# 9-1-FFmpeg过滤器框架分析





ffmpeg的filter用起来是和Gstreamer的plugin是一样的概念，通过avfilter_link，将各个创建好的filter按自己想要的次序链接到一起，然后avfilter_graph_config之后，就可以正常使用。
比较常用的滤镜有：scale、trim、overlay、rotate、movie、yadif。scale 滤镜用于缩放，trim 滤镜用于帧级剪切，overlay 滤镜用于视频叠加，rotate 滤镜实现旋转，movie 滤镜可以加载第三方的视频，yadif 滤镜可以去隔行。

# 1 主要结构体和API介绍

## AVFilterGraph–对filters系统的整体管理

重点

```
struct AVFilterGraph
{
    AVFilterContext **filters;
    unsigned nb_filters;
}
```

完整结构体

```
// 对filters系统的整体管理
typedef struct AVFilterGraph {
    const AVClass *av_class;
    AVFilterContext **filters;
    unsigned nb_filters;
    char *scale_sws_opts; ///< sws options to use for the auto-inserted scale filters
#if FF_API_LAVR_OPTS
    attribute_deprecated char *resample_lavr_opts;   ///< libavresample options to use for the auto-inserted resample filters
#endif
    /**
     * Type of multithreading allowed for filters in this graph. A combination
     * of AVFILTER_THREAD_* flags.
     *
     * May be set by the caller at any point, the setting will apply to all
     * filters initialized after that. The default is allowing everything.
     *
     * When a filter in this graph is initialized, this field is combined using
     * bit AND with AVFilterContext.thread_type to get the final mask used for
     * determining allowed threading types. I.e. a threading type needs to be
     * set in both to be allowed.
     */
    int thread_type;
    /**
     * Maximum number of threads used by filters in this graph. May be set by
     * the caller before adding any filters to the filtergraph. Zero (the
     * default) means that the number of threads is determined automatically.
     */
    int nb_threads;
    /**
```

```
30      * Opaque object for libavfilter internal use.
31      */
32     AVFilterGraphInternal *internal;
33     /**
34      * Opaque user data. May be set by the caller to an arbitrary va
   lue, e.g. to
35      * be used from callbacks like @ref AVFilterGraph.execute.
36      * Libavfilter will not touch this field in any way.
37      */
38     void *opaque;
39     /**
40      * This callback may be set by the caller immediately after allo
   cating the
41      * graph and before adding any filters to it, to provide a custo
   m
42      * multithreading implementation.
43      *
44      * If set, filters with slice threading capability will call thi
   s callback
45      * to execute multiple jobs in parallel.
46      *
47      * If this field is left unset, libavfilter will use its interna
   l
48      * implementation, which may or may not be multithreaded dependi
   ng on the
49      * platform and build options.
50      */
51     avfilter_execute_func *execute;
52     char *aresample_swr_opts; ///< swr options to use for the auto-i
   nserted aresample filters, Access ONLY through AVOptions
53     /**
54      * Private fields
55      *
56      * The following fields are for internal use only.
57      * Their type, offset, number and semantic can change without no
   tice.
58      */
59     AVFilterLink **sink_links;
60     int sink_links_count;
61     unsigned disable_auto_convert;
```

```
62 } AVFilterGraph;
```

## AVFilter–定义filter本身的能力

重点

const char *name;  // overlay

const AVFilterPad *inputs;

const AVFilterPad *outputs;

比如：

```
1  AVFilter ff_vf_overlay = {
2      .name          = "overlay",
3      .description   = NULL_IF_CONFIG_SMALL("Overlay a video source on
   top of the input."),
4      .preinit       = overlay_framesync_preinit,
5      .init          = init,
6      .uninit        = uninit,
7      .priv_size     = sizeof(OverlayContext),
8      .priv_class    = &overlay_class,
9      .query_formats = query_formats,
10     .activate      = activate,
11     .process_command = process_command,
12     .inputs        = avfilter_vf_overlay_inputs,
13     .outputs       = avfilter_vf_overlay_outputs,
14     .flags         = AVFILTER_FLAG_SUPPORT_TIMELINE_INTERNAL |
15                      AVFILTER_FLAG_SLICE_THREADS,
16 };
```

// 定义filter本身的能力，拥有的pads，回调函数接口定义

```
1  /**
2   * Filter definition. This defines the pads a filter contains, and
    all the
3   * callback functions used to interact with the filter.
```

```
 */
typedef struct AVFilter {
    /**
     * Filter name. Must be non-NULL and unique among filters.
     */
    const char *name;

    /**
     * A description of the filter. May be NULL.
     *
     * You should use the NULL_IF_CONFIG_SMALL() macro to define it.
     */
    const char *description;

    /**
     * List of inputs, terminated by a zeroed element.
     *
     * NULL if there are no (static) inputs. Instances of filters with
     * AVFILTER_FLAG_DYNAMIC_INPUTS set may have more inputs than present in
     * this list.
     */
    const AVFilterPad *inputs;
    /**
     * List of outputs, terminated by a zeroed element.
     *
     * NULL if there are no (static) outputs. Instances of filters with
     * AVFILTER_FLAG_DYNAMIC_OUTPUTS set may have more outputs than present in
     * this list.
     */
    const AVFilterPad *outputs;

    /**
     * A class for the private data, used to declare filter private AVOptions.
     * This field is NULL for filters that do not declare any optio
```

```
ns.
38      *
39      * If this field is non-NULL, the first member of the filter pr
ivate data
40      * must be a pointer to AVClass, which will be set by libavfilt
er generic
41      * code to this class.
42      */
43     const AVClass *priv_class;
44
45     /**
46      * A combination of AVFILTER_FLAG_*
47      */
48     int flags;
49
50     /*****************************************************************
***
51      * All fields below this line are not part of the public API. T
hey
52      * may not be used outside of libavfilter and can be changed an
d
53      * removed at will.
54      * New public fields should be added right above.
55      *****************************************************************
***
56      */
57
58     /**
59      * Filter pre-initialization function
60      *
61      * This callback will be called immediately after the filter co
ntext is
62      * allocated, to allow allocating and initing sub-objects.
63      *
64      * If this callback is not NULL, the uninit callback will be ca
lled on
65      * allocation failure.
66      *
67      * @return 0 on success,
68      *         AVERROR code on failure (but the code will be
```

```
69      *           dropped and treated as ENOMEM by the calling code)
70      */
71     int (*preinit)(AVFilterContext *ctx);
72
73     /**
74      * Filter initialization function.
75      *
76      * This callback will be called only once during the filter lif
   etime, after
77      * all the options have been set, but before links between filt
   ers are
78      * established and format negotiation is done.
79      *
80      * Basic filter initialization should be done here. Filters wit
   h dynamic
81      * inputs and/or outputs should create those inputs/outputs her
   e based on
82      * provided options. No more changes to this filter's inputs/ou
   tputs can be
83      * done after this callback.
84      *
85      * This callback must not assume that the filter links exist or
   frame
86      * parameters are known.
87      *
88      * @ref AVFilter.uninit "uninit" is guaranteed to be called eve
   n if
89      * initialization fails, so this callback does not have to clea
   n up on
90      * failure.
91      *
92      * @return 0 on success, a negative AVERROR on failure
93      */
94     int (*init)(AVFilterContext *ctx);
95
96     /**
97      * Should be set instead of @ref AVFilter.init "init" by the fi
   lters that
98      * want to pass a dictionary of AVOptions to nested contexts th
   at are
```

```
 99      * allocated during init.
100      *
101      * On return, the options dict should be freed and replaced wit
    h one that
102      * contains all the options which could not be processed by thi
    s filter (or
103      * with NULL if all the options were processed).
104      *
105      * Otherwise the semantics is the same as for @ref AVFilter.ini
    t "init".
106      */
107     int (*init_dict)(AVFilterContext *ctx, AVDictionary **options);
108
109     /**
110      * Filter uninitialization function.
111      *
112      * Called only once right before the filter is freed. Should de
    allocate any
113      * memory held by the filter, release any buffer references, et
    c. It does
114      * not need to deallocate the AVFilterContext.priv memory itsel
    f.
115      *
116      * This callback may be called even if @ref AVFilter.init "ini
    t" was not
117      * called or failed, so it must be prepared to handle such a si
    tuation.
118      */
119     void (*uninit)(AVFilterContext *ctx);
120
121     /**
122      * Query formats supported by the filter on its inputs and outp
    uts.
123      *
124      * This callback is called after the filter is initialized (so
     the inputs
125      * and outputs are fixed), shortly before the format negotiatio
    n. This
126      * callback may be called more than once.
127      *
```

```
128      * This callback must set AVFilterLink.out_formats on every inp
    ut link and
129      * AVFilterLink.in_formats on every output link to a list of pi
    xel/sample
130      * formats that the filter supports on that link. For audio lin
    ks, this
131      * filter must also set @ref AVFilterLink.in_samplerates "in_sa
    mplerates" /
132      * @ref AVFilterLink.out_samplerates "out_samplerates" and
133      * @ref AVFilterLink.in_channel_layouts "in_channel_layouts" /
134      * @ref AVFilterLink.out_channel_layouts "out_channel_layouts"
     analogously.
135      *
136      * This callback may be NULL for filters with one input, in whi
    ch case
137      * libavfilter assumes that it supports all input formats and p
    reserves
138      * them on output.
139      *
140      * @return zero on success, a negative value corresponding to a
    n
141      * AVERROR code otherwise
142      */
143     int (*query_formats)(AVFilterContext *);
144
145     int priv_size;      ///< size of private data to allocate for t
    he filter
146
147     int flags_internal; ///< Additional flags for avfilter internal
    use only.
148
149     /**
150      * Used by the filter registration system. Must not be touched
     by any other
151      * code.
152      */
153     struct AVFilter *next;
154
155     /**
156      * Make the filter instance process a command.
```

```
157      *
158      * @param cmd    the command to process, for handling simplicit
    y all commands must be alphanumeric only
159      * @param arg    the argument for the command
160      * @param res    a buffer with size res_size where the filter
    (s) can return a response. This must not change when the command is
    not supported.
161      * @param flags  if AVFILTER_CMD_FLAG_FAST is set and the comma
    nd would be
162      *               time consuming then a filter should treat it l
    ike an unsupported command
163      *
164      * @returns >=0 on success otherwise an error code.
165      *          AVERROR(ENOSYS) on unsupported commands
166      */
167     int (*process_command)(AVFilterContext *, const char *cmd, cons
    t char *arg, char *res, int res_len, int flags);
168
169     /**
170      * Filter initialization function, alternative to the init()
171      * callback. Args contains the user-supplied parameters, opaque
    is
172      * used for providing binary data.
173      */
174     int (*init_opaque)(AVFilterContext *ctx, void *opaque);
175
176     /**
177      * Filter activation function.
178      *
179      * Called when any processing is needed from the filter, instea
    d of any
180      * filter_frame and request_frame on pads.
181      *
182      * The function must examine inlinks and outlinks and perform a
    single
183      * step of processing. If there is nothing to do, the function
     must do
184      * nothing and not return an error. If more steps are or may be
185      * possible, it must use ff_filter_set_ready() to schedule anot
    her
```

```
     * activation.
     */
    int (*activate)(AVFilterContext *ctx);
} AVFilter;
```

## AVFilterContext–filter实例，管理filter与外部的联系

// filter实例，管理filter与外部的联系
重点
struct AVFilterContext
{
    const AVFilter *filter;
    char *name;

    AVFilterPad *input_pads;
    AVFilterLink **inputs;
    unsigned nb_inputs

    AVFilterPad *output_pads;
    AVFilterLink **outputs;
    unsigned nb_outputs;

    struct AVFilterGraph *graph;    // 从属于哪个AVFilterGraph
}
完整结构体

```
/** An instance of a filter */
struct AVFilterContext {
    const AVClass *av_class;        ///< needed for av_log() and fil
ters common options

    const AVFilter *filter;         ///< the AVFilter of which this
 is an instance

    char *name;                     ///< name of this filter instanc
e

    AVFilterPad   *input_pads;      ///< array of input pads
```

```
10      AVFilterLink **inputs;          ///< array of pointers to input
 links
11      unsigned    nb_inputs;          ///< number of input pads
12
13      AVFilterPad   *output_pads;     ///< array of output pads
14      AVFilterLink **outputs;         ///< array of pointers to output
links
15      unsigned    nb_outputs;         ///< number of output pads
16
17      void *priv;                     ///< private data for use by the
filter
18
19      struct AVFilterGraph *graph;    ///< filtergraph this filter bel
ongs to
20
21      /**
22       * Type of multithreading being allowed/used. A combination of
23       * AVFILTER_THREAD_* flags.
24       *
25       * May be set by the caller before initializing the filter to fo
rbid some
26       * or all kinds of multithreading for this filter. The default i
s allowing
27       * everything.
28       *
29       * When the filter is initialized, this field is combined using
 bit AND with
30       * AVFilterGraph.thread_type to get the final mask used for dete
rmining
31       * allowed threading types. I.e. a threading type needs to be se
t in both
32       * to be allowed.
33       *
34       * After the filter is initialized, libavfilter sets this field
 to the
35       * threading type that is actually used (0 for no multithreadin
g).
36       */
37      int thread_type;
38
```

```
39      /**
40       * An opaque struct for libavfilter internal use.
41       */
42      AVFilterInternal *internal;
43
44      struct AVFilterCommand *command_queue;
45
46      char *enable_str;               ///< enable expression string
47      void *enable;                   ///< parsed expression (AVExpr*)
48      double *var_values;             ///< variable values for the ena
    ble expression
49      int is_disabled;                ///< the enabled state from the
     last expression evaluation
50
51      /**
52       * For filters which will create hardware frames, sets the devic
    e the
53       * filter should create them in.  All other filters will ignore
     this field:
54       * in particular, a filter which consumes or processes hardware
     frames will
55       * instead use the hw_frames_ctx field in AVFilterLink to carry
     the
56       * hardware context information.
57       */
58      AVBufferRef *hw_device_ctx;
59
60      /**
61       * Max number of threads allowed in this filter instance.
62       * If <= 0, its value is ignored.
63       * Overrides global number of threads set per filter graph.
64       */
65      int nb_threads;
66
67      /**
68       * Ready status of the filter.
69       * A non-0 value means that the filter needs activating;
70       * a higher value suggests a more urgent activation.
71       */
72      unsigned ready;
```

```
73
74      /**
75       * Sets the number of extra hardware frames which the filter wil
   l
76       * allocate on its output links for use in following filters or
    by
77       * the caller.
78       *
79       * Some hardware filters require all frames that they will use f
   or
80       * output to be defined in advance before filtering starts.  For
   such
81       * filters, any hardware frame pools used for output must theref
   ore be
82       * of fixed size.  The extra frames set here are on top of any n
   umber
83       * that the filter needs internally in order to operate normall
   y.
84       *
85       * This field must be set before the graph containing this filte
   r is
86       * configured.
87       */
88      int extra_hw_frames;
89  };
```

## AVFilterLink–定义两个filters之间的联接

重点

```
struct AVFilterLink
{
    AVFilterContext *src;
    AVFilterPad *srcpad;

    AVFilterContext *dst;
    AVFilterPad *dstpad;

    struct AVFilterGraph *graph;
```

}

完整结构体

```
/**
 * A link between two filters. This contains pointers to the source and
 * destination filters between which this link exists, and the indexes of
 * the pads involved. In addition, this link also contains the parameters
 * which have been negotiated and agreed upon between the filter, such as
 * image dimensions, format, etc.
 *
 * Applications must not normally access the link structure directly.
 * Use the buffersrc and buffersink API instead.
 * In the future, access to the header may be reserved for filters
 * implementation.
 */
struct AVFilterLink {
    AVFilterContext *src;       ///< source filter
    AVFilterPad *srcpad;        ///< output pad on the source filter

    AVFilterContext *dst;       ///< dest filter
    AVFilterPad *dstpad;        ///< input pad on the dest filter

    enum AVMediaType type;      ///< filter media type

    /* These parameters apply only to video */
    int w;                      ///< agreed upon image width
    int h;                      ///< agreed upon image height
    AVRational sample_aspect_ratio; ///< agreed upon sample aspect ratio
    /* These parameters apply only to audio */
    uint64_t channel_layout;    ///< channel layout of current buffer (see libavutil/channel_layout.h)
    int sample_rate;            ///< samples per second
```

```
29
30      int format;                 ///< agreed upon media format
31
32      /**
33       * Define the time base used by the PTS of the frames/samples
34       * which will pass through this link.
35       * During the configuration stage, each filter is supposed to
36       * change only the output timebase, while the timebase of the
37       * input link is assumed to be an unchangeable property.
38       */
39      AVRational time_base;
40
41      /*****************************************************************
42       * All fields below this line are not part of the public API. They
43       * may not be used outside of libavfilter and can be changed and
44       * removed at will.
45       * New public fields should be added right above.
46       *****************************************************************
47       */
48      /**
49       * Lists of formats and channel layouts supported by the input and output
50       * filters respectively. These lists are used for negotiating the format
51       * to actually be used, which will be loaded into the format and
52       * channel_layout members, above, when chosen.
53       *
54       */
55      AVFilterFormats *in_formats;
56      AVFilterFormats *out_formats;
57
58      /**
59       * Lists of channel layouts and sample rates used for automatic
60       * negotiation.
61       */
```

```
62      AVFilterFormats  *in_samplerates;
63      AVFilterFormats *out_samplerates;
64      struct AVFilterChannelLayouts  *in_channel_layouts;
65      struct AVFilterChannelLayouts *out_channel_layouts;
66
67      /**
68       * Audio only, the destination filter sets this to a non-zero v
    alue to
69       * request that buffers with the given number of samples should
    be sent to
70       * it. AVFilterPad.needs_fifo must also be set on the correspon
    ding input
71       * pad.
72       * Last buffer before EOF will be padded with silence.
73       */
74      int request_samples;
75
76      /** stage of the initialization of the link properties (dimensi
    ons, etc) */
77      enum {
78          AVLINK_UNINIT = 0,      ///< not started
79          AVLINK_STARTINIT,       ///< started, but incomplete
80          AVLINK_INIT             ///< complete
81      } init_state;
82
83      /**
84       * Graph the filter belongs to.
85       */
86      struct AVFilterGraph *graph;
87
88      /**
89       * Current timestamp of the link, as defined by the most recent
90       * frame(s), in link time_base units.
91       */
92      int64_t current_pts;
93
94      /**
95       * Current timestamp of the link, as defined by the most recent
96       * frame(s), in AV_TIME_BASE units.
97       */
```

```
    int64_t current_pts_us;

    /**
     * Index in the age array.
     */
    int age_index;

    /**
     * Frame rate of the stream on the link, or 1/0 if unknown or v
ariable;
     * if left to 0/0, will be automatically copied from the first
 input
     * of the source filter if it exists.
     *
     * Sources should set it to the best estimation of the real fra
me rate.
     * If the source frame rate is unknown or variable, set this to
1/0.
     * Filters should update it if necessary depending on their fun
ction.
     * Sinks can use it to set a default output frame rate.
     * It is similar to the r_frame_rate field in AVStream.
     */
    AVRational frame_rate;

    /**
     * Buffer partially filled with samples to achieve a fixed/mini
mum size.
     */
    AVFrame *partial_buf;

    /**
     * Size of the partial buffer to allocate.
     * Must be between min_samples and max_samples.
     */
    int partial_buf_size;

    /**
     * Minimum number of samples to filter at once. If filter_frame
() is
```

```
131      * called with fewer samples, it will accumulate them in partia
    l_buf.
132      * This field and the related ones must not be changed after fi
    ltering
133      * has started.
134      * If 0, all related fields are ignored.
135      */
136     int min_samples;
137 
138     /**
139      * Maximum number of samples to filter at once. If filter_frame
    () is
140      * called with more samples, it will split them.
141      */
142     int max_samples;
143 
144     /**
145      * Number of channels.
146      */
147     int channels;
148 
149     /**
150      * Link processing flags.
151      */
152     unsigned flags;
153 
154     /**
155      * Number of past frames sent through the link.
156      */
157     int64_t frame_count_in, frame_count_out;
158 
159     /**
160      * A pointer to a FFFramePool struct.
161      */
162     void *frame_pool;
163 
164     /**
165      * True if a frame is currently wanted on the output of this fi
    lter.
166      * Set when ff_request_frame() is called by the output,
```

```
167      * cleared when a frame is filtered.
168      */
169     int frame_wanted_out;
170 
171     /**
172      * For hwaccel pixel formats, this should be a reference to the
173      * AVHWFramesContext describing the frames.
174      */
175     AVBufferRef *hw_frames_ctx;
176 
177 #ifndef FF_INTERNAL_FIELDS
178 
179     /**
180      * Internal structure members.
181      * The fields below this limit are internal for libavfilter's u
    se
182      * and must in no way be accessed by applications.
183      */
184     char reserved[0xF000];
185 
186 #else /* FF_INTERNAL_FIELDS */
187 
188     /**
189      * Queue of frames waiting to be filtered.
190      */
191     FFFrameQueue fifo;
192 
193     /**
194      * If set, the source filter can not generate a frame as is.
195      * The goal is to avoid repeatedly calling the request_frame()
     method on
196      * the same link.
197      */
198     int frame_blocked_in;
199 
200     /**
201      * Link input status.
202      * If not zero, all attempts of filter_frame will fail with the
203      * corresponding code.
204      */
```

```
205     int status_in;
206
207     /**
208      * Timestamp of the input status change.
209      */
210     int64_t status_in_pts;
211
212     /**
213      * Link output status.
214      * If not zero, all attempts of request_frame will fail with th
    e
215      * corresponding code.
216      */
217     int status_out;
218
219 #endif /* FF_INTERNAL_FIELDS */
220
221 };
```

**重点**

// 定义两个filters之间的联接
struct AVFilterLink
{
AVFilterContext *src;
AVFilterPad *srcpad;
AVFilterContext *dst;
AVFilterPad *dstpad;
struct AVFilterGraph *graph;
}

## AVFilterPad–定义filter的输入/输出接口

// 定义filter的输入/输出接口
重点
struct AVFilterPad
{
    const char *name;
    AVFrame *(*get_video_buffer)(AVFilterLink *link, int w, int h);
    AVFrame *(*get_audio_buffer)(AVFilterLink *link, int nb_samples);
    int (*filter_frame)(AVFilterLink *link, AVFrame *frame);

```
    int (*request_frame)(AVFilterLink *link);
}
```

完整结构体

```
/**
 * A filter pad used for either input or output.
 */
struct AVFilterPad {
    /**
     * Pad name. The name is unique among inputs and among outputs, but an
     * input may have the same name as an output. This may be NULL if this
     * pad has no need to ever be referenced by name.
     */
    const char *name;

    /**
     * AVFilterPad type.
     */
    enum AVMediaType type;

    /**
     * Callback function to get a video buffer. If NULL, the filter system will
     * use ff_default_get_video_buffer().
     *
     * Input video pads only.
     */
    AVFrame *(*get_video_buffer)(AVFilterLink *link, int w, int h);

    /**
     * Callback function to get an audio buffer. If NULL, the filter system will
     * use ff_default_get_audio_buffer().
     *
     * Input audio pads only.
     */
    AVFrame *(*get_audio_buffer)(AVFilterLink *link, int nb_samples)
```

```
;
32
33      /**
34       * Filtering callback. This is where a filter receives a frame w
ith
35       * audio/video data and should do its processing.
36       *
37       * Input pads only.
38       *
39       * @return >= 0 on success, a negative AVERROR on error. This fu
nction
40       * must ensure that frame is properly unreferenced on error if i
t
41       * hasn't been passed on to another filter.
42       */
43      int (*filter_frame)(AVFilterLink *link, AVFrame *frame);
44
45      /**
46       * Frame poll callback. This returns the number of immediately a
vailable
47       * samples. It should return a positive value if the next reques
t_frame()
48       * is guaranteed to return one frame (with no delay).
49       *
50       * Defaults to just calling the source poll_frame() method.
51       *
52       * Output pads only.
53       */
54      int (*poll_frame)(AVFilterLink *link);
55
56      /**
57       * Frame request callback. A call to this should result in some
 progress
58       * towards producing output over the given link. This should ret
urn zero
59       * on success, and another value on error.
60       *
61       * Output pads only.
62       */
63      int (*request_frame)(AVFilterLink *link);
```

```
64
65      /**
66       * Link configuration callback.
67       *
68       * For output pads, this should set the link properties such as
69       * width/height. This should NOT set the format property - that
    is
70       * negotiated between filters by the filter system using the
71       * query_formats() callback before this function is called.
72       *
73       * For input pads, this should check the properties of the link,
   and update
74       * the filter's internal state as necessary.
75       *
76       * For both input and output filters, this should return zero on
   success,
77       * and another value on error.
78       */
79      int (*config_props)(AVFilterLink *link);
80
81      /**
82       * The filter expects a fifo to be inserted on its input link,
83       * typically because it has a delay.
84       *
85       * input pads only.
86       */
87      int needs_fifo;
88
89      /**
90       * The filter expects writable frames from its input link,
91       * duplicating data buffers if needed.
92       *
93       * input pads only.
94       */
95      int needs_writable;
96  };
```

# AVFilterInOut–过滤器链输入/输出的链接列表

```
/**
 * A linked-list of the inputs/outputs of the filter chain.
 *
 * This is mainly useful for avfilter_graph_parse() / avfilter_graph_parse2(),
 * where it is used to communicate open (unlinked) inputs and outputs from and
 * to the caller.
 * This struct specifies, per each not connected pad contained in the graph, the
 * filter context and the pad index required for establishing a link.
 */
typedef struct AVFilterInOut {
    /** unique name for this input/output in the list */
    char *name;

    /** filter context associated to this input/output */
    AVFilterContext *filter_ctx;

    /** index of the filt_ctx pad to use for linking */
    int pad_idx;

    /** next input/input in the list, NULL if this is the last */
    struct AVFilterInOut *next;
} AVFilterInOut;
```

在AVFilter模块中定义了AVFilter结构，很个AVFilter都是具有独立功能的节点，如scale filter的作用就是进行图像尺寸变换，overlay filter的作用就是进行图像的叠加。

这里需要重点提的是两个特别的filter，**一个是buffer，一个是buffersink，**

- 滤波器buffer代表filter graph中的源头，原始数据就往这个filter节点输入的；
- 而**滤波器buffersink代表filter graph中的输出节点，处理完成的数据从这个filter节点输出。**

# 2 函数使用

// 获取FFmpeg中定义的filter，调用该方法前需要先调用avfilter_register_all();进行滤波器注册
AVFilter *avfilter_get_by_name(const char* name);

// 往源滤波器buffer中输入待处理的数据
int av_buffersrc_add_frame(AVFilterContext *ctx, AVFrame* frame);

// 从目的滤波器buffersink中获取处理完的数据
int av_buffersink_get_frame(AVFilterContext *ctx, AVFrame* frame);

// 创建一个滤波器图filter graph
AVFilterGraph *avfilter_graph_alloc(void);

// 创建一个滤波器实例AVFilterContext，并添加到AVFilterGraph中
int avfilter_graph_create_filter(AVFilterContext **filt_ctx, const AVFilter *filt,
const char *name, const char* args, void *opaque,
AVFilterGraph *graph_ctx);

// 连接两个滤波器节点
int avfilter_link(AVFilterContext *src, unsigned srcpad,
AVFilterContext *dst, unsigned dstpad);

# 3 AVFilter主体框架流程

在利用AVFilter进行音视频数据处理前先将在进行的处理流程绘制出来，现在以FFmpeg filter官方文档中的一个例子为例进行说明。

```
                [main]
input --> split ----------------------> overlay --> output
            |                             ^
            |[tmp]                  [flip]|
            +-----> crop --> vflip -------+
```

这个例子的处理流程如上所示，首先使用split滤波器将input流分成两路流（main和tmp），然后分别对两路流进行处理。对于tmp流，先经过crop滤波器进行裁剪处理，再经过flip滤波器进行垂直方向上的翻转操作，输出的结果命名为flip流。再将main流和flip流输入到overlay滤波器进行合成操作。上图的input就是

上面提过的buffer源滤波器，output就是上面的提过的buffersink滤波器。上图中每个节点都是一个AVFilterContext，每个连线就是AVFliterLink。所有这些信息都统一由AVFilterGraph来管理。

AVFilterInOut AVFilterGraph