参考网页：

https://gstreamer.freedesktop.org/

https://gstreamer.freedesktop.org/documentation/application-development/basics/init.html?gi-language=c

https://gstreamer.freedesktop.org/pkg/

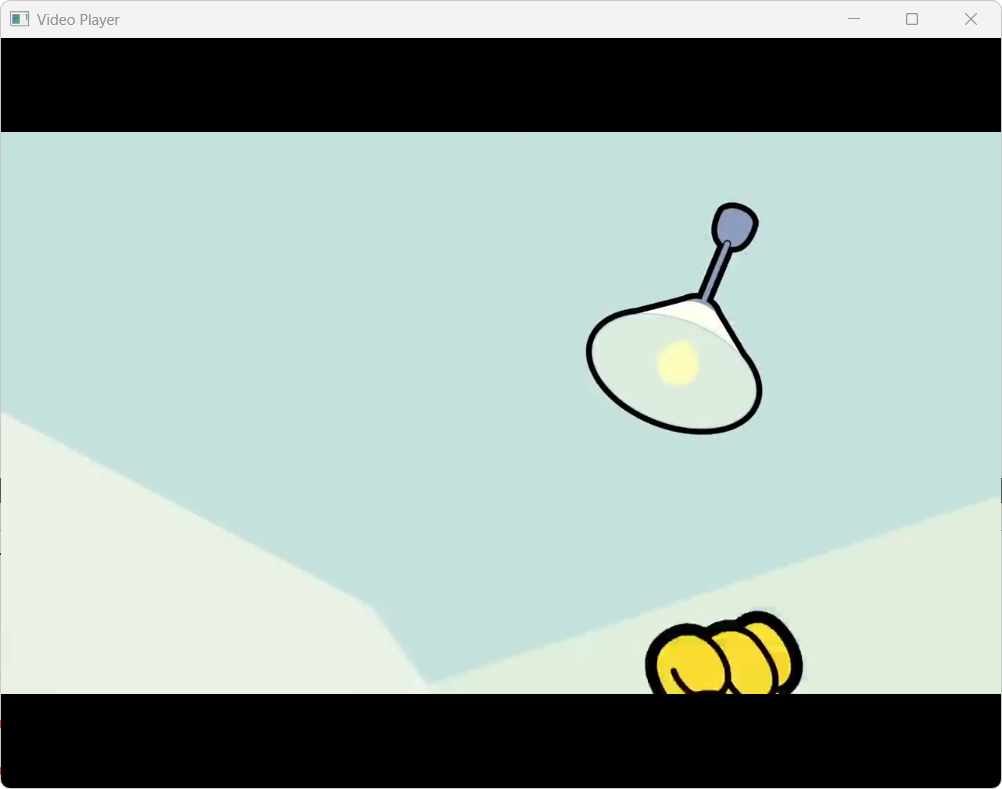
https://gitlab.freedesktop.org/gstreamer/gstreamer

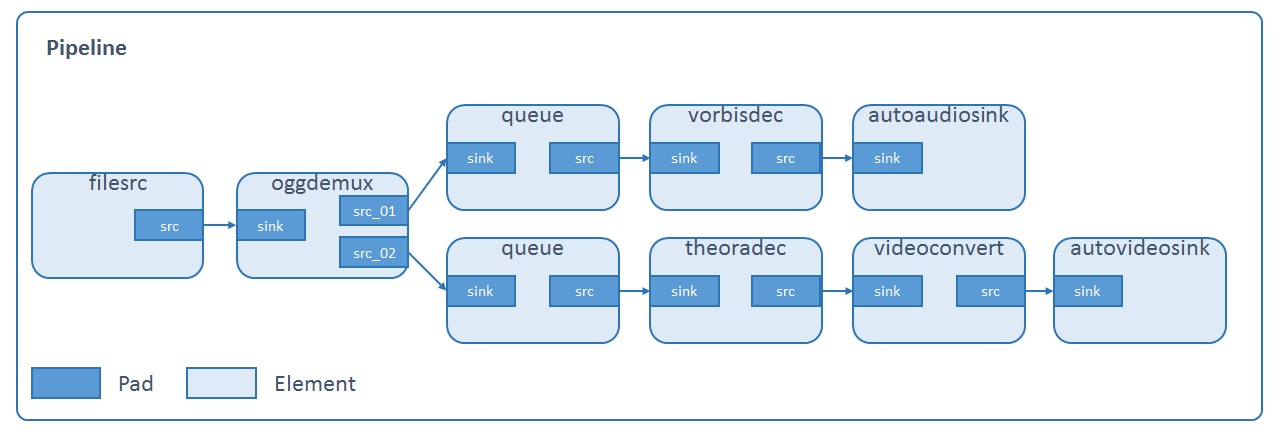
https://gstreamer.freedesktop.org/documentation/plugins\_doc.html?gi-language=c

Qt官方对于GStreamer的介绍

https://doc.qt.io/qt-6/qtmultimedia-gstreamer.html

Demo运行窗口示例（Qt-GStreamer）

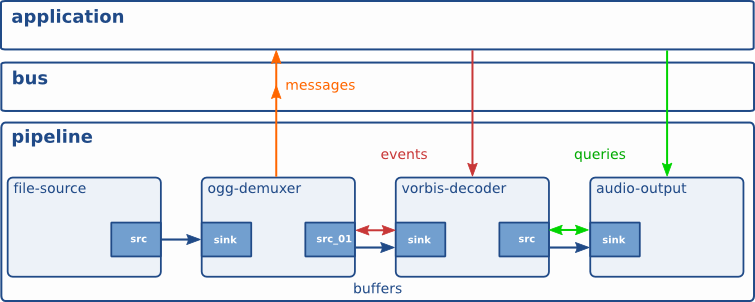




（Note：oggdemux在解析文件后，会将音频，视频通过不同的pad输出。）

**Bus线程：消息处理中心。**

在pipeline运行的过程中，各个element以及应用之间不可避免的需要进行数据消息的传输，gstreamer提供了bus系统以及多种消息类型（Buffers、Events、Messages，Queries）来达到此目的：



## **Bus**

Bus是gstreamer内部用于将消息从内部不同的streaming线程，传递到bus线程，再由bus所在线程将消息发送到应用程序。应用程序只需要向bus注册消息处理函数，即可接收到pipline中各element所发出的消息，使用bus后，应用程序就不用关心消息是从哪一个线程发出的，避免了处理多个线程同时发出消息的复杂性。

Buffers：sources到sinks的媒体数据流

Events：element到element

Message：element到app

Queries：app到element

# **gstreamer tools**

Gstreamer自带了gst-inspect-1.0和gst-launch-1.0等其他命令行工具

e.g. $ gst-inspect-1.0 playbin

用于创建及执行一个Pipline，因此通常使用gst-launch先验证相关功能，然后再编写相应应用。

一个pipeline的多个element之间通过 “!" 分隔，同时可以设置element及Cap的属性。例如：

播放音视频

gst-launch-1.0 playbin uri=file:///home/root/test.mp4

转码

gst-launch-1.0 filesrc location=/videos/sintel\_trailer-480p.ogv ! decodebin name=decode ! \

videoscale ! "video/x-raw,width=320,height=240" ! x264enc ! queue ! \

mp4mux name=mux ! filesink location=320x240.mp4 decode. ! audioconvert ! \

avenc\_aac ! queue ! mux.

Streaming

#Server

gst-launch-1.0 -v videotestsrc ! "video/x-raw,framerate=30/1" ! x264enc key-int-max=30 ! rtph264pay ! udpsink host=127.0.0.1 port=1234

#Client

gst-launch-1.0 udpsrc port=1234 ! "application/x-rtp, payload=96" ! rtph264depay ! decodebin ! autovideosink sync=false

# 第一个程序

<https://gitlab.freedesktop.org/gstreamer/gst-docs/blob/master/examples/tutorials/basic-tutorial-1.c>

下载链接中的源码，然后编译

gcc basic-tutorial-1.c -o basic-tutorial-1 `pkg-config --cflags --libs gstreamer-1.0`

编译成功后，我们可以得到可执行文件，执行 basic-tutorial-1.exe，会在弹出的窗口中，自动读取服务器上的sintel\_trailer-480p.webm视频文件并播放。如果网络环境不理想，在播放的过程中会经常处理缓冲状态，造成播放卡顿。也可以先下载媒体文件，将uri的http路径替换为本地uri（例如： uri=file:///home/john/sintel\_trailer-480p.webm）避免网络的影响。

## **源码分析**

### **GStreamer初始化**

/\* Initialize GStreamer \*/

gst\_init (&argc, &argv);

首先我们调用了gstreamer的初始化函数，该初始化函数必须在其他gstreamer接口之前被调用，gst\_init会负责以下资源的初始化：

1.初始化GStreamer库

2.注册内部element

3.加载插件列表，扫描列表中及相应路径下的插件

4.解析并执行命令行参数

5.在不需要gst\_init处理命令行参数时，我们可以讲NULL作为其参数，例如：gst\_init(NULL, NULL);

### **创建Pipeline**

/\* Build the pipeline \*/

pipeline = gst\_parse\_launch ("playbin uri=https://www.freedesktop.org/software/gstreamer-sdk/data/media/sintel\_trailer-480p.webm", NULL);

这一行是示例中的核心逻辑，展示了如何通过gst\_parse\_launch 创建一个playbin的pipeline，并设置播放文件的uri。

### **gst\_parse\_launch**

（在pipeline中，首先通过“source” element获取媒体数据，然后通过一个或多个element对编码数据进行解码，最后通过“sink” element输出声音和画面。通常在创建较复杂的pipeline时，我们需要通过gst\_element\_factory\_make来创建element，然后将其加入到GStreamer Bin中，并连接起来。）

当pipeline比较简单 并且我们不需要对pipeline中的element进行过多的控制时，我们可以采用gst\_parse\_launch 来简化pipeline的创建。

这个函数能够巧妙的将pipeline的文本描述转化为pipeline对象，我们也经常需要通过文本方式构建pipeline来查看GStreamer是否支持相应的功能，因此GStreamer提供了gst-launch-1.0命令行工具，极大的方便了pipeline的测试。

### **playbin**

pipeline中需要添加特定的element以实现相应的功能，在本例中，我们通过gst\_parse\_launch创建了只包含一个element的Pipeline。  
我们刚提到pipeline需要有“source”、“sink” element，为什么这里只需要一个playbin就够了呢？是因为**playbin element内部会根据文件的类型自动去查找所需要的“source”，“decoder”，”sink”并将它们连接起来**，同时提供了部分接口用于控制pipeline中相应的element。  
在playbin后，我们跟了一个uri参数，指定了我们想要播放的媒体文件地址，playbin会根据uri所使用的协议（“https://”，“ftp://”，“file://”等）自动选择合适的source element（此例中通过https方式）获取数据。

### 设置播放状态

/\* Start playing \*/

gst\_element\_set\_state (pipeline, GST\_STATE\_PLAYING);

这一行代码引入了一个新的概念“状态”（state）。每个GStreamer element都有相应都状态，我们目前可以简单的把状态与播放器的播放/暂停按钮联系起来，只有当状态处于PLAYING时，pipeline才会播放/处理数据。

这里gst\_element\_set\_state通过pipeline，将playbin的状态设置为PLAYING，使playbin开始播放视频文件。

### 等待播放结束

/\* Wait until error or EOS \*/

bus = gst\_element\_get\_bus (pipeline);

msg = gst\_bus\_timed\_pop\_filtered (bus, GST\_CLOCK\_TIME\_NONE, GST\_MESSAGE\_ERROR | GST\_MESSAGE\_EOS);

这几行会等待pipeline播放结束或者播放出错。我们知道GStreamer框架会通过bus，将所发生的事件通知到应用程序，因此，这里首先取得pipeline的bus对象，通过gst\_bus\_timed\_pop\_filtered 以同步的方式等待bus上的ERROR或EOS（End of Stream）消息，该函数收到消息后才会返回。

到目前为止，GStreamer会处理视频播放的所有工作（数据获取，解码，音视频同步，输出）。当到达文件末端（EOS）或出错（直接关闭播放窗口，断开网络）时，播放会自动停止。我们也可以在终端通过ctrl+c中断程序的执行。

### 释放资源

/\* Free resources \*/

if (msg != NULL)

gst\_message\_unref (msg);

gst\_object\_unref (bus);

gst\_element\_set\_state (pipeline, GST\_STATE\_NULL);

gst\_object\_unref (pipeline);

这里我们将不再使用的msg，bus对象进行销毁，并将pipeline状态设置为NULL（在NULL状态时GStreamer会释放为pipeline分配的所有资源），最后销毁pipeline对象。由于GStreamer是继承自GObject，所以需要通过gst\_object\_unref 来减少引用计数，当对象的引用计数为0时，函数内部会自动释放为其分配的内存。

不同接口会对返回的对象进行不同的处理，我们需要详细的阅读API文档，来决定我们是否需要对返回的对象进行释放。