**Dbstar阅读器模块开发文档**

Dbstar阅读器是以FBReader为核心，基于FBreader定制开发而成，目前能够支持EPUB、OEB、FB2(.zip)，TXT等电子书格式，主要包括图书解析，页面加载，页面绘制，翻页动画几大部分。

阅读器启动到打开图书大致流程如下：

程序通过DbStarPlayer启动，初始化完成阅读相关的设置和服务，程序通过如下代码完成阅读界面的设置，即ZLAndroidWidget的实例的设置，**final** ZLAndroidApplication androidApplication = (ZLAndroidApplication)getApplication();

**if** (androidApplication.myMainWindow == **null**) {

androidApplication.myMainWindow = **new** ZLAndroidApplicationWindow(myDbStarPlayerApp);

myDbStarPlayerApp.initWindow();

}

设置完后，注册了一系列的action用于响应用户的操作，然后在onStart函数中添加了启动图书加载的任务，正式开始进入图书的加载任务，

getCollection().bindToService(**this**, **new** Runnable() {

**public** **void** run() {

**new** Thread() {

**public** **void** run() {

openBook(getIntent(), getPostponedInitAction(), **false**);

myDbStarPlayerApp.getViewWidget().repaint();

}

}.start();

myDbStarPlayerApp.getViewWidget().repaint();

}

});

**private** **synchronized** **void** openBook(Intent intent, Runnable action, **boolean** force) {

**if** (!force && myBook != **null**) {

**return**;

}

myBook = SerializerUtil.*deserializeBook*(intent.getStringExtra(***BOOK\_KEY***));

**final** Bookmark bookmark =

SerializerUtil.*deserializeBookmark*(intent.getStringExtra(***BOOKMARK\_KEY***));

**if** (myBook == **null**) {

**final** Uri data = intent.getData();

**if** (data != **null**) {

myBook = createBookForFile(ZLFile.*createFileByPath*(data.getPath()));

}**else**{

myBook = createBookForFile(ZLFile.*createFileByPath*("/mnt/sdcard/myBooks/xiyouji.epub"));

}

}

myDbStarPlayerApp.openBook(myBook, bookmark, action);

}

开始正式打开图书。

openBooK代码清单如下所示：

**public** **void** openBook(**final** Book book, **final** Bookmark bookmark, **final** Runnable postAction) {

**if**(book==**null**){

mHandler.sendEmptyMessage(MSG\_OPEN\_BOOK\_FAILED);

**return**;

}

**if** (book != **null** || Model == **null**) {

runWithMessage("loadingBook", **new** Runnable() {

**public** **void** run() {

openBookInternal(book, bookmark, **false**);

**if** (book != **null**) {

book.addLabel(Book.***READ\_LABEL***);

Collection.saveBook(book, **false**);

curBook = book;

}

}

}, postAction);

}

}

从代码中可以发现直正执行加载图书任务是openBookInternal函数，代码清单如下

**synchronized** **void** openBookInternal(Book book, Bookmark bookmark, **boolean** force) {

**if** (book == **null**) {

Utils.*printLogError*(getClass().getSimpleName(), "Can't get book to open!!!");

mHandler.sendEmptyMessage(MSG\_OPEN\_BOOK\_FAILED);

**return**;

}**else**{

curBook = book;

}

**if** (!force && Model != **null** && book.equals(Model.Book)) {

**if** (bookmark != **null**) {

gotoBookmark(bookmark);

}

**return**;

}

onViewChanged();

BookTextView.setModel(**null**);

FootnoteView.setModel(**null**);

clearTextCaches();

Model = **null**;

System.*gc*();

System.*gc*();

**try** {

Model = BookModel.*createModel*(book);

Collection.saveBook(book, **false**);

ZLTextHyphenator.*Instance*().load(book.getLanguage());

BookTextView.setModel(Model.getTextModel());

setBookmarkHighlightings(BookTextView, **null**);

BookTextView.gotoPosition(Collection.getStoredPosition(book.getId()));

**if** (bookmark == **null**) {

setView(BookTextView);

} **else** {

gotoBookmark(bookmark);

}

Collection.addBookToRecentList(book);

**final** StringBuilder title = **new** StringBuilder(book.getTitle());

**if** (!book.authors().isEmpty()) {

**boolean** first = **true**;

**for** (Author a : book.authors()) {

title.append(first ? " (" : ", ");

title.append(a.DisplayName);

first = **false**;

}

title.append(")");

}

setTitle(title.toString());

initChapterList(Model.TOCTree);

} **catch** (BookReadingException e) {

processException(e);

}

getViewWidget().reset();

getViewWidget().repaint();

}

所有参数体测通过后，清除缓存，通过createModel创建图书存储结构，并读取图书内容存储到该结构。进入createModel会发现真正开始解析和存储是FormatPlugin类的public abstract boolean readModel(BookModel model)函数实现,进入readModel（以epub为例），首先读取epub文件的opf文件，即epub一系列文件的引用的入口，只有取得opf文件信息后才能通过opf文件里的信息找到书的封面，章节，图片等内容信息的地址(相对地址)，进入，然后进入OEBBookReader的readBook函数后通过read方法解析出opf文件信息，即epub包含的文件的地址并存储到html文件列表，这个时候到了真正解析图书内容的时候了，

**for** (String name : myHtmlFileNames) {

**final** ZLFile xhtmlFile = ZLFile.*createFileByPath*(myFilePrefix + name);

**if** (xhtmlFile == **null** || !xhtmlFile.exists()) {

**continue**;

}

**if** (count++ == 0 && xhtmlFile.getPath().equals(myCoverFileName)) {

**continue**;

}

**final** XHTMLReader reader = **new** XHTMLReader(myModelReader, myFileNumbers);

**final** String referenceName = reader.getFileAlias(MiscUtil.*archiveEntryName*(xhtmlFile.getPath()));

myModelReader.addHyperlinkLabel(referenceName);

myTOCLabels.put(referenceName, myModelReader.Model.BookTextModel.getParagraphsNumber());

**try** {

reader.readFile(xhtmlFile, referenceName + '#');

} **catch** (IOException e) {

**throw** **new** BookReadingException(e, xhtmlFile);

}

myModelReader.insertEndOfSectionParagraph();

}逐个的对html文件列表的文件进行解析并存储。解析的过程中会把相应的段落信息存储到model结构。然后再通过private void generateTOC()解析目录文件，生成目录文件信息。然后返回到openBookInternal，做一些初始化操作，比如跳转到上次阅读位置等，然后进入渲染流程。

图书解析完成后，进入加载渲染流程，渲染的时候实际上是通过canvas把需要显示的内容绘制到bitmap上完成。绘制的时候可以在添加背景，对需要绘制的内容进行结构排版，排版的过程实际上是根据页面的首或尾的游标进行向后或者向前移动并计算内容多少的过程，详细流程如下：

因为要进行翻页，而翻的时候可以同时看到两页的内容，所以需要2个bitmap来存储两页的内容，这两个bitmap可以在翻页的时候进行重复使用，只是绘制的内容不一样，从而实现不断翻页能看到页面内容变化，而不需要不断生成新的bitmap去存储新的页面内容。

绘制的时候主要是调用BitmapManager的函数

/\*\*

\* progress :read progress page index

\*/

**public** Bitmap getBitmap(ZLView.PageIndex index, EdgePosition edgepos, ZLView.PageIndex progress) {

/\*\*

\* drop cache for change background,

\* so now background will change every time

\* \*/

**for** (**int** i = 0; i < SIZE; ++i) {

**if** (index == myIndexes[i]) {

**if**(pageEntities[i].edgePos!=edgepos){

paintPage(i,index,edgepos,progress);

}

**if**(pageEntities[i].bmp!=**null**&&!pageEntities[i].bmp.isRecycled()){

**return** pageEntities[i].bmp;

}**else**{

/\*\*

\* exception the bitmap is recycled, rarely appear

\* should repaint

\* \*/

pageEntities[i].bmp = **null**;

pageEntities[i].bmp = Bitmap.*createBitmap*(myWidth, myHeight, Bitmap.Config.***ARGB\_8888***);

paintPage(i,index,edgepos,progress);

**return** pageEntities[i].bmp;

}

}

}

**final** **int** iIndex = getInternalIndex(index);

myIndexes[iIndex] = index;

**if** (pageEntities[iIndex].bmp == **null**) {

**try**{

Utils.*printLogError*(getClass().getSimpleName(), "pageEntities bmp==null create it iIndex="+iIndex);

pageEntities[iIndex].bmp = Bitmap.*createBitmap*(myWidth, myHeight, Bitmap.Config.***ARGB\_8888***);

}**catch** (OutOfMemoryError e) {

// **TODO**: handle exception

Utils.*printLogError*(getClass().getSimpleName(), "OutOfMemoryError "+e.getMessage());

e.printStackTrace();

System.*gc*();

System.*gc*();

pageEntities[iIndex].bmp = Bitmap.*createBitmap*(myWidth, myHeight, Bitmap.Config.***ARGB\_8888***);

}

}

paintPage(iIndex,index,edgepos,progress);

**return** pageEntities[iIndex].bmp;

}

从代码清单可以看到，绘制的时候首先会从bitmap数组，即页面信息中去检查是否需要重新绘制页面内容。如果需要绘制，则会调用paintPage函数进行绘制。 **private** **void** paintPage(**int** cacheindex, ZLView.PageIndex pageindex,EdgePosition edgepos,ZLView.PageIndex progress){

myWidget.drawOnBitmap(myWidth,myHeight,pageEntities[cacheindex].bmp, pageindex,edgepos,progress);

pageEntities[cacheindex].edgePos = edgepos;

}

可以看到程序马上进入drawOnBitmap函数，清单如下：

/\*\*

\* progress:read progress page index

\*/

**void** drawOnBitmap(**int** w, **int** h, Bitmap bitmap, ZLView.PageIndex index, EdgePosition edgepos, ZLView.PageIndex progress) {

**final** ZLView view = ZLApplication.*Instance*().getCurrentView();

**if** (view == **null**) {

**return**;

}

**if**(**null**==mPaintContext){

mPaintContext = **new** ZLAndroidPaintContext();

}

mPaintContext.setCanvas(**new** Canvas(bitmap));

mPaintContext.setSize(w,h,view.isScrollbarShown() ? getVerticalScrollbarWidth() : 0);

view.paint(mPaintContext, index,edgepos,progress);

}

获得一些环境参数后进入paint,实现绘制。清单如下：

@Override

**public** **synchronized** **void** paint(ZLPaintContext context, PageIndex pageIndex, EdgePosition edgepos,ZLView.PageIndex progress) {

setContext(context);

// final ZLFile wallpaper = getWallpaperFile();

// if (wallpaper != null) {

// context.clear(wallpaper, getWallpaperMode());

// } else {

// context.clear(getBackgroundColor());

// }

//画图片

drawBookBackground(context,edgepos);

**if** (myModel == **null** || myModel.getParagraphsNumber() == 0) {

**return**;

}

ZLTextPage page;

**final** Animation type = ZLApplication.*Instance*().getViewWidget().getCurAnimationType();

**if**(type==Animation.***realdouble***){

**final** ZLAndroidWidget widget = ((ZLAndroidLibrary)ZLAndroidLibrary.*Instance*()).getWidget();

**final** AnimationProvider animator = widget.getAnimationProvider();

**switch** (pageIndex) {

**default**:

**case** ***current***:

page = myCurrentPage;

Utils.*printLogError*(TAG, "paint called get myCurrentPage :"+myCurrentPage);

**if** (myCurrentPage.PaintState == PaintStateEnum.***NOTHING\_TO\_PAINT***&&

myCurrentRightPage.PaintState!=PaintStateEnum.***NOTHING\_TO\_PAINT***) {

Utils.*printLogInfo*(TAG, "myCurrentPage no content, myCurrentRightPage :"+myCurrentRightPage);

preparePaintInfo(myCurrentRightPage);

**if**(myCurrentRightPage.PaintState!=PaintStateEnum.***NOTHING\_TO\_PAINT***){

myCurrentPage.EndCursor.setCursor(myCurrentRightPage.StartCursor);

myCurrentPage.PaintState = PaintStateEnum.***END\_IS\_KNOWN***;

Utils.*printLogInfo*(TAG, "set myCurrentPage.EndCursor :"+myCurrentPage.EndCursor);

}

}

**break**;

**case** ***curright***:

page = myCurrentRightPage;

Utils.*printLogError*(TAG, "paint called get myCurrentRightPage :"+myCurrentRightPage);

**if** (myCurrentRightPage.PaintState == PaintStateEnum.***NOTHING\_TO\_PAINT***&&

myCurrentPage.PaintState!=PaintStateEnum.***NOTHING\_TO\_PAINT***) {

Utils.*printLogInfo*(TAG, "myCurrentRightPage no content, myCurrentPage :"+myCurrentPage);

preparePaintInfo(myCurrentPage);

**if**(myCurrentPage.PaintState!=PaintStateEnum.***NOTHING\_TO\_PAINT***){

myCurrentRightPage.StartCursor.setCursor(myCurrentPage.EndCursor);

myCurrentRightPage.PaintState = PaintStateEnum.***START\_IS\_KNOWN***;

Utils.*printLogInfo*(TAG, "set myCurrentRightPage.StartCursor :"+myCurrentRightPage.StartCursor);

}

}

**break**;

**case** ***previous***:

page = myPreviousPage;

Utils.*printLogError*(TAG, "paint called get myPreviousPage :"+myPreviousPage);

**if** (myPreviousPage.PaintState == PaintStateEnum.***NOTHING\_TO\_PAINT***) {

**if**(animator.getPageToScrollTo()==PageIndex.***next***){//后翻页

Utils.*printLogInfo*(TAG, "no content,to next, myCurrentRightPage :"+myCurrentRightPage);

preparePaintInfo(myCurrentRightPage);

myPreviousPage.StartCursor.setCursor(myCurrentRightPage.EndCursor);

myPreviousPage.PaintState = PaintStateEnum.***START\_IS\_KNOWN***;

Utils.*printLogInfo*(TAG, "set myPreviousPage.StartCursor :"+myPreviousPage.StartCursor);

}**else** **if**(animator.getPageToScrollTo()==PageIndex.***previous***){

Utils.*printLogInfo*(TAG, "no content,to previous myCurrentPage :"+myCurrentPage);

preparePaintInfo(myCurrentPage);

myPreviousPage.EndCursor.setCursor(myCurrentPage.StartCursor);

myPreviousPage.PaintState = PaintStateEnum.***END\_IS\_KNOWN***;

Utils.*printLogInfo*(TAG, "set myPreviousPage.EndCursor :"+myPreviousPage.EndCursor);

}

}

**break**;

**case** ***next***:

page = myNextPage;

Utils.*printLogError*(TAG, "paint called get myNextPage :"+myNextPage);

**if** (myNextPage.PaintState == PaintStateEnum.***NOTHING\_TO\_PAINT***) {

**if**(animator.getPageToScrollTo()==PageIndex.***next***){//后翻页

Utils.*printLogInfo*(TAG, "no content,to next, myPreviousPage :"+myPreviousPage+", PaintStateEnum:"+myPreviousPage.PaintState);

**if**(myPreviousPage.PaintState==PaintStateEnum.***NOTHING\_TO\_PAINT***&&

myCurrentRightPage.PaintState!=PaintStateEnum.***NOTHING\_TO\_PAINT***){

Utils.*printLogError*(TAG, "no content,to next, myPreviousPage has no content too, find it from myCurrentRightPage:"+myCurrentRightPage);

preparePaintInfo(myCurrentRightPage);

myPreviousPage.StartCursor.setCursor(myCurrentRightPage.EndCursor);

myPreviousPage.PaintState=PaintStateEnum.***START\_IS\_KNOWN***;

}

preparePaintInfo(myPreviousPage);

myNextPage.StartCursor.setCursor(myPreviousPage.EndCursor);

myNextPage.PaintState = PaintStateEnum.***START\_IS\_KNOWN***;

Utils.*printLogInfo*(TAG, "set myNextPage.StartCursor :"+myNextPage.StartCursor);

}**else** **if**(animator.getPageToScrollTo()==PageIndex.***previous***){//回翻

Utils.*printLogInfo*(TAG, "no content,to previous, myPreviousPage :"+myPreviousPage+", PaintStateEnum:"+myPreviousPage.PaintState);

**if**(myPreviousPage.PaintState==PaintStateEnum.***NOTHING\_TO\_PAINT***&&

myCurrentPage.PaintState!=PaintStateEnum.***NOTHING\_TO\_PAINT***){

Utils.*printLogError*(TAG, "no content,to previous, myPreviousPage has no content too, find it from myCurrentPage:"+myCurrentPage);

preparePaintInfo(myCurrentPage);

myPreviousPage.EndCursor.setCursor(myCurrentPage.StartCursor);

myPreviousPage.PaintState=PaintStateEnum.***END\_IS\_KNOWN***;

}

preparePaintInfo(myPreviousPage);

myNextPage.EndCursor.setCursor(myPreviousPage.StartCursor);

myNextPage.PaintState = PaintStateEnum.***END\_IS\_KNOWN***;

Utils.*printLogInfo*(TAG, "set myNextPage.EndCursor :"+myNextPage.EndCursor);

}

}

**break**;

}

}**else**{

**switch** (pageIndex) {

**default**:

**case** ***current***:

page = myCurrentPage;

**break**;

**case** ***previous***:

page = myPreviousPage;

**if** (myPreviousPage.PaintState == PaintStateEnum.***NOTHING\_TO\_PAINT***) {

preparePaintInfo(myCurrentPage);

myPreviousPage.EndCursor.setCursor(myCurrentPage.StartCursor);

myPreviousPage.PaintState = PaintStateEnum.***END\_IS\_KNOWN***;

}

**break**;

**case** ***next***:

page = myNextPage;

**if** (myNextPage.PaintState == PaintStateEnum.***NOTHING\_TO\_PAINT***) {

preparePaintInfo(myCurrentPage);

myNextPage.StartCursor.setCursor(myCurrentPage.EndCursor);

myNextPage.PaintState = PaintStateEnum.***START\_IS\_KNOWN***;

}

}

}

page.TextElementMap.clear();

preparePaintInfo(page);

**if** (page.StartCursor.isNull() || page.EndCursor.isNull()) {

**return**;

}

**final** ArrayList<ZLTextLineInfo> lineInfos = page.LineInfos;

**final** **int**[] labels = **new** **int**[lineInfos.size() + 1];

**int** x = getLeftMargin();

**int** y = getTopMargin();

**int** index = 0;

**for**(ImageEntity entity:page.ImageLists){

entity.setOverHeight(**true**);

}

**for** (ZLTextLineInfo info : lineInfos) {

prepareTextLine(page, info, x, y);

y += info.Height + info.Descent + info.VSpaceAfter;

**final** ImageEntity preImage = page.getPreviousImage();

**final** ImageEntity curImage = page.getCurrentImage();

**if**(curImage!=**null**&&curImage.lineInfo!=**null**&&curImage.lineInfo.ParagraphCursor!=**null**){

**if**(curImage.lineInfo.RealStartElementIndex==info.RealStartElementIndex&&

curImage.lineInfo.ParagraphCursor.Index==info.ParagraphCursor.Index){

**if**(preImage!=**null**&&preImage.isReduceTextLineWidth()){

preImage.setOverHeight(**true**);

y+=(preImage.getLineHeight()+preImage.lineInfo.Descent-preImage.getCurTextHeight());

}

}**else**{

**if**(curImage.isReduceTextLineWidth()&&info.isImageLine){

curImage.setOverHeight(**true**);

y+=(curImage.getLineHeight()+curImage.lineInfo.Descent-curImage.getCurTextHeight());

}

}

}

labels[++index] = page.TextElementMap.size();

}

x = getLeftMargin();

y = getTopMargin();

index = 0;

**final** Bookmark bookmark = page.getBookMark();

ZLTextFixedPosition pos = **null**;

**if**(bookmark!=**null**){

pos = **new** ZLTextFixedPosition(bookmark.ParagraphIndex, bookmark.ElementIndex, bookmark.CharIndex);

**if**(bookmark.myEnd.compareTo(page.StartCursor)<0||pos.compareTo(page.EndCursor)>0){

page.setBookMark(**null**);

}

}

**for** (ZLTextLineInfo info : lineInfos) {

drawHighlightings(page, edgepos, info, labels[index], labels[index + 1], x, y);

y += info.Height + info.Descent + info.VSpaceAfter;

++index;

}

**if**(edgepos==EdgePosition.***EDGE\_RIGHT***){

**if**(myRealBookmarkPage!=**null**&&myRealBookmarkPage.EndCursor.getParagraphIndex()==page.StartCursor.getParagraphIndex()&&

myRealBookmarkPage.EndCursor.getElementIndex()==page.StartCursor.getElementIndex()){

**if**(myRealBookmarkPage.getBookMark()!=**null**){

// if(!myRealBookmarkPage.isReleaseBookMark()){

drawBookMark(context,myRealBookmarkPage, **true**);

// }

}

// else if(myRealBookmarkPage.isReleaseBookMark()){

// drawBookMark(context, myRealBookmarkPage,false);

// }

}**else**{

**if**(page.getBookMark()!=**null**){

// if(!page.isReleaseBookMark()){

drawBookMark(context,page, **true**);

// }

}

// else if(page.isReleaseBookMark()){

// drawBookMark(context, page,false);

// }

}

}

**if**(isShowVolumeStatus){

**if**(type==Animation.***realdouble***){

**if**(pageIndex==PageIndex.***curright***){

drawVolume(context);

}

}**else** **if**(pageIndex==PageIndex.***current***){

drawVolume(context);

}

}

x = getLeftMargin();

y = getTopMargin();

index = 0;

**for**(ImageEntity entity:page.ImageLists){

entity.setOverHeight(**true**);

}

**for** (ZLTextLineInfo info : lineInfos) {

drawTextLine(page, info, labels[index], labels[index + 1]);

// y += info.Height + info.Descent + info.VSpaceAfter;

++index;

}

**final** ZLTextRegion selectedElementRegion = getSelectedRegion(page);

**if** (selectedElementRegion != **null** && myHighlightSelectedRegion) {

selectedElementRegion.draw(context);

}

drawSelectionCursor(context, getSelectionCursorPoint(page, ZLTextSelectionCursor.***Left***));

drawSelectionCursor(context, getSelectionCursorPoint(page, ZLTextSelectionCursor.***Right***));

**final** **int** curPage = computeTextPageNumber(getCurrentCharNumber(progress, **false**));

**final** **int** totalPage = computeTextPageNumber(sizeOfFullText());

**final** **int** curProgress = curPage\*100/totalPage;

drawBookFooter(context,curPage,totalPage,edgepos);

// if(mReadProgressChangeListener!=null){

// mReadProgressChangeListener.onReadProgressChange(curProgress);

// }

Message msg = mHandler.obtainMessage(MSG\_READ\_PROGRESS\_CHANGED);

msg.arg1 = curProgress;

mHandler.sendMessage(msg);

}

可以看到首先通过drawbackground绘制背景，然后主要是根据pageIndex参数，来加载上一页或者下一页，并设置移动当前页的段落的游标。绘制通过如下函数实现

**private** **synchronized** **void** preparePaintInfo(ZLTextPage page) {

page.setSize(getTextAreaWidth(), getTextAreaHeight(), page == myPreviousPage);

**if** (page.PaintState == PaintStateEnum.***NOTHING\_TO\_PAINT*** || page.PaintState == PaintStateEnum.***READY***) {

**return**;

}

**final** **int** oldState = page.PaintState;

**final** HashMap<ZLTextLineInfo,ZLTextLineInfo> cache = myLineInfoCache;

**for** (ZLTextLineInfo info : page.LineInfos) {

cache.put(info, info);

}

**switch** (page.PaintState) {

**default**:

**break**;

**case** PaintStateEnum.***TO\_SCROLL\_FORWARD***:

**if** (!page.EndCursor.isEndOfText()) {

**final** ZLTextWordCursor startCursor = **new** ZLTextWordCursor();

**switch** (myScrollingMode) {

**case** ScrollingMode.***NO\_OVERLAPPING***:

**break**;

**case** ScrollingMode.***KEEP\_LINES***:

page.findLineFromEnd(startCursor, myOverlappingValue);

**break**;

**case** ScrollingMode.***SCROLL\_LINES***:

page.findLineFromStart(startCursor, myOverlappingValue);

**if** (startCursor.isEndOfParagraph()) {

startCursor.nextParagraph();

}

**break**;

**case** ScrollingMode.***SCROLL\_PERCENTAGE***:

page.findPercentFromStart(startCursor, myOverlappingValue);

**break**;

}

**if** (!startCursor.isNull() && startCursor.samePositionAs(page.StartCursor)) {

page.findLineFromStart(startCursor, 1);

}

**if** (!startCursor.isNull()) {

**final** ZLTextWordCursor endCursor = **new** ZLTextWordCursor();

buildInfos(page, startCursor, endCursor,**true**);

**if** (!page.isEmptyPage() && (myScrollingMode != ScrollingMode.***KEEP\_LINES*** || !endCursor.samePositionAs(page.EndCursor))) {

page.StartCursor.setCursor(startCursor);

page.EndCursor.setCursor(endCursor);

**break**;

}

}

page.StartCursor.setCursor(page.EndCursor);

buildInfos(page, page.StartCursor, page.EndCursor,**true**);

}

**break**;

**case** PaintStateEnum.***TO\_SCROLL\_BACKWARD***:

**if** (!page.StartCursor.isStartOfText()) {

**switch** (myScrollingMode) {

**case** ScrollingMode.***NO\_OVERLAPPING***:

page.StartCursor.setCursor(findStartOfPrevousPage(page, page.StartCursor));

**break**;

**case** ScrollingMode.***KEEP\_LINES***:

{

ZLTextWordCursor endCursor = **new** ZLTextWordCursor();

page.findLineFromStart(endCursor, myOverlappingValue);

**if** (!endCursor.isNull() && endCursor.samePositionAs(page.EndCursor)) {

page.findLineFromEnd(endCursor, 1);

}

**if** (!endCursor.isNull()) {

ZLTextWordCursor startCursor = findStartOfPrevousPage(page, endCursor);

**if** (startCursor.samePositionAs(page.StartCursor)) {

page.StartCursor.setCursor(findStartOfPrevousPage(page, page.StartCursor));

} **else** {

page.StartCursor.setCursor(startCursor);

}

} **else** {

page.StartCursor.setCursor(findStartOfPrevousPage(page, page.StartCursor));

}

**break**;

}

**case** ScrollingMode.***SCROLL\_LINES***:

page.StartCursor.setCursor(findStart(page, page.StartCursor, SizeUnit.***LINE\_UNIT***, myOverlappingValue));

**break**;

**case** ScrollingMode.***SCROLL\_PERCENTAGE***:

page.StartCursor.setCursor(findStart(page, page.StartCursor, SizeUnit.***PIXEL\_UNIT***, page.getTextHeight() \* myOverlappingValue / 100));

**break**;

}

buildInfos(page, page.StartCursor, page.EndCursor,**false**);

**if** (page.isEmptyPage()) {

page.StartCursor.setCursor(findStart(page, page.StartCursor, SizeUnit.***LINE\_UNIT***, 1));

buildInfos(page, page.StartCursor, page.EndCursor,**false**);

}

}

**break**;

**case** PaintStateEnum.***START\_IS\_KNOWN***:

buildInfos(page, page.StartCursor, page.EndCursor,**true**);

**break**;

**case** PaintStateEnum.***END\_IS\_KNOWN***:

page.StartCursor.setCursor(findStartOfPrevousPage(page, page.EndCursor));

buildInfos(page, page.StartCursor, page.EndCursor,**false**);

**break**;

}

page.PaintState = PaintStateEnum.***READY***;

// **TODO**: cache?

myLineInfoCache.clear();

**if** (page == myCurrentPage) {

**final** Animation type = ZLApplication.*Instance*().getViewWidget().getCurAnimationType();

**if**(type==Animation.***realdouble***){

**if** (oldState != PaintStateEnum.***START\_IS\_KNOWN***) {

myPreviousPage.reset();

}

**if** (oldState != PaintStateEnum.***END\_IS\_KNOWN***) {

myCurrentRightPage.reset();

}

}**else**{

**if** (oldState != PaintStateEnum.***START\_IS\_KNOWN***) {

myPreviousPage.reset();

}

**if** (oldState != PaintStateEnum.***END\_IS\_KNOWN***) {

myNextPage.reset();

}

}

}

}

从代码中可以看到经过设置检测bitmap的宽高信息后，最后调用如下函数实现页面内容的排版和构建，清单如下

**private** **void** buildInfos(ZLTextPage page, ZLTextWordCursor start, ZLTextWordCursor result, **boolean** useCache) {

ZLTextWordCursor resultCopy= **new** ZLTextWordCursor(result);

result.setCursor(start);

**int** textAreaHeight = page.getTextHeight();

**if**(textAreaHeight<=0){

**return**;

}

page.LineInfos.clear();

page.ImageLists.clear();

**int** counter = 0;

/\*\*

\* is the word element occured before

\* \*/

**boolean** isUseCache = useCache;

**boolean** isReversed = **false**;

ZLTextLineInfo myReversedImageLineInfo = **null**;

// ImageEntity preImage = null;

ImageEntity curImage = **null**;

**do** {

resetTextStyle();

ZLTextParagraphCursor paragraphCursor = result.getParagraphCursor();

**if**(paragraphCursor==**null**){

Utils.*printLogError*(TAG, "Error, buildInfos paragraphCursor is null");

**break**;

}**else**{

**if**(myReversedImageLineInfo!=**null**&&myReversedImageLineInfo.ParagraphCursor!=**null**&&

myReversedImageLineInfo.ParagraphCursor.Index==paragraphCursor.Index){

result.moveTo(myReversedImageLineInfo.EndElementIndex,myReversedImageLineInfo.EndCharIndex);

isReversed = **false**;

}

}

**int** wordIndex = result.getElementIndex();

applyStyleChanges(paragraphCursor, 0, wordIndex);

ZLTextLineInfo info = **new** ZLTextLineInfo(paragraphCursor, wordIndex, result.getCharIndex(), getTextStyle());

**int** endIndex = info.ParagraphCursorLength;

**while** (info.EndElementIndex != endIndex) {

info = processTextLine(page, paragraphCursor, info.EndElementIndex, info.EndCharIndex, endIndex,isUseCache);

textAreaHeight -= info.Height + info.Descent;

curImage = page.getCurrentImage();

// preImage = page.getPreviousImage();

**if**(curImage!=**null**&&curImage.lineInfo!=**null**&&curImage.lineInfo.ParagraphCursor!=**null**){

**if**(curImage.lineInfo.RealStartElementIndex==info.RealStartElementIndex&&

curImage.lineInfo.ParagraphCursor.Index==info.ParagraphCursor.Index){

// if(preImage!=null&&preImage.isReduceTextLineWidth()){

// preImage.setOverHeight(true);

// textAreaHeight-=(preImage.getLineHeight()+preImage.lineInfo.Descent-preImage.getCurTextHeight());

// if(textAreaHeight<=0){

// /\*\*

// \* can't show another image,remove it

// \* \*/

// if(page.ImageLists.contains(curImage)){

// page.ImageLists.remove(curImage);

// page.setCurImage(page.ImageLists.size()-1);

// curImage = page.getCurrentImage();

// }

// }

// }

**if**(curImage.isReduceTextLineWidth()){

**if**(textAreaHeight>=0){

textAreaHeight+=info.Height + info.Descent;

info.Height = 0;

}**else**{

/\*\*

\* doesn't got enough room for the image, we should go back to get some room

\* \*/

**boolean** isContinue = **false**;

// if(preImage!=null){

// if(page.ImageLists.contains(curImage)){

// page.ImageLists.remove(curImage);

// page.setCurImage(page.ImageLists.size()-1);

// curImage = page.getCurrentImage();

// }

// }else{

**if**(!isReversed){

isReversed = !isReversed;

}**else**{

/\*\*

\* already reversed, break now

\* \*/

Utils.*printLogError*(getClass().getSimpleName(), "It had been Reversed, break right now!");

// result.setCursor(curImage.lineInfo.ParagraphCursor);

// result.moveTo(curImage.lineInfo.EndElementIndex, curImage.lineInfo.EndCharIndex);

**break**;

}

**int** removeCount = 0;

ZLTextLineInfo inf=**null**;

**int** dummyHeight = textAreaHeight;

**if**(dummyHeight\*curImage.lineInfo.Width+

(curImage.getLineHeight()+textAreaHeight)\*

(getTextAreaWidth()-curImage.lineInfo.Width)<0){

/\*\*

\* can't reverse situation

\* \*/

// if(page.LineInfos.size()<2&&!isUseCache){

// //only one line text, so show this image here

// page.StartCursor.setCursor(curImage.lineInfo.ParagraphCursor);

// page.StartCursor.moveToParagraphStart();

// page.LineInfos.clear();

// page.LineInfos.add(curImage.lineInfo);

// result.setCursor(page.StartCursor);

// result.moveTo(curImage.lineInfo.EndElementIndex, curImage.lineInfo.EndCharIndex);

// }else{

/\*\*

\* encounter another image,or previous image side text line

\*/

**if**(page.PaintState==PaintStateEnum.***END\_IS\_KNOWN***){

/\*\*

\* avoid lost contents

\* \*/

Utils.*printLogInfo*(getClass().getSimpleName(), "call checkLostContent 0");

textAreaHeight = checkLostContent(page, info, start, resultCopy, result,textAreaHeight);

**if**(textAreaHeight>=0){

textAreaHeight+=info.Height + info.Descent;

info.Height = 0;

result.moveTo(info.EndElementIndex, info.EndCharIndex);

**if**(!page.LineInfos.contains(info)){

page.LineInfos.add(info);

}

removeCount=0;

isContinue = **true**;

**break**;

}

}**else** **if**(page.ImageLists.contains(curImage)){

page.ImageLists.remove(curImage);

page.setCurImage(page.ImageLists.size()-1);

curImage = page.getCurrentImage();

}

// }

**break**;

}

**for**(**int** i=page.LineInfos.size()-1;i>=0;i--){

inf=page.LineInfos.get(i);

**if**(dummyHeight<0&&(inf.isImageLine||inf.isSideTextLine)){

**if**(curImage.lineInfo.ParagraphCursor.Index==inf.ParagraphCursor.Index&&

curImage.lineInfo.RealStartElementIndex==inf.RealStartElementIndex){

/\*\*

\* encounter line of current image , just remove it, but it impossible occur here

\* \*/

page.LineInfos.remove(inf);

**continue**;

}**else**{

// if(findImageFromLine(inf)>=0||inf.isSideTextLine){

/\*\*

\* encounter another image,

\*/

**if**(page.PaintState==PaintStateEnum.***END\_IS\_KNOWN***){

/\*\*

\* avoid lost contents

\* \*/

Utils.*printLogInfo*(getClass().getSimpleName(), "call checkLostContent 1");

textAreaHeight = checkLostContent(page, info, start, resultCopy, result,textAreaHeight);

**if**(textAreaHeight>=0){

textAreaHeight+=info.Height + info.Descent;

info.Height = 0;

result.moveTo(info.EndElementIndex, info.EndCharIndex);

**if**(!page.LineInfos.contains(info)){

page.LineInfos.add(info);

}

removeCount=0;

isContinue = **true**;

**break**;

}

}**else** **if**(page.ImageLists.contains(curImage)){

page.ImageLists.remove(curImage);

page.setCurImage(page.ImageLists.size()-1);

curImage = page.getCurrentImage();

}

removeCount=0;

**break**;

}

// }

}

dummyHeight+=inf.Height+inf.Descent;

removeCount++;

**if**(dummyHeight>=0){

// i--;

// if(i>=0){

// inf=page.LineInfos.get(i);

// removeCount++;

// textAreaHeight+=inf.Height+inf.Descent;

// }

**break**;

}

}

**if**(removeCount>0){

**if**(page.LineInfos.size()>=removeCount&&dummyHeight>=0){

**for**(**int** j=0;j<removeCount;j++){

/\*\*

\* if can successfully reposition this image, we can remove the line infos

\* \*/

page.LineInfos.remove(page.LineInfos.size()-1);

}

myReversedImageLineInfo = curImage.lineInfo;

textAreaHeight=dummyHeight;

textAreaHeight+=info.Height+info.Descent;

info.Height = 0;

page.LineInfos.add(info);

resetTextStyle();

result.setCursor(inf.ParagraphCursor);

result.moveTo(inf.StartElementIndex, inf.StartCharIndex);

paragraphCursor = inf.ParagraphCursor;

wordIndex = inf.StartElementIndex;

applyStyleChanges(paragraphCursor, 0, wordIndex);

info = **new** ZLTextLineInfo(paragraphCursor, wordIndex, inf.StartCharIndex, getTextStyle());

endIndex = info.ParagraphCursorLength;

isUseCache = **false**;

isContinue = **true**;

}**else**{

/\*\*

\* can't show this image at this page, show it to next page

\* \*/

**if**(page.ImageLists.contains(curImage)){

page.ImageLists.remove(curImage);

page.setCurImage(page.ImageLists.size()-1);

curImage = page.getCurrentImage();

}

inf = page.LineInfos.get(page.LineInfos.size()-1);

**if**(inf!=**null**){

result.setCursor(inf.ParagraphCursor);

result.moveTo(inf.EndElementIndex,inf.EndCharIndex);

}

removeCount=0;

**break**;

}

}

// }

**if**(isContinue)

**continue**;

}

}**else**{

**break**;

}

}**else**{

**if**(curImage.isReduceTextLineWidth()&&info.isImageLine){

curImage.setOverHeight(**true**);

textAreaHeight-=(curImage.getLineHeight()+curImage.lineInfo.Descent-curImage.getCurTextHeight());

**if**(textAreaHeight<0&&isReversed){

ZLTextLineInfo inf=**null**;

/\*\*

\* find last visible paragraph

\* \*/

**for**(**int** i=page.LineInfos.size()-1;i>=0;i--){

inf = page.LineInfos.get(i);

**if**(inf!=**null**&&inf.Height>0){

**break**;

}

}

**if**(inf!=**null**&&inf.ParagraphCursor.Index!=curImage.lineInfo.ParagraphCursor.Index){

/\*\*

\* current image not at the last line, should be reversed situation

\* \*/

**if**(curImage.lineInfo.ParagraphCursor.Index>=inf.ParagraphCursor.Index){

result.setCursor(curImage.lineInfo.ParagraphCursor);

result.moveTo(curImage.lineInfo.EndElementIndex, curImage.lineInfo.EndCharIndex);

**break**;

}

}

}

}

}

}

**if** (textAreaHeight < 0 && counter > 0) {

**if**(page.PaintState==PaintStateEnum.***END\_IS\_KNOWN***){

Utils.*printLogInfo*(getClass().getSimpleName(), "call checkLostContent 3");

textAreaHeight = checkLostContent(page, info, start, resultCopy, result, textAreaHeight);

**if**(textAreaHeight>=0){

textAreaHeight-=info.Height + info.Descent;

// info.Height = 0;

result.moveTo(info.EndElementIndex, info.EndCharIndex);

**if**(!page.LineInfos.contains(info)){

page.LineInfos.add(info);

}

**continue**;

}

}

**break**;

}

textAreaHeight -= info.VSpaceAfter;

result.moveTo(info.EndElementIndex, info.EndCharIndex);

**if**(!page.LineInfos.contains(info)){

page.LineInfos.add(info);

}

**if**(page.PaintState==PaintStateEnum.***END\_IS\_KNOWN***&&

resultCopy.compareTo(result)<=0){

// Utils.printLogError(TAG, "May get same content, break!!!++++++++++++++++++++++++++++++");

// Utils.printLogInfo(TAG, "result:"+result);

// Utils.printLogError(TAG, "resultCopy:"+resultCopy);

textAreaHeight=0;

}

**if** (textAreaHeight < 0) {

**break**;

}

counter++;

}

} **while** (result.isEndOfParagraph() && result.nextParagraph() && !result.getParagraphCursor().isEndOfSection() && (textAreaHeight >= 0));

resetTextStyle();

}

从代码中可以看到首先通过页面的游标取得页面的开始或者结束信息后，对页面内容进行正向或逆向构建，通过

**private** ZLTextLineInfo processTextLine(

ZLTextPage page,

ZLTextParagraphCursor paragraphCursor,

**final** **int** startIndex,

**final** **int** startCharIndex,

**final** **int** endIndex,

**boolean** useCache

) {

**boolean** isCover=**false**;

**boolean** isImageLine = **false**;

**boolean** isAddToCache = **true**;

**final** ZLPaintContext context = getContext();

**final** ZLTextLineInfo info = **new** ZLTextLineInfo(paragraphCursor, startIndex, startCharIndex, getTextStyle());

/\*\*

\* abandon cache

\* \*/

**final** ZLTextLineInfo cachedInfo = myLineInfoCache.get(info);

**if** (cachedInfo != **null**&&useCache) {

applyStyleChanges(paragraphCursor, startIndex, cachedInfo.EndElementIndex);

**return** cachedInfo;

}

**int** currentElementIndex = startIndex;

**int** currentCharIndex = startCharIndex;

**final** **boolean** isFirstLine = startIndex == 0 && startCharIndex == 0;

**boolean** isTitle=**false**;

**boolean** isStart=**false**;

**boolean** isEnd=**false**;

**boolean** isH1=**false**;

**if** (isFirstLine) {

ZLTextElement element = paragraphCursor.getElement(currentElementIndex);

**while** (isStyleChangeElement(element)) {

**if** (element **instanceof** ZLTextControlElement

&& (((ZLTextControlElement) element).Kind == FBTextKind.***TITLE***

|| ((ZLTextControlElement) element).Kind == FBTextKind.***H1*** || ((ZLTextControlElement) element).Kind == FBTextKind.***H2***)) {

isTitle = **true**;

**if**(((ZLTextControlElement) element).IsStart){

isStart=**true**;

}**else**{

isEnd=**true**;

}

//------------

**if**( ((ZLTextControlElement) element).Kind == FBTextKind.***H1*** ){

isH1=**true**;

}

}

applyStyleChangeElement(element);

++currentElementIndex;

currentCharIndex = 0;

**if** (currentElementIndex == endIndex) {

**break**;

}

element = paragraphCursor.getElement(currentElementIndex);

}

info.StartStyle = getTextStyle();

info.RealStartElementIndex = currentElementIndex;

info.RealStartCharIndex = currentCharIndex;

}

ZLTextStyle storedStyle = getTextStyle();

info.LeftIndent = getTextStyle().getLeftIndent();

**if** (isFirstLine && !isTitle) {

info.LeftIndent += getTextStyle().getFirstLineIndentDelta();

}

info.Width = info.LeftIndent;

**if** (info.RealStartElementIndex == endIndex) {

info.EndElementIndex = info.RealStartElementIndex;

info.EndCharIndex = info.RealStartCharIndex;

**return** info;

}

**int** newWidth = info.Width;

**int** newHeight = info.Height;

**int** newDescent = info.Descent;

**int** rightIndent = getTextStyle().getRightIndent();

**int** maxWidth = getTextAreaWidth() - rightIndent;

**int** maxHeight = getTextAreaHeight();

**final** ImageEntity curImage= page.getCurrentImage();

**if**(curImage!=**null**&&curImage.isReduceTextLineWidth()){

maxWidth-=curImage.width;

**if**(maxWidth<getSingleCharacterWidth()){//no enough space to show a single character

info.EndElementIndex = currentElementIndex;

info.EndCharIndex = currentCharIndex;

info.Height = curImage.getLineHeight()+curImage.lineInfo.Descent-curImage.getCurTextHeight();

storedStyle = getTextStyle();

curImage.setOverHeight(**true**);

**return** info;

}

}

**boolean** wordOccurred = **false**;

**boolean** isVisible = **false**;

**int** lastSpaceWidth = 0;

**int** internalSpaceCounter = 0;

**boolean** removeLastSpace = **false**;

ImageEntity image=**null**;

**int** elementWidth = 0;

**int** elementHeight = 0;

**int** offsetHeight=0;

**do** {

ZLTextElement element = paragraphCursor.getElement(currentElementIndex);

elementWidth = getElementWidth(element, currentCharIndex);

newWidth +=elementWidth;

elementHeight = getElementHeight(element);

**if**(newHeight>0){

offsetHeight = elementHeight-newHeight;

}

newHeight = Math.*max*(newHeight, elementHeight);

newDescent = Math.*max*(newDescent, getElementDescent(element));

**if** (element == ZLTextElement.***HSpace***) {

**if** (wordOccurred) {

wordOccurred = **false**;

internalSpaceCounter++;

lastSpaceWidth = context.getSpaceWidth();

newWidth += lastSpaceWidth;

}

} **else** **if** (element **instanceof** ZLTextWord) {

wordOccurred = **true**;

isVisible = **true**;

}**else** **if** (isStyleChangeElement(element)) {

applyStyleChangeElement(element);

**if** (element **instanceof** ZLTextControlElement

&& (((ZLTextControlElement) element).Kind == FBTextKind.***TITLE***

|| ((ZLTextControlElement) element).Kind == FBTextKind.***H1*** || ((ZLTextControlElement) element).Kind == FBTextKind.***H2***)) {

isTitle = **true**;

**if**(((ZLTextControlElement) element).IsStart){

isStart=**true**;

}**else**{

isEnd=**true**;

}

}

} **else** **if** (element **instanceof** ZLTextImageElement) {

wordOccurred = **true**;

isVisible = **true**;

**if**(((ZLTextImageElement) element).IsCover){

isCover=**true**;

}**else** {

**final** **int** imageWidth = elementWidth;

**final** **int** imageHeight = elementHeight;

**boolean** isBreak = **false**;

**boolean** isEmbededImage= **false**;

**if**(newWidth<=maxWidth&&imageWidth>0&&imageHeight>0){

**if**(((ZLTextImageElement) element).align==ImageAlign.***IMAGE\_ALIGN\_LEFT***||

((ZLTextImageElement) element).align==ImageAlign.***IMAGE\_ALIGN\_RIGHT***){

**if**(imageWidth<getTextAreaWidth()){

**if**(currentElementIndex>startIndex&&info.Width>0&&

info.Width>info.LeftIndent&&info.Height>0){

//got something ahead, should turn to next line

isBreak = **true**;

}**else**{

isEmbededImage = **true**;

}

}**else** **if**(imageWidth>=getTextAreaWidth()){

//only equal may occur

((ZLTextImageElement) element).align = ImageAlign.***IMAGE\_ALIGN\_CENTER***;

}

}

}**else**{

**if**(newWidth>maxWidth){

**if**(info.LeftIndent>0&&info.Width==info.LeftIndent&&

info.Height==0&&info.Width+elementWidth>maxWidth&&

(curImage==**null**||!curImage.isReduceTextLineWidth())){

/\*\*

\* picture too big to shows at the first line, just show it whatever

\* \*/

isBreak = **false**;

**if**(((ZLTextImageElement) element).align==ImageAlign.***IMAGE\_ALIGN\_LEFT***||

((ZLTextImageElement) element).align==ImageAlign.***IMAGE\_ALIGN\_RIGHT***){

isEmbededImage = **true**;

}

newWidth -=info.LeftIndent;

info.LeftIndent = 0;

}**else**{

isBreak = **true**;

info.isImageLine = **true**;

}

}

}

**if**(isEmbededImage){

**if**(curImage!=**null**&&curImage.isReduceTextLineWidth()){

**if**(!TextUtils.*isEmpty*(((ZLTextImageElement) element).Id)&&

((ZLTextImageElement) element).Id.equals(curImage.getPath())){

isBreak = **false**;

}**else**{

isBreak = **true**;

**if**(info.EndElementIndex==info.StartElementIndex){

info.isImageLine = **true**;

isAddToCache = **false**;

}

}

// info.Height = newHeight-elementHeight+

// curImage.getLineHeight()+curImage.lineInfo.VSpaceAfter-curImage.getCurTextHeight();

// curImage.setOverHeight(true);

}**else**{

image= **new** ImageEntity(info,

((ZLTextImageElement) element).Id,

imageWidth,

imageHeight,

((ZLTextImageElement) element).align);

**if**(!page.hasImage(image)){

page.ImageLists.add(image);

page.setCurImage(page.ImageLists.size()-1);

}**else**{

page.resetImageTextHeight(image);

}

}

}

**if**(((ZLTextImageElement) element).align!=ImageAlign.***IMAGE\_ALIGN\_TEXT***&&!isBreak){

isImageLine = **true**;

info.isImageLine = **true**;

}

**if**(image!=**null**||

((ZLTextImageElement) element).align==ImageAlign.***IMAGE\_ALIGN\_CENTER***||

newWidth>maxWidth||isBreak){

info.IsVisible = isVisible;

**if**(!isBreak){

info.Width = newWidth;

**if** (info.Height < newHeight) {

info.Height = newHeight;

}

**if**(offsetHeight>0){

info.Height=newHeight-offsetHeight;

}

**if** (info.Descent < newDescent) {

info.Descent = newDescent;

}

info.EndElementIndex = currentElementIndex+1;

}**else**{

info.EndElementIndex = currentElementIndex;

// info.isImageLine = true;

}

info.EndCharIndex = currentCharIndex;

info.SpaceCounter = internalSpaceCounter;

storedStyle = getTextStyle();

removeLastSpace = !wordOccurred && (internalSpaceCounter > 0);

**break**;

}

}

}

**if** (newWidth > maxWidth) {

**if** (info.EndElementIndex != startIndex || element **instanceof** ZLTextWord) {

info.IsVisible = isVisible;

info.Width = newWidth-elementWidth;

**if** (info.Height < newHeight) {

info.Height = newHeight;

}

**if**(offsetHeight>0){

info.Height=newHeight-offsetHeight;

}

**if** (info.Descent < newDescent) {

info.Descent = newDescent;

}

info.EndElementIndex = currentElementIndex;

info.EndCharIndex = currentCharIndex;

info.SpaceCounter = internalSpaceCounter;

storedStyle = getTextStyle();

removeLastSpace = !wordOccurred && (internalSpaceCounter > 0);

**break**;

}

}

ZLTextElement previousElement = element;

++currentElementIndex;

currentCharIndex = 0;

**boolean** allowBreak = currentElementIndex == endIndex;

**if** (!allowBreak) {

element = paragraphCursor.getElement(currentElementIndex);

allowBreak = ((!(element **instanceof** ZLTextWord) || previousElement **instanceof** ZLTextWord) &&

!(element **instanceof** ZLTextImageElement) &&

!(element **instanceof** ZLTextControlElement));

}

**if** (allowBreak) {

info.IsVisible = isVisible;

info.Width = newWidth;

**if** (info.Height < newHeight) {

info.Height = newHeight;

}

**if** (info.Descent < newDescent) {

info.Descent = newDescent;

}

info.EndElementIndex = currentElementIndex;

info.EndCharIndex = currentCharIndex;

info.SpaceCounter = internalSpaceCounter;

storedStyle = getTextStyle();

removeLastSpace = !wordOccurred && (internalSpaceCounter > 0);

}

} **while** (currentElementIndex != endIndex);

**if** (currentElementIndex != endIndex &&

(isHyphenationPossible() || info.EndElementIndex == startIndex)) {

ZLTextElement element = paragraphCursor.getElement(currentElementIndex);

**if** (element **instanceof** ZLTextWord) {

**final** ZLTextWord word = (ZLTextWord)element;

newWidth -= getWordWidth(word, currentCharIndex);

info.Width = newWidth;

**int** spaceLeft = maxWidth - newWidth;

**if** ((word.Length > 3 && spaceLeft > 2 \* context.getSpaceWidth())

|| info.EndElementIndex == startIndex) {

ZLTextHyphenationInfo hyphenationInfo = ZLTextHyphenator.*Instance*().getInfo(word);

**int** hyphenationPosition = word.Length - 1;

**int** subwordWidth = 0;

**for**(; hyphenationPosition > currentCharIndex; hyphenationPosition--) {

**if** (hyphenationInfo.isHyphenationPossible(hyphenationPosition)) {

subwordWidth = getWordWidth(

word,

currentCharIndex,

hyphenationPosition - currentCharIndex,

word.Data[word.Offset + hyphenationPosition - 1] != '-'

);

**if** (subwordWidth <= spaceLeft) {

**break**;

}

}

}

**if** (hyphenationPosition == currentCharIndex && info.EndElementIndex == startIndex) {

hyphenationPosition = word.Length == currentCharIndex + 1 ? word.Length : word.Length - 1;

subwordWidth = getWordWidth(word, currentCharIndex, word.Length - currentCharIndex, **false**);

**for**(; hyphenationPosition > currentCharIndex + 1; hyphenationPosition--) {

subwordWidth = getWordWidth(

word,

currentCharIndex,

hyphenationPosition - currentCharIndex,

word.Data[word.Offset + hyphenationPosition - 1] != '-'

);

**if** (subwordWidth <= spaceLeft) {

**break**;

}

}

}

**if** (hyphenationPosition > currentCharIndex) {

info.IsVisible = **true**;

info.Width = newWidth + subwordWidth;

**if** (info.Height < newHeight) {

info.Height = newHeight;

}

**if** (info.Descent < newDescent) {

info.Descent = newDescent;

}

info.EndElementIndex = currentElementIndex;

info.EndCharIndex = hyphenationPosition;

info.SpaceCounter = internalSpaceCounter;

storedStyle = getTextStyle();

removeLastSpace = **false**;

}

}

}

}

**if** (removeLastSpace) {

info.Width -= lastSpaceWidth;

info.SpaceCounter--;

}

setTextStyle(storedStyle);

**if** (isFirstLine&&!isCover) {

info.Height += info.StartStyle.getSpaceBefore();

//info.Height+=myContext.getDedentHegiht();

**if**(info.isImageLine&&info.Height>maxHeight){

info.Height = maxHeight;

**if**((info.Height+info.Descent)>maxHeight){

info.Descent=0;

}

}

}

**if** (info.isEndOfParagraph()) {

info.VSpaceAfter = getTextStyle().getSpaceAfter();

}

**if**(isTitle&&isStart){

info.Height+=Utils.*dip2px*(((ZLAndroidLibrary)ZLAndroidLibrary.*Instance*()).getWidget().getContext(),20);

}

**if**(isTitle&&isEnd){

info.VSpaceAfter=Utils.*dip2px*(((ZLAndroidLibrary)ZLAndroidLibrary.*Instance*()).getWidget().getContext(),20);

}

**if**(isH1){

info.VSpaceAfter+=100;

}

**if**(isImageLine&&image!=**null**){

**if**(info.Height<maxHeight){

image.setLineHeight(info.Height);

}**else**{

image.setLineHeight(image.height);

info.Height = image.height;

}

}

**if**(curImage!=**null**&&curImage.isReduceTextLineWidth()){

**if**(isImageLine){

//got another image while no enough text to fill the empty space

// curImage.setOverHeight(true);

}**else**{

curImage.addTextLineHeight(info.Height+info.Descent);

info.isSideTextLine = **true**;

}

}

**if** ((info.EndElementIndex != endIndex || endIndex == info.ParagraphCursorLength)&&useCache&&isAddToCache) {

myLineInfoCache.put(info, info);

}

**return** info;

}

函数生成页面中每一行的内容及宽高等信息，并存储起来。在生成过程中针对游标遍历到的元素信息进行不同的处理，从而实现页面中各种不同的格式的变化，比如文字颜色，文字大小，文字类型等。

每生成一行后，对剩余的页面内容高度进行重新计算。多个不同的行的信息构成了整个页面，当页面内容高度不能容纳新的一行或者已经为负时结束当前页面结构的计算，并记录开始或者结束的游标位置。

当页面结构计算完成后回到paint函数，调用

**private** **void** prepareTextLine(ZLTextPage page, ZLTextLineInfo info, **int** x, **int** y) {

**boolean** isImageLine = **false**;

y = Math.*min*(y + info.Height, getTopMargin() + page.getTextHeight() - 1);

**final** ZLPaintContext context = getContext();

**final** ZLTextParagraphCursor paragraphCursor = info.ParagraphCursor;

setTextStyle(info.StartStyle);

**int** spaceCounter = info.SpaceCounter;

**int** fullCorrection = 0;

**boolean** endOfParagraph = info.isEndOfParagraph();

**boolean** wordOccurred = **false**;

**boolean** changeStyle = **true**;

x += info.LeftIndent;

**int** rightIndent = getTextStyle().getRightIndent();

**int** maxWidth = getTextAreaWidth();

**final** ImageEntity curImage = page.getCurrentImage();

**if**(curImage!=**null**&&

curImage.isReduceTextLineWidth()){

maxWidth-=curImage.width;

}

**if**(info.isImageLine){

endOfParagraph = **true**;

}

**if**(maxWidth<=0){

maxWidth +=curImage.width;

curImage.setOverHeight(**true**);

}

**switch** (getTextStyle().getAlignment()) {

**case** ZLTextAlignmentType.***ALIGN\_RIGHT***:

x += maxWidth - rightIndent - info.Width;

**break**;

**case** ZLTextAlignmentType.***ALIGN\_CENTER***:

x += (maxWidth - rightIndent - info.Width) / 2;

**break**;

**case** ZLTextAlignmentType.***ALIGN\_JUSTIFY***:

**if** (!endOfParagraph && (paragraphCursor.getElement(info.EndElementIndex) != ZLTextElement.***AfterParagraph***)) {

fullCorrection = maxWidth - getTextStyle().getRightIndent() - info.Width;

}

**break**;

**case** ZLTextAlignmentType.***ALIGN\_LEFT***:

**case** ZLTextAlignmentType.***ALIGN\_UNDEFINED***:

**break**;

}

**final** ZLTextParagraphCursor paragraph = info.ParagraphCursor;

**final** **int** paragraphIndex = paragraph.Index;

**final** **int** endElementIndex = info.EndElementIndex;

**int** charIndex = info.RealStartCharIndex;

ZLTextElementArea spaceElement = **null**;

**int** xMax = getLeftMargin()+maxWidth;

**int** xMin = x;

**if**(curImage!=**null**&&curImage.isReduceTextLineWidth()){

**if**(curImage.align==ImageAlign.***IMAGE\_ALIGN\_LEFT***&&maxWidth!=getTextAreaWidth()){

//image left

xMax += curImage.width;

xMin += curImage.width;

x=xMin;

}

}

info.xMax = xMax;

info.xMin = xMin;

**for** (**int** wordIndex = info.RealStartElementIndex; wordIndex < endElementIndex; ++wordIndex, charIndex = 0) {

**final** ZLTextElement element = paragraph.getElement(wordIndex);

**final** **int** width = getElementWidth(element, charIndex);

**if** (element == ZLTextElement.***HSpace***) {

**if** (wordOccurred && (spaceCounter > 0)) {

**final** **int** correction = fullCorrection / spaceCounter;

**final** **int** spaceLength = context.getSpaceWidth() + correction;

**if** (getTextStyle().isUnderline()) {

spaceElement = **new** ZLTextElementArea(

paragraphIndex, wordIndex, 0,

0, // length

**true**, // is last in element

**false**, // add hyphenation sign

**false**, // changed style

getTextStyle(), element, x, x + spaceLength, y, y

);

} **else** {

spaceElement = **null**;

}

x += spaceLength;

fullCorrection -= correction;

wordOccurred = **false**;

--spaceCounter;

}

} **else** **if** (element **instanceof** ZLTextWord ) {

**final** **int** height = getElementHeight(element);

**final** **int** descent = getElementDescent(element);

**final** **int** length = element **instanceof** ZLTextWord ? ((ZLTextWord)element).Length : 0;

**if** (spaceElement != **null**) {

page.TextElementMap.add(spaceElement);

spaceElement = **null**;

}

page.TextElementMap.add(**new** ZLTextElementArea(

paragraphIndex, wordIndex, charIndex,

length - charIndex,

**true**, // is last in element

**false**, // add hyphenation sign

changeStyle, getTextStyle(), element,

x, x + width - 1, y - height + 1, y + descent

));

changeStyle = **false**;

wordOccurred = **true**;

}**else** **if**(element **instanceof** ZLTextImageElement){

**final** **int** height = getElementHeight(element);

**final** **int** descent = getElementDescent(element);

**final** **int** length = 0;

**if**(!((ZLTextImageElement) element).IsCover){

**if**(width<=0||height<=0){

}**else**{

ImageEntity image = **new** ImageEntity(info,((ZLTextImageElement) element).Id,

width, height, ((ZLTextImageElement) element).align);

**if**(image.isEmbedImage()){

**if**(!page.hasImage(image)){

page.ImageLists.add(image);

page.setCurImage(page.ImageLists.size()-1);

}**else**{

page.resetImageTextHeight(image);

}

**if**(image.lineInfo.Height==0){

y+=height;

}

}

**if**(curImage!=**null**&&curImage.isReduceTextLineWidth()&&!curImage.equals(image)){

y+=(curImage.getLineHeight()+curImage.lineInfo.Descent-curImage.getCurTextHeight());

**if**(image.align==ImageAlign.***IMAGE\_ALIGN\_RIGHT***){

xMax+= curImage.width;

}

}

**if**(image.align!=ImageAlign.***IMAGE\_ALIGN\_TEXT***){

isImageLine = **true**;

}

**switch**(image.align){

**case** ***IMAGE\_ALIGN\_CENTER***:

//align middle

x=(getContext().getWidth()-width)/2;

**break**;

**case** ***IMAGE\_ALIGN\_LEFT***:

//align left

x=getLeftMargin()+getImageBoundary();

**break**;

**case** ***IMAGE\_ALIGN\_RIGHT***:

//align right

x=xMax-width+getImageBoundary();

**break**;

**case** ***IMAGE\_ALIGN\_TEXT***:

**break**;

**case** ***IMAGE\_ALIGN\_MID***:

**break**;

**default**:

**break**;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if** (spaceElement != **null**) {

page.TextElementMap.add(spaceElement);

spaceElement = **null**;

}

page.TextElementMap.add(**new** ZLTextElementArea(

paragraphIndex, wordIndex, charIndex,

length - charIndex,

**true**, // is last in element

**false**, // add hyphenation sign

changeStyle, getTextStyle(), element,

x, x + width - 1, y - height + getImageBoundary(), y + descent

));

changeStyle = **false**;

wordOccurred = **true**;

}**else** **if** (isStyleChangeElement(element)) {

applyStyleChangeElement(element);

changeStyle = **true**;

}

x += width;

}

**if** (!endOfParagraph) {

**final** **int** len = info.EndCharIndex;

**if** (len > 0) {

**final** **int** wordIndex = info.EndElementIndex;

**final** ZLTextWord word = (ZLTextWord)paragraph.getElement(wordIndex);

**final** **boolean** addHyphenationSign = word.Data[word.Offset + len - 1] != '-';

**final** **int** width = getWordWidth(word, 0, len, addHyphenationSign);

**final** **int** height = getElementHeight(word);

**final** **int** descent = context.getDescent();

page.TextElementMap.add(

**new** ZLTextElementArea(

paragraphIndex, wordIndex, 0,

len,

**false**, // is last in element

addHyphenationSign,

changeStyle, getTextStyle(), word,

x, x + width - 1, y - height + 1, y + descent

)

);

}

}

**if**(curImage!=**null**&&curImage.isReduceTextLineWidth()){

**if**(isImageLine){

//got another image while no enough text to fill the empty space

// curImage.setOverHeight(true);

}**else**{

curImage.addTextLineHeight(info.Height+info.Descent);

}

}

}

函数根据之前计算的结构信息，逐个计算行中每一个元素的坐标，并存储。在这个函数中可以对行的文字间的间距进行调整以实现最后页面内容展示的时候行的内容能对首尾对齐。

最后调用函数

**private** **void** drawTextLine(ZLTextPage page, ZLTextLineInfo info, **int** from, **int** to) {

**final** ZLPaintContext context = getContext();

**final** ZLTextParagraphCursor paragraph = info.ParagraphCursor;

**int** index = from;

**final** **int** endElementIndex = info.EndElementIndex;

**int** charIndex = info.RealStartCharIndex;

**for** (**int** wordIndex = info.RealStartElementIndex; wordIndex != endElementIndex && index < to; ++wordIndex, charIndex = 0) {

**final** ZLTextElement element = paragraph.getElement(wordIndex);

**final** ZLTextElementArea area = page.TextElementMap.get(index);

**if** (element == area.Element) {

++index;

**if** (area.ChangeStyle) {

setTextStyle(area.Style);

}

**final** **int** areaX = area.XStart;

**final** **int** areaY = area.YEnd - getElementDescent(element) - getTextStyle().getVerticalShift();

**if** (element **instanceof** ZLTextWord) {

drawWord(

areaX, areaY, (ZLTextWord)element, charIndex, -1, **false**,

mySelection.isAreaSelected(area)

? getSelectionForegroundColor() : getTextColor(getTextStyle().Hyperlink)

);

} **else** **if** (element **instanceof** ZLTextImageElement) {

**final** ZLTextImageElement imageElement = (ZLTextImageElement)element;

Size maxSize;

**if**(!imageElement.IsCover){

maxSize = **new** Size(getPictureWidthMax(imageElement.align), getTextAreaHeight());

}**else**{

maxSize = getTextAreaSize();

}

context.drawImage(

areaX, areaY,

imageElement.ImageData,

maxSize,

imageElement.IsCover

? ZLPaintContext.ScalingType.***FitMaximum***

: ZLPaintContext.ScalingType.***IntegerCoefficient***

);

} **else** **if** (element == ZLTextElement.***HSpace***) {

**final** **int** cw = context.getSpaceWidth();

/\*

context.setFillColor(getHighlightingColor());

context.fillRectangle(

area.XStart, areaY - context.getStringHeight(),

area.XEnd - 1, areaY + context.getDescent()

);

\*/

**for** (**int** len = 0; len < area.XEnd - area.XStart; len += cw) {

context.drawString(areaX + len, areaY, ***SPACE***, 0, 1);

}

}

}

}

**if** (index != to) {

ZLTextElementArea area = page.TextElementMap.get(index++);

**if** (area.ChangeStyle) {

setTextStyle(area.Style);

}

**final** **int** start = info.StartElementIndex == info.EndElementIndex

? info.StartCharIndex : 0;

**final** **int** len = info.EndCharIndex - start;

**final** ZLTextWord word = (ZLTextWord)paragraph.getElement(info.EndElementIndex);

drawWord(

area.XStart, area.YEnd - context.getDescent() - getTextStyle().getVerticalShift(),

word, start, len, area.AddHyphenationSign,

mySelection.isAreaSelected(area)

? getSelectionForegroundColor() : getTextColor(getTextStyle().Hyperlink)

);

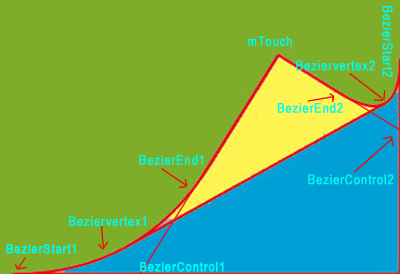
}

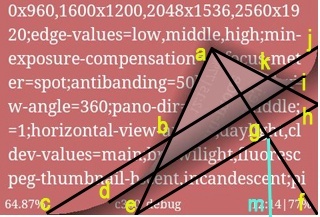
}

根据计算得出的坐标，逐个元素的绘制到bitmap上。最后在paint函数最后完成页脚，书签等的绘制。这样，页面加载到页面绘制的流程就结束了。返回的bitmap直接显示出来就行了。

翻页动画的实现实际上是通过在显示bitmap的时候对bitmap的位置等参数来实现的，如阅读器中实现的滑动，移动方式的翻页动画。

而真实翻页动画的模拟实现是利用canvas的path绘制结合Bezier曲线的路径计算来实现的，动画移动的过程是通过Scroller类的public void startScroll(int startX, int startY, int dx, int dy, int duration)函数来实现的，然后再滚动的过程中加入bezier曲线路径的绘制，加上一些shape的绘制，实现了真实的翻页效果。Bezier原理图如下图所示





主要实现过程如下：

首先根据触摸点计算需要卷起的是哪一个角，

**public** **void** calcCornerXY(**float** x, **float** y) {

printLogInfo(TAG,"calcCornerXY start x="+x+", y="+y);

**float** leftX = 0;

**float** rightX = 0;

**float** topY = 0;

**float** bottomY =0;

**if**(mReadMode==***MODE\_ONE\_PAGE***){

leftX = mPositionRectDst.left;

rightX = mPageWidth+leftX;

topY = mPositionRectDst.top;

bottomY = mPageHeight+topY;

mCornerX = rightX;

**if**(y<topY+(bottomY-topY)\*2/5.0f&&

mFlipMode==***ONEFLIPLEFT***){

mCornerY = topY;

mFlatFlip = **false**;

mIsRTandLB = **true**;

}**else** **if**(y>topY+(bottomY-topY)\*3/5.0f&&

mFlipMode==***ONEFLIPLEFT***){

mCornerY = bottomY;

mFlatFlip = **false**;

mIsRTandLB = **false**;

}**else**{

mCornerY = topY;

mFlatFlip = **true**;

mIsRTandLB = **true**;

}

}**else** **if**(mReadMode==***MODE\_TWO\_PAGE***){

leftX = mPositionRectDst.left-mPageWidth;

rightX = mPageWidth+mPositionRectDst.left;

topY = mPositionRectDst.top;

bottomY = mPageHeight+topY;

**if** (x <mPositionRectDst.left&&x>mPositionRectDst.left-mPageWidth)

mCornerX = leftX;

**else**

mCornerX = rightX;

**if**(y<topY+(bottomY-topY)\*2/5.0f){

mCornerY = topY;

mFlatFlip = **false**;

}**else** **if**(y>topY+(bottomY-topY)\*3/5.0f){

mCornerY = bottomY;

mFlatFlip = **false**;

}**else**{

mCornerY = topY;

mFlatFlip = **true**;

mIsRTandLB = **true**;

}

**if** ((mCornerX == leftX && mCornerY == bottomY)

|| (mCornerX == rightX && mCornerY == topY))

mIsRTandLB = **true**;

**else**

mIsRTandLB = **false**;

}

printLogInfo(TAG,"calcCornerXY end mCornerX="+mCornerX+", mCornerY="+mCornerY);

}

计算好后，调用

**public** **void** startAnimation() {

**int** dx = 0;

**int** dy = 0;

**int** top = -1;

/\*\*

\* Calculate duration by distance

\* \*/

**int** tmpDuration = 0;

**if**(mReadMode==***MODE\_ONE\_PAGE***){

**int** left = -1;

left = mPositionRectDst.left;

top = mPositionRectDst.top;

// dx 水平方向滑动的距离，负值会使滚动向左滚动

// dy 垂直方向滑动的距离，负值会使滚动向上滚动

**if** (mCornerX > left&&mFlipMode==***ONEFLIPLEFT***) {

dx = -(**int**) (mPageWidth + mTouch.x-left);

tmpDuration = (-dx\*mDuration)/(2\*mPageWidth);

} **else** {

dx = (**int**) (mPageWidth+left - mTouch.x/\* + mPageWidth\*/);

tmpDuration = (dx\*mDuration)/(mPageWidth);

}

**if** (mCornerY > top) {

dy = (**int**) (mPageHeight - mTouch.y+mPositionRectDst.top);

} **else** {

dy = (**int**) (1f - mTouch.y+mPositionRectDst.top); // 防止mTouch.y最终变为0

}

}**else**{

**int** mid = mPositionRectDst.left;

top = mPositionRectDst.top;

**if**(mCornerX>mid){

dx = -(**int**) (mPageWidth + mTouch.x-mid);

tmpDuration = (-dx\*mDuration)/(2\*mPageWidth);

}**else** {

dx = (**int**) (mPageWidth + mid - mTouch.x);

tmpDuration = (dx\*mDuration)/(2\*mPageWidth);

}

**if** (mCornerY > top) {

dy = (**int**) (mPageHeight - mTouch.y+mPositionRectDst.top);

} **else** {

dy = (**int**) (1f - mTouch.y+mPositionRectDst.top); // 防止mTouch.y最终变为0

}

}

mEffectState = ***PAGEFLIPPEDANIMATION***;

printLogInfo(TAG,"startAnimation mTouch.x="+mTouch.x+", dx="+dx+", dy="+dy);

mScroller.startScroll((**int**) mTouch.x, (**int**) mTouch.y, dx, dy,tmpDuration);

mTargetView.postInvalidate();

}

开始执行翻页动画。然后每次页面更新的时候通过如下函数

**private** **void** calcBezierPoints(**float** leftbound,**float** rightbound,

**float** offsetx, **float** offsety){

// printLogInfo(TAG,"calcBezierPoints start leftbound="+leftbound+", rightbound="+rightbound);

mMiddleX = (mTouch.x + mCornerX) / 2;

mMiddleY = (mTouch.y + mCornerY) / 2;

mBezierControl1.x = (**float**) (mMiddleX - (mCornerY - mMiddleY)

\* (mCornerY - mMiddleY) / getDivider(mCornerX,mMiddleX));

mBezierControl1.y = mCornerY;

mBezierControl2.x = mCornerX;

mBezierControl2.y = (**float**) (mMiddleY - (mCornerX - mMiddleX)

\* (mCornerX - mMiddleX) / getDivider(mCornerY, mMiddleY));

mBezierStart1.x = mBezierControl1.x - (mCornerX - mBezierControl1.x)/ 2;

mBezierStart1.y = mCornerY;

**if**(mReadMode==***MODE\_TWO\_PAGE***){

**if**(mBezierStart1.x<leftbound||mBezierStart1.x>rightbound){

**if** (mBezierStart1.x < leftbound){

mBezierStart1.x = leftbound;

}**else** **if**(mBezierStart1.x>rightbound){

mBezierStart1.x =rightbound;

}

}

}

// 当mBezierStart1.x < 0或者mBezierStart1.x > 480时

// 如果继续翻页，会出现BUG故在此限制

**if** (mTouch.x > leftbound && mTouch.x < rightbound) {

**if** (mBezierStart1.x < leftbound || mBezierStart1.x > rightbound) {

**if** (mBezierStart1.x < leftbound)

mBezierStart1.x = rightbound - mBezierStart1.x;

**float** f1 = Math.*abs*(mCornerX - mTouch.x);

**float** f2 = rightbound \* f1 / mBezierStart1.x;

mTouch.x = Math.*abs*(mCornerX - f2);

**float** f3 = Math.*abs*(mCornerX - mTouch.x)

\* Math.*abs*(mCornerY - mTouch.y) / f1;

mTouch.y = Math.*abs*(mCornerY - f3);

mMiddleX = (mTouch.x + mCornerX) / 2;

mMiddleY = (mTouch.y + mCornerY) / 2;

mBezierControl1.x = (**float**) (mMiddleX - (mCornerY - mMiddleY)

\* (mCornerY - mMiddleY) / getDivider(mCornerX, mMiddleX));

mBezierControl1.y = mCornerY;

mBezierControl2.x = mCornerX;

mBezierControl2.y = (**float**) (mMiddleY - (mCornerX - mMiddleX)

\* (mCornerX - mMiddleX) / getDivider(mCornerY, mMiddleY));

mBezierStart1.x = mBezierControl1.x - (mCornerX - mBezierControl1.x) / 2;

}

}

mBezierStart2.x = mCornerX;

mBezierStart2.y = mBezierControl2.y - (mCornerY - mBezierControl2.y)/ 2;

mTouchToCornerDis = (**float**) Math.*hypot*((mTouch.x - mCornerX),

(mTouch.y - mCornerY));

mBezierEnd1 = getCross(mTouch, mBezierControl1, mBezierStart1,mBezierStart2);

mBezierEnd2 = getCross(mTouch, mBezierControl2, mBezierStart1,mBezierStart2);

mBeziervertex1.x = (mBezierStart1.x + 2 \* mBezierControl1.x + mBezierEnd1.x) / 4;

mBeziervertex1.y = (2 \* mBezierControl1.y + mBezierStart1.y + mBezierEnd1.y) / 4;

mBeziervertex2.x = (mBezierStart2.x + 2 \* mBezierControl2.x + mBezierEnd2.x) / 4;

mBeziervertex2.y = (2 \* mBezierControl2.y + mBezierStart2.y + mBezierEnd2.y) / 4;

**if**(offsetx>0){

mBezierStart1.x+=offsetx;

mBezierControl1.x+=offsetx;

mBezierEnd1.x+=offsetx;

mBeziervertex1.x+=offsetx;

mBezierStart2.x+=offsetx;

mBezierControl2.x+=offsetx;

mBezierEnd2.x+=offsetx;

mBeziervertex2.x+=offsetx;

mTouch.x+=offsetx;

mCornerX+=offsetx;

}

**if**(offsety>0){

mBezierStart1.y+=offsety;

mBezierControl1.y+=offsety;

mBezierEnd1.y+=offsety;

mBeziervertex1.y+=offsety;

mBezierStart2.y+=offsety;

mBezierControl2.y+=offsety;

mBezierEnd2.y+=offsety;

mBeziervertex2.y+=offsety;

mTouch.y+=offsety;

mCornerY+=offsety;

}

// mTouchPrevious.x = mTouch.x;

// mTouchPrevious.y = mTouch.y;

// printLogError(TAG, "final mMiddleX=" + mMiddleX + " mMiddleY=" + mMiddleY);

// printLogInfo(TAG, "final mPageWidth=" + mPageWidth + " mPageHeight=" + mPageHeight);

// printLogInfo(TAG, "final mPositionRectDst=" + mPositionRectDst);

// printLogInfo(TAG, "final mTouchX=" + mTouch.x + " mTouchY=" + mTouch.y);

// printLogInfo(TAG, "final mCornerX=" + mCornerX + " mCornerY=" + mCornerY);

// printLogInfo(TAG, "final mBezierStart1.x=" + mBezierStart1.x + " mBezierStart1.y=" + mBezierStart1.y);

// printLogInfo(TAG, "final mBezierStart2.x=" + mBezierStart2.x + " mBezierStart2.y=" + mBezierStart2.y);

// printLogInfo(TAG, "final mBezierControl1.x=" + mBezierControl1.x+ " mBezierControl1.y=" + mBezierControl1.y);

// printLogInfo(TAG, "final mBezierControl2.x=" + mBezierControl2.x+ " mBezierControl2.y=" + mBezierControl2.y);

// printLogInfo(TAG, "final mBezierEnd1.x=" + mBezierEnd1.x + " mBezierEnd1.y="+ mBezierEnd1.y);

// printLogInfo(TAG, "final mBezierEnd2.x=" + mBezierEnd2.x + " mBezierEnd2.y="+ mBezierEnd2.y);

// printLogInfo(TAG, "final mBeziervertex1.x=" + mBeziervertex1.x + " mBeziervertex1.y=" + mBeziervertex1.y);

// printLogError(TAG, "final mBeziervertex2.x=" + mBeziervertex2.x + " mBeziervertex2.y=" + mBeziervertex2.y);

}

根据当前卷起的角和scroll的距离计算得出Bezier曲线的各个点的坐标。计算好后，分别依次根据当前计算出的Bezier曲线的点的坐标计算出当前页要绘制的路径并绘制

**private** **void** drawCurrentPageArea(Canvas canvas, Bitmap bitmap, Path path) {

// printLogInfo(TAG,"drawCurrentPageArea bitmap="+bitmap);

mPath0.reset();

mPath0.moveTo(mBezierStart1.x, mBezierStart1.y);

mPath0.quadTo(mBezierControl1.x, mBezierControl1.y, mBezierEnd1.x,

mBezierEnd1.y);

mPath0.lineTo(mTouch.x, mTouch.y);

mPath0.lineTo(mBezierEnd2.x, mBezierEnd2.y);

mPath0.quadTo(mBezierControl2.x, mBezierControl2.y, mBezierStart2.x,

mBezierStart2.y);

mPath0.lineTo(mCornerX, mCornerY);

mPath0.close();

canvas.save();

canvas.clipPath(mPath0, Region.Op.***XOR***);

**if**(mFlipMode==***TWOFLIPRIGHT***){

canvas.drawBitmap(bitmap, mPositionRectDst.left-bitmap.getWidth(), mPositionRectDst.top, mPaint);

}**else**{

canvas.drawBitmap(bitmap, mPositionRectDst.left, mPositionRectDst.top, mPaint);

}

canvas.restore();

}

计算出下一页面及生成阴影

/\*\*

\* 下页页面及阴影

\*/

**private** **void** drawNextPageAreaAndShadow(Canvas canvas, Bitmap bitmap) {

// printLogInfo(TAG,"drawNextPageAreaAndShadow bitmap="+bitmap);

mPath1.reset();

mPath1.moveTo(mBezierStart1.x, mBezierStart1.y);

mPath1.lineTo(mBeziervertex1.x, mBeziervertex1.y);

mPath1.lineTo(mBeziervertex2.x, mBeziervertex2.y);

mPath1.lineTo(mBezierStart2.x, mBezierStart2.y);

mPath1.lineTo(mCornerX, mCornerY);

mPath1.close();

mDegrees = (**float**) Math.*toDegrees*(Math.*atan2*(mBezierControl1.x

- mCornerX, mBezierControl2.y - mCornerY));

**int** leftx = 0;

**int** rightx = 0;//take control of width of the shadow

GradientDrawable mBackShadowDrawable;

**if** (mIsRTandLB) {

**if**(mReadMode==***MODE\_TWO\_PAGE***){

**if**(mFlipMode==***TWOFLIPLEFT***&&mTouch.x<mPositionRectDst.left){

leftx = (**int**) mBezierStart1.x;

rightx = (**int**) (mBezierStart1.x - (mPositionRectDst.left-mPageWidth-mTouch.x)/3);

}**else** **if**(mFlipMode==***TWOFLIPRIGHT***&&mTouch.x>mPositionRectDst.left){

leftx = (**int**) mBezierStart1.x;

rightx = (**int**) (mBezierStart1.x - (mTouch.x-mPositionRectDst.right)/3);

}**else**{

leftx = (**int**) (mBezierStart1.x);

rightx = (**int**) (mBezierStart1.x + mTouchToCornerDis / 4);

}

}**else**{

leftx = (**int**) (mBezierStart1.x);

rightx = (**int**) (mBezierStart1.x + mTouchToCornerDis / 4);

}

mBackShadowDrawable = mBackShadowDrawableLR;

} **else** {

**if**(mReadMode==***MODE\_TWO\_PAGE***){

**if**(mFlipMode==***TWOFLIPLEFT***&&mTouch.x<mPositionRectDst.left){

leftx = (**int**) (mBezierStart1.x + (mPositionRectDst.left-mPageWidth-mTouch.x)/3);

rightx = (**int**) mBezierStart1.x;

}**else** **if**(mFlipMode==***TWOFLIPRIGHT***&&mTouch.x>mPositionRectDst.left){

leftx = (**int**) (mBezierStart1.x + (mTouch.x-mPositionRectDst.right)/3);

rightx = (**int**) mBezierStart1.x;

}**else**{

leftx = (**int**) (mBezierStart1.x - mTouchToCornerDis / 4);

rightx = (**int**) mBezierStart1.x;

}

}**else**{

leftx = (**int**) (mBezierStart1.x - mTouchToCornerDis / 4);

rightx = (**int**) mBezierStart1.x;

}

mBackShadowDrawable = mBackShadowDrawableRL;

}

canvas.save();

canvas.clipPath(mPath0);

canvas.clipPath(mPath1, Region.Op.***INTERSECT***);

**if**(mFlipMode!=***TWOFLIPRIGHT***){

canvas.drawBitmap(bitmap,mPositionRectDst.left, mPositionRectDst.top, mPaint);

}

canvas.rotate(mDegrees, mBezierStart1.x, mBezierStart1.y);

mBackShadowDrawable.setBounds(leftx, (**int**) mBezierStart1.y, rightx,

(**int**) (mMaxLength + mBezierStart1.y));

mBackShadowDrawable.draw(canvas);

canvas.restore();

}

当前页背面阴影

/\*\*

\* 当前页阴影

\*/

**public** **void** drawCurrentPageShadow(Canvas canvas) {

// printLogInfo(TAG,"drawCurrentPageShadow called");

**double** degree;

**if** (mIsRTandLB) {

degree = Math.***PI***/4 - Math.*atan2*(mBezierControl1.y - mTouch.y,

mTouch.x - mBezierControl1.x);

} **else** {

degree = Math.***PI***/4 - Math.*atan2*(mTouch.y - mBezierControl1.y,

mTouch.x - mBezierControl1.x);

}

// 翻起页阴影顶点与touch点的距离

**double** d1 = (**float**) 25 \* 1.414 \* Math.*cos*(degree);

**double** d2 = (**float**) 25 \* 1.414 \* Math.*sin*(degree);

**float** x = (**float**) (mTouch.x + d1);

**float** y;

**if** (mIsRTandLB) {

y = (**float**) (mTouch.y + d2);

} **else** {

y = (**float**) (mTouch.y - d2);

}

mPath1.reset();

mPath1.moveTo(x, y);

mPath1.lineTo(mTouch.x, mTouch.y);

mPath1.lineTo(mBezierControl1.x, mBezierControl1.y);

mPath1.lineTo(mBezierStart1.x, mBezierStart1.y);

mPath1.close();

**float** rotateDegrees;

canvas.save();

canvas.clipPath(mPath0, Region.Op.***XOR***);

canvas.clipPath(mPath1, Region.Op.***INTERSECT***);

**int** leftx = 0;

**int** rightx = 0;

GradientDrawable mCurrentPageShadow;

**if** (mIsRTandLB) {

leftx = (**int**) (mBezierControl1.x);

rightx = (**int**) mBezierControl1.x + 25;

mCurrentPageShadow = mFrontShadowDrawableVLR;

} **else** {

leftx = (**int**) (mBezierControl1.x - 25);

rightx = (**int**) mBezierControl1.x + 1;

mCurrentPageShadow = mFrontShadowDrawableVRL;

}

rotateDegrees = (**float**) Math.*toDegrees*(Math.*atan2*(mTouch.x

- mBezierControl1.x, mBezierControl1.y - mTouch.y));

canvas.rotate(rotateDegrees, mBezierControl1.x, mBezierControl1.y);

mCurrentPageShadow.setBounds(leftx,

(**int**) (mBezierControl1.y - mMaxLength), rightx,

(**int**) (mBezierControl1.y));

mCurrentPageShadow.draw(canvas);

canvas.restore();

**int** upY = 0;

**int** downY = 0;

mPath1.reset();

mPath1.moveTo(x, y);

mPath1.lineTo(mTouch.x, mTouch.y);

mPath1.lineTo(mBezierControl2.x, mBezierControl2.y);

mPath1.lineTo(mBezierStart2.x, mBezierStart2.y);

mPath1.close();

canvas.save();

canvas.clipPath(mPath0, Region.Op.***XOR***);

canvas.clipPath(mPath1, Region.Op.***INTERSECT***);

**if** (mIsRTandLB) {

upY = (**int**) (mBezierControl2.y);

downY = (**int**) (mBezierControl2.y + 25);

mCurrentPageShadow = mFrontShadowDrawableHTB;

} **else** {

upY = (**int**) (mBezierControl2.y - 25);

downY = (**int**) (mBezierControl2.y + 1);

mCurrentPageShadow = mFrontShadowDrawableHBT;

}

rotateDegrees = (**float**) Math.*toDegrees*(

Math.*atan2*(mBezierControl2.y - mTouch.y, mBezierControl2.x - mTouch.x));

canvas.rotate(rotateDegrees, mBezierControl2.x, mBezierControl2.y);

**float** temp;

**if** (mBezierControl2.y < 0)

temp = mBezierControl2.y - mPageHeight;

**else**

temp = mBezierControl2.y;

**int** hmg = (**int**) Math.*hypot*(mBezierControl2.x, temp);

**if** (hmg > mMaxLength){

mCurrentPageShadow.setBounds((**int**) (mBezierControl2.x - 25) - hmg,

upY,(**int**) (mBezierControl2.x + mMaxLength) - hmg,downY);

}

**else**{

mCurrentPageShadow.setBounds(

(**int**) (mBezierControl2.x - mMaxLength), upY,

(**int**) (mBezierControl2.x), downY);

}

mCurrentPageShadow.draw(canvas);

canvas.restore();

}

计算当前页面的背面路径并绘制

**private** **void** drawCurrentBackArea(Canvas canvas, Bitmap bitmap) {

// printLogInfo(TAG,"drawCurrentBackArea bitmap="+bitmap);

**int** MidXOfStAndCtrl = (**int**) (mBezierStart1.x + mBezierControl1.x) / 2;

**float** DistXBetweenMidAndCtrl = Math.*abs*(MidXOfStAndCtrl - mBezierControl1.x);

**int** MidYOfStartAndControl = (**int**) (mBezierStart2.y + mBezierControl2.y) / 2;

**float** DistYBetweenMidAndCtrl = Math.*abs*(MidYOfStartAndControl - mBezierControl2.y);

**float** MinDistance = Math.*min*(DistXBetweenMidAndCtrl, DistYBetweenMidAndCtrl);

mPath1.reset();

mPath1.moveTo(mBeziervertex2.x, mBeziervertex2.y);

mPath1.lineTo(mBeziervertex1.x, mBeziervertex1.y);

mPath1.lineTo(mBezierEnd1.x, mBezierEnd1.y);

mPath1.lineTo(mTouch.x, mTouch.y);

mPath1.lineTo(mBezierEnd2.x, mBezierEnd2.y);

mPath1.close();

GradientDrawable mFolderShadowDrawable;

**int** left;

**int** right;

**if** (mIsRTandLB) {

left = (**int**) (mBezierStart1.x - 1);

right = (**int**) (mBezierStart1.x + MinDistance + 1);

mFolderShadowDrawable = mFolderShadowDrawableLR;

} **else** {

left = (**int**) (mBezierStart1.x - MinDistance - 1);

right = (**int**) (mBezierStart1.x + 1);

mFolderShadowDrawable = mFolderShadowDrawableRL;

}

canvas.save();

canvas.clipPath(mPath0);

canvas.clipPath(mPath1, Region.Op.***INTERSECT***);

// //画边角缺失颜色

// mPaint.setAlpha(0xff);

// mPaint.setColor(computeColor(bitmap));

// canvas.drawPath(mPath1, mPaint);

// canvas.save();

**float** DistanceBetweenTwoCtrls = (**float**) Math.*hypot*(mCornerX - mBezierControl1.x,

mBezierControl2.y - mCornerY);

//if A is the angle of two line segments,one line segment is terminated by two control points,

//and the other one is terminated by corner and the control point one

**float** cosA = (mCornerX - mBezierControl1.x) / DistanceBetweenTwoCtrls;

**float** sinA = (mBezierControl2.y - mCornerY) / DistanceBetweenTwoCtrls;//sinA

mMatrixArray[0] = 1 - 2 \* sinA \* sinA;//cos2A

mMatrixArray[1] = 2 \* cosA \* sinA;//sin2A

mMatrixArray[3] = mMatrixArray[1];//sin2A

mMatrixArray[4] = 1 - 2 \* cosA \* cosA;//-cos2A

mMatrix.reset();

mMatrix.setValues(mMatrixArray);

**if**(mFlipMode==***TWOFLIPRIGHT***){

mMatrix.preTranslate(mPositionRectDst.left-bitmap.getWidth(), mPositionRectDst.top);

}**else**{

mMatrix.preTranslate(mPositionRectDst.left, mPositionRectDst.top);

}

mMatrix.preTranslate(-mBezierControl1.x, -mBezierControl1.y);

mMatrix.postTranslate(mBezierControl1.x, mBezierControl1.y);

**if**(mReadMode==***MODE\_ONE\_PAGE***){

mPaint.setAlpha(0x40);

}**else**{

mPaint.setAlpha(0xff);

}

canvas.drawBitmap(bitmap, mMatrix, mPaint);

mPaint.setAlpha(0xff);

canvas.rotate(mDegrees, mBezierStart1.x, mBezierStart1.y);

mFolderShadowDrawable.setBounds(left, (**int**) mBezierStart1.y, right,

(**int**) (mBezierStart1.y + mMaxLength));

mFolderShadowDrawable.draw(canvas);

canvas.restore();

}

完成一轮绘制后，不断更新页面又开始下轮计算下一bezier曲线的各点，并计算各页面的path及阴影进行绘制。直到翻动到指定的位置为止，这一连续的不断绘制更新即实现了整个翻页动画。