1.mupdf.h

整个库的核心功能函数声明在mupdf.h头文件中,该头文件包含了一些转换函数(将库内定义的一些变量转换为int,float等基本格式),以及这个库提供的最主要的功能函数:解析与渲染PDF

MuPDF提供了两种打开PDF文件的方式:
pdf_document *pdf_open_document(fz_context *ctx, const char *filename);
pdf_document *pdf_open_document_with_stream(fz_stream *file);
分别是从路径与给定内容流的方式打开PDF文档

读取页号与统计页数量的一些函数:
int pdf_lookup_page_number(pdf_document *doc, pdf_obj *pageobj);
int pdf_count_pages(pdf_document *doc);

加载与释放page资源的函数
pdf_page *pdf_load_page(pdf_document *doc, int number);
void pdf_free_page(pdf_document *doc, pdf_page *page);

最后,是实现了最主要功能的函数
void pdf_run_page(pdf_document *doc, pdf_page *page, fz_device *dev, fz_matrix ctm, fz_cookie *cookie);//解释加载一个page的内容并将其呈现在设备上

mupdf官网给出的使用这个库的流程大致是这样子的:

```
usemupdf(const char* path)//初始化文件
{
    ctx=fz_new_context(NULL,NULL,FZ_STORE_UNLIMITED);//初始化context
   doc= fz_open_document(ctx,(char*)path);//读取PDF文件并提取出其中信息放入fz_document类的结构体中
    page_number=fz_count_pages(doc);//统计PDF文件的页数
    for (int i=0;i<page_number;i++)//对每一页进行处理
    {
        page=fz_load_page(doc,i);//加载页面内容
        rect=fz_bound_page(doc,page);//获取页面的rectangle对象
        this->rect=rect;
```

```
            //设置页面转置矩阵
              ctm.a=1;
              ctm.b=0;
              ctm.c=0;
              ctm.d=1;
              ctm.e=0;
              ctm.f=0;

pix=fz_new_pixmap(ctx,fz_find_device_colorspace(ctx,"DeviceRGB"),rect.x1-
rect.x0,rect.y1-rect.y0);
              fz_clear_pixmap_with_value(ctx,pix,0xFF);//将背景设置为白色
              dev=fz_new_draw_device(ctx,pix);//创建绘图设备
              fz_run_page(doc,page,dev,ctm,0);//将页面绘制到pixmap生成的绘图设备上
              this->PageCount++;
        }
}
```

因此void pdf_run_page(pdf_document *doc, pdf_page *page, fz_device *dev, fz_matrix ctm, fz_cookie *cookie);函数就是我们需要重点去看源码的

2.mupdf-internal.h

这个头文件存放了MuPDF库的一些核心结构体

比如:Document结构体

```
struct pdf_document_s//PDF整体架构结构体
{
        fz_document super;

        fz_context *ctx;
        fz_stream *file;

        int version;//PDF版本号
        int startxref;//交叉引用表xref table的偏移地址
        int file_size;//文件大小
        pdf_crypt *crypt;
        pdf_obj *trailer;//文件尾trailer对象
```

```
	pdf_ocg_descriptor *ocg;

	int len;
	pdf_xref_entry *table;//交叉引用表xref table

	int page_len;
	int page_cap;
	pdf_obj **page_objs;//存放各个页的object地址
	pdf_obj **page_refs;

	pdf_lexbuf_large lexbuf;
};
```

以及一些对内容流的分类枚举变量以及操作函数

PDF function的处理函数:

```
pdf_function *pdf_load_function(pdf_document *doc, pdf_obj *ref);
void pdf_eval_function(fz_context *ctx, pdf_function *func, float *in, int inlen, float *out, int outlen);
pdf_function *pdf_keep_function(fz_context *ctx, pdf_function *func);
void pdf_drop_function(fz_context *ctx, pdf_function *func);
unsigned int pdf_function_size(pdf_function *func);
```

XObject的处理函数与结构体:

```
struct pdf_xobject_s
{
	fz_storable storable;
	fz_matrix matrix;
	fz_rect bbox;
	int isolated;
	int knockout;
	int transparency;
	fz_colorspace *colorspace;
	pdf_obj *resources;
	fz_buffer *contents;
	pdf_obj *me;
};
```

```
pdf_xobject *pdf_load_xobject(pdf_document *doc, pdf_obj *obj);
pdf_xobject *pdf_keep_xobject(fz_context *ctx, pdf_xobject *xobj);
void pdf_drop_xobject(fz_context *ctx, pdf_xobject *xobj);
/* SumatraPDF: allow to synthesize XObjects (cf. pdf_create_annot) */
pdf_xobject *pdf_create_xobject(fz_context *ctx, pdf_obj *dict);
```

以及PDFpage结构体

```
struct pdf_page_s //保存PDF页面信息
{
	fz_matrix ctm; /* calculated from mediabox and rotate */
	fz_rect mediabox;
	int rotate;
	int transparency;//透明度
	pdf_obj *resources;//资源
	fz_buffer *contents;//内容流
	fz_link *links;
	pdf_annot *annots;
};
```

3.pdf_interpret.c

这个文件里面的函数基本上都是解析page内容的,先看最外层函数:

3925行

```
void
pdf_run_page(pdf_document *xref, pdf_page *page, fz_device *dev, fz_matrix ctm,
fz_cookie *cookie)
{
	pdf_run_page_with_usage(xref, page, dev, ctm, "View", cookie);
}
```

调用了pdf_run_page_with_usage(xref, page, dev, ctm, "View", cookie);

3849行

```
void
```

```
pdf_run_page_with_usage(pdf_document *xref, pdf_page *page, fz_device *dev,
fz_matrix ctm, char *event, fz_cookie *cookie)
{
	fz_context *ctx = dev->ctx;
	pdf_csi *csi;
	pdf_annot *annot;
	int flags;

	ctm = fz_concat(page->ctm, ctm);//转换矩阵


	if (page->transparency)//透明页面
		fz_begin_group(dev, fz_transform_rect(ctm, page->mediabox), 1, 0, 0, 1);

	csi = pdf_new_csi(xref, dev, ctm, event, cookie, NULL);
	fz_try(ctx)
	{
		pdf_run_buffer(csi, page->resources, page->contents);//解析页面contents
	}
	fz_catch(ctx)
	{
		pdf_free_csi(csi);
		fz_throw(ctx, "cannot parse page content stream");//无法解析页面内容流
	}
	pdf_free_csi(csi);

	if (cookie && cookie->progress_max != -1)
	{
		int count = 1;
		for (annot = page->annots; annot; annot = annot->next)
			count++;
		cookie->progress_max += count;
	}

	for (annot = page->annots; annot; annot = annot->next)
```

```
	{
		/* Check the cookie for aborting */
		if (cookie)
		{
			if (cookie->abort)
				break;
			cookie->progress++;
		}

		flags = pdf_to_int(pdf_dict_gets(annot->obj, "F"));

		/* TODO: NoZoom and NoRotate */
		if (flags & (1 << 0)) /* Invisible */
			continue;
		if (flags & (1 << 1)) /* Hidden */
			continue;
		if (!strcmp(event, "Print") && !(flags & (1 << 2))) /* Print */
			continue;
		if (!strcmp(event, "View") && (flags & (1 << 5))) /* NoView */
			continue;

		csi = pdf_new_csi(xref, dev, ctm, event, cookie, NULL);
		if (!pdf_is_hidden_ocg(pdf_dict_gets(annot->obj, "OC"), csi, page->resources))
		{
			fz_try(ctx)
			{
				pdf_run_xobject(csi, page->resources, annot->ap, annot->matrix);
			}
			fz_catch(ctx)
			{
				pdf_free_csi(csi);
				fz_throw(ctx, "cannot parse annotation appearance stream");
			}
```

```
            }
            pdf_free_csi(csi);
        }

        if (page->transparency)
            fz_end_group(dev);
}
```

这里面主要功能函数:pdf_run_buffer(csi, page->resources, page->contents);//解析页面contents

mupdf是将PDF每一页的contents读入buffer中再进行解析的

3807行

```
static void
pdf_run_buffer(pdf_csi *csi, pdf_obj *rdb, fz_buffer *contents)
{
    fz_context *ctx = csi->dev->ctx;
    pdf_lexbuf_large *buf;
    fz_stream * file = NULL;
    int save_in_text;

    fz_var(buf);
    fz_var(file);

    if (contents == NULL)
        return;

    fz_try(ctx)
    {
        buf = fz_malloc(ctx, sizeof(*buf)); /* we must be re-entrant for type3 fonts
```

我们必须为Type3字体重入*/

```
        buf->base.size = PDF_LEXBUF_LARGE;
        file = fz_open_buffer(ctx, contents);
        save_in_text = csi->in_text;
        csi->in_text = 0;
        fz_try(ctx)
```

```
            {
                pdf_run_stream(csi, rdb, file, &buf->base);//绘制内容流
            }
            fz_catch(ctx)
            {
                fz_warn(ctx, "Content stream parsing error - rendering truncated");//
内容流解析错误-呈现截断
            }
            csi->in_text = save_in_text;
        }
        fz_always(ctx)
        {
            fz_close(file);
            fz_free(ctx, buf);
        }
        fz_catch(ctx)
        {
            fz_throw(ctx, "cannot parse context stream");
        }
}
```

可以看到,主要调用了pdf_run_stream(csi, rdb, file, &buf->base)函数来绘制内容流

3643行

```
static void
pdf_run_stream(pdf_csi *csi, pdf_obj *rdb, fz_stream *file, pdf_lexbuf *buf)
```

该函数功能是对内容流进行分类并调用相应函数绘制

该函数中3698行调用了pdf_lex函数

```
tok = pdf_lex(file, buf);
```

pdf_lex函数进行分类操作，根据从流中读取的符号内容将流数据分成不同的类别，tok作为返回值，根据不同类别的内容，返回不同的数据

PS:分类结果参照mupdf-internal.h中的92行

```
    enum//对流内部的数据进行分类
    {
        PDF_TOK_ERROR, PDF_TOK_EOF,
        PDF_TOK_OPEN_ARRAY, PDF_TOK_CLOSE_ARRAY,
```

```
        PDF_TOK_OPEN_DICT, PDF_TOK_CLOSE_DICT,
        PDF_TOK_OPEN_BRACE, PDF_TOK_CLOSE_BRACE,
        PDF_TOK_NAME, PDF_TOK_INT, PDF_TOK_REAL, PDF_TOK_STRING,
    PDF_TOK_KEYWORD,
        PDF_TOK_R, PDF_TOK_TRUE, PDF_TOK_FALSE, PDF_TOK_NULL,
        PDF_TOK_OBJ, PDF_TOK_ENDOBJ,
        PDF_TOK_STREAM, PDF_TOK_ENDSTREAM,
        PDF_TOK_XREF, PDF_TOK_TRAILER, PDF_TOK_STARTXREF,
        PDF_NUM_TOKENS
    };
```

pdf_interpret.c文件3734行根据tok的值对内容流进行分类处理,如下所示

```
        switch (tok)
         {
         case PDF_TOK_ENDSTREAM://流结尾
         case PDF_TOK_EOF:
                return;

         case PDF_TOK_OPEN_ARRAY://数组开始
                if (!csi->in_text)
                {
                        csi->obj = pdf_parse_array(csi->xref, file, buf);//解析数组
                        /* RJW: "cannot parse array" */
                }
                else
                {
                        in_array = 1;
                }
                break;

         case PDF_TOK_OPEN_DICT:
                csi->obj = pdf_parse_dict(csi->xref, file, buf);//处理字典
                /* RJW: "cannot parse dictionary" */
                break;
```

```
		case PDF_TOK_NAME:
			fz_strlcpy(csi->name, buf->scratch, sizeof(csi->name));//处理name类
型变量
			//printf("%s\n",csi->name);
			break;

		case PDF_TOK_INT:
			csi->stack[csi->top] = buf->i;
			csi->top ++;
			break;

		case PDF_TOK_REAL:
			csi->stack[csi->top] = buf->f;
			csi->top ++;
			break;

		case PDF_TOK_STRING://处理String类型变量
			if (buf->len <= sizeof(csi->string))
			{
				memcpy(csi->string, buf->scratch, buf->len);
#ifdef debug
				printf("%s\n",csi->string);
#endif
				csi->string_len = buf->len;
			}
			else
			{
				csi->obj = pdf_new_string(ctx, buf->scratch, buf->len);
			}
			break;

		case PDF_TOK_KEYWORD:
			pdf_run_keyword(csi, rdb, file, buf->scratch);
			/* RJW: "cannot run keyword" */
			pdf_clear_stack(csi);
```

```
            break;

        default:
            fz_throw(ctx, "syntax error in content stream");
        }
```

我们需要注意的是pdf_run_keyword(csi, rdb, file, buf->scratch)函数,这部分内容流就是PDFcontent里面的命令,记录了大部分PDF图形或文字对象的属性参数

3503行
pdf_run_keyword(pdf_csi *csi, pdf_obj *rdb, fz_stream *file, char *buf)
该函数就是根据解析出的keyword执行对应的PDF操作
该部分代码核心逻辑如下所示:

```
switch (key)
    {
    case A('"'): pdf_run_dquote(csi); break;
    case A('\''): pdf_run_squote(csi); break;
    case A('B'): pdf_run_B(csi); break;//路径绘制 填充然后涂抹路径 使用非零缠绕数原则
    case B('B','*'): pdf_run_Bstar(csi); break;//路径绘制 填充然后涂抹路径 使用奇偶原则
    case C('B','D','C'): pdf_run_BDC(csi, rdb); break;
    case B('B','I'):
        pdf_run_BI(csi, rdb, file);
        /* RJW: "cannot draw inline image" */
        break;
    case C('B','M','C'): pdf_run_BMC(csi); break;
    case B('B','T'): pdf_run_BT(csi); break;//文本对象 开始一个文本对象，初始化文本矩阵Tm和文本行矩阵Tlm
    case B('B','X'): pdf_run_BX(csi); break;
    case B('C','S'): pdf_run_CS(csi, rdb); break;
    case B('D','P'): pdf_run_DP(csi); break;
    case B('D','o'):
        fz_try(ctx)
        {
```

```
                pdf_run_Do(csi, rdb);
            }
            fz_catch(ctx)
            {
                fz_warn(ctx, "cannot draw xobject/image");
            }
            break;
        case C('E','M','C'): pdf_run_EMC(csi); break;
        case B('E','T'): pdf_run_ET(csi); break;//文本对象 完成一个文本对象，结束一个文本矩阵（下层尚不明了）
        case B('E','X'): pdf_run_EX(csi); break;
        case A('F'): pdf_run_F(csi); break;//路径绘制 相当于f，为了兼容性存在
        case A('G'): pdf_run_G(csi); break;//色彩空间设置 G空间
        case A('J'): pdf_run_J(csi); break;//图形状态操作 设置线端口样式
        case A('K'): pdf_run_K(csi); break;
        case A('M'): pdf_run_M(csi); break;//图形状态操作 设置斜切极限
        case B('M','P'): pdf_run_MP(csi); break;
        case A('Q'): pdf_run_Q(csi); break;//图形状态操作 删除堆栈中最新存储的状态来还原图形状态
        case B('R','G'): pdf_run_RG(csi); break;//色彩空间设置 RGB空间
        case A('S'): pdf_run_S(csi); break;//路径绘制 涂抹（路径绘制均调用pdf_show_path函数，只不过参数不同，效果不同）
        case B('S','C'): pdf_run_SC(csi, rdb); break;
        case C('S','C','N'): pdf_run_SC(csi, rdb); break;
        case B('T','*'): pdf_run_Tstar(csi); break;//文本位置操作 移动到下一行首部
        case B('T','D'): pdf_run_TD(csi); break;//文本位置操作 移动到下一行的开始
        case B('T','J'): pdf_run_TJ(csi); break;//文本显示操作
        case B('T','L'): pdf_run_TL(csi); break;//文本行距
        case B('T','c'): pdf_run_Tc(csi); break;//文本字符间距
        case B('T','d'): pdf_run_Td(csi); break;//文本位置操作 移动到下一行的开始
        case B('T','f'):文本字体
            fz_try(ctx)
            {
                pdf_run_Tf(csi, rdb);
            }
```

```
            fz_catch(ctx)
            {
                fz_warn(ctx, "cannot set font");
            }
            break;
        case B('T','j'): pdf_run_Tj(csi); break;//文本显示操作 显示字符串
        case B('T','m'): pdf_run_Tm(csi); break;//文本位置操作 设置文本矩阵Tm和文本行矩阵Tlm
        case B('T','r'): pdf_run_Tr(csi); break;
        case B('T','s'): pdf_run_Ts(csi); break;
        case B('T','w'): pdf_run_Tw(csi); break;//文本字间距
        case B('T','z'): pdf_run_Tz(csi); break;
        case A('W'): pdf_run_W(csi); break;//路径剪辑 通过使当前路径与其相交修改当前路径，非零绕数规则
        case B('W','*'): pdf_run_Wstar(csi); break;//路径剪辑 通过使当前路径与其相交修改当前路径，奇偶规则
        case A('b'): pdf_run_b(csi); break;//路径绘制 关闭，填充，并涂抹路径，非零缠绕规则
        case B('b','*'): pdf_run_bstar(csi); break;//路径绘制 关闭，填充，并涂抹路径，奇偶原则
        case A('c'): pdf_run_c(csi); break;//路径构造 向当前路径加入一条三次贝塞尔曲线
        case B('c','m'): pdf_run_cm(csi); break;//图形状态操作 指定CTM数组
        case B('c','s'): pdf_run_cs(csi, rdb); break;
        case A('d'): pdf_run_d(csi); break;//图形状态操作 设置虚线模式
        case B('d','0'): pdf_run_d0(csi); break;
        case B('d','1'): pdf_run_d1(csi); break;
        case A('f'): pdf_run_f(csi); break;//路径绘制 填充路径，使用非零缠绕数规则决定填充的区域
        case B('f','*'): pdf_run_fstar(csi); break;//路径绘制 使用奇偶填充原则
        case A('g'): pdf_run_g(csi); break;//色彩空间设置 g空间
        case B('g','s')://图形状态操作 设置图形状态中指定的参数
            fz_try(ctx)
            {
                pdf_run_gs(csi, rdb);
            }
```

```
		fz_catch(ctx)
		{
			fz_warn(ctx, "cannot set graphics state");
		}
		break;
	case A('h'): pdf_run_h(csi); break;//路径构造 从当前点到子路径的起点连接一条直线段，关闭子路径
	case A('i'): pdf_run_i(csi); break;//图形状态操作 设置平面度公差
	case A('j'): pdf_run_j(csi); break;//图形状态操作 设置线连接样式
	case A('k'): pdf_run_k(csi); break;
	case A('l'): pdf_run_l(csi); break;//路径构造 从当前点向（x，y）追加一条直线线段，新的当前点修改为（x，y）
	case A('m'): pdf_run_m(csi); break;//路径构造 将当前点移动到坐标（x，y），开始一个新的子路径
	case A('n'): pdf_run_n(csi); break;//路径绘制 不填充涂抹。，直接结束路径对象
	case A('q'): pdf_run_q(csi); break;//图形状态操作 保存图形状态堆栈中的当前操作
	case B('r','e'): pdf_run_re(csi); break;//路径构造 向当前路径中加入一个矩形作为封闭的子路径
	case B('r','g'): pdf_run_rg(csi); break;
	case B('r','i'): pdf_run_ri(csi); break;//图形状态操作 设置图像显示方式
	case A('s'): pdf_run(csi); break;//路径绘制 涂抹并关闭路径
	case B('s','c'): pdf_run_sc(csi, rdb); break;
	case C('s','c','n'): pdf_run_sc(csi, rdb); break;
	case B('s','h'):
		fz_try(ctx)
		{
			pdf_run_sh(csi, rdb);
		}
		fz_catch(ctx)
		{
			fz_warn(ctx, "cannot draw shading");
		}
		break;
	case A('v'): pdf_run_v(csi); break;//路径构造 向当前路径加入一条三次贝塞尔曲线
	case A('w'): pdf_run_w(csi); break;//图形状态操作 设置线宽
```

```
case A('y'): pdf_run_y(csi); break;//路径构造 向当前路径加入一条三次贝塞尔曲线
default:
    if (!csi->xbalance)
        fz_warn(ctx, "unknown keyword: '%s'", buf);
    break;
}
```

这些命令均可以在PDF reference文档中找到

表 4.1 操作类别

类别	操作	表	页码
一般图形状态	w, J, j, M, d, ri, i, gs	4.7	189
特殊图形状态	q, Q, cm	4.7	189
路径构造	m, l, c, v, y, h, re	4.9	196
路径绘制	S, s, f, F, f*, B, B*, b, b*, n	4.10	200
裁剪路径	W, W*	4.11	205
文本对象	BT, ET	5.4	375
文本状态	Tc, Tw, Tz, TL, Tf, Tr, Ts	5.2	368
文本配置	Td, TD, Tm, T*	5.5	376
文本显示	Tj, TJ,’ ,”	5.6	377
类型 3 字体	d0, d1	5.10	392
颜色	CS, cs, SC, SCN, sc, scn, G, g, RG, rg, K, k	4.24	257
阴影模式	sh	4.27	273
内嵌图像	BI, ID, EI	4.42	322
外在对象	Do	4.37	302
标记内容	MP, DP, BMC, BDC, EMC	10.7	779
兼容性	BX, EX	3.29	127

至此,整体的逻辑已经大体清晰了,具体的命令跟进到上述switch里面的函数即可取得相应的值

4.特殊对象

然而,PDF还存在几个特殊的对象

(1).图形状态参数对象:大部分情况下,简单的图形是在上述content内容流中由这些指令序列描述的,但是,由于PDF早期版本的原因,有部分属性存在于图形状态参数字典中

尽管在图形状态中的一些参数可以被独立的操作设置，如表 4.7 所示，但是其他的不行。后者只能被一半的图形状态操作 gs（PDF1.2）设定。提供给这个操作的操作说是一个包含定义一个或多个图形状态参数的图形状态参数字典的名字。这个名字在当前源字典的 ExtGState 子字典中可查找。(ExtGState，针对延伸图形状态，是一种早期版本的 PDF 的剩余)。

pdf_interpret.c文件中

1773行函数

static void pdf_run_extgstate(pdf_csi *csi, pdf_obj *rdb, pdf_obj *extgstate)用来处理图形状态参数集,

这个函数内部提取了相应图形状态参数字典中的指令,包括透明度在内的许多参数都可以从这里找到

(2)PDF函数

在使用渐变色的时候,PDF会使用函数来保存渐变色的参数,比如RGB值从(0,0,0)变换到(255,255,255),这些变换在PDF中是由函数进行表示的.

pdf_shade.c文件中的函数

fz_shade *pdf_load_shading(pdf_document *xref, pdf_obj *dict)

用来加载shade阴影用以实现渐变色

相同文件中的函数

static void pdf_sample_shade_function(fz_context *ctx, fz_shade *shade, int funcs, pdf_function **func, float t0, float t1)

用来处理函数的内容,进行对函数的分类解析,内部调用了pdf_function的

void pdf_eval_function(fz_context *ctx, pdf_function *func, float *in, int inlen, float *out, int outlen)

函数,判断function类型,对不同的function进行分类处理,PDF中有四种类型的function

所以上述函数核心内容为

```
switch(func->type)//根据类型处理
 {
 case SAMPLE: eval_sample_func(ctx, func, in, out); break;
 case EXPONENTIAL: eval_exponential_func(ctx, func, *in, out); break;
```

```
case STITCHING: eval_stitching_func(ctx, func, *in, out); break;
case POSTSCRIPT: eval_postscript_func(ctx, func, in, out); break;
}
```

据此可以根据类型跳转到相应函数中

5.总结

至此,我们完成了MuPDF的读取与解析操作部分源码的讲解

PDF解析的过程可以看作是这样的

PDF文件->Document对象->page(多个)->对每个page,分类其内容流->解析相应的指令内容流->根据

指令调用不同的函数进行指令的处理->存在一些特殊的图形对象(这些对象的参数与内容流所在的content位于不同的object中,需要调用跨object的函数去提取相应信息)->得到内存中的一个完整对象并进行渲染