

Integrando o banco de dados ao servidor Web

Para dar liga

De nada adianta ter um banco de dados e um servidor Web se não houver algo para integrá-los. O PHP é a linguagem preferida de muitos webmasters, e atualmente é uma das estrelas do código aberto no mundo.

por **Rúben Lício**

Em 1995, Rasmus Lerdorf achou que seria interessante ter estatísticas geradas de forma dinâmica em seu currículo online. Para isso, criou um sistema de scripts simples e veloz, capaz de realizar somente algumas funções básicas, a fim de atender às suas necessidades. Logo percebeu as possibilidades que isso lhe proporcionava, e começou a aperfeiçoar o script, com diversos recursos. Rasmus necessitava de cada vez mais tempo para adicionar recursos e ajustar os já existentes, quando pensou que seria muito mais fácil se houvesse pessoas colaborando no desenvolvimento. Decidiu abrir o código-fonte do *PHP/FI* (que teve seu nome mudado para *PHP* na versão 3). Desde então, diversas pessoas ao redor do mundo começaram a contribuir e usar essa (então) nova linguagem. A decisão de abrir o código-fonte fez com que o PHP crescesse de forma muito rápida. Hoje em dia, o PHP é uma das linguagens de script mais fáceis de se aprender, e também uma das mais usadas. A evolução do código-fonte pode ser vista em [1], e a história completa em [2].

O PHP, por si só, é apenas um interpretador de scripts, sendo necessário um servidor web para que ele seja visto

no navegador. Neste artigo, usaremos como exemplo o *Apache 2.2* como servidor web e o *MySQL 5.1* como servidor de banco de dados.

Scripts mais seguros

Após seguir as instruções do quadro **Apache mais seguro**, o usuário do Apache será *www*, e os arquivos do servidor web terão permissão de leitura para esse usuário, a fim de serem processa-

dos pelo servidor. Podemos atribuir 444 como permissão para todos os arquivos abaixo de *htdocs*, ou, para maior segurança, mudar o usuário e grupo dos arquivos para *www* e atribuir 400 como permissão a eles.

```
chown www.www /usr/local/apache/htdocs/ -R
chmod 400 /usr/local/apache/htdocs -R
```

Lembre-se que, caso o usuário do Apache não tenha permissão

Apache mais seguro

Por padrão, o Apache rodará sob o usuário *root*, o que não é seguro, pois se alguém conseguir invadir o sistema através do Apache, terá acesso irrestrito ao sistema. Para resolvermos isso, basta adicionarmos um usuário e grupo exclusivos para o *daemon* do Apache:

```
# groupadd www www
# useradd -g www www
```

Depois, altere o seu arquivo *httpd.conf*, onde provavelmente já há um usuário e um grupo ativados. Troque-os pelo usuário e grupo que adicionamos acima desta forma:

```
User www
Group www
```

Pronto, agora tudo que o Apache fizer no seu sistema será como usuário *www*, impedindo assim que ele faça qualquer estrago no sistema.

para ler seus scripts, isso causará um erro na hora de requisitá-los no navegador.

Para mais informações sobre o Apache, a documentação oficial pode ser obtida em [3].

Instalando o PHP 5.1

A última versão do PHP está disponível em [6]. Compilaremos nosso PHP manualmente, usando a versão `bz2` do código-fonte. Descompacte o arquivo baixado com `tar -jxf php-5.1.4.tar.bz2`, e depois entre no diretório criado com `cd php-5.1.4`.

Configure-o para ser um módulo dinâmico do Apache com a opção `--with-apxs2`, e também para poder ser executado por linha de comando, com `--enable-cli`, e ainda para ati-

var o suporte a MySQL e *MySQLi*. A *MySQLi* é uma biblioteca mais recente e mais rápida, e por isso seu uso é recomendado, no lugar da antiga *MySQL*.

```
./configure \
--with-apxs2=/usr/local/apache/bin/apxs \
--enable-cli \
--with-mysql=/usr/local/mysql/bin \
--with-mysqli=/usr/local/mysql/bin/
mysql_config
```

O PHP necessita de um programa chamado *Flex* [7] para ser compilado. Caso a configuração peça, você pode instalá-lo usando apenas `./configure && make && make install`.

Agora que o PHP já está configurado, basta compilarmos e instalarmos:

```
make && make install
```

Para quem usa distribuições baseadas no Debian, é possível instalar tudo com um único comando:

```
apt-get install mysql-client mysql-server
➤ apache2 libapache2-mod-php5 php5
➤ php5-mysql php5-cli
```

Esses pacotes estão em praticamente qualquer repositório do Debian. Todavia, caso você não os encontre, procure em [8].

Apache, esse é o PHP

Por si só, o Apache não interpreta nativamente a extensão `.php`. Temos que fazer isso manualmente. Edite o arquivo `httpd.conf`, adicionando ao final do arquivo a linha:

Instalação e segurança do MySQL 5.1

O *MySQL* 5.1, que está na versão beta no momento da escrita deste artigo, pode ser baixado em [4]. A versão binária se encontra no final da página. Com o arquivo no formato *tarball* em mãos, entre no diretório onde foi baixado o arquivo e execute:

```
# tar -zxf mysql-5.1.11-beta.tar.gz
# cd mysql-5.1.11-beta
# ./configure --prefix=/usr/local/mysql
# make
# make install
```

Utilizamos apenas a opção `--prefix` para definir o local onde os binários serão instalados. Com o *MySQL* instalado, devemos criar o arquivo de configuração:

```
# cp support-files/my-medium.cnf /etc/my.cnf
```

Para que o servidor funcione corretamente, precisamos instalar um banco de dados para configurações dos demais bancos do seu servidor. Esse banco por padrão se chama `mysql`.

```
# /usr/local/mysql/bin/./mysql_install_db
```

Para iniciar o banco de dados para testes, use:

```
# /usr/local/mysql/bin/./mysqld_safe
```

Caso não tenham ocorrido problemas, o *MySQL* deve estar ativo. Para confirmar, use os seguintes comandos:

```
$ mysql -uroot
mysql> show databases;
```

Isso mostrará as bases de dados existentes no seu servidor. É importante que haja um banco com o nome `mysql`: lembre-se que ele é usado pelo próprio *MySQL* para configurar os usuários, entre outras coisas. Se não estiver corretamente instalado, o funcionamento de todo o servidor *MySQL* ficará comprometido. Use `quit` para sair do console do *MySQL*.

O daemon do *MySQL* por padrão roda sob `root`. Devemos começar criando um usuário só para executar o servidor *MySQL*:

```
# groupadd mysql
# useradd -g mysql mysql
```

Com o usuário criado, já se pode iniciar o servidor rodando sobre ele da seguinte forma:

```
# /usr/local/mysql/bin/./mysqld_safe --user=mysql
```

Outra opção é especificar o usuário `mysql` como padrão no arquivo de configuração do *MySQL*. Para isso, só precisamos acrescentar a linha à sessão `[mysqld]` de `/etc/my.cnf`:

```
user=mysql
```

Podemos também tornar os dados físicos mais seguros, mudando o usuário e grupo para `mysql`. Dessa forma, apenas o daemon será capaz de acessar seus dados:

```
# chown -R root.mysql /usr/local/mysql
# chmod -R o-rwx /usr/local/mysql
```

Ainda temos brechas de segurança com relação aos usuários do *MySQL*, já que, por padrão, nem mesmo o usuário `root` possui senha. Entre no prompt do *MySQL* e digite:

```
mysql> DELETE FROM mysql.user WHERE User = '';
mysql> FLUSH PRIVILEGES;
mysql> SELECT Host, User FROM mysql.user;
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('rootlocalhost');
mysql> SET PASSWORD FOR 'root'@'192.168.0.2' = PASSWORD('rootoutropc2');
mysql> FLUSH PRIVILEGES;
```

Note que `root` recebeu senhas diferentes para `localhost` e `192.168.0.2`. O *MySQL* permite que você atribua senhas diferentes para o mesmo usuário, de acordo com a máquina de origem da requisição. Note também que você pode trocar o usuário `root` por qualquer outro nome de sua preferência, tornando mais difícil uma possível tentativa de invasão.

Depois desses comandos executados, tente entrar novamente usando o comando `mysql -uroot`. Você verá que ele vai negar seu acesso. Agora você irá necessitar de senha toda vez que entrar no *MySQL*, como no exemplo:

```
mysql -u root -p rootlocalhost
```

Para maiores informações sobre o *MySQL*, pode-se acessar a documentação oficial em [5].

```
AddType application/x-httpd-php .php
```

Caso você queira, também é possível especificar a extensão `.html` como um script PHP. Entretanto, lembre-se que isso fará com que o Apache passe a interpretação de todos os arquivos *HTML* para o PHP, o que provavelmente será um desperdício de recursos do sistema.

Uma dica adicional é incluir também a extensão `.inc` aos arquivos interpretados pelo PHP, a fim de garantir a segurança para alguns desenvolvedores que usem esses arquivos de inclusão.

Estrutura MVC

O MVC (*Model-View-Control*) é uma forma de organização de código que promove uma separação da estrutura de uma aplicação qualquer em três partes principais. Essas partes são: a regra de negócios (*Model*), a interface da aplicação (*View*) e o controle da execução (*Control*).

Isso significa que cada classe deve se limitar exclusivamente àquilo que se propõe a fazer, mantendo tudo em seu contexto a fim de tornar transparente a localização de cada parte do código. Isso possibilita que se mude uma das camadas do seu sistema sem que as demais parem de funcionar. Além disso, esse recurso

fornece as bases para que o código seja construído por diversas pessoas ao mesmo tempo, sem que elas se preocupem tanto com a comunicação. Vamos ver uma definição mais exata de cada camada:

♦ **View:** É toda a interface do sistema, podendo ser tanto para pessoas quanto para outros programas (API). Exemplificando, um sistema pode ter uma interface por linha de comando, uma por Web e outra por Web-service em XML. Temos então o mesmo sistema com três interfaces diferentes, uma independente da outra, e todas independentes da regra de negócios e do fluxo de execução. Essa camada deve saber como apresentar

Exemplo 1: A classe controladora da operação de login.

```
01 <?php
02
03 class LoginController
04 {
05     private $view = null;
06     private $model = null;
07
08     function __construct($acao)
09     {
10         $this->view = new LoginView();
11         $this->model = new LoginModel();
12
13         if (method_exists($this, $acao))
14             $this->{$acao}();
15         else
16             $this->loginForm();
17     }
18
19     public function validarLogin()
20     {
21         global $_POST;
22
23         $dados = $_POST;
24         $resultadoValidacao = $this->model->validaUsuario($dados);
25         if ($resultadoValidacao) {
26             // Redireciona para pagina interna do sistema
27         } else {
28             $this->loginForm();
29         }
30     }
31
32     public function loginForm()
33     {
34         $this->view->formLogin();
35     }
36
37     public function cadastrarLoginForm()
38     {
39         $this->view->formNovoUsuario();
40     }
41
42     public function alterarLoginForm()
43     {
44         global $_GET;
45
46         $idUserio = $_GET['idu'];
47
48         $dados = $this->model->recuperarUsuario($idUserio);
49
50         $this->view->formEditarUsuario($dados);
51     }
52     public function alterarLogin();
53     public function cadastrarLogin();
54 }
55
56 $lg = new LoginController($_GET['acao']);
57
58 ?>
59
60 <?php
61
62 class LoginModel
63 {
64     private $link = null;
65
66     public function __construct()
67     {
68         $this->link = mysqli_pconnect('localhost','root','rootlocalhost');
69         mysqli_select_db($this->link, 'mvc');
70     }
71
72     public function validaUsuario($dados)
73     {
74         $sql = 'SELECT count(*) as total
75             FROM usuarios
76             WHERE usuario = "'.$dados['usuario'].'"
77             .\' AND senha = "'.$dados['senha'].'"';
78
79         $query = mysqli_query($this->link, $sql);
80         $row = mysqli_num_rows($query);
81
82         if ($row == 1)
83             return true;
84         else
85             return false;
86     }
87
88     public function inserirUsuario($dados);
89     public function alterarUsuario($dados);
90     public function listarUsuarios();
91     public function deletarUsuario($idUserio);
92     public function recuperarUsuario($idUserio);
93 }
94
95 ?>
```

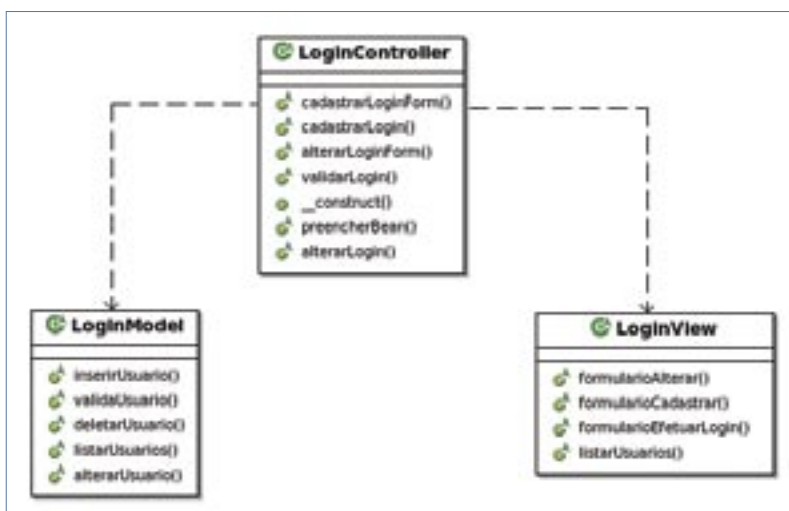


Figura 1 Três camadas distintas para o processamento do login de um usuário, de acordo com a estrutura MVC.

nosso sistema para quem o acessar, nada mais que isso.

▶ **Model:** Define o que o sistema vai fazer e como vai fazer, ou seja, as regras de negócio do sistema. Por exemplo, nosso sistema pode cadastrar, editar ou deletar um registro, gerar um relatório dos últimos dez registros modificados, adicionados ou deletados, contar visualizações de cada registro etc. O sistema fará isso através de um banco de dados. Todo cadastro terá de ser enviado também por e-mail para xxx@site.com.br, junto com um relatório dos últimos visitantes. Podemos ter inúmeros recursos, independentemente da forma como será a interface e do momento em que será executado. Essa camada deve saber somente como executar cada recurso do sistema.

▶ **Control:** faz o controle entre o usuário, as Views e o Model. Por exemplo, nosso usuário acabou de entrar no sistema. O controle é acionado e verifica que a ação pertinente é exibir o formulário de login. O controle não sabe como fazer isso, somente sabe o que tem que fazer, e então chama a View, que sabe como exibir o formulário de login. O usuário tenta fazer um login, e o controle sabe que, quando isso acontece, tem que verificar com o Model se ele foi bem sucedido. Em caso de sucesso, o usuário deve ser redirecionado para a página inicial do sistema; caso negativo, uma tela de erro deve ser exibida. O Control deve saber apenas as possíveis ações a serem executadas, e não como elas serão implementadas. A implementação fica a cargo da View e do Model.

Isso não significa que devemos simplesmente tentar dividir um módulo em três arquivos, ou três classes, ou algo do tipo. A divisão requer no mínimo três camadas pois, sem elas, não teríamos um MVC.

Projetos grandes com PHP e MVC

Na nossa aplicação do [exemplo 1](#), usaremos três classes, divididas em três arquivos. Não será uma aplicação completa. Há muitas camadas que podem ser implementadas, as quais mencionarei posteriormente. Por questões didáticas, não as implementaremos agora. Na [figura 1](#), podemos ver um diagrama das três classes exemplificadas.

Nosso sistema de exemplo servirá para demonstrar como funciona um MVC em PHP. Não me aprofundarei em como funciona o PHP, nem tampouco em como fazer sua programação específica.

O sistema funcionará da seguinte forma: toda vez que for iniciado, será instanciada a classe de controle, e a ação será passada como argumento. O construtor da classe de controle receberá essa ação e tentará achar um método nela mesma referente a essa ação. Caso esse método exista, receberá do construtor o controle; caso não exista, mostraremos o método padrão, que é `loginForm()`. Poderíamos ter disparado um erro ou registrado a ação inválida para que fosse analisada mais tarde.

A divisão de ações foi feita propositalmente dessa forma para demonstrar uma das melhores práticas em programação orientada a objetos, que é a definição de cada método como algo extremamente

específico, sendo preferível que não faça nada além de uma ação absolutamente específica, e não fuja de forma alguma do contexto para o qual se destina.

Isso promove uma reutilização de código muito maior, por evitar métodos muito amplos, que fazem tantas coisas que acabam sendo inutilizáveis em outras situações. Quando um método é decomposto em vários métodos pequenos e específicos, podemos, caso não consigamos reaproveitar o método principal, construir um semelhante que reutilize todos os pequenos métodos, ou boa parte deles. Isso também evita a duplicidade de código, auxiliando assim a manutenção dos scripts.

Como exemplo, vamos examinar nossa classe de controle. Veja que no método `validarLogin()`, quando é retornado que o login é inválido (falso), nós exibimos o formulário novamente, através do método `loginForm()`, que é o mesmo usado quando iniciamos o sistema. Poderíamos ter apenas copiado o conteúdo do método, que aparentemente jamais necessitará de algo mais que isso. Entretanto, da forma como fizemos, caso precisemos mudá-lo por algum motivo (por exemplo, nosso formulário usa um outro método para exibi-lo), podemos trocar somente um lugar, sendo que o restante continuará funcionando normalmente.

Outro exemplo é nossa classe de Model. Suponha que queiramos validar os dados antes de enviá-los para o banco. Construiríamos um método que validasse os dados, e o usaríamos tanto no método `inserirUsuario()` quanto no `alterarUsuario()`. Ambos teriam as mesmas funcionalidades, sem qualquer duplicidade de código.

Outro ponto importante é a forma como foi modelada a controladora. Veja que o construtor recebe a ação ao invés de recuperá-la por `POST` ou `GET`. Isso foi feito justamente para maximizar a reutilização de código! Se tivessem sido usados `POST`, `GET` ou qualquer outra forma direta de recuperar a ação, ficaríamos limitados a essa forma. O jeito utilizado torna a classe transparente ao resto do sistema (ou sistemas), podendo utilizá-la em diversos sistemas diferentes, sem ter que alterar nenhuma linha. Posso ter num sistema uma chamada de ação por `GET`, enquanto em outro a chamada ocorre por um web-service (que é chamado pelo cliente), e num terceiro sistema a ação é definida por um arquivo XML. Qualquer que seja o método específico do sistema, a classe não será alterada. Isso se torna muito importante para a construção de grandes sistemas, pois facilita

enormemente a manutenção. Esse é um dos métodos mais utilizados para criar classes abstratas, pois podemos atualizar uma classe sem interferirmos no funcionamento do resto do sistema.

Note também que existe um método correspondente a cada ação possível, que será chamado de acordo com a ação informada. Isso foi feito para podermos adicionar recursos a essa classe sem termos que alterar o construtor, podendo tratar cada ação de forma inteiramente personalizada.

Além disso tudo, temos o contexto bem explícito. Veja que a controladora não faz absolutamente nada além de delegar as ações pertinentes às classes de Model e View; podemos a qualquer hora trocar o Model, ou o View, sem deixarmos que nossa controladora pare de funcionar.

A classe Model cria uma conexão com o banco de dados ao ser instanciada. Isso serve apenas para demonstrar uma forma de fazer, pois, numa aplicação real, é recomendado que nunca se utilize acesso direto ao banco de dados. Você pode usar alguma classe de abstração de dados pronta como `pear::MDB2` [9].

Talvez você esteja em dúvida a respeito da implementação, na classe `model`, de métodos muito similares aos da controladora, ao invés de fazer a imple-

mentação, diretamente, nela própria, eliminando assim a necessidade de mais uma classe e mais um arquivo. A razão é a facilidade na manutenção e até na criação dessa classe:

- Na criação, por possibilitar o desenvolvimento simultâneo por mais de uma pessoa na controladora, nas Views e no Model; também individualmente em cada uma dessas classes, por manter um contexto menor, tornando mais claro e rápido o desenvolvimento. O Model só terá as regras de negócio, não sendo necessário se preocupar em como será exibida a informação, nem em como será feito o controle do fluxo.

- Na manutenção, por ser muito mais fácil e rápido mexer em arquivos pequenos e, principalmente, porque podemos mudar todas as regras de negócio do sistema, sem alterar absolutamente nada na View e na controladora. Imagine, por exemplo, que nosso cliente não queira mais que os dados de administradores estejam alocados em um banco de dados, e sim num arquivo XML; poderíamos criar uma outra classe `model` específica para XML, e mudar a controladora para chamar a nova classe. Nossa antiga classe continuaria lá e poderia ser utilizada em outro projeto, ou mesmo

nesse, caso o cliente mudasse de idéia novamente. Nessa última hipótese, gastaríamos poucos minutos para voltar tudo para a forma anterior, já que só precisaríamos mudar a controladora para chamar novamente o Model do banco de dados. ■

Mais Informações

- | | |
|-----|--|
| [1] | Museu PHP: http://museum.php.net |
| [2] | História do PHP e projetos relacionados:
http://php.net/history |
| [3] | Documentação do Apache 2.2:
http://httpd.apache.org/docs/2.2/ |
| [4] | MySQL 5.1: http://dev.mysql.com/downloads/mysql/5.1.html |
| [5] | Documentação do MySQL:
http://dev.mysql.com/doc/ |
| [6] | Código-fonte e binários do PHP:
http://www.php.net/downloads.php |
| [7] | Flex: http://flex.sourceforge.net/ |
| [8] | Apt-get: http://apt-get.org |
| [9] | MDB2: http://pear.php.net/package/MDB2 |

Chega de tantos downloads! Adquira já nossos DVDs completos!

DVD SUSE 10.0 Pro
Oferta especial de lançamento!
R\$29,90 + frete



DVD Mandriva 2006
Oferta especial de lançamento!
R\$29,90 + frete

Estreme Gaming
Distribuição completa +
jogos com aceleração 3D!
R\$29,90 + frete



Acesse agora o nosso site www.linuxmagazine.com.br ou peça pelo telefone: 11 2161-5400.