Capítulo 29. Zend_Pdf

Traduzido por Flávio Gomes da Silva Lisboa

Sumário

29.1.Introdução.

O módulo Zend_Pdf é um engenho de manipulação de PDF (Portable Document Format) escrito totalmente em PHP 5. Ele pode carregar documentos existentes, criar novos, modificar e gravar documentos modificados. Assim ele pode ajudar qualquer aplicação PHP a preparar dinamicamente documentos em um PDF por meio da modificação do modelo existente ou gerar um documento a partir de um rascunho. O módulo Zend_Pdf suporta as seguintes características:

- Cria um novo documento ou carrega um existente. [2]
- Recupera a revisão especificada do documento.
- Manipula páginas dentro do documento. Altera ordem de páginas, adiciona novas páginas e remove páginas de um documento.
- Diferentes primitivas de desenho (linhas, retângulos, polígonos, círculos, elipses e setores).
- Desenho de texto usando qualquer uma das 14 fontes padrão (embutidas) ou suas próprias fontes TrueType customizadas.
- Rotações.
- Desenho de imagens. [3]
- Atualização de arquivo PDF incremental.

[2] Documentos PDF V1.4 (Acrobat 5) são suportados para carregamento agora.

[3] Imagens JPG, PNG [Acima de 8 bits por canal+Alfa] e TIFF são suportadas.

29.2.Criando e carregando documentos PDF.

A classe `Zend_Pdf` representa um documento PDF e fornece funcionalidade no nível de documento.

Para criar um novo documento um novo objeto `Zend_Pdf` deve ser criado.

A classe `Zend_Pdf` também fornece dois métodos estáticos para carregar um PDF existente. São eles `Zend_Pdf::load()` e `Zend_Pdf::parse()`. Ambos os dois retornam um objeto Zend_Pdf como um resultado ou lançam uma exceção se um erro ocorrer.

Exemplo 29.1. Criar um novo PDF ou carregar um existente.

```
<?php
...
//Cria um novo documento PDF.
$pdf1 = new Zend_Pdf();
// Carrega um documento PDF de um arquivo.
$pdf2 = Zend_Pdf::load($fileName);
// Carrega um documento PDF de uma variável string.
$pdf3 = Zend_Pdf::parse($pdfString);
...
```

O formato de arquivo PDF suporta atualização incremental. Assim, cada vez que um documento é atualizado, um nova revisão do documento é criada. O módulo Zend_Pdf suporta a recuperação de uma revisão específica.

Uma revisão pode ser especificada como um segundo parâmetro para os métodos `Zend_Pdf::load()` e `Zend_Pdf::parse()` ou requisitada por chamada a `Zend_Pdf::rollback()` [4].

Exemplo 29.2. Requisitando revisões específicas de um documento PDF.

```
<?php
...
//Carrega última revisão do documento.
$pdf1 = Zend_Pdf::load($fileName, 1);
//Carrega última revisão do documento.
$pdf2 = Zend_Pdf::parse($pdfString, 1);
//Carrega a primeira revisão do documento.
$pdf3 = Zend_Pdf::load($fileName);
$revisions = $pdf3->revisions();
$pdf3->rollback($revisions - 1);
...
```

[4] O método `Zend_Pdf::rollback()` deve ser invocado antes de quaisquer mudanças, aplicadas ao documento. Caso contrário o comportamento é indefinido.

29.3. Gravar mudanças no documento PDF.

Há dois métodos, que fornecem a gravação de mudanças no documento PDF. São eles `Zend_Pdf::save()` e `Zend_Pdf::render()`.

`Zend_Pdf::save($filename, $updateOnly = false)` grava o documento PDF em um arquivo. Se $updateOnly é verdadeiro, então somente o novo segmento de arquivo PDF é adicionado ao arquivo. Caso contrário, o arquivo é sobrescrito.

`Zend_Pdf::render($newSegmentOnly = false)` retorna o documento PDF como um string. Se $newSegmentOnly é verdadeiro, então somente o novo segmento de arquivo PDF é retornado.

Exemplo 29.3. Gravar documento PDF.

```
<?php
...
//Carrega documento PDF.
$pdf = Zend_Pdf::load($fileName);
...
// Atualiza documento PDF
$pdf->save($fileName, true);
// Grava documento como um novo arquivo
$pdf->save($newFileName);
// Retorna o documento PDF como uma variável string.
$pdfString = $pdf->render();
...
```

29.4.Pintura.

29.4.1.Geometria.

PDF usa a mesma geometria que PostScript. Ele inicia do canto inferior esquerdo da página e por padrão é medido em pontos (1/72 de um inch ou 2,54cm) .

O tamanho da página pode ser recuperado de um objeto page:

```
<?php
$width  = $pdfPage->getWidth();
$height = $pdfPage->getHeight();
```

29.4.2.Cores.

PDF tem capacidades poderosas para representação de cores. O módulo Zend_Pdf suporta Escala de Cinza, RGB e CMYK. Qualquer uma delas pode ser usada em qualquer lugar, onde o objeto `Zend_Pdf_Color` é requerido. As classes `Zend_Pdf_Color_GrayScale`, `Zend_Pdf_Color_Rgb` e `Zend_Pdf_Color_Cmyk` fornecem essa funcionalidade:

```php
<?php
// $grayLevel (números float). 0.0 (preto) - 1.0 (branco)
$color1 = new Zend_Pdf_Color_GrayScale($grayLevel);
// $r, $g, $b (números float). 0.0 (intensidade mínima) - 1.0 (intensidade
máxima)
$color2 = new Zend_Pdf_Color_Rgb($r, $g, $b);
// $c, $m, $y, $k (números float). 0.0 (intensidade mínima) - 1.0 (intensidade
máxima)
$color3 = new Zend_Pdf_Color_Cmyk($c, $m, $y, $k);
```

Cores estilo HTML também são fornecidades com a classe `Zend_Pdf_Color_Html`:

```php
<?php
$color1 = new Zend_Pdf_Color_Html('#3366FF');
$color2 = new Zend_Pdf_Color_Html('silver');
$color3 = new Zend_Pdf_Color_Html('forestgreen');
```

29.4.3. Desenho de Formas.

Todas as operações de desenho podem ser feitas em um contexto de página PDF.

A classe `Zend_Pdf_Page` fornece um conjunto de primitivas de desenho:

```php
<?php
/**
 * Desenha uma linha de x1,y1 a x2,y2.
 *
 * @param float $x1
 * @param float $y1
 * @param float $x2
 * @param float $y2
 */
public function drawLine($x1, $y1, $x2, $y2);
```

```php
<?php
/**
 * Desenha um retângulo.
 *
 * Tipos de preenchimento:
 * Zend_Pdf_Page::SHAPE_DRAW_FILL_AND_STROKE - preenche o retângulo e a borda
(padrão)
 * Zend_Pdf_Page::SHAPE_DRAW_STROKE      - contorna o retângulo
 * Zend_Pdf_Page::SHAPE_DRAW_FILL        - preenche o retângulo
 *
 * @param float $x1
 * @param float $y1
 * @param float $x2
 * @param float $y2
 * @param integer $fillType
 */
public function drawRectangle($x1, $y1, $x2, $y2,
$fillType = Zend_Pdf_Page::SHAPE_DRAW_FILL_AND_STROKE);
```

```php
<?php
/**
 * Desenha um polígono.
 *
 * Se $fillType é Zend_Pdf_Page::SHAPE_DRAW_FILL_AND_STROKE ou
Zend_Pdf_Page::SHAPE_DRAW_FILL,
 * então o polígino é automaticamente fechado.
 * Veja detalhadamente a descrição destes métodos na documentação PDF
 * (seção 4.4.2 Operadores de Pintura de Caminho, Preenchimento)
 *
 * @param array $x  - vetor de float (as coordenadas X dos vértices)
 * @param array $y  - vetor de float (as coordenadas Y dos vértices)
 * @param integer $fillType
 * @param integer $fillMethod
 */
public function drawPolygon($x, $y,
$fillType = Zend_Pdf_Page::SHAPE_DRAW_FILL_AND_STROKE,
$fillMethod = Zend_Pdf_Page::FILL_METHOD_NON_ZERO_WINDING);
```

```php
<?php
/**
 * Desenha um círculo centrado em x, y com um raio radius.
 *
 * Ângulos são especificados em radianos
 *
 * Assinaturas do método:
 * drawCircle($x, $y, $radius);
 * drawCircle($x, $y, $radius, $fillType);
 * drawCircle($x, $y, $radius, $startAngle, $endAngle);
 * drawCircle($x, $y, $radius, $startAngle, $endAngle, $fillType);
 *
 *
 * Isso não é realmente um círculo, porque PDF suporta somente curvas de Bezier.
 * Mas é uma boa aproximação.
 * Difere de um círculo real em um máximo de 0.00026 medidas de raio
 * (nos ângulos PI/8, 3*PI/8, 5*PI/8, 7*PI/8, 9*PI/8, 11*PI/8, 13*PI/8 e 15*PI/8
).
 * Em 0, PI/4, PI/2, 3*PI/4, PI, 5*PI/4, 3*PI/2 e 7*PI/4 é exatamente uma
tangente a um círculo.
 *
 * @param float $x
 * @param float $y
 * @param float $radius
 * @param mixed $param4
 * @param mixed $param5
 * @param mixed $param6
 */
public function drawCircle($x, $y, $radius, $param4 = null, $param5 = null,
$param6 = null);
```

```php
<?php
/**
 * Desenha uma elipse dentro de um retângulo especificado.
 *
 * Assinaturas do métodos:
 * drawEllipse($x1, $y1, $x2, $y2);
```

```
 * drawEllipse($x1, $y1, $x2, $y2, $fillType);
 * drawEllipse($x1, $y1, $x2, $y2, $startAngle, $endAngle);
 * drawEllipse($x1, $y1, $x2, $y2, $startAngle, $endAngle, $fillType);
 *
 * Ângulos são especificados em radianos
 *
 * @param float $x1
 * @param float $y1
 * @param float $x2
 * @param float $y2
 * @param mixed $param5
 * @param mixed $param6
 * @param mixed $param7
 */
public function drawEllipse($x1, $y1, $x2, $y2, $param5 = null, $param6 = null,
$param7 = null);
```

29.4.4. Desenho de Texto.

Operações de pintura de texto também existem no contexto de uma página PDF. Você pode pintar uma simples linha de texto em qualquer posição na página fornecendo as coordenadas x e y da linha base. O tipo e tamanho de fontes atuais são usados para operações de pintura de texto (veja descrição detalhada abaixo).

```
<?php
/**
 * Desenha uma linha de texto na posição especificada.
 *
 * @param string $text
 * @param float $x
 * @param float $y
 * @param string $charEncoding (opcional) Codificação de caracteres do texto
fonte.
 *   Padroniza para a localidade atual.
 * @throws Zend_Pdf_Exception
 */
public function drawText($text, $x, $y, $charEncoding = '');
```

Example 29.4. Draw a string on the page.

```
<?php
...
$pdfPage->drawText('Alô, Mundo!', 72, 720);
...
```

Por definição, tipos string de texto são interpretados usando o método de codificação de caracteres da localidade atual. Se você tem um tipo string que usa um método de codificação diferente (tal como um tipo string UTF-8 lido de um arquivo no disco, ou um tipo string MacRoman obtido de um banco de dados legado), você pode indicar a codificação de caracteres em tempo de desenho e Zend_Pdf irá manipular a conversão para você. Você pode forneceder strings de fonte em qualquer método de codificação suportado pela função iconv() do PHP:

Exemplo 29.5. Desenho de um string codificado em UTF-8 em uma página.

```
<?php
...
// Lê um string codificado em UTF-8 do disco
$unicodeString = fread($fp, 1024);
// Desenha o string na página
$pdfPage->drawText($unicodeString, 72, 720, 'UTF-8');
...
```

29.4.5. Usando fontes.

`Zend_Pdf_Page::drawText()` usa o tipo e tamanho de fonte da página atual, que são configurados com o método `Zend_Pdf_Page::setFont()`:

```
<?php
/**
 * Configura a fonte atual.
 *
 * @param Zend_Pdf_Resource_Font $font
 * @param float $fontSize
 */
public function setFont(Zend_Pdf_Resource_Font $font, $fontSize);
```

Documentos PDF suportam fontes PostScript Type 1 e TrueType, assim como dois tipos PDF especializados, Type 3 e fontes compostas. Há também 14 fontes padrão Type 1 internas em todo visualizador PDF: Courier (4 estilos), Helvetica (4 estilos), Times (4 estilos), Symbol, e Zapf Dingbats.

Zend_Pdf atualmente suporta as 14 fontes padrão PDF assim como suas próprias fontes TrueType customizadas. Objetos Font são obtidos por meio de dois métodos de fabricação: `Zend_Pdf_Font::fontWithName($fontName)` para as 14 fontes padrão PDF ou `Zend_Pdf_Font::fontWithPath($filePath)` para fontes customizadas.

Exemplo 29.6. Cria uma fonte padrão.

```
<?php
...
// Cria nova fonte
$font = Zend_Pdf_Font::fontWithName(Zend_Pdf_Font::FONT_HELVETICA);
// Aplica fonte
$pdfPage->setFont($font, 36);
...
```

Constantes para os 14 nomes de fonte padrão PDF são definidas na classe `Zend_Pdf_Font`:

- Zend_Pdf_Font::FONT_COURIER
- Zend_Pdf_Font::FONT_COURIER_BOLD

- Zend_Pdf_Font::FONT_COURIER_ITALIC
- Zend_Pdf_Font::FONT_COURIER_BOLD_ITALIC
- Zend_Pdf_Font::FONT_TIMES
- Zend_Pdf_Font::FONT_TIMES_BOLD
- Zend_Pdf_Font::FONT_TIMES_ITALIC
- Zend_Pdf_Font::FONT_TIMES_BOLD_ITALIC
- Zend_Pdf_Font::FONT_HELVETICA
- Zend_Pdf_Font::FONT_HELVETICA_BOLD
- Zend_Pdf_Font::FONT_HELVETICA_ITALIC
- Zend_Pdf_Font::FONT_HELVETICA_BOLD_ITALIC
- Zend_Pdf_Font::FONT_SYMBOL
- Zend_Pdf_Font::FONT_ZAPFDINGBATS

Você pode usar qualquer fonte TrueType individual (que normalmente tem uma extensão '.ttf') ou uma fonte OpenType (extensão '.otf') se ela contém esboços TrueType. Atualmente não suportado, mas planejado para uma futura atualização estão os arquivos .dfont do Mac OS X e Microsoft TrueType Collection (extensão '.ttc').

Para usar uma fonte TrueType, você deve fornecer o caminho completo para o programa da fonte. Se a fonte não puder ser lida por alguma razão, ou se não é uma fonte TrueType font, o método de fabricação irá lançar uma exceção:

Exemplo 29.7. Cria uma fonte TrueType.

```
<?php
...
// Cria uma nova fonte
$goodDogCoolFont = Zend_Pdf_Font::fontWithPath('/path/to/GOODDC__.TTF');
// Aplica fonte
$pdfPage->setFont($goodDogCoolFont, 36);
...
```

Por definição, fontes customizadas serão embutidas no documento PDF resultante. Isso permite ao recipientes ver a página como planejado, mesmo que elas não tenham as fontes apropriadas instaladas em seu sistema. Se você está preocupado com o tamanho do arquivo, você pode requisitar que o programa da fonte não seja embutido passando a opção "do not embed' para o método de fabricação:

Exemplo 29.8. Cria uma fonte TrueType, mas não a embute no documento PDF.

```
<?php
...
// Cria a nova fonte
$goodDogCoolFont = Zend_Pdf_Font::fontWithPath('/path/to/GOODDC__.TTF',
Zend_Pdf_Font::EMBED_DONT_EMBED);
// Aplica a fonte
$pdfPage->setFont($goodDogCoolFont, 36);
...
```

Se o programa da fonte não for embutido mas o recipiente do arquivo PDF tem a fonte instalada em seu sistema, eles verão o documento como planejado. Se eles não tem a fonte correta instalada, a aplicação visualizadora de PDF irá fazer o melhor para sintetizar uma substituição.

Algumas fontes tem regras de licenciamento muito específicas que evitam que elas sejam embutidas em documentos PDF. Assim, você não será pego desprevenido por isso, se tentar usar uma fonte que não pode ser embutida, o método de fabricação irá lançar uma exceção.

Você pode ainda usar estas fontes, mas você deve ou passar a flag não embutida como descrito acima, ou você pode simplesmente suprimir a exceção:

Exemplo 29.9. Não lança uma exceção para fontes que não podem ser embutidas.

```
<?php
...
$font = Zend_Pdf_Font::fontWithPath('/path/to/unEmbeddableFont.ttf',
Zend_Pdf_Font::EMBED_SUPPRESS_EMBED_EXCEPTION);
...
```

Essa técnica de supressão é preferida se você permite a um usuário final escolher suas próprias fontes. Fontes que podem ser embutidas em documentos PDF, serão; aquelas que não puderem, não serão.

Programas de fonte podem ser realmente grandes, alguns alcançando décimos de megabytes. Por definição, todas as fontes embutidas são comprimidas usando o esquema de compressão Flate, resultando em uma economia de espaço de 50% em média. Se, por alguma razão, você não quiser comprimir o programa de fonte, você pode desabilitá-lo com uma opção:

Exemplo 29.10. Não comprime uma fonte embutida.

```
<?php
...
$font = Zend_Pdf_Font::fontWithPath('/path/to/someReallyBigFont.ttf',
Zend_Pdf_Font::EMBED_DONT_COMPRESS);
...
```

Finalmente, quando necessário, você pode combinar as opções embutidas pelo uso do operador de manipulação de bits OR:

Exemplo 29.11. Combinando opções de fonte embutida.

```
<?php
...
$font = Zend_Pdf_Font::fontWithPath($someUserSelectedFontPath,
(Zend_Pdf_Font::EMBED_SUPPRESS_EMBED_EXCEPTION|
Zend_Pdf_Font::EMBED_DONT_COMPRESS));
...
```

29.4.6. Desenho de Imagem.

A classe `Zend_Pdf_Page` fornece o método drawImage() para desenhar imagens:

```
<?php
/**
 * Desenha uma imagem na posição especificada na página.
 *
 * @param Zend_Pdf_Resource_Image $image
 * @param float $x1
 * @param float $y1
 * @param float $x2
 * @param float $y2
 */
public function drawImage(Zend_Pdf_Resource_Image $image, $x1, $y1, $x2, $y2);
```

Objetos imagem devem ser criados com o método `Zend_Pdf_Image::imageWithPath($filePath)` (imagens JPG, PNG e TIFF são suportadas agora):

Exemplo 29.12. Desenho de imagem.

```
<?php
...
// carrega imagem
$image = Zend_Pdf_Image::imageWithPath('my_image.jpg');
$pdfPage->drawImage($image, 100, 100, 400, 300);
...
```

Importante! O suporte a JPEG requer que a extensão PHP GD seja configurada.

Igualmente Importante! O suporte a PNG requer que a extensão ZLIB seja configurada para trabalhar com imagens de canal Alpha.

Procure a documentação PHP para informações detalhadas (http://www.php.net/manual/en/ref.image.php). (http://www.php.net/manual/en/ref.zlib.php).

29.4.7. Estilo de desenho de linha.

O estilo de desenho de linha é definido pelo comprimento da linha, cor da linha e padrão de preenchimento da linha. Todos os parâmetros podem ser associados pelos métodos de classe `Zend_Pdf_Page`:

```
<?php
/** Configura a cor da linha. */
public function setLineColor(Zend_Pdf_Color $color);
/** Configura o comprimento da linha. */
public function setLineWidth(float $width);
/**
 * Configura o padrão de preenchimento da linha.
```

```
     *
     * O padrão é um vetor de floats: array(on_length, off_length, on_length,
    off_length, ...)
     * Fase é o deslocamento do início da linha.
     *
     * @param array $pattern
     * @param array $phase
     */
    public function setLineDashingPattern($pattern, $phase = 0);
```

29.4.8. Estilo de preenchimento.

Os métodos `Zend_Pdf_Page::drawRectangle()`, `Zend_Pdf_Page::drawPoligon()`, `Zend_Pdf_Page::drawCircle()` e `Zend_Pdf_Page::drawEllipse()` levam o argumento `$fillType` como um parâmetro opcional. Ele pode ser:

- Zend_Pdf_Page::SHAPE_DRAW_STROKE – contorna a forma
- Zend_Pdf_Page::SHAPE_DRAW_FILL – somente preenche a forma
- Zend_Pdf_Page::SHAPE_DRAW_FILL_AND_STROKE – preenche e contorna (comportamento padrão)

Os métodos `Zend_Pdf_Page::drawPoligon()` também levam um parâmetro adicional `$fillMethod`:

- Zend_Pdf_Page::FILL_METHOD_NON_ZERO_WINDING (comportamento padrão)

  PDF reference descreve essa regra como o seguinte:

  > A regra do número sinuoso diferente de zero[1] determina se um dado ponto está dentro de um caminho pelo desenho conceitual de um raio daquele ponto ao infinito em qualquer direção e o imediato exame dos lugares onde um segmento do caminho cruza o raio. Começando a contar de 0, a regra adiciona 1 cada vez que um segmento de caminho cruza o raio da esquerda para a direita. Depois de contar todos os cruzamentos, se o resultado é 0 então o ponto está fora do caminho; caso contrário, está dentro. Nota: O método ora descrito não especifica o que fazer se um segmento do caminho coincide ou é tangente ao raio escolhido. Uma vez que a direção do raio é arbitrária, a regra simplesmente escolhe um raio que não encontra tais intersecções problemáticas. Para simples caminhos convexos, o regra do número sinuoso diferente de zero define dentro e fora como seria intuitivamente esperado. Os casos mais interessantes são aqueles envolvendo caminhos complexos ou autointerseccionável como os mostrados na figura 4.10 (em PDF Reference). Para um caminho consistindo de uma estrela de cinco pontas, desenhada com cinco segmentos de linha reta conectados interseccionando um ao outro, a regra considera dentro como a área inteira envolvida pela estrela, incluindo o pentágono no centro. Para um caminho composto de dois círculos concêntricos, as áreas envolvidas por ambos os círculos são consideradas dentro, dado que ambos são desenhados na mesma direção. Se os círculos são desenhados em direções opostas, somente a forma de "rosquinha"[2] entre eles está dentro, de acordo com a regra; o "buraco da

1 Não havia nenhum matemático presente pra me dar uma sugestão melhor
2 Homer ia adorar

rosquinha" está fora.

- Zend_Pdf_Page::FILL_METHOD_EVEN_ODD

  PDF reference descreve essa regra como o seguinte:

  > Uma alternativa para a regra do número sinuoso diferente de zero é a regra par-ímpar. Essa regra determina a "internalidade" de um ponto pelo desenho de um raio daquele ponto em qualquer direção e a simples contagem do número de segmentos de caminho que cruzam o raio, independete da direção. Se esse número é ímpar, o ponto está dentro; se é par, o ponto está fora. Isso produz os mesmo resultados que a regra do número sinuoso diferente de zero para caminhos com formas simples, mas produz resultados diferentes para formas mais complexas. A figura 4.11 (em PDF Reference) mostra os efeitos de aplicar a regra par-ímpar para caminhos complexos. Para a estrela de cinco pontas, a regra considera os pontos triangulares dentro do caminho, mas não o pentágono no centro. Para dois círculos concêntricos, somente a forma de "rosquinha" dentro dos dois círculos será considerada como dentro, independente das direções nas quais os círculos foram desenhados.

29.4.9. Rotações.

Um página PDF pode ser rotacionada antes de aplicar qualquer operação de desenho. Ela pode ser feita pelo método`Zend_Pdf_Page::rotate()`:

```
<?php
/**
 * Rotaciona a págine ao redor do ponto ($x,$y) pelo ângulo especificado (em
radianos).
 *
 * @param float $angle
 */
public function rotate($x, $y, $angle);
```

29.4.10. Grava/restaura o estado gráfico.

A qualquer momento o estado gráfico da página (fonte atual, tamanho da fonte, cor da linha, cor de preenchimento, estilo de linha, rotação de página, área de recorte) podem ser gravado e então restaurado. A operação de gravar adiciona dados a pilha de estado gráfico, enquanto a operação de restaurar recupera-os de lá.

Há dois métodos na classe `Zend_Pdf_Page` para essas operações:

```
<?php
/**
 * Grava o estado gráfico deste página.
 * Isso tira um instantâneo do estilo, posição, area de recorte aplicados
atualmente e
 * qualquer rotação/translação/escalonamento que tenha sido aplicado.
 */
public function saveGS();
/**
 * Restaura o estado gráfico que foi salvo com a última chamada a saveGS().
```

```
 */
public function restoreGS();
```

29.4.11. Área de recorte de desenho.

Tanto o padrão PDF quanto o módulo Zend_Pdf suportam área de recorte. A área de recorte atual limita as regiões da página afetada pelos operadores de desenho. É uma página inteira, inicialmente.

A classe `Zend_Pdf_Page` fornece um conjunto de métodos para operações de recorte.

```
<?php
/**
 * Intersecciona a área de recorte atual com um retângulo.
 *
 * @param float $x1
 * @param float $y1
 * @param float $x2
 * @param float $y2
 */
public function clipRectangle($x1, $y1, $x2, $y2);
```

```
<?php
/**
 * Intersecciona a área de recorte atual com um polígono.
 *
 * @param array $x  - vetor de float (as coordenadas X do vértice)
 * @param array $y  - vetor de float (as coordenadas Y do vértice)
 * @param integer $fillMethod
 */
public function clipPolygon($x, $y, $fillMethod =
Zend_Pdf_Page::FILL_METHOD_NON_ZERO_WINDING);
```

```
<?php
/**
 * Intersecciona a área de recorte atual com um círculo.
 *
 * @param float $x
 * @param float $y
 * @param float $radius
 * @param float $startAngle
 * @param float $endAngle
 */
public function clipCircle($x, $y, $radius, $startAngle = null, $endAngle =
null);
```

```
<?php
/**
 * Intersecciona a área de recorte atual com uma elipse.
 *
 * Assinaturas do método:
 * drawEllipse($x1, $y1, $x2, $y2);
```

```
     * drawEllipse($x1, $y1, $x2, $y2, $startAngle, $endAngle);
     *
     * @todo process special cases with $x2-$x1 == 0 or $y2-$y1 == 0
     *
     * @param float $x1
     * @param float $y1
     * @param float $x2
     * @param float $y2
     * @param float $startAngle
     * @param float $endAngle
     */
    public function clipEllipse($x1, $y1, $x2, $y2, $startAngle = null, $endAngle =
    null);
```

29.4.12. Estilos.

A classe `Zend_Pdf_Style` fornece a funcionalidade de estilos.

Os estilos podem ser usados para armazenar um conjunto de parâmetros de estado gráfico e aplicá-los a uma página PDF por meio de uma operação:

```
<?php
/**
 * Configura o estilo usado para futuras operações de desenho nesta página
 *
 * @param Zend_Pdf_Style $style
 */
public function setStyle(Zend_Pdf_Style $style);
/**
 * Return the style, applied to the page.
 *
 * @return Zend_Pdf_Style|null
 */
public function getStyle();
```

A classe `Zend_Pdf_Style` fornece um conjunto de métodos para configurar ou obter diferentes parâmetros de estado gráfico:

```
<?php
/**
 * Configura a cor da linha.
 *
 * @param Zend_Pdf_Color $color
 */
public function setLineColor(Zend_Pdf_Color $color);
```

```
<?php
/**
 * Obtém a cor da linha.
 *
 * @return Zend_Pdf_Color|null
 */
public function getLineColor();
```

```php
<?php
/**
 * Configura o comprimento da linha.
 *
 * @param float $width
 */
public function setLineWidth($width);
```

```php
<?php
/**
 * Obtém o comprimento da linha.
 *
 * @return float
 */
public function getLineWidth();
```

```php
<?php
/**
 * Configura o padrão de preenchimento da linha.
 *
 * @param array $pattern
 * @param float $phase
 */
public function setLineDashingPattern($pattern, $phase = 0);
```

```php
<?php
/**
 * Obtém o padrão de preenchimento da linha.
 *
 * @return array
 */
public function getLineDashingPattern();
```

```php
<?php
/**
 * Obtém a fase de preenchimento da linha
 *
 * @return float
 */
public function getLineDashingPhase();
```

```php
<?php
/**
 * Configura a cor de preenchimento.
 *
 * @param Zend_Pdf_Color $color
 */
public function setFillColor(Zend_Pdf_Color $color);
```

```php
<?php
/**
 * Obtém a cor de preenchimento.
 *
 * @return Zend_Pdf_Color|null
 */
public function getFillColor();
```

```php
<?php
/**
 * Configura a fonte atual.
 *
 * @param Zend_Pdf_Resource_Font $font
 * @param float $fontSize
 */
public function setFont(Zend_Pdf_Resource_Font $font, $fontSize);
```

```php
<?php
/**
 * Modifica o tamanho de fonte atual
 *
 * @param float $fontSize
 */
public function setFontSize($fontSize);
```

```php
<?php
/**
 * Obtém a fonte atual.
 *
 * @return Zend_Pdf_Resource_Font $font
 */
public function getFont();
```

```php
<?php
/**
 * Obtém o tamanho de fonte atual.
 *
 * @return float $fontSize
 */
public function getFontSize();
```

29.5. Informações do Documento e Metadados.

Um documento PDF pode incluir informações gerais tais como título do documento, autor, e datas de criação e modificação.

Historicamente esta informação é armazenada usando uma estrutura de informação especial. Essa estrutura está disponível para leitura e escrita como um vetor associativo usando a propriedade

pública `properties` dos objetos Zend_Pdf:

```
<?php
$pdf = Zend_Pdf::load($pdfPath);
echo $pdf->properties['Title'] . "\n";
echo $pdf->properties['Author'] . "\n";
$pdf->properties['Title'] = 'New Title.';
$pdf->save($pdfPath);
```

As seguintes chaves são definidas pelo padrão PDF v1.4 (Acrobat 5):

- *Title* - string, opcional, o título do documento.
- *Author* - string, opcional, o nome da pessoa que criou o documento.
- *Subject* - string, opcional, o assunto do documento.
- *Keywords* - string, opcional, palavras-chave associadas com o documento.
- *Creator* - string, opcional, se o documento foi convertido para PDF de outro formato, o nome da aplicação (por exemplo, Adobe FrameMaker®) que criou o documento original do qual foi convertido.
- *Producer* - string, opcional, se o documento foi convertido para PDF de outro formato, o nome da aplicação (por exemplo, Acrobat Distiller) que o converteu para PDF.
- *CreationDate* - string, opcional, a data e a hora em que o documento foi criado, no seguinte formato: "D:YYYYMMDDHHmmSSOHH'mm'", onde:
    - *YYYY* é o ano.
    - *MM* é o mês.
    - *DD* é o dia (01–31).
    - *HH* é a hora (00–23).
    - *mm*is o minuto (00–59).
    - *SS* é o segundo (00–59).
    - *O* é o relacionamento da hora local com a Hora Universal (UT), denotado por um dos caracteres +, −, ou Z (veja abaixo).
    - *HH* seguido por ' é o valor absoluto do intervalo da UT em horas (00–23).
    - *mm* seguido por ' é o valor absoluto do intervalo da UT em minutos (00–59).

  O caráter apóstrofo (') depois de HH e mm é parte da sintaxe. Todos os campos depois do ano são opcionais. (O prefixo D:, embora também opcional, é altamente recomendável.) Os valores padrão para MM e DD são ambos 01; todos os outros campos numéricos são definidos com valores zero. Um sinal de mais (+) como valor do campo O significa que a hora local é maior do que UT, um sinal de menos (−) que a hora local é menor que UT, e a letra Z que a hora local é igual a UT. SE nenhuma informação é especificada, o relacionamento da hora específica com a UT é considerado desconhecido. Se a zona de tempo é conhecida ou não, o restante da data deve ser especificado em hora local.

  Por exemplo, 23 de dezembro de 1998, às 07h52min, na hora padrão do Pacífico dos Estados Unidos, é representada pelo string "D:199812231952−08'00'".
- *ModDate* - string, opcional, a data e a hora em que o documento foi mais recentemente

modificado, no mesmo formato que *CreationDate*.

- *Trapped* - boolean, opcional, indica se o documento tem sido modificado para incluir informação de aprisionamento.
    - *true* – O documento tem sido completamente aprisionado; nenhum outro aprisionamento é necessário.
    - *false* – O documento não sido aprisionado ainda; qualquer aprisionamento desejado ainda deve ser feito.
    - *null* – Ou é desconhecido se o documento tem sido aprisionado ou ele tem sido aprisionado parcialmente; algum aprisionamento adicional pode ainda ser necessário.

Desde PDF v 1.6 que os metadados podem ser armazenados em um documento XML especial anexado ao PDF (XMP - Extensible Metadata Platform).

Esse documento XML pode ser recuperado e anexado ao PDF com os métodos `Zend_Pdf::getMetadata()` e `Zend_Pdf::setMetadata($metadata)`:

```
<?php
$pdf = Zend_Pdf::load($pdfPath);
$metadata = $pdf->getMetadata();
$metadataDOM = new DOMDocument();
$metadataDOM->loadXML($metadata);
$xpath = new DOMXPath($metadataDOM);
$pdfPreffixNamespaceURI = $xpath->query('/rdf:RDF/rdf:Description')->item(0)-
>lookupNamespaceURI('pdf');
$xpath->registerNamespace('pdf', $pdfPreffixNamespaceURI);
$titleNode = $xpath->query('/rdf:RDF/rdf:Description/pdf:Title')->item(0);
$title = $titleNode->nodeValue;
...
$titleNode->nodeValue = 'New title';
$pdf->setMetadata($metadataDOM->saveXML());
$pdf->save($pdfPath);
```

Propriedades comuns do documento são duplicadas na estrutura Info e nos metadados do documento (se apresentados). É responsabilidade da aplicação do usuário agora mantê-las sincronizadas.

29.6. Exemplo de uso do módulo Zend_Pdf.

Esta seção fornece um exemplo de uso do módulo.

Este exemplo pode ser encontrado no arquivo `demos/Zend/Pdf/demo.php`.

Há também o arquivo `test.pdf`, que pode ser usado com esta demo para propósitos de teste.

Exemplo 29.13. Demo de uso do módulo Zend_Pdf.

```
<?php
/**
 * @package Zend_Pdf
 * @subpackage demo
 */
/** Zend_Pdf */
```

```
require_once 'Zend/Pdf.php';
if (!isset($argv[1])) {
    echo "USAGE: php demo.php <pdf_file> [<output_pdf_file>]\n";
    exit;
}
try {
    $pdf = Zend_Pdf::load($argv[1]);
} catch (Zend_Pdf_Exception $e) {
    if ($e->getMessage() == 'Can not open \'' . $argv[1] . '\' file for
reading.') {
      // Cria um novo PDF se o arquivo não existir
      $pdf = new Zend_Pdf();
      if (!isset($argv[2])) {
            // força a completa reescrita do arquivo (ao invés de atualizá-lo)
            $argv[2] = $argv[1];
      }
    } else {
       // Lança uma exceção se não for a a exceção "Can't open file"
       throw $e;
    }
}
//------------------------------------------------------------------------------
------
// Ordem reversa da página
$pdf->pages = array_reverse($pdf->pages);
// Cria um novo estilo
$style = new Zend_Pdf_Style();
$style->setFillColor(new Zend_Pdf_Color_Rgb(0, 0, 0.9));
$style->setLineColor(new Zend_Pdf_Color_GrayScale(0.2));
$style->setLineWidth(3);
$style->setLineDashingPattern(array(3, 2, 3, 4), 1.6);
$style->setFont(Zend_Pdf_Font::fontWithName(Zend_Pdf_Font::FONT_HELVETICA_BOLD),
32);
// Cria um novo objeto de imagem
$stampImage = Zend_Pdf_Image::imageWithPath(dirname(__FILE__) . '/stamp.jpg');
// Marca página como modificada
foreach ($pdf->pages as $page){
    $page->saveGS();
    $page->setStyle($style);
    $page->rotate(0, 0, M_PI_2/3);
    $page->saveGS();
    $page->clipCircle(550, -10, 50);
    $page->drawImage($stampImage, 500, -60, 600, 40);
    $page->restoreGS();
    $page->drawText('Modificado por Zend Framework!', 150, 0);
    $page->restoreGS();
}
// Adiciona uma nova página gerada pelo objeto Zend_Pdf (a página é anexada ao
documento especificado)
$pdf->pages[] = ($page1 = $pdf->newPage('A4'));
// Adiciona uma nova página gerada pelo objeto Zend_Pdf_Page (a página não é
anexada ao documento)
$pdf->pages[] = ($page2 = new
Zend_Pdf_Page(Zend_Pdf_Page::SIZE_LETTER_LANDSCAPE));
// Cria uma nova fonte
$font = Zend_Pdf_Font::fontWithName(Zend_Pdf_Font::FONT_HELVETICA);
// Aplica a fonte e desenha o texto
$page1->setFont($font, 36);
$page1->drawText('Helvetica 36 text string', 60, 500);
// Usa o objeto fonte para outro página
$page2->setFont($font, 24);
```

```
$page2->drawText('Helvetica 24 text string', 60, 500);
// Usa outra fonte
$page2->setFont(Zend_Pdf_Font::fontWithName(Zend_Pdf_Font::FONT_TIMES_ROMAN),
32);
$page2->drawText('Times-Roman 32 text string', 60, 450);
// Desenha um retângulo
$page2->setFillColor(new Zend_Pdf_Color_GrayScale(0.8));
$page2->setLineColor(new Zend_Pdf_Color_GrayScale(0.2));
$page2->setLineDashingPattern(array(3, 2, 3, 4), 1.6);
$page2->drawRectangle(60, 400, 400, 350);
// Desenha um círculo
$page2->setLineDashingPattern(Zend_Pdf_Page::LINE_DASHING_SOLID);
$page2->setFillColor(new Zend_Pdf_Color_Rgb(1, 0, 0));
$page2->drawCircle(85, 375, 25);
// Desenha setores
$page2->drawCircle(200, 375, 25, 2*M_PI/3, -M_PI/6);
$page2->setFillColor(new Zend_Pdf_Color_Cmyk(1, 0, 0, 0));
$page2->drawCircle(200, 375, 25, M_PI/6, 2*M_PI/3);
$page2->setFillColor(new Zend_Pdf_Color_Rgb(1, 1, 0));
$page2->drawCircle(200, 375, 25, -M_PI/6, M_PI/6);
// Desenha uma elipse
$page2->setFillColor(new Zend_Pdf_Color_Rgb(1, 0, 0));
$page2->drawEllipse(250, 400, 400, 350);
$page2->setFillColor(new Zend_Pdf_Color_Cmyk(1, 0, 0, 0));
$page2->drawEllipse(250, 400, 400, 350, M_PI/6, 2*M_PI/3);
$page2->setFillColor(new Zend_Pdf_Color_Rgb(1, 1 , 0));
$page2->drawEllipse(250, 400, 400, 350, -M_PI/6, M_PI/6);
// Desenha e preecnhe um polígono
$page2->setFillColor(new Zend_Pdf_Color_Rgb(1, 0 ,1));
$x = array();
$y = array();
for ($count = 0; $count < 8; $count++) {
    $x[] = 140 + 25*cos(3*M_PI_4*$count);
    $y[] = 375 + 25*sin(3*M_PI_4*$count);
}
$page2->drawPolygon($x, $y,
                    Zend_Pdf_Page::SHAPE_DRAW_FILL_AND_STROKE,
                    Zend_Pdf_Page::FILL_METHOD_EVEN_ODD);
// Desenha uma linha
$page2->setLineWidth(0.5);
$page2->drawLine(60, 375, 400, 375);
//--------------------------------------------------------------------------------
------
if (isset($argv[2])) {
   $pdf->save($argv[2]);
} else {
   $pdf->save($argv[1], true /* atualiza */);
}
```