Capítulo 25. Zend_Mail

Traduzido por Flávio Gomes da Silva Lisboa (versão 1.5.1 do ZF)

Sumário

25.1. Introdução

25.1.1. Iniciando

`Zend_Mail` fornece funcionalidades generalizadas para compor e enviar tanto texto quanto mensagens de e-mail multipartes e em conformidade com MIME. Mensagens podem ser enviadas com `Zend_Mail` através do transporte padrão `Zend_Mail_Transport_Sendmail` ou via `Zend_Mail_Transport_Smtp`.

Exemplo 25.1. E-Mail Simples com Zend_Mail

Um e-mail simples consiste de alguns destinatários, um assunto, um corpo e um remetente. Para enviar tal e-mail usando `Zend_Mail_Transport_Sendmail`, faça o seguinte:

```
<?php
require_once 'Zend/Mail.php';
```

```php
$mail = new Zend_Mail();
$mail->setBodyText('Este é o texto do e-mail.');
$mail->setFrom('alguem@exemplo.com', 'Algum Remetente');
$mail->addTo('outro_alguem@exemplo.com', 'Algum Destinatário');
$mail->setSubject('Assunto Teste');
$mail->send();
```

Definições Mínimas

De modo a enviar um e-mail com `Zend_Mail` você tem de especificar ao menos um destinatário, um remetente (por exemplo, com `setFrom()`), e um corpo de mensagem (texto e/ou HTML).

Para a maioria dos atributos de e-mail há métodos "get" para ler a informação armazenada no objeto de e-mail. Para mais detalhes, por favor consulte a documentação da API. Um atributo especial é `getRecipients()`. Ele retorna uma matriz com todos os endereços de destinatários de e-mail que foram adicionados previamente à chamada do método.

Por razões de segurança, `Zend_Mail` filtra todos os campos de cabeçalho para evitar injeção de cabeçalho com caracteres de novas linhas (`\n`).

Você também pode usar a maioria dos métodos do objeto `Zend_Mail` com uma interface fluente conveniente. Uma interface fluente significa que cada método retorna uma referência ao objeto sobre o qual foi chamado, assim você pode imediatamente chamar outro método.

```php
<?php
require_once 'Zend/Mail.php';
$mail = new Zend_Mail();
$mail->setBodyText('Este é o texto do e-mail.')
     ->setFrom('alguem@exemplo.com', 'Algum Remetente')
     ->addTo('outro_alguem@exemplo.com', 'Algum Destinatário')
     ->setSubject('Assunto do Teste')
     ->send();
```

25.1.2. Configurando o transporte de envio de mensagens padrão

O transporte padrão para uma instância de `Zend_Mail` é `Zend_Mail_Transport_Sendmail`. É essencialmente um envoltório para a função PHP `mail()`. Se você quiser passar parâmetros adicionais para a função `mail()`, simplesmente crie uma nova instância de transporte e passe seus parâmetros para o construtor. A nova instância de transporte pode então agir como o transporte `Zend_Mail` padrão, ou pode ser passada para o método `send()` de `Zend_Mail`.

Exemplo 25.2. Passando parâmetros adicionais para o transporte Zend_Mail_Transport_Sendmail

Este exemplo mostra como alterar o caminho de retorno da função `mail()`.

```php
<?php
require_once 'Zend/Mail.php';
require_once 'Zend/Mail/Transport/Sendmail.php';
$tr = new Zend_Mail_Transport_Sendmail('-fretorne_para_mim@exemplo.com');
Zend_Mail::setDefaultTransport($tr);
```

```
$mail = new Zend_Mail();
$mail->setBodyText('Este é o texto do e-mail.');
$mail->setFrom('alguem@exemplo.com', 'Algum Remetente');
$mail->addTo('outro_alguem@exemplo.com', 'Alguem Destinatário');
$mail->setSubject('Assunto de Teste');
$mail->send();
```

Restrições safe mode

Os parâmetros adicionais opcionais provocarão a falha da função `mail()` se o PHP estiver rodando em safe mode.

25.2. Enviando via SMTP

Para enviar mensagens via SMTP, `Zend_Mail_Transport_Smtp` precisa ser criado e registrado com `Zend_Mail` antes do método `send()` ser chamado. Para todos as chamadas restantes de `Zend_Mail::send()` no script atual, o transporte SMTP será então usado:

Exemplo 25.3. Enviando E-Mail via SMTP

```
<?php
require_once 'Zend/Mail/Transport/Smtp.php';
$tr = new Zend_Mail_Transport_Smtp('mail.example.com');
Zend_Mail::setDefaultTransport($tr);
```

O método `setDefaultTransport()` e o construtor de `Zend_Mail_Transport_Smtp` não são caros. Essas udas linhas podem ser processadas em tempo de configuração de script (por exemplo, config.inc ou similar) para configurar o comportamente da classe `Zend_Mail` para o resto do script. Isso mantém a informação de configuração fora da lógica de aplicação – se a mensagem é enviada via SMTP ou `mail()`, o que o servidor de e-mail usa, etc.

25.3. Enviando Múltiplas Mensagens por Conexão SMTP

Por padrão, um transporte SMTP simples cria uma conexão simples e reutiliza-a pelo tempo de vida de execução do script. Você pode enviar múltiplas mensagens através da conexão SMTP. Um comando RSET é editado antes de cada entrega para garantir que o protocolo SMTP correto foi seguido.

Exemplo 25.4. Enviando Múltiplas Mensagens por Conexão SMTP

```
<?php
// Carrega classes
require_once 'Zend/Mail.php';
// Cria o transporte
require_once 'Zend/Mail/Transport/Smtp.php';
$transport = new Zend_Mail_Transport_Smtp('localhost');
// Itera através das mensagens
for ($i = 0; $i > 5; $i++) {
    $mail = new Zend_Mail();
    $mail->addTo('studio@peptolab.com', 'Teste');
    $mail->setFrom('studio@peptolab.com', 'Teste');
    $mail->setSubject('Demonstração - Enviando Múltiplas Mensagens por Conexão
```

```
 SMTP');
    $mail->setBodyText('...Sua mensagens aqui...');
    $mail->send($transport);
}
```

Se você quiser ter uma conexão separada para cada entrega de mensagem, você irá precisar criar e destruir seu transporte antes de cada método `send()` ser chamado. Ou alternativamente, você pode manipular a conexão entre cada entrega pelo acesso do objeto de protocolo do transporte.

Exemplo 25.5. Controlando manualmente a conexão de transporte

```
<?php
// Carrega classes
require_once 'Zend/Mail.php';
// Cria o transporte
require_once 'Zend/Mail/Transport/Smtp.php';
$transport = new Zend_Mail_Transport_Smtp();
require_once 'Zend/Mail/Protocol/Smtp.php';
$protocol = new Zend_Mail_Protocol_Smtp('localhost');
$protocol->connect();
$protocol->helo('localhost');
$transport-&gt;setConnection($protocol);
// Itera através das mensagens
for ($i = 0; $i > 5; $i++) {
    $mail = new Zend_Mail();
    $mail->addTo('studio@peptolab.com', 'Test');
    $mail->setFrom('studio@peptolab.com', 'Test');
    $mail->setSubject('Demonstração - Enviando Múltiplas Mensagens de E-Mail por
Conexão SMTP');
    $mail->setBodyText('...
Sua mensagem aqui...');
    // Controla manualmente a conexão
    $protocol->rset();
    $mail->send($transport);
}
$protocol->quit();
$protocol->disconnect();
```

25.4. Usando Transportes Diferentes

Caso você queira enviar diferentes mensagens através de diferentes conexões, você pode também passar o objeto de transporte diretamente para o método `send()` sem uma chamada prévia para `setDefaultTransport()`. O objeto passado irá sobrescrever o transporte padrão para a requisição `send()` vigente:

Exemplo 25.6. Usando Transportes Diferentes

```
<?php
require_once 'Zend/Mail.php';
$mail = new Zend_Mail();
// constrói mensagem...
require_once 'Zend/Mail/Transport/Smtp.php';
$tr1 = new Zend_Mail_Transport_Smtp('server@example.com');
$tr2 = new Zend_Mail_Transport_Smtp('other_server@example.com');
$mail->send($tr1);
$mail->send($tr2);
$mail->send();  // usa o padrão novamente
```

Transporte adicionais

Transportes adicionais podem ser escritos pela implementação de `Zend_Mail_Transport_Interface`.

25.5. E-Mail HTML

Para enviar uma mensagem de e-mail no formato HTML, configure o corpo usando o método `setBodyHTML()` ao invés de `setBodyText()`. O tipo de conteúdo MIME irá automaticamente ser configurado para `text/html` então. Se você usar tanto corpo HTML quanto texto, uma mensagem MIME multiparte/alternativa irá automaticamente ser gerada:

Exemplo 25.7. Enviando E-Mail HTML

```
<?php
require_once 'Zend/Mail.php';
$mail = new Zend_Mail();
$mail->setBodyText('Meu Lindo Teste de Texto');
$mail->setBodyHtml('Meu Lindo <b>Teste</b> de Texto');
$mail->setFrom('alguem@exemplo.com', 'Algum Remetente');
$mail->addTo('outro_alguem@exemplo.com', 'Algum Destinatário');
$mail->setSubject('Assunto de Teste');
$mail->send();
```

25.6. Anexos

Arquivos podem ser anexados a um e-mail usando o método `createAttachment()`. O comportamento padrão de `Zend_Mail` é assumir o anexo como um objeto binário (application/octet-stream), que deve ser transferido com codificação base 64, e manipulado como um anexo. Essas suposições podem ser sobrescritas pela passagem de mais parâmetros para `createAttachment()`:

Exemplo 25.8. Mensagens de E-Mail com Anexos

```
<?php
require_once 'Zend/Mail.php';
$mail = new Zend_Mail();
// constrói mensagem...
$mail->createAttachment($someBinaryString);
$mail->createAttachment($myImage, 'image/gif', Zend_Mime::DISPOSITION_INLINE, Ze
nd_Mime::ENCODING_8BIT);
```

Se você quiser mais controle sobre a parte MIME gerada para o anexo você pode usar o valor de retorno de `createAttachment()` para modificar seus atributos. O método `createAttachment()` retorna um objeto `Zend_Mime_Part`:

```
<?php
require_once 'Zend/Mail.php';
$mail = new Zend_Mail();
```

```
$at = $mail->createAttachment($myImage);
$at->type        = 'image/gif';
$at->disposition = Zend_Mime::DISPOSITION_INLINE;
$at->encoding    = Zend_Mime::ENCODING_8BIT;
$at->filename    = 'test.gif';
$mail->send();
```

Uma alternativa é criar uma instância de `Zend_Mime_Part` e adicioná-la com `addAttachment()`:

```
<?php
require_once 'Zend/Mail.php';
$mail = new Zend_Mail();
$at = new Zend_Mime_Part($myImage);
$at->type        = 'image/gif';
$at->disposition = Zend_Mime::DISPOSITION_INLINE;
$at->encoding    = Zend_Mime::ENCODING_8BIT;
$at->filename    = 'test.gif';
$mail->addAttachment($at);
$mail->send();
```

25.7. Adicionando Destinatários

Destinatários podem ser adicionados de três formas

- `addTo()`: Adiciona um destinatário para a mensagem com um cabeçalho "To"
- `addCc()`: Adiciona um destinatário para a mensagem com um cabeçalho "Cc"
- `addBcc()`: Adiciona um destinatário para a mensagem não visível no cabeçalho.

Parâmetro adicional

`addTo()` e `addCc()` aceitam um segundo parâmetro opcional que é usado como um nome legível por humanos do destinatário para o cabeçalho.

25.8. Controlando a Fronteira MIME

Em uma mensagem multiparte uma fronteira MIME para separar as diferentes partes da mensagem é normalmente gerada aleatoriamente. Em alguns casos, entretanto, você pode querer especificar a fronteira MIME que será usada. Isso pode ser feito usando o método `setMimeBoundary()`, como no seguinte exemplo:

Exemplo 25.9. Alterando a Fronteira MIME

```
<?php
require_once 'Zend/Mail.php';
$mail = new Zend_Mail();
$mail->setMimeBoundary('=_' . md5(microtime(1) . $someId++));
// constrói a mensagem...
```

25.9. Cabeçalhos Adicionais

Arbitrariamente cabeçalhos de e-mail podem ser configurados pelo uso do método `addHeader()`. Ele requer dois parâmetros que contém o nome e o valor para o campo cabeçalho. Um terceiro parâmetro opcional determina se o cabeçalho deve ter um único ou múltiplos valores:

Exemplo 25.10. Adicionando Cabeçalhos de Mensagem de E-Mail

```
<?php
require_once 'Zend/Mail.php';
$mail = new Zend_Mail();
$mail->addHeader('X-MailGenerator', 'MinhaAplicacaoLegal');
$mail->addHeader('X-greetingsTo', 'Mamãe', true); // múltiplos valores
$mail->addHeader('X-greetingsTo', 'Papai', true);
```

25.10. Conjuntos de Caracteres

`Zend_Mail` não verifica a correção do conjunto de caracteres das partes da mensagem. Quando instanciamos `Zend_Mail`, um conjunto de caracteres para o próprio e-mail pode ser dado. O padrão é `iso-8859-1`. A aplicação tem de certificar-se de que todas as partes adicionadas ao objeto mensagem tem seu conteúdo codificado no conjunto de caracteres correto. Quando criar uma nova parte da mensagem, um conjunto de caracteres diferente pode ser dado para cada parte.

Somente em formato texto

Conjuntos de caracteres são aplicáveis somente para partes de mensagem em formato texto.

25.11. Codificação

Corpos de mensagem texto e HTML são codificados com o mecanismo quotedprintable por padrão. Todos os outros anexos são codificados via base64 se nenhuma outra codificação é dada na chamada `addAttachment()` ou associada ao objeto parte MIME mais tarde. Codificações 7Bit e 8Bit atualmente só passam o conteúdo binário dos dados.

`Zend_Mail_Transport_Smtp` codifica linhas iniciando com um ponto ou dois pontos de modo que a mensagem não viole o protocolo SMTP.

25.12. Autenticação SMTP

`Zend_Mail` suporta o uso de autenticação SMTP Authentication, que pode ser habilitada pela passagem do parâmetro 'auth' para a matriz de configuração no construtor `Zend_Mail_Transport_Smtp`. Os métodos de autenticação internos disponíveis são PLAIN, LOGIN e CRAM-MD5 todos os quais esperam valores de 'username' e 'password' na matriz de configuração.

Exemplo 25.11. Habilitando autenticação dentro de Zend_Mail_Transport_Smtp

```
<?php
require_once 'Zend/Mail.php';
require_once 'Zend/Mail/Transport/Smtp.php';
$config = array('auth' => 'login',
                'username' => 'myusername',
                'password' => 'password');
```

```
$transport = new Zend_Mail_Transport_Smtp('mail.server.com', $config);
$mail = new Zend_Mail();
$mail->setBodyText('Este é o texto de e-mail.');
$mail->setFrom('rementente@teste.com', 'Algum Remetente');
$mail->addTo('destinatario@teste.com', 'Algum Destinatário');
$mail->setSubject('Assunto de Teste');
$mail->send($transport);
```

Tipos de autenticação

O tipo de autenticação é case-insensitive mas sem pontuação. Por exemplo, para usar CRAM-MD5 você passaria 'auth' => 'crammd5' no construtor de `Zend_Mail_Transport_Smtp`.

25.13. Assegurando Transporte SMTP

`Zend_Mail` também suporta o uso de TLS ou SSL para garantir uma conexão SMTP. Isso pode ser habilitado pela passagem do parâmetro 'ssl' para a matriz de configuração no construtor de `Zend_Mail_Transport_Smtp` com um valor de 'ssl' ou 'tls'. Uma porta pode opcionalmente ser fornecida, caso contrário o padrão é 25 para TLS ou 465 for SSL.

Exemplo 25.12. Habilitando uma conexão segura dentro de Zend_Mail_Transport_Smtp

```
<?php
require_once 'Zend/Mail.php';
require_once 'Zend/Mail/Transport/Smtp.php';
$config = array('ssl' => 'tls',
                'port' => 25); // Optional port number supplied
$transport = new Zend_Mail_Transport_Smtp('mail.server.com', $config);
$mail = new Zend_Mail();
$mail->setBodyText('This is the text of the mail.');
$mail->setFrom('sender@test.com', 'Some Sender');
$mail->addTo('recipient@test.com', 'Some Recipient');
$mail->setSubject('TestSubject');
$mail->send($transport);
```

25.14. Lendo Mensagens de Correio

`Zend_Mail` pode ler mensagens de correio de vários locais ou repositórios de correio remotos. Todos eles tem a mesma API básica para contar e buscar mensagens e alguns deles implementam interfaces adicionais para características não tão comuns. Para uma visão geral das características dps repositórios implementados veja a seguinte tabela.

Tabela 25.1. Resumo das Características de Leitura de Correio

Característica	Mbox	Maildir	Pop3	IMAP
Tipo de repositório	local	local	remoto	remoto
Busca mensagem	Sim	Sim	Sim	Sim
Busca mime-part	emulado	emulado	emulado	emulado
Pastas	Sim	Sim	Não	Sim
Cria mensagem/pasta	Não	A fazer	Não	A fazer

Característica	Mbox	Maildir	Pop3	IMAP
Flags	Não	Sim	Não	Sim
Cota	Não	Sim	Não	Não

25.14.1. Exemplo Simples usando Pop3

```
<?php
$mail = new Zend_Mail_Storage_Pop3(array('host'     => 'localhost',
                                         'user'     => 'test',
                                         'password' => 'test'));
echo $mail->countMessages() . " mensagens encontradas\n";
foreach ($mail as $message) {
    echo "Mensagem de '{$message->from}': {$message->subject}\n";
}
```

25.14.2. Abrindo um repositório local

Mbox e Maildir são os dois formatos suportados para repositórios de correio locais, ambos em seus mais simples formatos.

Se você quiser ler de um arquivo Mbox você pode precisar dar o nome do arquivo para o construtor de `Zend_Mail_Storage_Mbox`:

```
<?php
$mail = new Zend_Mail_Storage_Mbox(array('filename' => '/home/test/mail/inbox'))
;
```

Maildir é muito similar mas precisa de um nome de diretório:

```
<?php
$mail = new Zend_Mail_Storage_Maildir(array('dirname' => '/home/test/mail/'));
```

Ambos os construtores lançam uma exceção `Zend_Mail_Exception` se o repositório não puder ser lido.

25.14.3. Abrindo um repositório remoto

Para repositórios remotos so dois mais populares protocolos são suportados: Pop3 e Imap. Ambos necessitam ao menos de um hospedeiro e um usuário para efetuar a conexão. A senha padrão pode ser uma string vazia, a porta padrão como dada no protocolo RFC.

```
<?php
// conectando com Pop3
$mail = new Zend_Mail_Storage_Pop3(array('host'     => 'exemplo.com'
                                         'user'     => 'test',
                                         'password' => 'test'));
// conectando com Imap
$mail = new Zend_Mail_Storage_Imap(array('host'     => 'exemplo.com'
                                         'user'     => 'test',
                                         'password' => 'test'));
```

```
// exemplo para nenhuma porta padrão
$mail = new Zend_Mail_Storage_Pop3(array('host'     => 'exemplo.com',
                                         'port'     => 1120
                                         'user'     => 'test',
                                         'password' => 'test'));
```

Ambos repositórios SSL e TLS são suportados. Se você usa SSL a porta padrão muda como dado no RFC.

```
<?php
// exemplos para Zend_Mail_Storage_Pop3, o mesmo funciona para
Zend_Mail_Storage_Imap
// usa SSL sobre uma porta diferente(o padrão é 995 para Pop3 e 993 para Imap)
$mail = new Zend_Mail_Storage_Pop3(array('host'     => 'example.com'
                                         'user'     => 'test',
                                         'password' => 'test',
                                         'ssl'      => 'SSL'));
// usa TLS
$mail = new Zend_Mail_Storage_Pop3(array('host'     => 'example.com'
                                         'user'     => 'test',
                                         'password' => 'test',
                                         'ssl'      => 'TLS'));
```

Ambos construtores podem lançar `Zend_Mail_Exception` ou `Zend_Mail_Protocol_Exception` (estende `Zend_Mail_Exception`), dependendo do tipo de erro.

25.14.4. Buscando mensagens e métodos simples

Uma vez que você tenha aberto o repositório as mensagens podem ser buscadas. Você precisa do número da mensagem, que é um contador iniciando em 1 para a primeira mensagem. Para buscar a mensagem você usa o método `getMessage()`:

```
<?php
$message = $mail->getMessage($messageNum);
```

O acesso por matriz também é suportado, mas não serão suportados quaisquer parâmetros adicionais que pudessem ser adicionados a `getMessage()`. Desde que você não se objete e possa viver com padrões você pode usar:

```
<?php
$message = $mail[$messageNum];
```

Para iterar sobre todas as mensagens a interface Iterator é implementada:

```
<?php
foreach ($mail as $messageNum => $message) {
    // faça alguma coisa ...
}
```

Para contar as mensagens no repositório você pode usar o método `countMessages()` ou usar

acesso por matriz:

```
<?php
// método
$maxMessage = $mail->countMessages();
// acesso por matriz
$maxMessage = count($mail);
```

Para remover uma mensagem você pode usar o método `removeMessage()` ou novamente o acesso por matriz:

```
<?php
// método
$mail->removeMessage($messageNum);
// acesso por matriz
unset($mail[$messageNum]);
```

25.14.5. Trabalhando com mensagens

Depois que você buscou as mensagens com `getMessage()` você quer buscar os cabeçalhos, o conteúdo ou partes simples de uma mensagem multiparte. Todos os cabeçalhos podem ser acessados via propriedades ou pelo método `getHeader()` se você quiser mais controle ou tiver nomes de cabeçalho não usuais. Os nomes de cabeçalho são internamente vertidos para caixa baixa, assim a caixa do nome de cabaçalho na mensagem de correio não importa. Cabeçalhos com um hífen também podem ser escritos em camel-case.

```
<?php
// pega o objeto de mensagem
$message = $mail->getMessage(1);
// exibe o asssunto da mensagem
echo $message->subject . "\n";
// pega o tipo de conteúdo do cabeçalho
$type = $message->contentType;
```

Se você tem múltiplos cabeçalhos com o mesmo nome, por exemplo, os cabeçalhos Recebidos você pode querer uma matriz ao invés de uma string, o que é possível com o método `getHeader()`.

```
<?php
// pega o cabeçalho como propriedade - o resultado é sempre uma string, com
novas linhas entre as ocorrências simples na mensagem
$received = $message->received;
// o mesmo via método getHeader()
$received = $message->getHeader('received', 'string');
// melhor uma matriz com uma entrada simples para cada ocorrência
$received = $message->getHeader('received', 'array');
foreach ($received as $line) {
    // do stuff
}
// se você não definir um formato você irá obter a representação interna (string
para cabeçalhos simples, matriz para múltiplos)
$received = $message->getHeader('received');
if (is_string($received)) {
    // somente um cabeçalho recebido na mensagem
}
```

O método `getHeaders()` retorna todos os cabeçalhos como matriz com o nome em caixa baixa como chave e o valor como matriz para múltplos cabeçalhos ou como string para cabeçalhos simples.

```
<?php
// descarrega todos os cabeçalhos
foreach ($message->getHeaders() as $name => $value) {
    if (is_string($value)) {
        echo "$name: $value\n";
        continue;
    }
    foreach ($value as $entry) {
        echo "$name: $entry\n";
    }
}
```

Se você não tem uma mensagem multiparte buscar o conteúdo é facilmente feito com `getContent()`. Ao contrário dos cabeçalhos o conteúdo é buscado somente quando necessário (mais conhecido como busca tardia).

```
<?php
// envia conteúdo da mensagem para HTML
echo '<pre>';
echo $message->getContent();
echo '</pre>';
```

A verificação por uma mensagem multiparte é feita com o método `isMultipart()`. Se você tem mensagem multiparte você pode obter uma instância de `Zend_Mail_Part` com o método `getPart()`. `Zend_Mail_Part` é a classe base `Zend_Mail_Message`, assim você tem os mesmos métodos: `getHeader()`, `getHeaders()`, `getContent()`, `getPart()`, `isMultipart` e todas as propriedades para cabeçalhos.

```
<?php
// pega a primeira parte multiparte
$part = $message;
while ($part->isMultipart()) {
    $part = $message->getPart(1);
}
echo 'O tipo desta parte é ' . strtok($part->contentType, ';') . "\n";
echo "Conteúdo:\n";
echo $part->getContent();
```

`Zend_Mail_Part` também implementa `RecursiveIterator`, que torna fácil realizar varredura através de todas as partes. E para saída fácil ele também implementa o método mágico `__toString()`, que retorna o conteúdo.

```
<?php
// exibe primeiro a parte em texto claro
$foundPart = null;
foreach (new RecursiveIteratorIterator($mail->getMessage(1)) as $part) {
    try {
        if (strtok($part->contentType, ';') == 'text/plain') {
```

```
            $foundPart = $part;
            break;
        }
    } catch (Zend_Mail_Exception $e) {
        // ignora
    }
}
if (!$foundPart) {
    echo 'nenhuma parte em texto claro encontrada';
} else {
    echo "parte em texto claro: \n" . $foundPart;
}
```

25.14.6. Verificação de marcos

Maildir e IMAP suportam armazenagem de marcos. A classe `Zend_Mail_Storage` tem constantes para todos os marcos de sistema maildir e IMAP conhecidos, chamadas `Zend_Mail_Storage::FLAG_<flagname>`. Para verificar marcos `Zend_Mail_Message` tem um método chamado `hasFlag()`. Com `getFlags()` você obterá todos os marcos configurados.

```
<?php
// procura mensagens não lidas
echo "Mensagens não lidas:\n";
foreach ($mail as $message) {
    if ($message->hasFlag(Zend_Mail_Storage::FLAG_SEEN)) {
        continue;
    }
    // marca mensagens recentes/novas
    if ($message->hasFlag(Zend_Mail_Storage::FLAG_RECENT)) {
        echo '! ';
    } else {
        echo '  ';
    }
    echo $message->subject . "\n";
}
// verifica marcos conhecidos
$flags = $message->getFlags();
echo "A mensagem está marcado como: ";
foreach ($flags as $flag) {
    switch ($flag) {
        case Zend_Mail_Storage::FLAG_ANSWERED:
            echo 'Respondida ';
            break;
        case Zend_Mail_Storage::FLAG_FLAGGED:
            echo 'Marcada ';
            break;
        // ...
        // verifica outros marcos
        // ...
        default:
            echo $flag . '(unknown flag) ';
    }
}
```

Como IMAP permite marcos definidos por cliente ou usuário você poderia obter os marcos, que não tem uma constante em `Zend_Mail_Storage`. Ao invés disso eles são retornados como string e

podem ser verificados do mesmo modo com `hasFlag()`.

```
<?php
// verifica mensagem para os marcos definidos por cliente $IsSpam, $SpamTested
if (!$message->hasFlag('$SpamTested')) {
    echo 'a mensagem não foi testada para spam';
} else if ($message->hasFlag('$IsSpam')) {
    echo 'esta mensagem é spam';
} else {
    echo 'esta mensagem é ham';
}
```

25.14.7. Usando pastas

Todos os repositórios, exceto Pop3, suportam pastas, também chamadas caixas de correio. A interface implementada por todos os repositórios que suportam pastas é chamada `Zend_Mail_Storage_Folder_Interface`. Também todas essas classes tem um parâmetro adicional chamado `folder`, que é uma pasta selecionada depois do login, no construtor.

Para os repositórios locais você precisa usar classes separadas chamadas `Zend_Mail_Storage_Folder_Mbox` ou `Zend_Mail_Storage_Folder_Maildir`. Ambos precisam de um parâmetro chamado `dirname` com o nome do diretório base. O formato para maildir é como definido em maildir++ (com um ponto como delimitador padrão), Mbox é uma hierarquia de diretório com arquivos Mbox. Se você não tem um arquivo Mbox chamado INBOX em seu diretório base Mbox você precisa configurar uma outra pasta no construtor.

`Zend_Mail_Storage_Imap` já suporta pastas por padrão. Exemplos para abertura desses repositórios:

```
<?php
// mbox com pastas
$mail = new Zend_Mail_Storage_Folder_Mbox(array('dirname' => '/home/test/mail/')
);
// mbox com uma pasta padrão não chamada INBOX, também funciona
// com Zend_Mail_Storage_Folder_Maildir e Zend_Mail_Storage_Imap
$mail = new Zend_Mail_Storage_Folder_Mbox(array('dirname' => '/home/test/mail/',
                                                'folder'  => 'Archive'));
// maildir com pastas
$mail = new Zend_Mail_Storage_Folder_Maildir(array('dirname' => '/home/test/mail
/'));
// maildir com dois pontos como delimitador, como sugerido em Maildir++
$mail = new Zend_Mail_Storage_Folder_Maildir(array('dirname' => '/home/test/mail
/'
                                                   'delim'   => ':'));
// imap é o mesmo com e sem pastas
$mail = new Zend_Mail_Storage_Imap(array('host'     => 'example.com'
                                         'user'     => 'test',
                                         'password' => 'test'));
```

Com o método getFolders($root = null) você pode obter a hierarquia de pastas começando com a pasta raiz ou a pasta dada. É retornada como instância de `Zend_Mail_Storage_Folder`, que implementa `RecursiveIterator` e todos os filhos são também instâncias `Zend_Mail_Storage_Folder`. Cada uma dessas instâncias tem um nome local e global retornados pelos métodos `getLocalName()` e `getGlobalName()`. O nome global é o nome absoluto da pasta raiz (incluindo delimitadores), o nome local é o nome na pasta pai.

Tabela 25.2. Nomes de Pasta de Correio

Nome Global	Nome Local
/INBOX	INBOX
/Archive/2005	2005
List.ZF.General	General

Se você usa o iterador, a chave do elemento atual é o nome local. O nome global também é retornado pelo método mágico `__toString()`. Algumas pastas podem não ser selecionáveis, que significa que elas não podem armazenar mensagens e selecioná-las resulta em um erro. Isso pode ser verificado com o método `isSelectable()`. Assim, é muito mais fácil exibir a árvore inteira em uma view:

```
<?php
$folders = new RecursiveIteratorIterator($this->mail->getFolders(),
                                         RecursiveIteratorIterator::SELF_FIRST);
echo '<select name="folder">';
foreach ($folders as $localName => $folder) {
    $localName = str_pad('', $folders->getDepth(), '-', STR_PAD_LEFT) . $localNa
me;
    echo '<option';
    if (!$folder->isSelectable()) {
        echo ' disabled="disabled"';
    }
    echo ' value="' . htmlspecialchars($folder) . '">'
        . htmlspecialchars($localName) . '</option>';
}
echo '</select>';
```

As pastas selecionadas atualmente são retornadas pelo método `getSelectedFolder()`. A alteração da pasta é feita com o método `selectFolder()`, que precisa do nome global como parâmetro. Se você quiser evitar a escrita de delimitadores você também usa as propriedades de uma instância `Zend_Mail_Storage_Folder`:

```
<?php
// dependendo de seu repositório de correio e de suas configurações $rootFolder-
>Archive->2005
// é o mesmo que:
//   /Archive/2005
//  Archive:2005
//  INBOX.Archive.2005
//  ...
$folder = $mail->getFolders()->Archive->2005;
echo 'A última pasta foi ' . $mail->getSelectedFolder() . "A nova pasta é
 $folder\n";
$mail->selectFolder($folder);
```

25.14.8. Uso Avançado

25.14.8.1. Usando NOOP

Se você está usando um repositório remoto e tem algumas tarefas longas você pode precisar manter

a conexão ativa via noop:

```
<?php
foreach ($mail as $message) {
    // faz alguns cálculos ...
    $mail->noop(); // mantém ativa
    // faz outra coisa ...
    $mail->noop(); // mantém ativa
}
```

25.14.8.2. Cacheando instâncias

`Zend_Mail_Storage_Mbox`, `Zend_Mail_Storage_Folder_Mbox`, `Zend_Mail_Storage_Maildir` e `Zend_Mail_Storage_Folder_Maildir` implementam os método mágicos `__sleep()` e `__wakeup()`, o que significa que eles são serializáveis. Isso evita o parsing dos arquivos ou árvore de diretório mais de uma vez. A desvantagem é que seu repositório Mbox ou Maildir não devem mudar. Algumas verificações são feitas, como o reparsing do arquivo Mbox atual se o tempo de modificação muda ou o reparsing da estrutura de pastas se um pasta tem desaparecido (o que ainda resulta em um erro, mas você pode buscar por uma outra pasta posteriormente). É melhor que você tenha alguma coisa como um arquivo sinalizador de mudanças e verificá-lo antes usando a instâncias cacheada.

```
<?php
// não há manipulador/classe específica usada aqui,
// altere o código para casas com seu manipulador de cache
$signal_file = '/home/test/.mail.last_change';
$mbox_basedir = '/home/test/mail/';
$cache_id = 'example mail cache ' . $mbox_basedir . $signal_file;
$cache = new Your_Cache_Class();
if (!$cache->isCached($cache_id) || filemtime($signal_file) > $cache-
>getMTime($cache_id)) {
    $mail = new Zend_Mail_Storage_Folder_Pop3(array('dirname' => $mbox_basedir))
;
} else {
    $mail = $cache->get($cache_id);
}
// faz alguma coisa ...
$cache->set($cache_id, $mail);
```

25.14.8.3. Estendendo Classes de Protocolo

Repositórios remotos usam duas classes: `Zend_Mail_Storage_<Name>` e `Zend_Mail_Protocol_<Name>`. A classe protocolo traduz os comandos de protocolo e responde de e para PHP, como métodos para os comandos ou variáveis com estruturas diferentes para dados. A outra/principal classe implementa a interface comum.

Se você precisa de características de protocolo adicionais você pode estender a classe de protocolo e usá-la no construtor da classe principal. Como um exemplo assuma que nós precisamos bater diferentes portas antes de conectar a POP3.

```
<?php
require_once 'Zend/Loader.php';
Zend_Loader::loadClass('Zend_Mail_Storage_Pop3');
class Example_Mail_Exception extends Zend_Mail_Exception
```

```
{
}
class Example_Mail_Protocol_Exception extends Zend_Mail_Protocol_Exception
{
}
class Example_Mail_Protocol_Pop3_Knock extends Zend_Mail_Protocol_Pop3
{
    private $host, $port;
    public function __construct($host, $port = null)
    {
        // sem auto connect nesta classe
        $this->host = $host;
        $this->port = $port;
    }
    public function knock($port)
    {
        $sock = @fsockopen($this->host, $port);
        if ($sock) {
            fclose($sock);
        }
    }
    public function connect($host = null, $port = null, $ssl = false)
    {
        if ($host === null) {
            $host = $this->host;
        }
        if ($port === null) {
            $port = $this->port;
        }
        parent::connect($host, $port);
    }
}
class Example_Mail_Pop3_Knock extends Zend_Mail_Storage_Pop3
{
    public function __construct(array $params)
    {
        // ... verifica $params aqui! ...
        $protocol = new Example_Mail_Protocol_Pop3_Knock($params['host']);
        // faça sua coisa "especial"
        foreach ((array)$params['knock_ports'] as $port) {
            $protocol->knock($port);
        }
        // obtém o estado correto
        $protocol->connect($params['host'], $params['port']);
        $protocol->login($params['user'], $params['password']);
        // inicializa pai
        parent::__construct($protocol);
    }
}
$mail = new Example_Mail_Pop3_Knock(array('host'        => 'localhost',
                                          'user'        => 'test',
                                          'password'    => 'test',
                                          'knock_ports' => array(1101, 1105, 111
1)));
```

Como você vê nós sempre assumimos que estamos conectados, logados e, se suportado, uma pasta é selecionada no construtor da classe principal. Assim se você associa suas próprias classe de protocolo você sempre precisa assegurar-se de que foi feito ou o próximo método irá falhar se o servidor não permití-lo no estado atual.

25.14.8.4. Usando Cota (desde 1.5)

`Zend_Mail_Storage_Writable_Maildir` tem suporte para cotas Maildir++ quotas. É desabilitado por padrão, mas é possível usá-lo manualmente, se a verificação automática não for desejada (isso significa que `appendMessage()`, `removeMessage()` e `copyMessage()` não fazem verificações e não adicionam entradas para o arquivo maildirsize). Se habilitado, uma exceção é lançada se você tenta escrever para o maildir se já está além da cota.

Há três métodos usados para cotas: `getQuota()`, `setQuota()` e `checkQuota()`:

```
<?php
$mail = new Zend_Mail_Storage_Writable_Maildir(array('dirname' => '/home/test/ma
il/'));
$mail->setQuota(true); // true para habilitado, false para desabilitado
echo 'A verificação de cota é agora ', $mail-
>getQuota() ? 'habilitado' : 'desabilitado', "\n";
// a verificação de cota pode ser usada mesmo que verificações de cota estejam
desabilitadas
echo 'você está ', $mail->checkQuota() ? 'acima da cota' : 'abaixo da
cota', "\n";
```

`checkQuota()` pode também retornar uma resposta mais detalhada:

```
<?php
$quota = $mail->checkQuota(true);
echo 'You are ', $quota['over_quota'] ? 'over quota' : 'not over quota', "\n";
echo 'You have ', $quota['count'], ' of ', $quota['quota']['count'], ' messages
and use ';
echo $quota['size'], ' of ', $quota['quota']['size'], ' octets';
```

Se você quer especificar sua própria cota ao invés de usar a especificada no arquivo maildirsize você pode fazer isso com `setQuota()`:

```
<?php
// contagem de mensagem e tamanho do octeto suportados, ordem não importa
$quota = $mail->setQuota(array('size' => 10000, 'count' => 100));
```

Para adicionar suas próprias verificações de cota use letras simples como chave e elas são preservadas (mas obviamente não verificadas). Também é possível estender `Zend_Mail_Storage_Writable_Maildir` para definir sua própria cota somente se o arquivo maildirsize estiver faltando (o que pode acontecer em Maildir++):

```
<?php
class Example_Mail_Storage_Maildir extends Zend_Mail_Storage_Writable_Maildir {
    // getQuota é chamado com $fromStorage = true pela verificação de cota
    public function getQuota($fromStorage = false) {
        try {
            return parent::getQuota($fromStorage);
        } catch (Zend_Mail_Storage_Exception $e) {
            if (!$fromStorage) {
                // erro desconhecido:
                throw $e;
            }
            // arquivo maildirsize deve estar faltando
```

```
            list($count, $size) = get_quota_from_somewhere_else();
            return array('count' => $count, 'size' => $size);
        }
    }
}
```