_Toc20622)

[2.1.5.3.1.3. ThoroughlyAbandonTime 11](#_Toc145)

[2.1.5.3.2. ep::stage::ServerSendbackController 11](#_Toc29027)

[2.1.5.3.2.1. Event 11](#_Toc14096)

[2.1.5.3.2.1.1. UnderloadEvent 11](#_Toc7627)

[2.1.5.3.2.2. Method 12](#_Toc27901)

[2.1.5.3.2.2.1. AddSendback 12](#_Toc6730)

[2.1.5.3.2.2.2. DoEvents 12](#_Toc30568)

[2.1.5.3.2.2.3. On 12](#_Toc10978)

[2.1.5.3.2.2.4. Run 12](#_Toc22346)

[2.1.5.4. 客户端 12](#_Toc21879)

[2.1.6. 虚拟服务器套接字层 12](#_Toc23492)

[2.1.6.1. ep::stage::StageVirtualSocket 12](#_Toc5203)

[2.1.8. 安全套接字层（N-SSL） 13](#_Toc19480)

[2.1.8.1. Cipher 13](#_Toc16513)

[2.1.8.1.1. rc4\_mod\_VT\_subtract 13](#_Toc23156)

[2.1.9. 可选压缩机制 13](#_Toc27075)

[2.1.9.1. Algorithm 14](#_Toc26858)

[2.1.9.1.1. LZ77-gzip 14](#_Toc14170)

[2.1.9.2. Interfaces 14](#_Toc17959)

[2.1.9.2.1. DefinedTypes 14](#_Toc18394)

[2.1.9.2.1.1. ep::stage::StageSocket 14](#_Toc8990)

[2.1.9.2.1.2. ep::stage::StageClient 14](#_Toc28708)

[2.1.9.2.1.3. ep::stage::StageSocketCommunication 14](#_Toc11467)

[2.1.9.2.2. APIs 14](#_Toc2843)

[2.1.9.2.2.1. IsEnabledCompressedMessages 14](#_Toc29967)

[2.1.9.2.2.2. EnabledCompressedMessages 14](#_Toc8834)

[2.1.9.2.2.3. DisabledCompressedMessages 14](#_Toc25573)

[2.2. 客户端 14](#_Toc27627)

[2.2.1. 虚拟交换机 14](#_Toc8775)

[3. 部署 14](#_Toc8303)

[3.1. epsvr 14](#_Toc820)

[3.1.1. References 14](#_Toc16476)

[3.1.1.1. centos7 14](#_Toc8428)

[3.1.1.2. libc 14](#_Toc7631)

[3.1.1.3. g-libc 14](#_Toc22367)

[3.2. usage 14](#_Toc13876)

[3.3. STAGE适配器配置格式(缺省) 14](#_Toc116)

[3.4. LinkType 14](#_Toc21060)

[3.4.1. gamesvr 0 14](#_Toc30829)

[3.4.2. chatsvr 1 14](#_Toc29748)

[3.4.3. dbsvr 2 14](#_Toc28617)

[3.4.4. logsvr 3 15](#_Toc17217)

[3.4.5. ranksvr 4 15](#_Toc32096)

[3.4.6. mailsvr 5 15](#_Toc23516)

[3.5. Macro.xml 15](#_Toc23182)

[3.6. 附件 15](#_Toc11130)

[3.7. 示例 15](#_Toc29983)

[3.7.1. 引入STAGE基础设施类库常用的头文件 15](#_Toc19236)

[3.7.2. 构建并运行一个STAGE服务器主机实例](#_Toc6799)

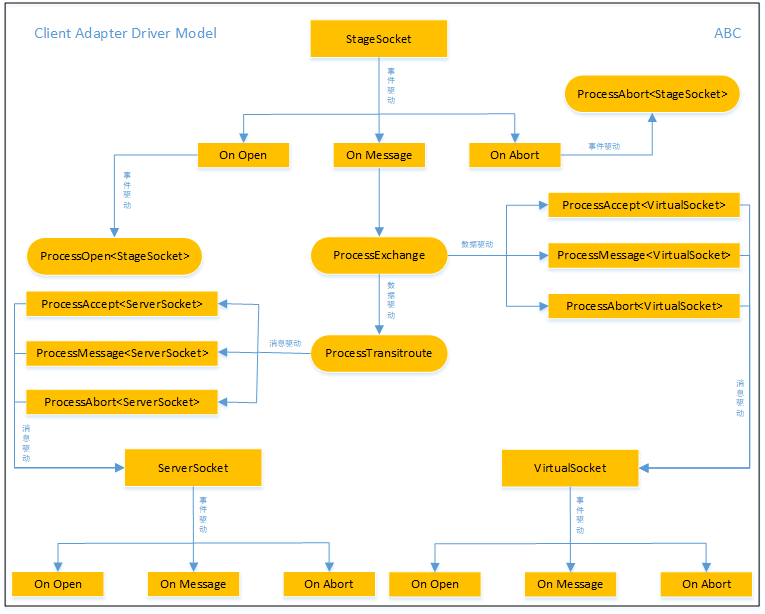
[3.7.3. 构建并运行一个STAGE服务适配器实例 15](#_Toc27053)

[3.7.4. 构建并运行一个STAGE服务器间交换机 16](#_Toc14225)

1. 概述
   1. 核心组件
      1. ep::Object
      2. ep::Reference
      3. ep::Hosting

本地托管主机，驱动整套基础框架与设施的工作主机管理对象，可向外弹性扩展中间件，线程隔离，流程串化。

* + 1. ep::threading::Timer
    2. ep::threading::Stranding  
        搁浅线（搁置线），是保证在多线程环境下代码执行顺序串化保持有序的工具类， 我们可以在任何一个时机向“stranding”提交（Post）需要被串化执行处理的委托处理 器，“stranding”的另外一点是它是由外部调用方决策驱动的，这与“boost::asio::strand” 由“boost::asio::io\_context”直接驱动的形式不同。
    3. ep::net::Socket
    4. ep::net::SocketListener
  1. 适配器
     1. Adapter driver model



* 1. 服务器

1. 要点
   1. 服务器
      1. 虚拟交换机

内部服务器与对公客户端之间相互进行网络通信的虚拟套接字链路交换机，每个不同的连接类型的内部服务器，STAGE服务器都会为其构建且维护一组对应的虚拟链路交换机实例。

* + 1. 文件描述符

STAGE可接受的最大对公客户端套接字的数量，最大只允许接受65535个客户端套接字数量，另外一点在设置最大套文件描述数量为0时，STAGE则视其极限上限为65535个文件描述。

须知STAGE服务器本身是无法指定最大文件描述数量的，其数量是由STAGE适配器于STAGE服务器主机之间鉴权时决定的，但这仅仅只影响到同“平台”、“同服”、“同链接类型”的服务器交换机实例。

STAGE设置的最大文件描述符数量是允许运行时变更，但须注意文件描述被减小时，STAGE服务器并不会对强制以剔除已经在线的客户端数量，以保持最新设定的最大文件描述数量限制的客户端数量保持一致性。

* + 1. Control Acceptor

STAGE提供一组用于“控制STAGE对公客户端套接字接受器”的函数集，在很多的场景下面我们都希望在内部服务器未完成初始化部署之前或者某些特定情况，不希望接受来自任何对公客户端请求建立的链接时；可以利用“Control Acceptor”提供的APIs；以下列出STAGE提供用于“控制接受器”的接口（适配器）。

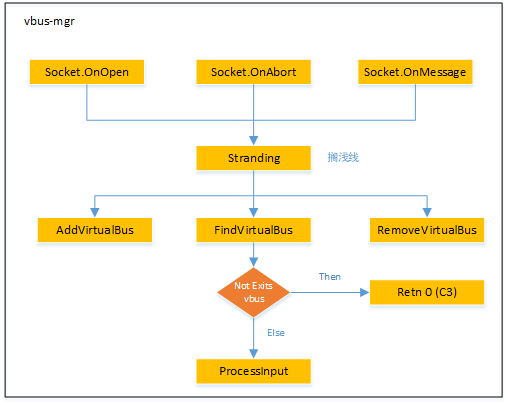
1、PreventAcceptClient() 阻止接受客户端实例

2、PermittedAcceptClient() 允许接受客户端实例

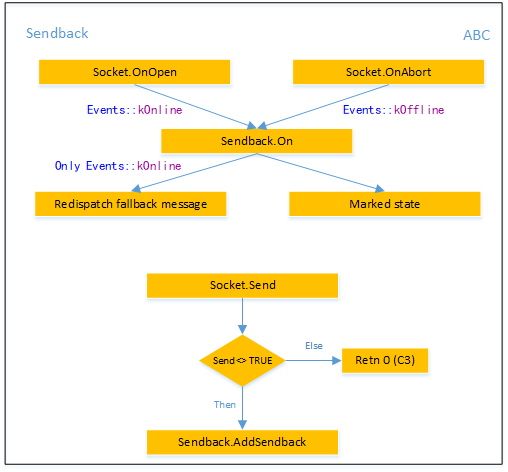
3、IsAllowAcceptClient() 是否允许接受客户端

* + 1. ServerExchanger
       1. 概述

服务器之间的虚拟bus/vbus通道交换器设备，其作用为现有服务器框架提供兼容与现有应用tbus/tapp框架的游戏服务器程序之间保持一些关键性APIs/Interfaces兼容，须知，它不属于STAGE组件部分，是第三方对于具体业务需求进行的派生扩展的模式组件。



* + 1. Sendback
       1. 示意图



* + - 1. 概述

发送回退机制用于确保服务器与服务器之间发送数据可在一定限度约束之内保持可靠链路传输。

发送回退机制只保证，堆积在STAGE适配器或者STAGE服务器中间件之间的发送失败回退消息在保持时序的情况发送目标的以太网卡上面，但这不意味着发送回退机制会保证目标能够正确处理。

这并不是Actor通信模型构建的SendEcho机制，由应用层或者框架层Auto-Ack 确保并提供有效的Echo-Ack。

* + - 1. 服务器
         1. ep::stage::ServerSendbackSettings

MaxSendQueueSize

最大发送队列大小，指发送回退可以堆积消息的总个数。

MaxSendBufferSize

最大发送缓冲区大小，指发送回退可以被堆积在缓冲区的多个消息数据。

ThoroughlyAbandonTime

回退强制抛弃时间，指自从链路中断起计时达到强制抛弃时候后则彻底丢弃发送回退的全部消息数据，设置为0则意味着永远不会丢弃。

* + - * 1. ep::stage::ServerSendbackController

Event

UnderloadEvent

过载事件，指当发送失败可回退的消息堆积超出了由发送回退设置选项约定的最大承载时，诱发。

Method

AddSendback

添加发送失败回退的数据报文，须知必须在两者链路不通的情况下才可被调用，否则可能引发一些疑难杂症。

DoEvents

驱动一次发送回退控制器消化一些额外事件

On

驱动发送失败被回退的消息报文数据，在目的主机在线时投递堆积的报文到目的主机网卡上面，离线则记录中断链路的时间等相关。

Run

运行服务器发送回退控制器。

* + - 1. 客户端
    1. 虚拟服务器套接字层
       1. ep::stage::StageVirtualSocket

虚拟的客户端套接字，提供安全并且可靠的客户端虚套接字；具有一个完整套接字实例的主要行为与驱动，它彻底屏蔽了繁琐且复杂的物理套接字通信层，同时虚拟套接字接口设计实现都是允许可被替换或者移植的，并且提供规范清晰一致性的统一接口，这为易于后期扩展与开发人员改善提供了可能。  
 虚拟套接字层应用的比较广泛的场景，主要集中在Web后台服务器与Web应用宿主服务器上面用的比较多。

比如：

1. “Kestrel”与“ASP.NET Core”
2. “IIS”与“ASP.NET”
3. “Apache”与“Java EE”

虚拟套接字的技术提出理由主要为了分离物理套接字层与虚拟套接字层之间的直接联系，减少物理套接字对于内部服务器的资源开销，由其Web服务器这类型几乎都是短链接请求，对于服务器程序资源的时钟周期开销是非常严重的（它必定存在大量的TIME\_WAIT\_1、TIME\_WAIT\_2、CLOSE\_WAIT、SYN\_RECVD）；

访问“物理套接字”的接口会造成程序频繁的“断入内核态（“INT 2E”或者“sysenter”指令）”，纵然异步的套接字并不意味着程序不需要“断入断出内核态”也不意味着这些开销在内核就跟应用程序不相干。

这部分开销是属于应用程序进程的（只不过是内核态上面）同时值得一提套接字访问每次都是有锁的一次只能一个线程进入内核套接字池访问PCB。

例如TCP\_PCB、UDP\_PCB；每次请求获取localTaken的开销是无法省却的（一个线程【TIB】在访问另一个线程【TIB】则需要等待竞争，两个物理套接字实例访问它们可能竞争临界锁也可能不需要，内核协议栈是多核工作模式）

虚拟套接字层一般会通过“管线（Pipeline）”、“双向通道MMF /(IPC)”、“UDP数据报”的方法把来自物理套接字传输的数据，传输到内部的子服务器，而其优点主要为以下列出的几个方面。

1. 潜在的内核对象句柄泄漏（例如套接字）
2. 减少维护物理层套接字的必要开销
3. 减少CPU断入内核态与断出用户态的切换
4. INT 3可能造成大量套接字损坏
5. 减少安全防护措施所带来的额外开销
6. 提供可选客户端请求分离分发

主要接口例如：

事件：

1. OnOpen 套接字建立打开事件
2. OnMessage 套接字消息到达事件
3. OnAbort 套接字中断事件

属性（函数型）：

1. GetLocalEndPoint() 获取套接字本地点对点端点地址
2. GetRemoteEndPoint() 获取套接字远程点对点端点地址
3. GetHandle() 获取套接字文件描述符
4. GetLinkType() 获取套接字的链接类型
5. Available() 反应套接字是否还是可用的
6. GetPtr() 获取套接字的共享指针
7. IsDisposed() 反应套接字持有的资源是否已经被释放
8. GetSynchronizingObject() 获取套接字同步对象
9. GetTransmissionObject() 获取物理套接字传输对象
10. GetHosting() 获取本地主机对象

方法：

1、Abort() 强制中断套接字实例（中断不等于关闭）

2、Close() 强制关闭套接字实例

3、Send(...) 套接字发送消息报文到对端

4、Dispose() 强制关闭套接字实例

* + 1. 虚拟客户端套接字层
       1. ep::stage::StageServerSocket  
           虚拟的服务器套接字，派生自“虚拟套接字”同样具有“虚拟套接字”提供的一系 列接口与行为驱动，但是其本身则额外扩展了一些核心的特性。

例如：

属性（函数型）：

1. GetPlatform() 获取游戏平台的名称编号
2. GetServerNo() 获取游戏平台服务器编号
3. GetPointer() 获取服务器套接字的共享指针
   * 1. 安全套接字层（N-SSL）

STAGE为数据报提供安全加密消息密文，除“心跳”、“鉴权”协议无需加密的以外，任何与对公客户端之间的消息都经历随机生成的报文协议加密特征；当然在强烈的加密方式都只能防范君子而无法防范小人，但这并不在STAGE设计的考虑范围之内，我们只能尽可能给小人增加分解的复杂度与时间成本，这本身就是一个零和游戏，在权衡效率与安全性的基础上，我们只可以选择相对较为恰当的办法。

* + - 1. Cipher
         1. rc4\_mod\_VT\_subtract
    1. 可选压缩机制

STAGE提供可选压缩机制，即在服务器应用启用压缩选项时，STAGE在发送通信报文之前会对其报文进行可选压缩，即先尝试压缩设其压缩后的报文超出未压缩报文时，我们则视为报文无需被压缩，反之则标记报文已经被压缩。

服务器可以禁用压缩选项时，则服务器不会对报文进行可选压缩，这会显著减少STAGE在发送通信报文上面所花费的开销，而且服务器禁用可选压缩并不会影响到对公客户端，例如必须与服务器保持一致禁用可选压缩选项。

* + - 1. Algorithm
         1. LZ77-gzip
      2. Interfaces
         1. DefinedTypes

ep::stage::StageSocket

ep::stage::StageClient

ep::stage::StageSocketCommunication

* + - * 1. APIs

IsEnabledCompressedMessages

检查当前套接或通信层是否启用压缩消息，缺省为启用

EnabledCompressedMessages

启用可选压缩消息机制

DisabledCompressedMessages

禁用可选压缩消息机制（但不影响客户端启用可选压缩机制）

* 1. 客户端
     1. 虚拟交换机

1. 部署
   1. epsvr
      1. References
         1. centos7
         2. libc
         3. g-libc
   2. usage

epsvr --client-port=6666 --server-port=7777 --backlog=10000 --max-concurrent=4

* 1. STAGE适配器配置格式(缺省)

<Stage> <!--STAGE配置根-->

<HostName>172.0.6.138</HostName> <!--STAGE服务器IPV4地址-->

<Port>7777</Port> <!--STAGE服务器端口地址-->

<LinkType>0</LinkType> <!--STAGE链接类型-->

<Platform>lan\_android</Platform> <!--STAGE平台名称-->

<SvrAreaNo>1</SvrAreaNo> <!--STAGE区域服编码-->

<ServerNo>9997</ServerNo> <!--STAGE平台服编码-->

<MaxFiledescriptor>3000</MaxFiledescriptor> <!--STAGE最大文件描述符-->

<MaxConcurrent>8</MaxConcurrent> <!--STAGE最大通道并发指数-->

<!--STAGE最大同帧工作处理并发指数-->

<MaxWorkingHandlingConcurrent>100</MaxWorkingHandlingConcurrent>

</Stage>

* 1. LinkType
     1. gamesvr 0
     2. chatsvr 1
     3. dbsvr 2
     4. logsvr 3
     5. ranksvr 4
     6. mailsvr 5
  2. Macro.xml

<macrosgroup name="TBUSFuncDef" >

<macro name="TBUS\_FUNC\_GAME" value="0" desc="Game Svr"/>

<macro name="TBUS\_FUNC\_CHAT" value="1" desc="Chat Svr"/>

<macro name="TBUS\_FUNC\_DB" value="2" desc="DBSvr"/>

<macro name="TBUS\_FUNC\_LOG" value="3" desc="Log Svr"/>

<macro name="TBUS\_FUNC\_RANK" value="4" desc="Rank Svr"/>

<macro name="TBUS\_FUNC\_MAIL" value="5" desc="Mail Svr"/>

</macrosgroup>

* 1. 附件



* 1. 示例
     1. 引入STAGE基础设施类库常用的头文件

#include "util/stage/core/Hosting.h"

#include "util/stage/net/Socket.h"

#include "util/stage/net/SocketListener.h"

#include "util/stage/threading/TimerScheduler.h"

#include "util/stage/stage/StageSocket.h"

#include "util/stage/stage/StageClient.h"

#include "util/stage/stage/StageSocketListener.h"

#include "util/stage/stage/StageSocketCommunication.h"

* + 1. 构建并运行一个STAGE服务器主机实例  
        int main(int argc, char\* argv[])

{

std::shared\_ptr<ep::Hosting> hosting = ep::make\_shared\_object<ep::Hosting>(4);

hosting->Run();

ep::threading::TimerScheduler::PreparedGlobalScheduler(hosting);

auto listener =

ep::make\_shared\_object<ep::stage::StageSocketCommunication>(hosting,

1000, 6666, 7777);

listener->Listen();

return getchar();

}

* + 1. 构建并运行一个STAGE服务适配器实例

auto& stageConfiguration = gs\_stEnv.pstConf->stStage;

auto stageClient = ep::make\_shared\_object<ep::stage::StageClient>(

hosting,

stageConfiguration.szHostName,

stageConfiguration.wPort,

stageConfiguration.bLinkType,

stageConfiguration.szPlatform,

stageConfiguration.wSvrAreaNo,

stageConfiguration.wServerNo,

stageConfiguration.wMaxFiledescriptor,

stageConfiguration.wMaxConcurrent);

if (!stageClient ) {

throw std::*runtime\_error*("Unable to build a valid instance of the stage client object");

}

if (!stageClient ->Listen()) {

throw std::*runtime\_error*("Could not run-sub the listen for the stage client object instance");

}

* + 1. 构建并运行一个STAGE服务器间交换机

std::shared\_ptr<ep::stage::StageClient> stageClient;

auto exchanger = ep::make\_shared\_object<ServerExchanger>(stageClient);

if (!exchanger) {

throw std::*runtime\_error*("Unable to build an available ServerExchanger");

}