以 client.c 为例说明 stream 中各个函数的调用关系。文件中的超链接可以链接到函数的定义处，并且提供了相关的说明。

Client.c	stream.c	Stream-provider.h	Stream-tcp.c
struct stream* p_stream		Struct stream{...}	
stream_open	Stream_open(...)	Struct Stream_class{...}	
	Stream_lookup_class(..)		Tcp_stream_class{...}
	Struct stream classes[]		Tcp_open(...)
Stream_connect	Stream_connect(...)		
	Scs_connecting(...)		
Stream_send	Stream_send(...)		
Stream_close	Stream_close(...)		

```
int main()
{
    struct stream* p_stream;
    const char* stream_name = "tcp:127.0.0.1:1234";
    char pnum[4];
    uint8_t dscp = 1;
    int actualsend = 0;
    int index = 0;

    while(1)
    {
        if(stream_open(stream_name,&p_stream,dscp))
        {
            printf("stream open failure!\n");
        }
        else
        {
            printf("stream open sucessed!\n");

            if(stream_connect(p_stream))
            {
                printf("stream connect failure!\n");
            }
            else
            {
                printf("stream connect successed!\n");
                sprintf(pnum,"%d",index);
                actualsend = stream_send(p_stream,pnum,sizeof(pnum));
                index++;
                if(actualsend < 0)
                {
```

```
                printf("stream send failure!\n");
            }
            else if(actualsend == 0)
            {
                printf("stream send 0 bytes!\n");
            }
            else
            {
                printf("stream actual send %d bytes,Data:%s\n",actualsend,pnum);
            }
        }
    }
    stream_close(p_stream);
    sleep(1);
  }
  return 0;
}
```

结构体：Struct stream{...}：

stream 类型的结构体在 stream-provider.h 文件中进行了定义，定义如下图所示：Stream 结构体重包含了另一个结构体的定义， 即 stream_class{...}， 还定义了 Stream 类型的状态 state ， 名称 name，远程/本地 ip 以及端口号 port 等一些参数。

stream Structure in Stream-provider.h at line 7 (10 lines)

```
struct stream {
    const struct stream_class *class;
    int state;
    int error;
    ovs_be32 remote_ip;
    ovs_be16 remote_port;
    ovs_be32 local_ip;
    ovs_be16 local_port;
    char *name;
};
```

函数：Stream_open(...)：

Stream_open(...) 函数定义在 Stream.c 文件中， 函数的实现如下图所示。此函数的基本功能把连接的 stream 连接到一个远程的 stream，如果成功则返回 0，否则会返回一个错误的值。Stream_open()函数会完成 stream 的一部分连接工作。Stream_open()函数中调用了 stream_look_up_class(...) 函数。

stream_open Function in Stream.c at line 166 (32 lines)

```
/* Attempts to connect a stream to a remote peer.  'name' is a connection name
 * in the form "TYPE:ARGS", where TYPE is an active stream class's name and
 * ARGS are stream class-specific.
 *
 * Returns 0 if successful, otherwise a positive errno value.  If successful,
 * stores a pointer to the new connection in '*streamp', otherwise a null
 * pointer.  */

int
stream_open(const char *name, struct stream **streamp, uint8_t dscp)
{
    const struct stream_class *class;
    struct stream *stream;
    char *suffix_copy;
    int error;

    /*COVERAGE_INC(stream_open);*/

    /* Look up the class. */
    error = stream_lookup_class(name, &class);
    if (!class) {
        goto error;
    }

    /* Call class's "open" function. */
    suffix_copy = xstrdup(strchr(name, ':') + 1);
    error = class->open(name, suffix_copy, &stream, dscp);
    free(suffix_copy);
    if (error) {
        goto error;
    }

    /* Success. */
    *streamp = stream;
    return 0;

error:
    *streamp = NULL;
    return error;
} ? end stream_open ?
```

函数：Stream_connect(...)：

Stream_connect() 函数定义在stream.c文件中，函数的实现形式如下图所示。

Stream_connect()函数会继续完成 stream_open()函数中未完成的连接操作。如果连接成功则返回0， 如果连接失败则返回一个错误的值。stream_connect()中主要通过scs_connecting()完成连接的建。

stream_connect Function in Stream.c at line 302 (25 lines)

```
/* Tries to complete the connection on 'stream'.  If 'stream''s connection is
 * complete, returns 0 if the connection was successful or a positive errno
 * value if it failed.  If the connection is still in progress, returns
 * EAGAIN. */
int
stream_connect(struct stream *stream)
{
    enum stream_state last_state;

    do {
        last_state = stream->state;
        switch (stream->state) {
        case SCS_CONNECTING:
            scs_connecting(stream);
            break;

        case SCS_CONNECTED:
            return 0;

        case SCS_DISCONNECTED:
            return stream->error;

        default:
            NOT_REACHED();
        }
    } while ((unsigned)stream->state != last_state);

    return EAGAIN;
} ? end stream_connect ?
```

```
int
stream_open(const char *name, struct stream **streamp, uint8_t dscp)
{
    const struct stream_class *class;
    struct stream *stream;
    char *suffix_copy;
    int error;

    /*COVERAGE_INC(stream_open);*/

    /* Look up the class. */
    error = stream_lookup_class(name, &class);
    if (!class) {
        goto error;
    }

    /* Call class's "open" function. */
    suffix_copy = xstrdup(strchr(name, ':') + 1);
    error = class->open(name, suffix_copy, &stream, dscp);
    free(suffix_copy);
    if (error) {
        goto error;
    }

    /* Success. */
    *streamp = stream;
    return 0;

error:
    *streamp = NULL;
    return error;
} ? end stream_open ?
```

函数：Stream_send()　：

Stream_send() 函数定义在 stream.c 文件中。函数的实现形式如图所示.。函数的功能是

在 stream 上发送 buffer 中的 n 个字节的内容。如果成功，会返回实际发送的字节数，否则则会返回一个错误的值（负数）。值得注意的是 stream_send 函数不是阻塞式的。

stream_send Function in Stream.c at line 357 (8 lines)

```
/* Tries to send up to 'n' bytes of 'buffer' on 'stream', and returns:
 *
 *    - If successful, the number of bytes sent (between 1 and 'n'). 0 is
 *      only a valid return value if 'n' is 0.
 *
 *    - On error, a negative errno value.
 *
 * The send function will not block. If no bytes can be immediately accepted
 * for transmission, it returns -EAGAIN immediately. */
int
stream_send(struct stream *stream, const void *buffer, size_t n)
{
    int retval = stream_connect(stream);
    return (retval ? -retval
            : n == 0 ? 0
            : (stream->class->send)(stream, buffer, n));
}
```

函数 ：stream_close() ：

Stream_close() 函数定义在 stream.c 文件中。函数的实现形式如图所示.。函数功能是关闭 stream 的传输通道。

stream_close Function in Stream.c at line 235 (9 lines)

```
void
stream_close(struct stream *stream)
{
    if (stream != NULL) {
        char *name = stream->name;
        (stream->class->close)(stream);
        free(name);
    }
}
```

结构体：stream_class{...}：

Stream_class{} 类型的结构体在 stream-provider.h 文件中进行了定义，定义如下图所示。它被封装成 stream 类型的一个内嵌的结构体。其中提供了多个函数指针， 如 connect(), recv(), send(), 等。

stream_class Structure in Stream-provider.h at line 25 (13 lines)

```
struct stream_class {
    const char *name;
    bool needs_probes;
    int (*open)(const char *name, char *suffix, struct stream **streamp,
            uint8_t dscp);
    void (*close)(struct stream *stream);
    int (*connect)(struct stream *stream);
    ssize_t (*recv)(struct stream *stream, void *buffer, size_t n);
    ssize_t (*send)(struct stream *stream, const void *buffer, size_t n);
    void (*run)(struct stream *stream);
    void (*run_wait)(struct stream *stream);
    void (*wait)(struct stream *stream, enum stream_wait_type type);
};
```

函数 ：stream_look_up_class(...) ：

Stream_lookup_class() 函数定义在stream.c文件中，函数的实现形式如下图所示。函数通过传入的 name 为stream结构体中的stream->class进行赋值操作。在此结构体中， 利用了stream_classes[]结构体数组为stream_class结构体进行赋值。

stream_lookup_class Function in Stream.c at line 126 (23 lines)

```
/* Given 'name', a stream name in the form "TYPE:ARGS", stores the class
 * named "TYPE" into '*classp' and returns 0.  Returns EAFNOSUPPORT and stores
 * a null pointer into '*classp' if 'name' is in the wrong form or if no such
 * class exists. */
static int
stream_lookup_class(const char *name, const struct stream_class **classp)
{
    size_t prefix_len;
    size_t i;

   /* check_stream_classes();*/

    *classp = NULL;
    prefix_len = strcspn(name, ":");
    if (name[prefix_len] == '\0') {
        return EAFNOSUPPORT;
    }
    for (i = 0; i < ARRAY_SIZE(stream_classes); i++) {
        const struct stream_class *class = stream_classes[i];
        if (strlen(class->name) == prefix_len
            && !memcmp(class->name, name, prefix_len)) {
            *classp = class;
            return 0;
        }
    }
    return EAFNOSUPPORT;
} ?  end stream_lookup_class ?
```

函数 scs_connecting()：

Scs_connection() 函数定义在stream.c文件中，函数的实现形式如下图所示。函数通过stream->class->connect进行连接， 判断上一步是否成功， 并且修改stream结构体中state的字段的值。

```
scs_connecting    Function in Stream.c at line 285 (12 lines)
static void
scs_connecting(struct stream *stream)
{
    int retval = (stream->class->connect)(stream);
    ovs_assert(retval != EINPROGRESS);
    if (!retval) {
        stream->state = SCS_CONNECTED;
    } else if (retval != EAGAIN) {
        stream->state = SCS_DISCONNECTED;
        stream->error = retval;
    }
}
```

结构体数组 ：Struct stream_classes[] ：

Stream_classes 结构体数组定义在 stream.c 文件中，函数的实现形式如下图所示。结构体数组中调用了 tcp_stream_class 为其赋值。

```
static const struct stream_class *stream_classes[] = {
    &tcp_stream_class,
    /* &unix_stream_class,
#ifdef HAVE_OPENSSL
    &ssl_stream_class,
#endif*/
};
```

结构体 ： tcp_stream_class{} ：

Tcp_stream_class 结构体定义在 stream-tcp.c 文件中，函数的实现形式如下图所示。结构体数组中调用了 tcp_stream_class 为其赋值。

```
const struct stream_class tcp_stream_class = {
    "tcp",                      /* name */
    true,                       /* needs_probes */
    tcp_open,                   /* open */
    NULL,                       /* close */
    NULL,                       /* connect */
    NULL,                       /* recv */
    NULL,                       /* send */
    NULL,                       /* run */
    NULL,                       /* run_wait */
    NULL,                       /* wait */
};
```

结构体 ： tcp_open() ：

Tcp_open()函数定义在 stram-tcp.c 文件中，函数的实现形式如下图所示。此函数调用了 inet_open_active()函数建立一个 tcp 连接。

```
static int
tcp_open(const char *name, char *suffix, struct stream **streamp, uint8_t dscp)
{
    struct sockaddr_in sin;
    int fd, error;

    error = inet_open_active(SOCK_STREAM, suffix, 0, &sin, &fd, dscp);
    if (fd >= 0) {
        return new_tcp_stream(name, fd, error, &sin, streamp);
    } else {
 /*     VLOG_ERR("%s: connect: %s", name, ovs_strerror(error)); */
        return error;
    }
}
```

结构体 ： inet_open_active() ：

inet_open_active()函数定义在 Util.c 文件中，函数的实现形式如下图所示。此函数调用了 Unix 系统编程中的 socket()函数建立了一个最原始的 socket 连接。

```
/* Opens a non-blocking IPv4 socket of the specified 'style' and connects to
 * 'target', which should be a string in the format "<host>[:<port>]".  <host>
 * is required.  If 'default_port' is nonzero then <port> is optional and
 * defaults to 'default_port'.
 *
 * 'style' should be SOCK_STREAM (for TCP) or SOCK_DGRAM (for UDP).
 *
 * On success, returns 0 (indicating connection complete) or EAGAIN (indicating
 * connection in progress), in which case the new file descriptor is stored
 * into '*fdp'.  On failure, returns a positive errno value other than EAGAIN
 * and stores -1 into '*fdp'.
 *
 * If 'sinp' is non-null, then on success the target address is stored into
 * '*sinp'.
 *
 * 'dscp' becomes the DSCP bits in the IP headers for the new connection.  It
 * should be in the range [0, 63] and will automatically be shifted to the
 * appropriately place in the IP tos field. */
int
inet_open_active(int style, const char *target, uint16_t default_port,
                 struct sockaddr_in *sinp, int *fdp, uint8_t dscp)
{
    struct sockaddr_in sin;
    int fd = -1;
    int error;

    /* Parse. */
    if (!inet_parse_active(target, default_port, &sin)) {
        error = EAFNOSUPPORT;
        goto exit;
    }

    /* Create non-blocking socket. */
    fd = socket(AF_INET, style, 0);
    if (fd < 0) {
 /*     VLOG_ERR("%s: socket: %s", target, ovs_strerror(errno));*/
        error = errno;
        goto exit;
    }
    error = set_nonblocking(fd);
    if (error) {
        goto exit;
    }
```

结构体 stream_class 的结构体关系图如图所示：