此函数是链接RDP服务器的主要逻辑函数, 主要进行链接服务器前的客户端接口绑定, channels的连接前处理, 链接, 连接后的客户端接口处理, channels连接后处理,链接失败处理逻辑

1.重置一些必要的数据包括setting,event,channelError

```
/* We always set the return code to 0 before we start the connect sequence*/
instance->ConnectionCallbackState = CLIENT_STATE_INITIAL;
connectErrorCode = 0;
instance->context->LastError = FREERDP_ERROR_SUCCESS;
clearChannelError(instance->context);
ResetEvent(instance->context->abortEvent);
rdp = instance->context->rdp;
settings = instance->settings;
```

2,设置默认的命令

```
if (!freerdp_settings_set_default_order_support(settings))
    return FALSE;
```

3.调用连接前的函数,即wf_pre_connect, 此函数主要做了三件事情, 获取本机显示器分辨率信息, 加载channels信息, 设置键盘信息

```
IFCALLRET(instance->PreConnect, status, instance);
instance->ConnectionCallbackState = CLIENT_STATE_PRECONNECT_PASSED;
```

- 获取本机分辨率, 根据是否全屏, 采用不同的逻辑

```
if (wfc->percentscreen > 0)
{
    desktopWidth = (GetSystemMetrics(SM_CXSCREEN) * wfc->percentscreen) / 100;
    settings->DesktopWidth = desktopWidth;
    desktopHeight = (GetSystemMetrics(SM_CYSCREEN) * wfc->percentscreen) / 100;
    settings->DesktopHeight = desktopHeight;
}

if (wfc->fullscreen)
{
    if (settings->UseMultimon)
    {
        desktopWidth = GetSystemMetrics(SM_CXVIRTUALSCREEN);
        desktopHeight = GetSystemMetrics(SM_CYVIRTUALSCREEN);
    }
    else
    {
        desktopWidth = GetSystemMetrics(SM_CXSCREEN);
        desktopHeight = GetSystemMetrics(SM_CYSCREEN);
    }
}
```

- 将分辨率信息赋值给setting

```
 *        otherwise the screen will crash when connecting to an XP desktop. */
desktopWidth = (desktopWidth + 3) & (~3);

if (desktopWidth != settings->DesktopWidth)
{
    freerdp_set_param_uint32(settings, FreeRDP_DesktopWidth, desktopWidth);
}

if (desktopHeight != settings->DesktopHeight)
{
    freerdp_set_param_uint32(settings, FreeRDP_DesktopHeight, desktopHeight);
}
```

- 加载channels信息, 主要从static表中获取对应channels的初始化接口, 然后绑定init open等回调, 最后调用对应的entry接口, 初始化channels。其他文档中会详细介绍。

```
if (!freerdp_client_load_addins(context->channels, instance->settings))
    return -1;
```

- 设置键盘信息

```
freerdp_set_param_uint32(settings, FreeRDP_KeyboardLayout,
                         (int) GetKeyboardLayout(0) & 0x0000FFFF);
```

- 这个目前还不清楚什么作用

```
PubSub_SubscribeChannelConnected(instance->context->pubSub,
                                 wf_OnChannelConnectedEventHandler);
PubSub_SubscribeChannelDisconnected(instance->context->pubSub,
                                    wf_OnChannelDisconnectedEventHandler);
```

4,如果3.执行成功, 调用channels的pre函数, 内部遍历所有有效的channelsclientdata, 执行pChannelInitEventProc。此接口再FreeRDP_VirtualChannelInit或者FreeRDP_VirtualChannelInitex中绑定。这两个init函数, 再freerdp_channels_client_load或freerdp_channels_client_load_ex中绑定, 并通过pChannelClientData->entryEx()调用, entryEx有分为静态和动态, 之后会详细介绍。

```
if (status)
    status2 = freerdp_channels_pre_connect(instance->context->channels,
                                           instance);
```

5.如果键盘是KBD_JAPANESE_INPUT_SYSTEM_MS_IME2002类型的, 需要进一步指定keyboardtype keyboardsubtype和keybooardfunctionkey

```
if (settings->KeyboardLayout == KBD_JAPANESE_INPUT_SYSTEM_MS_IME2002)
{
    settings->KeyboardType = 7;
    settings->KeyboardSubType = 2;
    settings->KeyboardFunctionKey = 12;
}
```

6.pre函数处理出现错误, 错误处理逻辑。为什么不放在keyboard参数赋值之前, 并不清楚。

```
if (!status || (status2 != CHANNEL_RC_OK))
{
    if (!freerdp_get_last_error(rdp->context))
        freerdp_set_last_error(instance->context, FREERDP_ERROR_PRE_CONNECT_FAILED);

    WLog_ERR(TAG, "freerdp_pre_connect failed");
    goto freerdp_connect_finally;
}
```

7. 链接RDP服务器函数接口, 包括协议选择, socket建立等

```
status = rdp_client_connect(rdp);
```

8.为记录remoteFX的dump信息创建文件

```
if (instance->settings->DumpRemoteFx)
{
    instance->update->pcap_rfx = pcap_open(instance->settings->DumpRemoteFxFile,
                                           TRUE);

    if (instance->update->pcap_rfx)
        instance->update->dump_rfx = TRUE;
}
```

9.链接成功之后的处理逻辑pointer_cache_register_callbacks是干嘛的目前不清楚,PostConnect的调用实际上就是调用wf_post_connect, 此接口之后详述。freerdp_channels_post_connect()接口目前还不是特别明白,貌似也是处理一些channel相关的event

```
if (status)
{
    pointer_cache_register_callbacks(instance->context->update);
    IFCALLRET(instance->PostConnect, status, instance);
    instance->ConnectionCallbackState = CLIENT_STATE_POSTCONNECT_PASSED;

    if (status)
        status2 = freerdp_channels_post_connect(instance->context->channels, instance);
}
```

10. 如果链接失败, 根据错误码在尝试链接一次, 否则直接退出, 清理环境

```
else
{
    if (freerdp_get_last_error(instance->context) == FREERDP_ERROR_CONNECT_TRANSPORT_FAILED)
        status = freerdp_reconnect(instance);
    else
        goto freerdp_connect_finally;
}
```

11. 同样的再次链接依然不成功, 则退出

```
if (!status || (status2 != CHANNEL_RC_OK)
    || !update_post_connect(instance->update))
{
    WLog_ERR(TAG, "freerdp_post_connect failed");

    if (!freerdp_get_last_error(rdp->context))
        freerdp_set_last_error(instance->context, FREERDP_ERROR_POST_CONNECT_FAILED);

    status = FALSE;
    goto freerdp_connect_finally;
}
```

12. 如果使用remoteApp文件形式的, 不断循环, 处理画面。当然需要先打开remotefx文件。目前推测这个文件中的数据会不断被写入update_recv_surfcmds中写入。update_recv_surfcmds中的数据来源于fastpath_recv_update()。这里并不确定我的推测, 需要进一步验证。

```
while (pcap_has_next_record(update->pcap_rfx) && status)
{
    pcap_get_next_record_header(update->pcap_rfx, &record);

    if (!(s = StreamPool_Take(rdp->transport->ReceivePool, record.length)))
        break;

    record.data = Stream_Buffer(s);
    pcap_get_next_record_content(update->pcap_rfx, &record);
    Stream_SetLength(s, record.length);
    Stream_SetPosition(s, 0);

    if (!update_begin_paint(update))
        status = FALSE;
    else
    {
        if (update_recv_surfcmds(update, s) < 0)
            status = FALSE;

        if (!update_end_paint(update))
            status = FALSE;
    }

    Stream_Release(s);
}
```

13.重置event, 环境清理等一般退出或者出错执行。

```
    SetEvent(rdp->transport->connectedEvent);
freerdp_connect_finally:
    EventArgsInit(&e, "freerdp");
    e.result = status ? 0 : -1;
    PubSub_OnConnectionResult(instance->context->pubSub, instance->context, &e);

    if (!status)
        freerdp_disconnect(instance);

    return status;
```