edição #5 - ago/2008

PHP MAGAZINE

A sua revista digital de PHP

Conhecimento Científico
PHP e o conhecimento científico pelo mundo

PHP-GTK
proteja seus códigos

SQL Injection
vulnerabilidades e prevenção

Arquitetura
PHP, MVC e AJAX

PHP
confira sua participação no FISL 9.0

ISSN 1981-044X

FISL 9.0
Confira os números do maior de todos

www.phpmagazine.org.br

sumário

PHP-GTK

SQL Injection

Arquitetura

editorial

Apresentação

Equipe

Caixa de Entrada

Chamadas

artigos

PHP e o Conhecimento Científico ao redor do mundo
por Leandro Schwarz

Protegendo códigos em PHP-GTK
por Pablo Dall'Oglio

***SQL Injection* em PHP: vulnerabilidades e prevenção**
por Ismael Rocha

Desenvolvendo em três camadas com PHP, MVC e AJAX
por Almir Neto e Otávio Calaça

Os números do FISL 9.0

9° Fórum de Software Livre e muito PHP
por Helton Ritter

apresentação

Prezado leitor,

A equipe da PHP Magazine inicia esta edição pedindo desculpas pela demora na publicação deste exemplar. Os participantes da produção da revista são voluntários e, portanto, nem sempre dispõem de tempo livre para trabalhar em prol da comunidade. Nossa 5ª edição vem acompanhada de uma grande novidade. Graças ao apoio do Instituto de Desenvolvimento de Software Livre (IDSL – www.idsl.org.br), conseguimos mudar para o domínio www.phpmagazine.org.br. A partir de agora, o IDSL passa a nos apoiar institucionalmente, caracterizando ainda mais o nosso papel social.

Nesta edição contamos com artigos em três áreas distintas. Leandro Schwarz, que participou em todas as edições da revista, apresenta um artigo sobre a linguagem PHP e sua expressão diante do conhecimento científico pelo mundo. Nosso parceiro Pablo Dall'Oglio, nos prestigia mais uma vez com um belo artigo sobre PHP-GTK, demonstrando como aplicar segurança em seus aplicativos. Ismael Rocha também participa com um artigo sobre segurança, com vulnerabilidades e prevenção de *SQL Injection*. Almir Neto e Otávio Calaça participaram com um artigo sobre arquitetura em três camadas. Nosso agradecimento a esses autores que dedicaram seu tempo para contribuir com a revista.

Agradecemos a vocês pelo apoio, pois este é o nosso maior incentivo para trabalhar mais e mais. Vale lembrar que nosso portal já está próximo de atingir 10.000 usuários, um grande marco para todos.

Gostaríamos de registrar a participação e agradecer ao Helton Ritter, que nos últimos meses se fez presente nas atividades da revista.

Tenha uma boa leitura! Aguardamos suas sugestões e críticas.

Equipe da PHP Magazine

Editores
Flávio Zacharias Fagundes, zach@phpmagazine.org.br
Ricardo Aragão, ricardoaragao@phpmagazine.org.br

Administração
Flávio Zacharias Fagundes, zach@phpmagazine.org.br
Ricardo Aragão, ricardoaragao@phpmagazine.org.br

Comercial
Flávio Zacharias Fagundes
Ricardo Aragão

Projeto gráfico
Flávio Zacharias Fagundes
Ricardo Aragão

Revisão técnica
Ricardo Aragão da Silva
Flávio Zacharias Fagundes
Helton Ritter

Revisão - Língua Portuguesa
Camilla Carvalho, camilla@phpmagazine.org.br

PHP MAGAZINE
ISSN 1981-044X

www.phpmagazine.org.br

Comercial
comercial@phpmagazine.org.br

Contato/Suporte
contato@phpmagazine.org.br

Marketing
mkt@phpmagazine.org.br

Submissão de artigo

Primeiramente gostaria de parabenizá-los pelo excelente trabalho e manifestar meu total apoio. Sou desenvolvedor PHP e trabalho em uma instituição de segurança/insegurança da informação. Construí um manual de políticas e práticas de programação segura em PHP (que também serve para outras linguagens voltadas pra Web) e gostaria de divulgar o meu trabalho. Como é extenso, pois possui exemplos práticos detalhados e informações sobre as diversas vulnerabilidades, sugiro uma coluna na PHP Magazine sobre programação segura, onde a cada edição seria publicado um artigo tratando de uma determinada vulnerabilidade.

Ismael Rocha :: Brasília - DF

Recebemos este e-mail há algum tempo e trocamos várias mensagens com o leitor. Ismael demonstrou-se realmente interessado, desenvolvendo um tema e, após alguns feedbacks, apresentamos seu artigo nesta edição. Nosso muito obrigado pela sua participação. Que sirva de exemplo aos demais leitores.

Sugestão

Acompanho a revista desde a primeira edição. Quero dar os parabéns pelas matérias que têm me ajudado muito. Gostaria de pedir uma matéria sobre PHPReports, pois ele é um utilitário de relatórios muito eficaz. Eu mesmo serei o mais beneficiado com isso, pois tenho muitos relatórios para desenvolver e pouco conhecimento no assunto.

Leandro Coelho

Sugestão anotada, Leandro. Este tema sempre é do interesse de todos, mas infelizmente até o momento não foi submetido para a revista. Aproveitamos a oportunidade para comunicar que iniciantes também podem contribuir com artigos, pois nossa equipe está disposta a orientá-los na pesquisa e desenvolvimento do tema abordado. Acreditamos que a visão de um iniciante, bem como o empenho em publicar o trabalho e contribuir com a comunidade, servirá de modelo a ser seguido pelos demais.

Mais sugestões

Sou pesquisador na área de percepção auditiva e áudio digital. Tenho trabalhado com PHP há algum tempo e estou trabalhando com projetos na área de percepção auditiva e e-learning com aplicações browser-based. Fica a sugestão de um artigo sobre como construir aplicações Web com PHP, usando módulos e templates na forma que o Joomla ou o PHPNuke fazem. Parabéns pela revista!!!

Rael Toffolo

Agradecemos sua participação, Rael! Toda sugestão é bem-vinda. Na segunda edição apresentamos um artigo sobre a utilização do Mambo, um gerenciador de conteúdo muito semelhante ao Joomla. Que sua mensagem sirva de incentivo para a submissão de um artigo relacionado ao tema.

Novos Colaboradores

Já venho acompanhando todo o trabalho de vocês. Achei muito interessante o surgimento da PHP Magazine. Li todas as edições e fiquei muito encantado com todas. A quarta edição está ótima. Tenho experiência de quatro anos com a Linguagem PHP e interesse em colaborar. Que procedimentos devo tomar?

Josué Camelo dos Santos :: Anápolis-GO

Estamos sempre buscando pessoas interessadas em colaborar. Você pode participar submetendo algum artigo de seu interesse ou integrando-se a nossa equipe de colaboradores, atividade que deve ser assumida como um compromisso. Em breve estaremos disponibilizando para todos os interessados um documento de diretrizes, onde ficará definida a forma de participação da equipe da revista.

Interesse em participação

Estou interessado em ajudar a revista. Eu poderia escrever sobre integração do Adobe Flex com o PHP. O que acham?

Ricardo Cerqueira :: Caruaru-PE

Estamos aguardando sua participação, Ricardo! E, como sempre, nos colocamos à disposição para auxiliá-lo no que for necessário Basta entrar em contato.

Elogios

Parabéns a toda equipe pelo belo trabalho!

Raquel Dezidério Souto

É com grande satisfação que recebemos suas mensagens de agradecimento e apoio. Para quem desconhece: a Raquel participou da segunda edição da revista com o artigo "PHP nas Geociências" e, desde então, tem acompanhado nosso trabalho.

ENVIE E-MAIL PARA

contato@phpmagazine.org.br

Só serão publicadas mensagens com os seguintes dados do remetente: nome completo, cidade e estado.
A redação se reserva o direito de selecionar e adaptar as mensagens sem alterar o conteúdo geral.

O que é o CONAPHP?

CONAPHP significa COngresso NAcional de PHP. É um grande evento de PHP que consiste em várias palestras de PHP apresentadas por palestrantes nacionais e internacionais reputados.

Quando?

O CONAPHP 2008 vai decorrer nos dias 18 e 19 de Outubro.

Onde?

O CONAPHP 2008 está inserido no CONISLI 2008. O CONISLI é um evento sobre Software Livre muito popular no Brasil que sempre teve a forte presença da comunidade de PHP desde 2005. Este ano a comunidade de PHP do Brasil se empenhou para organizar no CONISLI um grande evento de PHP com o nome CONAPHP.
O CONISLI 2008 terá lugar nas da FIAP - Faculdade de Informática e Administração Paulista - em São Paulo com endereço Av. Lins de Vasconcelos, 1264, bairro Aclimação.

Quem?

Quem melhor conhece os interesses e necessidades da comunidade de PHP, são os próprios membros da comunidade. Por isso, o CONAPHP é organizado exclusivamente por membros bem ativos da comunidade de PHP Brasileira.

O tamanho da comunidade de PHP no mundo está estimado em 4,5 milhões de profissionais que trabalham no desenvolvimento e manutenção de mais de 20 milhões de sites. A comunidade de PHP no Brasil tem cerca de 5% do total global, portanto são mais de 200.000 profissionais de PHP brasileiros.

Devido ao seu grande tamanho, já se pode contar que a comunidade de PHP do Brasil venha participar em peso no CONAPHP. O CONISLI 2008 tem uma expectativa de audiência de 1.500 congressistas, dos quais pelo menos 500 deverão vir para assistir às palestras do CONAPHP.

Quem vem palestrar?

Como demonstração da sua força e capacidade de organização, a comunidade de PHP do Brasil está trazendo para este evento 2 palestrantes internacionais, para além de vários outros nomes conhecidos da cena de PHP do Brasil.

Andrei ZmievskiAndrei Zmievski é um reputado desenvolvedor no núcleo de PHP. Ele é mais conhecido por ter sido o criador do sistema de *templates Smarty* e pela extensão de criação de aplicações *desktop* PHP-GTK.

Este ano ele vem falar do PHP 6, nomeadamente da maior mudança da qual ele foi o principal arquiteto: suporte nativo a Unicode. Este recurso vai facilitar muito o desenvolvimento de aplicações internacionalizadas em que o texto usa alfabetos distintos dos usados na Europa e Américas, como por exemplo os usados nos paises da Ásia, Leste Europeu, tais como Rússia, China, Japão, Coreia, etc..

César RodasCésar Rodas é o outro palestrante internacional convidado. Apesar de ser ainda um jovem universitário que estuda no Paraguai, César já participou em atividades dignas de grande destaque, como o fato de ter sido o vencedor do *PHP Programming Innovation Award* de 2007 organizado pelo site PHPClasses.org.

Neste evento César virá falar sobre o seu mais recente trabalho de destaque incluido na iniciativa *Google Summer of Code (GSoC)*. Trata-se do desenvolvimento de recursos avançados para o popular sistema de blogs WordPress que permitem lidar com arquivos de conteúdo de grande porte em sistemas mais apropriados para esse efeito, como o YouTube, Amazon S3, Google Picasa, etc..

PARTICIPE
Está disponível a seção para inscrições, garanta já sua participação.
Você pode apoiá-lo divulgando em seu site e obtendo um pequeno espaço para sua logomarca no site do portal do evento.
Verifique as formas de inscrições, saiba mais clicando aqui.

Eventos
A PHP Magazine apóia a iniciativa de eventos como cursos gratuitos, palestras e fóruns. Para divulgar na revista, envie informações sobre o evento para contato@phpmagazine.org.br , que teremos prazer em contribuir com a divulgação.

por Leandro Schwarz

PHP e o Conhecimento Científico ao redor do mundo.

A linguagem PHP é amplamente utilizada em *scripts* para fins comerciais. Usa-se no desenvolvimento dos carrinhos de compras, das vitrines de lojas virtuais, dos sítios de leilões e similares. Através de tecnologias híbridas, como AJAX, também é possível a sua utilização como aplicações de bate-papo, por exemplo, do serviço de atendimento ao cliente via *Chat*. Muito difundido também é o uso da linguagem PHP em *scripts* de fóruns e listas de discussões; pouco se conhece, entretanto, sobre o uso da linguagem PHP como ferramenta de auxílio à difusão de pesquisas acadêmicas. Sendo assim, o artigo pretende um breve esclarecimento sobre o uso da linguagem para a difusão da ciência e tecnologia no Brasil e no mundo.

O objetivo deste artigo não é desmerecer linguagens de programação como as linguagens C, C++ ou Delphi, e nem as linguagens que já são muito utilizadas para este fim, como a linguagem Java. O objetivo principal deste artigo é mostrar e incentivar a utilização da linguagem PHP como ferramenta para a difusão do conhecimento científico.

Grande parte dos programadores em PHP utiliza a linguagem como suporte à busca e população de bases de dados. Pode-se verificar tal fato com uma simples busca na Internet por *scripts* em PHP sobre temas como "*login* de cliente", "loja virtual" e "contador de visitas". O número de resultados ultrapassa em muito a busca por qualquer *script* relacionado a calculadoras, gráficos matemáticos e similares.

A razão incontestável para este fato é o uso muito mais difundido de lojas virtuais e ferramentas de *e-commerce* do que das ferramentas matemáticas. Até porque, o mercado de trabalho dos programadores PHP está voltado para este nicho.

O que cabe aos programadores de PHP é saber que a linguagem também é utilizada dentro do meio científico como ferramenta de *interface* entre bases de dados e pesquisadores, possibilitando que experimentos científicos sejam realizados de forma mais eficaz.

Serão apresentadas sete aplicações da linguagem PHP no meio acadêmico. Todas foram publicadas em revistas de divulgação científica, nacionais e internacionais. Um breve resumo de cada aplicação será feito.

1. Transporte de plasma (Itália 2003)

Foi desenvolvido um equipamento para estudar o transporte de plasma, o aquecimento do plasma e a corrente na presença do aquecimento adicional ocasionado pela radiofreqüência. As descargas de plasma duram em torno de 1,5 segundos e, em modo normal de operação, um experimento é realizado a cada 20 ou 25 minutos.

Devido à natureza experimental do equipamento e da complexidade do fenômeno envolvido, a confiabilidade do experimento é obtida com a validação manual de cada experimento por meio da verificação da validade de cada medição e da procura por falhas nos sensores. Considerando-se que cada experimento gera entre 1.200 e 2.000 medições, grande quantidade de tempo e recursos humanos deve ser alocada para a validação de cada experimento.

Para automatizar o sistema, uma rede neural artificial foi treinada para validar cada experimento, possibilitando que mais experimentos possam ser realizados em um mesmo dia, com custos mais baixos. A rede neural desenvolvida roda em um servidor LINUX RedHat e armazena os dados em um banco de dados relacional MySQL. O núcleo da validação do experimento foi feito em MATLAB 6.

Os dados armazenados no banco de dados podem ser acessados pelos usuários por meio de uma *interface* Web. A *interface* foi desenvolvida em navegadores da Internet por apresentar-se mais leve que os softwares usuais. A interface foi desenvolvida em PHP 4.1.2, e roda em servidor Apache 1.3.12.

Através da *interface* Web é possível detectar a funcionalidade do sistema através de uma visualização gráfica baseada nos resultados de validação de um dado experimento. Um esquema de caixas coloridas permite identificarem-se sensores falhos, que aparecem em uma cor diferenciada. Clicando-se em cima de cada caixa, uma janela *pop-up* surge com os dados e a localização do sensor.

Outra funcionalidade do sistema permite que o usuário procure no banco de dados por experimentos arquivados de qualidade, fornecendo os dados do experimento que interessem ao pesquisador. Os usuários também podem selecionar um subconjunto de dados arquivados, especificando um sensor [Buceti, 2003].

2. Reações nucleares (China 2004)

Os dados nucleares compreendem tanto as propriedades do núcleo dos átomos, bem como as leis fundamentais que regem as iterações nucleares.

Embora alguns dos institutos nucleares, empresas e departamentos das universidades chinesas tenham construído suas próprias bases de dados e ferramentas, não havia uma base de dados chinesa disponível na Internet até 2001.

As maiores bases de dados internacionais armazenam os dados em arquivos de texto em formato ASCII com padronizações diferentes, por exemplo, os formatos *Evaluated Nuclear Data File* (ENDF) e *Evaluated Nuclear Structure Data File* (ENSDF) Estes formatos são, em geral, complexos para usuários não-especializados, o que apresenta grandes dificuldades na verificação e processamento de arquivos grandes.

O sistema desenvolvido, chamado NDOS (*Nuclear Data On-line Services*), inclui um repositório de dados central utilizando um banco de dados relacional e permite a busca interativa dos dados através de tecnologia baseada na WEB.

O sistema NDOS também pode reconstruir facilmente os dados nucleares em formatos padronizados, tais como ENDF e ENSDF, a partir do repositório central de dados. Esta característica oferece flexibilidade para compara dados proveniente de bases de dados diferentes e o processamento *on-line* dos dados.

O sistema roda em plataforma Linux e foi desenvolvido em PHP com a base de dados MySQL. A base de dados e os serviços principais do sistema NDOS residem em uma estação de trabalho PC, mantido pelo *Institute of Heavy Ion Physics* (IHIP) da Universidade de Pequim. A maioria das funções de gerenciamento da base de dados, tais como a adição de dados, podem ser realizados remotamente.

Uma ferramenta para "plotar" gráficos foi desenvolvida utilizando a biblioteca JPGraph para PHP. A ferramenta permite que o usuário "plote" gráficos com os dados provenientes da base de dados ou dos dados processados através de alguma das outras ferramentas do sistema. Pode-se "plotar" gráficos utilizando dados de diversas bases de dados em um mesmo gráfico, com cores diferentes para cada curva, permitindo a sua comparação. As escalas, os títulos e os eixos X e Y podem ser escolhidos pelos usuários de maneira simplificada. Os eixos podem ser exibidos em escala linear ou logarítmica, pois os gráficos em escala logarítmica apresentam a informação de maneira mais eficaz para sessões de baixa energia. Os gráficos podem ser salvos em formato *Postscript* ou imagens GIF no computador do usuário, para visualização posterior ou impressão. Os gráficos disponíveis estão no formato de gráficos 2-D, entretanto, gráficos 3-D serão desenvolvidos para visualização em perspectiva das distribuições angulares *versus* energia incidente ou distribuições de energia secundárias *versus* energia incidente [Fan, 2004].

3. Mapeamento genético (Suíça 2004)

Um grande desafio na biologia atual é a determinação da função genética em larga escala. Os padrões estabelecidos e o vocabulário controlado facilitam a integração dos dados experimentais com o trabalho computacional. As bases de dados estruturadas e as ferramentas de busca possibilitam meios de caracterizar a função de um dado gene.

O seqüenciamento genético completo da planta ornamental *Arabidopsis* foi alcançado no ano de 2000 e permitiu monitorar a expressão desta planta em escala genética usando *microarrays*. O *array* ATH1, por exemplo, representa aproximadamente 23.750 genes da *Arabidopsis*. Milhares de *arrays* foram processados desde então.

A combinação múltiplas bases de dados gera dúvidas quanto à sua compatibilidade, em particular quando se comparam dados de diferentes organismos.

Um sistema *on-line* foi desenvolvido para facilitar a descoberta da função genética, compreendendo uma base de dados de expressão genética e um conjunto de ferramentas de busca e análise, chamado GENEVESTIGATOR. O GENEVESTIGATOR foi desenvolvido como sendo uma ferramenta *on-line* amigável para análise de dados de expressões genéticas em larga escala. Consiste de uma base de dados MySQL e um servidor rodando *scripts* em linguagem PHP.

O GENEVESTIGATOR é acessado gratuitamente para todas as instituições acadêmicas. Uma vez que a base de dados possui tanto dados públicos quanto confidenciais, um sistema usuários com perfil *dual fio* criado para gerenciar o acesso de usuários de perfil público e privado. Todos os usuários devem se registrar e iniciar uma sessão com usuário e senha. Os dados coletados para registro são utilizados apenas para administração da base de dados e para aperfeiçoar o funcionamento do sistema. As informações pessoais não são compartilhadas com terceiros [Zimmermann, 2004].

4. Tele-consulta médica (Turquia 2004)

A equipe médica de regiões remotas tem grande dificuldade para conseguir a opinião de médicos especialista em uma determinada área, principalmente porque estes,

normalmente, trabalham nos grandes centros urbanos. Portanto, os pacientes de zonas remotas devem viajar para os centros urbanos à procura de opinião especializada.

A telemedicina é a transferência de dados médicos eletrônicos, como imagens de alta resolução, sons, vídeos e sinais bioelétricos, entre duas regiões. Sistemas de tele-consulta são utilizados por médicos para consultar pacientes de regiões remotas em tempo real ou em momentos mais convenientes ao médico.

As clínicas de tele-consulta são equipadas de forma a realizar os exames básicos, como pressão sanguínea, temperatura corporal, freqüência cardíaca, pesagem etc. Equipamentos de ECG, EEG, máquinas de raios-X e ultra -sonografias também podem ser fornecidos. As clínicas empregam clínicos gerais, enfermeiras e técnicos especializados na utilização e manutenção dos equipamentos e computadores. Os médicos locais e as enfermeiras utilizam o sistema para enviar os dados dos pacientes para uma base de dados central.

Os médicos especialistas localizados nos grandes complexos médicos utilizam o sistema para monitorar os pacientes através de páginas de Internet. Após a análise dos dados enviados pelas clínicas locais, os especialistas podem enviar seus laudos pelos meios de comunicação convencionais (telefone, fax, e-mail etc.).

O sistema foi desenvolvido em PHP, principalmente pela gratuidade da linguagem. Outras vantagens da linguagem para o sistema incluem a independência de plataforma (rodando em sistemas UNIX, Windows e Macintosh com apenas poucas modificações no código), a velocidade, o baixo uso de memória, a facilidade de aprendizado, a *interface* intuitiva (permitindo que os comandos PHP sejam inseridos por entre as *tags* HTML) e a grande difusão e suporte na comunidade *open-source*. Além destas vantagens, a PHP é uma linguagem robusta, possuindo características avançadas de programação. Diferente de outras linguagens, a PHP permite a integração com bibliotecas externas.

Um problema em potencial pertinente à transmissão de dados médicos é a segurança. Para garantir o sigilo e a integridade dos dados, os usuários devem acessar a base de dados através de usuário e senha. Uma vez autenticado, uma sessão é criada para armazenar os dados do usuário e um *cookie* é criado na máquina do cliente. Na sessão estão armazenados o usuário e a senha, entretanto, o endereço de IP, o tempo de acesso e a informação do navegador também podem ser armazenados. Os dados da sessão são codificados através da função HASH **md5()**, que possui taxa de colisão 2^{-128}, o que é um número bem baixo. Quando o usuário realiza, sai do sistema ou fecha seu navegador, a sessão criada é apagada. Além disso, cada usuário está associado a um nível de acesso, definindo quem pode ler, escrever e atualizar os dados na base de dados.

A HTML pura não é capaz de desenhar figuras, portanto, uma forma de apresentar imagens médicas em páginas da Internet é desenhá-las no servidor e enviá-las para o navegador em um formato compatível. Para isto, foi utilizada a biblioteca gráfica GD ☐ uma biblioteca escrita em ANSI C, gratuita que permite a criação de imagens de forma dinâmica.

Como característica adicional do sistema, a interface gráfica permite ao usuário converter a imagem para uma matriz binária antes da transmissão para o cliente. Este processo fornece uma matriz de 256 níveis de cores para cada *pixel*, permitindo seu *download* para o computador do cliente, onde ferramentas de processamento digital de sinais podem ser utilizadas. Além de gerenciar texto e imagens, o sistema também suporta dados multimídia, como filmes de ultra-sonografia, tomografia computadorizada e ressonância magnética e áudio de sons cardíacos e respiratórios, reproduzidos em uma janela do navegador [Kuntalp, 2004].

5. Identificação de espécies (Áustria 2005)

O gênero de fungos *Trichoderma* (*Hypocreales*, *Ascomycota*) contém fungos freqüentemente encontrados em madeira em decomposição e no solo, compondo a maior parte da biomassa total dos fungos. Algumas espécies deste gênero são economicamente importantes na produção de enzimas industriais, antibióticos e como bioagentes aplicados no controle dos patógenos de plantas.

As espécies *Longibrachiatum* são conhecidas como patógenos oportunistas de mamíferos imunocomprometidos, incluindo seres humanos e diversos indivíduos da espécie são encontrados como contaminantes de ambientes domésticos.

Estas diversas implicações dos *Trichoderma* na sociedade humana tornam a identificação das espécies um tópico de grande importância.

Devido à homoplasia, a determinação morfológica é difícil mesmo para especialistas, o que tem freqüentemente resultado no uso incorreto do nome das espécies associadas à produção de enzimas, ao biocontrole de patógenos, à infecção humana e à formação de metabólitos secundários.

O artigo apresenta um sistema de codificação de barras baseado nos oligonucleotídeos da cadeia de DNA como um método rápido para a identificação das espécies *Hypocrea* e *Trichoderma*. O método originou um sistema chamado TrichOKey v 1.0, com *interface* Web amigável baseado em uma biblioteca com poucas, porém essencialmente invariáveis e específicos entre espécies, seqüências de nucleotídeos dos *loci* ITS1 e 2.

A biblioteca de marcadores de oligonucleotídeos foi armazenada em uma base de dados MySQL. O programa TrichOKEY v. 1.0 foi escrito na linguagem PHP (PHP4) dentro do código HTML das páginas. A base de dados e os *scripts* estão localizados no endereço eletrônico http://www.isth.info ☐ um portal sobre a taxonomia *Hypocrea*/*Trichoderma*, desenvolvido pelo grupo dos autores. O servidor roda com o sistema operacional Linux, versão do *kernel* 2.4.21-15.0.4.ELsmp, com servidor Apache versão 1.3.33, PHP versão 4.3.10 e servidor

de base de dados MySQL versão 4.0.22-standard.

Os autores planejam a expansão do sistema para contemplar a identificação de quaisquer espécies, sendo também adaptado para os indivíduos futuramente encontrados [Druzhinina, 2005].

6. Sensoriamento Remoto (Brasil 2007)

Na Universidade Federal do Rio Grande do Sul (UFRGS), Leonardo Monteiro Brys e Sérgio Florêncio de Souza desenvolveram um sistema na Internet para disponibilizar conceitos e definições de técnicas de Processamento Digital de Imagens Orbitais (PDI).

Os autores justificam a escolha da linguagem PHP o fato de ser uma linguagem de programação *server-side*, possuir boa apresentação ao usuário e eficiente suporte matemático e suporte a um grande número de banco de dados. Outras vantagens incluem a facilidade de implementar, em PHP, códigos escritos em linguagem C, C++ e Perl e a gratuidade da linguagem.

O artigo faz referência à implementação da página cálculos geodésicos on-line como ferramenta de suporte ao Geoprocessamento no Curso de Engenharia Cartográfica da UFRGS

O sistema desenvolvido permite que o usuário faça o *upload* de uma imagem em esquema de cores RGB e outra em esquema pancromático, as quais serão misturadas dinamicamente. O *script* transforma o sistema RGB em sistema HSV (*Hue*, *Saturation*, *Value* – Tonalidade, Saturação e Brilho) e combina as imagens, preservando as características espectrais da imagem RGB e as características espaciais da imagem pancromática [Brys, 2007].

7. Densidade de probabilidade (Estados Unidos 2007)

A proposta do artigo foi explorar a utilização da linguagem PHP para proporcionar representações visuais para a distribuição de probabilidade de órbitas caóticas.

Foi desenvolvido um sistema em PHP que permitia as seguintes funcionalidades: seleção automática de um sistema de cálculo; seleção do sistema de cálculo pelo usuário; a exibição da distribuição densidade de probabilidade na forma de tabela; e os gráficos da densidade de probabilidade.

Todos os sistemas são enviados a partir de formulários HTML no método POST para a página em PHP, na qual cada sistema de cálculo é representado por uma função de cálculo específica. Para o acesso às variáveis postadas, foi utilizada a variável $_REQUEST.

Os gráficos foram implementados em uma página em separado utilizando a biblioteca GD. O artigo disponibiliza parte do código [Wang, 2007].

Considerações finais

Este artigo apontou as várias aplicações não comerciais da linguagem PHP, demonstrando a possibilidade de implementação de sistemas para divulgação do conhecimento e desenvolvimento científico.

Espera-se que, em relação ao desenvolvimento de interfaces gráficas, suporte matemático e processamento de imagens on-line, este documento tenha apontado a linguagem PHP como uma alternativa viável aos profissionais do ramo científico das empresas, universidades e centros tecnológicos brasileiros.

Referências e links sugeridos

BRYS, L.M.; SOUZA, S.F. *Aplicação da linguagem PHP em sensoriamento remoto*. In: Anais XIII Simpósio Brasileiro de Sensoriamento Remoto, Florianópolis: 2007.

BUCETI, G.; CENTIOLI, C.; IANNONE, F.; PANELLA, M.; RIZZO, A.;VITALE, V. *A rating system for post pulse data validation. Fusion Engineering and Design*. 2003.

DRUZHININA, I.S.; KOPCHINSKIY, A.G.; KOMOJ, M.; BISSETT, J.; SZAKACS, G.; KUBICEK, G.P. *An oligonucleotide barcode for species identification in Trichoderma and Hypocrea*. Fungal Genetics and Biology, v. 42, 2005.

FAN, T.S.; GUO, Z.Y.; YE, W.G.; LIU, W.L.; FENG, Y.G.; SONG, X.X.; HUANG, G.; LIU, T.J.;HONG, Y.J.; LIU, C.; CHEN, J.X.; TANG, G.Y.; SHI, Z.M.; HUANG, X.L.; CHEN, J.E. *NDOS: nuclear data online services. Annals of Nuclear Energy,* v. 31, 2004.

KUNTALP, M.; AKAR, O. *A simple and low-cost Internet-based teleconsultation system that could effectively solve the health care access problems in underserved areas of developing countries.* Computer Methods and Programs in Biomedicine, nº 75, 2004.

WANG, Z.J.; YANG, H.H.; DING, J. *Webgraphics for the computation of invariant measures.* Applied Mathematics and Computation, v. 187, 2007.

ZIMMERMANN, P.; HIRSCH-HOFFMANN, M.; HENNING, L.; GRUISSEM, W. GENEVESTIGATOR. *Arabidopsis mi-*

Leandro Schwarz - leandroschwarz@gmail.com
Engenheiro eletricista pela Universidade Federal de Santa Catarina (UFSC). Mestre em Engenharia Elétrica (área de concentração: Engenharia Biomédica) pela UFSC. Atuando desde 2000 com desenvolvimento Web, possui sólidos conhecimentos em PHP e MySQL.
Atualmente é professor do Departamento de Design do Centro de Artes (CEART) da Universidade do Estado de Santa Catarina (UDESC), e colaborador do Laboratório de Instrumentação (Labin) do Centro de Ciências da Saúde e do Esporte (CEFID) da UDESC.

por Pablo Dall'Oglio

Protegendo códigos em PHP-GTK

Este artigo apresenta algumas formas simples de proteger o acesso ao código-fonte de aplicações desenvolvidas em PHP-GTK.

O PHP é uma linguagem que nasceu na Web e é neste ambiente que é mais adotado. O ambiente Web é regido pela arquitetura cliente-servidor, em que há o código do programa interpretado pelo servidor e o resultado de sua execução. Normalmente nesse modelo, o resultando é código HTML para ser visualizado no navegador do cliente. Neste caso, a própria arquitetura cliente-servidor impede o usuário de ter acesso ao código-fonte do programa, a menos que a pessoa tenha acesso ao servidor da aplicação.

Quando desenvolvemos uma aplicação gráfica em PHP-GTK, rodamos a aplicação de forma local, utilizando recursos da máquina cliente, da mesma forma que em aplicações em Delphi ou Visual Basic. Mas, no caso do PHP-GTK, o código da aplicação é interpretado diretamente pelo executável do PHP. Desta forma, o código-fonte fica exposto ao cliente, caso ele queira investigá-lo.

O objetivo de se proteger um código em PHP-GTK pode ser comercial, impedindo que o cliente tenha acesso indevido ao código-fonte e realize cópias piratas, mas também pode evitar que um usuário "curioso" investigue o código do mesmo e interfira no seu funcionamento, tomando conhecimento de senhas de acesso ao banco de dados, por exemplo. As técnicas que iremos estudar se aplicam tanto para a utilização com PHP-GTK, quanto para utilização no desenvolvimento de uma aplicação PHP Web.

1. O programa

Para demonstrar as técnicas de proteção de código, em primeiro lugar vamos construir uma pequena aplicação a ser protegida, chamada "app.php".

A nossa aplicação de exemplo é extremamente simples. Ela está toda contida em uma classe chamada *Application*. A classe *Application* na verdade é filha da classe *GtkWindow*, logo, pelo mecanismo de herança, ela é uma janela.

Dentro da janela *Application*, criaremos uma caixa vertical (*GtkVBox*) e dentro desta caixa vertical colocaremos dois *widgets*. O primeiro ($this->nome) é um objeto *GtkEntry* para digitação de um nome qualquer. Já o segundo, ($botao) é um botão que ao ser clicado executará o método *onClick*().

A tarefa do método *onClick* é exibir uma janela de mensagem com o nome digitado pelo usuário.

```
<?php
class Application extends GtkWindow
{
private $nome;
      /*
      * método construtor
      * cria a janela e os campos
      */
      function __construct()
      {
            parent::__construct();
            $this->set_title('Aplicação');
            // cria uma caixa vertical
            $vbox = new GtkVBox;
            // campo para digitação do nome
            $this->nome = new GtkEntry;
            // cria o botão
            $botao = new GtkButton('Exibe');
            // conecta o botão à ação
            $botao->connect('clicked',
            array($this, 'onClick'));
            // adiciona os campos na vbox
            $vbox->add($this->nome);
            $vbox->add($botao);
            // adiciona a vbox na janela
            $this->add($vbox);
      }
      /*
      * método onClick
      * Exibe o nome digitado pelo usuário
      */
      function onClick()
      {
      $dlg =new GtkMessageDialog(null,
      Gtk::DIALOG_MODAL,
      Gtk::MESSAGE_INFO, Gtk::BUTTONS_OK,
      $this->nome->get_text());
      $dlg->run();
      $dlg->destroy();
      }
}
?>
```

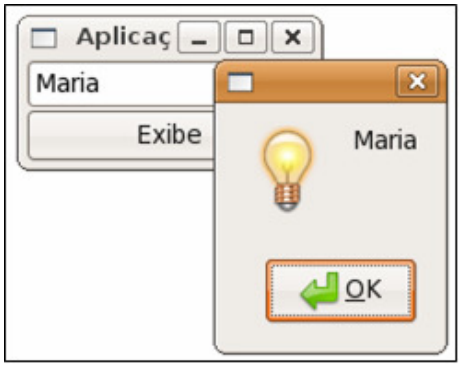


Figura 1 - Aplicação de exemplo

Agora, para executar tal aplicação, teremos de escrever um segundo programa para incluir a classe "app.php" e, então, instanciar um objeto desta classe e executar seu método *show_all*(), para exibir a janela.

```
<?php
 // inclui a classe
 include_once 'app.php';
 // instancia um objeto
 $app=new Application;
 // exibe a janela
 $app->show_all();
 Gtk::Main();
?>
```

2. Compactação

A forma mais simples de se proteger o código de um programa é compactando-o. Este mecanismo protege o acesso ao programa de grande parte dos usuários, mas é facilmente reversível, tendo em vista que basta o usuário final da aplicação descobrir que ele deve descompactar um arquivo para ter acesso ao seu código-fonte.

Para compactar a aplicação, iremos utilizar o aplicativo GZ do Linux, que compacta o programa "app.php", gerando um outro, chamado "app.php.gz".

```
# gzip app.php
```

Agora, para utilizar o programa ao invés de simplesmente executarmos o comando *include_once* sobre o arquivo compactado, precisamos indicar qual o *wrapper* utilizado para sua interpretação. Neste caso, o *wrapper* é "compress.zlib". O arquivo é descompactado instantaneamente e a classe fica disponível.

```
<?php
// inclui a classe compactada
include_once 'compress.zlib://app.php.gz';
// instancia um objeto
$app=new Application;
// exibe a janela
$app->show_all();
Gtk::Main();
?>
```

3. Criptografia Simples

Podemos melhorar um pouco mais o algoritmo anterior, criptografando o código da aplicação além de compactá-la. Para tal, iremos escrever um pequeno programa "empacotador" da aplicação. Este programa irá ler o código da aplicação por meio da função *file_get_contents*(), para então remover as tags de início e fim do <?php?> com a função *substr*(). Após, iremos codificar o fonte do programa com a função *base64_encode*(). Note que aqui poderíamos estar utilizando métodos mais poderosos de criptografia. Ao final, gravamos o código compactado e criptografado no arquivo "app.php.cri".

```
<?php
$code = file_get_contents('app.php');
$code = substr(trim($code),6, -2);
$code = base64_encode($code);
file_put_contents('compress.zlib://
app.php.cri', $code);
?>
```

Agora, para podermos executar tal programa criptografado, temos de escrever outro programa. Este programa irá ler o conteúdo do arquivo criptografado, chamado "app.php.cri", descriptografá-lo e executar este código por meio da função *eval*(). Após, o funcionamento do programa continua o mesmo.

```
<?php
// inclui a classe criptografada
$code =
file_get_contents('compress.zlib://
app.php.cri');
eval(base64_decode($code));
// instancia um objeto
$app=new Application;
// exibe a janela
$app->show_all();
Gtk::Main();
?>
```

Você deve ter percebido que as duas técnicas demonstradas até aqui são bastante simples. Da mesma forma que são simples de utilizar são simples de serem desfeitas, uma vez que o usuário terá acesso ao código-fonte que irá descriptografar e executar o código fonte. Desta forma, facilmente utilizamos um comando *echo*() no lugar do *eval*() para exibirmos o código-fonte em tela. Claro que nenhum usuário final faz isto, mas qualquer desenvolvedor com conhecimentos básicos em PHP pode fazer isto.

4. bcompiler

O *bcompiler* é uma extensão do PHP que faz parte do PECL (*The PHP Extension Community Library*), que é um repositório de extensões comunitárias para PHP, escritas em linguagem C. Funciona assim como o PEAR (*PHP Extension and Application Repository*), que é um repositório de extensões escritas em PHP.

O *bcompiler* foi escrito com o objetivo de codificar arquivos, classes ou funções por completo em uma aplicação proprietária.

De acordo com o autor da *bcompiler*, Alan Knowles, a utilização da extensão pode melhorar o desempenho em até 30%, embora o código codificado seja bem maior em termos de tamanho físico ocupado pelo arquivo em

relação à aplicação normal. Em termos de proteção de código, é seguro dizer que é impossível recriar o exato código-fonte original.

Para utilizar a *bcompiler*, primeiro devemos instalá-la. Alguns pré-requisitos devem ser satisfeitos em alguns ambientes. No caso do sistema operacional Ubuntu, será necessário instalar algumas bibliotecas relacionadas à compactação de arquivos (bzip2).

```
# apt-get install libbz2-dev bzip2
```

Após isto, podemos proceder para a instalação da *bcompiler* em si. Para tal, utilizamos o binário "pecl", que acompanha a instalação do PHP. Para tal, digitamos o seguinte comando a partir do diretório onde os binários do PHP estão instalados.

```
# <php>/bin/pecl install bcompiler-0.8
```

Agora que temos a *bcompiler* instalada, teremos de escrever dois programas. O primeiro programa irá gerar os *byte-codes* e o segundo irá interpretá-los. Para distribuir o programa ao usuário final, precisaremos distribuir apenas os *byte-codes* e o programa interpretador.

Este primeiro programa listado a seguir é o codificador, sua função é ler o conteúdo de nosso programa "app.php", codificá-lo e gravar o conteúdo codificado dentro de um arquivo chamado "compiled.phpb", que além de codificado será também compactado. Para codificar o arquivo, utilizamos a função *bcompiler_write_file* (), além de outras com o objetivo de criar o cabeçalho e rodapé do arquivo.

```
<?php
// carrega a extensão bcompiler
dl('bcompiler.so');
// define o arquivo a ser gravado os byte-
codes
$fh = fopen("compress.zlib://
compiled.phpb","w");
// escreve o cabeçalho do arquivo
bcompiler_write_header($fh);
// codifica o arquivo 'app.php'
bcompiler_write_file($fh, 'app.php');
// escreve o rodapé do arquivo
bcompiler_write_footer($fh);
fclose($fh);
?>
```

Após rodarmos o programa codificador, o arquivo "compiled.phpb" será gerado. O conteúdo do arquivo codificado se parecerá em muito com o trecho listado a seguir, ou seja, totalmente ilegível.

```
80><8a>^C2.Ï^U^P^PqAA^P^Q^EÄ^E<9
Ñ5òß<94>^Lø^Z`M^[§h^A½8E`<9f>^T!
PÎ¨ôZl<88>LÊM<83>¯,6¡ª^]<90>9B<9
<88><82>ç²<99>TÀ»¯!`
c¶í6^B<9e>4ö3 c<8d><9a>^Bõà½yz
ê¤Ûêôó,£4ÝÕ'^SõÙÃëÃ/Eô©Í-<98><89
\+4@+¤¥pVàWcVhÀ-H~O<8c>Øêðõ¯DñÉÕ
>7M^NÔL® <99>7<82>ì7<86>3^Zu¦^U
```

Após codificarmos nosso programa, podemos escrever o programa interpretador que será distribuído juntamente com os *byte-codes* (compiled.phpb). O programa interpretador irá carregar a extensão "bcompiler.so", que disponibiliza as funções da *bcompiler*, abrir o arquivo de *byte-codes* por meio da função *fopen*() e interpretar o arquivo codificado por meio da função *bcompiler_read*(). Note que esta função disponibiliza para aplicação exatamente o mesmo código que foi codificado. Desta forma, o programador terá neste momento acesso à classe "*Application*", que faz parte do arquivo original "app.php" que foi codificado. Após executarmos a *bcompiler_read*(), fechamos o arquivo e então já podemos instanciar objetos da classe *Application*. No final do programa instanciamos um objeto e exibimos ele em tela pelo método *show_all*(). Neste momento o programa será executado da mesma forma que em nosso primeiro exemplo onde interpretávamos diretamente o código-fonte.

```
<?php
// carrega a bcompiler
dl('bcompiler.so');
// abre o arquivo compactado
$fh = fopen("compress.zlib://
compiled.phpb","r");
// lê os byte-codes
bcompiler_read($fh);
// fecha o arquivo
fclose($fh);
// intancia o objeto
$app=new Application;
// exibe a janela
$app->show_all();
Gtk::Main();
?>
```

Considerações finais

Neste artigo estudamos algumas técnicas muito simples de se proteger o acesso ao código-fonte até abordarmos a *bcompiler*, uma extensão bastante complexa e poderosa do PHP. Sabemos da existência de diversas ferramentas comerciais com o objetivo de proteger ou ofuscar códigos em PHP. Mesmo assim, focamos este artigo em soluções simples, livres e que funcionassem não apenas no ambiente Web, mas também para quem desenvolve em PHP-GTK.

Referências e links sugeridos

[PHP-GTK Brasil] – http://www.php-gtk.com.br
[Livro PHP-GTK] – http://www.php-gtk.com.br/book
[Site do Autor] – http://www.pablo.blog.br
[Site do PECL] – http://pecl.php.net
[Site bcompiler] – http://www.php.net/bcompiler

Pablo Dall'Oglio - pablo@dalloglio.net
Pablo Dall'Oglio é bacharel em Informática pela UNISINOS. Autor dos livros "PHP Programando com Orientação a Objetos" e "PHP-GTK Criando Aplicações Gráficas com PHP", pela editora Novatec. Também foi o criador do PHP-GTK Brasil. Atualmente, é diretor de tecnologia e proprietário da Adianti Solutions (www.adianti.com.br).

por Ismael Rocha

SQL Injection em PHP: vulnerabilidades e prevenção.

Detalharemos algumas vulnerabilidades referentes ao ataque de injeção de comandos SQL em aplicações escritas em PHP, suas conseqüências e algumas boas práticas para sua prevenção.

Em diversos momentos o foco principal é atender a todos os requisitos definidos pelo cliente num prazo realmente curto. Em algumas ocasiões, por falta de planejamento ou porque realmente o desenvolvedor não se atém a detalhes, o projeto acaba deixando algumas lacunas de segurança abertas após a publicação.

Umas das falhas mais comuns e encontradas facilmente no mudo Web é a injeção de comandos SQL, conhecido como *SQL Injection*. Apresentaremos aqui o que é essa falha e quais as conseqüências que a exploração da falha pode nos trazer, bem como práticas para preveni-la.

1. O que é *SQL Injection*?

Sql Injection é um tipo de ataque que consiste na injeção de uma consulta SQL via entrada de dados do cliente para a aplicação. Um ataque bem sucedido pode ler dados sigilosos de uma base de dados, modificá-los (*update/insert/delete*), executar operações de administração no banco de dados (ex: *shutdown*) e, em alguns casos, executar comandos no sistema operacional (ex: *SQL Server xp_cmdshell). Um exemplo de execução seria=> exec xp_cmdshell 'dir c:'*, O ataque também pode fazer leitura (*Mysql load_file*) de arquivos do sistema.

As principais conseqüências de uma injeção de SQL bem sucedidas podem ser:

- **Confidencialidade**: banco de dados geralmente guardam informações confidenciais. A perda de confidencialidade é um freqüente problema com as vulnerabilidades de *SQL Injection*;

- **Autenticação**: se a aplicação não faz os devidos tratamentos nos dados que são "inputados" pelo usuário e já encaminha tais dados para a *query* do banco de dados para verificação de credenciais, é possível acessar o sistema sem conhecimento prévio de senha/usuário.

- **Autorização**: se alguma informação de autorização está guardada em um banco de dados SQL, pode ser possível mudar a informação por meio da exploração de uma vulnerabilidade de *SQL Injection*;

- **Integridade**: assim como é possível ler informações confidenciais, também é possível fazer mudanças ou até mesmo excluir informações usando ataque de *SQL Injection*.

2. Demonstração de implementações vulneráveis a *SQL Injection*

Tomaremos como exemplo de código vulnerável um *script* simples para fazer autenticação na aplicação a partir de um usuário e senha fornecidos por um cliente. Também será apresentado outro *script* cuja principal função é demonstrar ao usuário uma notícia, bem como seu título e autor. Essas informações são recuperadas da base de dados por meio de parâmetros fornecidos via métodos *POST* ou *GET.*

Veja abaixo um exemplo de *script* vulmerável de autenticação (*login*):

```
<?php
// Script: artigo1.php (Sript de autenticação)
// Funcao: demonstrar vulnerabilidade de SQL I
njection
 $login = $_POST['login'];
 $senha = $_POST['senha'];
 mysql_connect "localhost", "root", "ismael");
 mysql_select_db("teste");

 $query = "SELECT * FROM users WHERE login_use
r='$login' AND password_user='$senha'";

 $consulta = mysql_query($query) or die
(mysql_error());
 $linhas = mysql_num_rows($consulta);
 if ($linhas > 0)
 {
   echo "Seja bem-vindo!!";
 }
 else
 {
   echo "Login e/ou senha invalidos!!";
 }
?>
```

No nosso formulário submeteremos as seguintes informações conforme mostrado na tela abaixo:

Figura 1 - Formulário de *Login*

Observe que no campo "senha" informamos a seguinte instrução:

```
'OR 1='1
```

Da forma como os dados foram submetidos, a *query* montada dinamicamente no *script* acima, ficaria da seguinte forma:

```
SELECT * FROM users WHERE login_user='qualquercoisa' AND password_user=' ' or 1='1'
```

Dessa maneira, a afirmação acima é verdadeira, pois a consulta tenta encontrar algum registro se a condição campo *login_user* for igual a "qualquercoisa" e *password_user*="" **ou** 1 for igual a 1.

Da maneira como foi implementado o *script*, a autenticação acontece a partir do momento em que a consulta retorna alguma linha. Sendo assim, no caso mencionado, todas as linhas da tabela *users* foram retornadas, o que, conseqüentemente, nos permitiu a autenticação na aplicação.

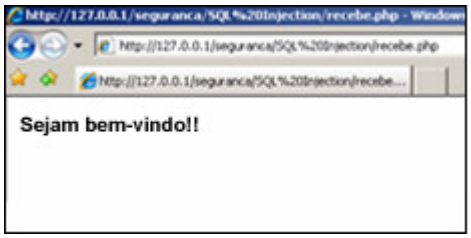


Figura 2 - Autenticação realizada com sucesso

Abaixo, damos um exemplo de código do script vulnerável de notícias:

```
<?php
    // Script: artigo1.php (Script de noticias)
    // Função: demonstrar vulnerabilidade de SQL Injection

    mysql_connect("localhost", "root", "" );
    mysql_select_db("teste");

    if ( !isset($_GET['id']) )
        {
        echo "A notícia a ser lida não foi informada!";
        exit;
        }
        else
      {
        $id = $_GET['id'];
    }

    $query    = "SELECT titulo, conteudo, autor FROM noticias WHERE id=$id";
    $consulta = mysql_query($query);
    $linhas   = mysql_num_rows($consulta);

    if ( $linhas==0 )
    {
        echo "Não foi possível recuperar a notícia informada!!";
    }
    else
    {
        echo mysql_result($consulta, 0, 'titulo')."<br><br>";
        echo mysql_result($consulta, 0, 'conteudo');
echo mysql_result($consulta, 0, 'autor')."<br><br>";
    }
?>
```

Na figura 3, nosso script funcionando com uma requisição normal:

Figura 3 – Código de notícias em funcionamento normal

Agora injetaremos a seguinte instrução SQL através do parâmetro ID, passado via GET para a aplicação, com objetivo de fazer a união com um segundo *select*, nos trazendo alguns campos da tabela *user* do banco MySQL:

```
UNION SELECT user,host,password FROM mysql.user LIMIT 1,1
```

A figura 4 demonstra o resultado após a inserção da instrução SQL acima.

Figura 4 – Host, usuário e senha retornados

A instrução injetada nos retornou os campos *user*, *login* e *password,* oriundos da base MySQL, que são os usuários válidos que se conectam ao servidor de banco de dados. Explorando essa falha é possível obter os usuários e senhas do SGDB. Observe que nesse caso, a aplicação vulnerável se conecta como *root* ao SGDB. Por isso é possível fazer a leitura da base de dados MySQL. Sendo assim, a senha criptografada é passível de ataque de força bruta para recuperá-la.

Em uma outra situação, poderíamos fazer uso da base de dados *information_schema* (disponível a partir da versão 5.0) para levantar informações diversas, tais como nome de bancos, tabelas, colunas entre outras coisas sobre as quais o usuário da aplicação conectado ao SGDB possui privilégios. Veja o exemplo:

```
UNION SELECT table_name, table_schema,
column_name FROM information_schema.columns LIMIT 286,1
```

Figura 5 – No exemplo acima, a injeção de comandos nos retorna o nome de uma tabela, nome de um banco e nome de um campo dessa tabela.

Com a injeção da instrução acima, obtivemos como resultado os campos *table_name*, *table_schema*, *column_name* do banco de dados *information_schema*. Verificamos que existe uma base de dados chamada *helpdesk*, uma tabela chamada *tickets_up* e um campo chamado *tamanho_anexo*. Um atacante poderia facilmente montar o desenho de todos os bancos de dados em que o usuário da aplicação conecta do SGDB possui permissão e, conseqüentemente, ler as informações contidas nessas bases após o conhecimento do nome das tabelas e campos.

3. Medidas preventivas

Há diversas formas de se prevenir contra a injeção de comandos SQL. Pode-se utilizar desde recursos da própria *engine* PHP, como *magic_quotes_gpc* (em alguns casos), até o uso funções diversas como escape de dados (*mysql_real_scape_string*, *addslashes)*, além das *prepared statements*.

3.1 Os recursos de escape

Para aqueles que não conhecem, o *Magic Quotes* é um recurso do PHP em que são inseridos automaticamente caracteres de escape (\) em todos os dados enviados para um *script* PHP, via POST ou GET e *Cookie*. Com esta opção habilitada no *php.ini*, nosso *script* que faz a autenticação de usuários estaria protegido, pois, ao inserir a instrução 'OR 1='1, o próprio PHP adicionaria \ antes das aspas, fazendo com que a consulta enviada ao banco ficasse da seguinte forma:

```
SELECT * FROM users WHERE
login_user='qualquercoisa' AND pass-
word_user='\'or 1=\'1'
```

Entretanto, como foi observado, há casos de injeção de instruções em que um atacante não necessita de aspas, como no exemplo do *script* de notícia e, sendo assim, funções de escape como o *addslashes()*, *mysql_real_escape_strings()* se mostrariam ineficientes para conter tais inserções.

É preciso observar que conforme manual do PHP, o recurso *Magic Quotes* tornou-se obsoleto e foi removido do PHP 6.0. Confiar neste recurso é extremamente **não** recomendado.

3.2 Validando os dados recebidos

Nos exemplos mostrados não há qualquer tipo de validação dos dados enviados pelo cliente. A recomendação é sempre validar os dados recebidos por um usuário, checando/convertendo seu tipo de acordo com o campo a ser consultado, delimitando o tamanho de acordo com o campo para consulta e aplicando expressão regular sobre os mesmos.

No *script* de notícias, o problema da injeção de instrução SQL seria resolvido facilmente apenas convertendo o valor de $_GET['id'] para um inteiro, usando a função *intval()*. No caso, o PHP ao converter **1 UNION ...** retornaria apenas 1. No exemplo de autenticação, caso o sistema de senhas da aplicação, por exemplo, permitisse apenas números e letras, poderia-se utilizar expressão regular para checar de os dados fornecidos. Funções como *preg_match()*, *strlen()*, *intval()*, *intfloat()*, dentre outras, são bastante úteis no auxílio da validação dos dados.

3.2 Prepared statements (medida eficaz contra SQL Injection)

Prepared statement é a habilidade de definir uma consulta uma vez e executá-la diversas vezes com diferentes parâmetros. Sua principal função é separar a lógica da consulta SQL dos dados recebidos para a montagem da *query* dinamicamente. Sendo assim, a nossa *query* do script de autenticação ficaria o como:

```
SELECT * FROM users WHERE login_user=? AND
password_user=?
```

Substituímos as variáveis *$login* e *$senha* por duas interrogações. Ao enviar a *query* "preparada" para o banco de dados, ele fica no aguardo dos parâmetros para fazer a comparação (nesse caso) e também da instrução para executar a *query*.

Dessa forma, se enviássemos algo como **'OR 1='1**, esse valor apenas seria comparado com o campo *password_user* assim que fizéssemos a "ligação" do parâmetro aguardado com a variável *$password*, pois, a lógica da *query* já foi montada previamente e está apenas a espera de um parâmetro. O recurso de *prepared statements* pode ser utilizado em diversas consultas SQL como *updates*, *inserts*, *deletes*.

Utilizada amplamente, a biblioteca *MySQLD* infelizmente não possui esse tipo de recurso, entretanto, há diversas outras bibliotecas que trabalham com *prepared statements,* como a *MySQLI* e *PDO*. Abaixo, segue um exemplo do mesmo *script* utilizando *MySQLI*. Abaixo, um código com instruções *Prepared Statements*.

```
<?php
 //Script:artigo1.php (Script de autenticação)
 $login = $_POST['login'];
 $senha = $_POST['senha'];

 //Instancia objeto para conexão com banco
 $bd = new mysqli
("localhost" , "user_ismael", "ismael", "teste
" );

  $query = "SELECT * FROM users WHERE login_
user=? AND password_user=?";

  //Método para envio de querys preparadas
  $stm = $bd->prepare($query);
  //Método  para  fazer a "ligação"
  //de conteúdo aos parâmetros esperados
  //pela query preparada
  $stm->bind_param("ss", $login, $senha);
  // Método para executar a query
  $stm->execute();

  $stm->store_result();
  $linhas = $stm->num_rows;

  if ($linhas > 0)
  {
     echo "Seja bem-vindo!!";
  }
  else
  {
     echo "Login e/ou senha invalidos!!";
  }
?>
```

Considerações finais

É inegável o poder de destruição de um ataque bem sucedido de *SQL Injection*. Como visto, um atacante pode burlar um sistema de *login*, obter dados de diversos databases do SGDB, bem como ler arquivos de sistema, descobrir senhas etc. A prevenção não é algo tão complicado. As principais dicas são: sempre validar a entrada de dados oriundas dos clientes e utilizar as *prepared statements*. A utilização de *frameworks* também pode contribuir para a redução da vulnerabilidade, visto que alguns possuem mecanismos de filtragem de dados que utilizam diversos recursos, inclusive expressões regulares. Aos programadores mais experientes fica a dica de revisarem seus códigos, pois a correria do dia-a-dia faz com que deixemos brechas às vezes imperceptíveis e, segundo o Gartner, órgão bastante reconhecido na área de pesquisas tecnológicas, 75% dos ataques bem sucedidos ocorrem via aplicações Web.

Referências e links sugeridos

[PHP] - http://www.php.net/manual/pt_BR/book.mysqli.php
http://www.php.net/manual/pt_BR/book.pdo.php
[OSWAP]
http://www.owasp.org/index.php/SQL_injection
[GARTNER]
http://www.gartner.com/it/about_gartner.jsp

Ismael Rocha - ismaelrg@gmail.com
Graduando em Sistemas de Informação pela Grupo Anhaguera, atua há mais de 5 anos na área de desenvolvimento de sistemas em diversas linguagens. Atualmente, trabalha com segurança da informação

Criando Aplicações Gráficas com PHP

por Almir Neto e Otávio Calaça

Desenvolvendo em três camadas com PHP, MVC e AJAX

Neste artigo, apresenta-se o conceito de desenvolvimento em três camadas com PHP. Para tal, será demonstrada a metodologia do MVC e sua integração com AJAX.

O MVC (*Model-View-Controller*) é um Padrão de Arquitetura de Software criado com o objetivo de aumentar a produtividade, separando a lógica do negócio da apresentação com o uso de controladores e implementando alguns *Design Patterns* (Padrões de desenvolvimento de software). Sua maior vantagem é a facilidade de manutenção do código, pois oferece estrutura de diretórios, arquivos e classes criadas.

Já o AJAX (*Asynchronous Javascript And XML*) é um conjunto de tecnologias que tem como intuito tornar mais dinâmica e interativa a camada de apresentação (*View*), fazendo com que o usuário tenha uma experiência muito melhor com o aplicativo.

Se utilizados da maneira correta, PHP, MVC e AJAX permitem o desenvolvimento de aplicações com alta produtividade, interatividade e qualidade.

1. Design Patterns

Design Patterns é o termo em inglês para Padrões de Projeto de *Software*, que nada mais é do que uma padronização no desenvolvimento de software.

Antes de tudo, é preciso saber que um *Design Pattern* é um conceito e não uma tecnologia. Sendo assim, não está ligado obrigatoriamente com nenhuma linguagem de programação ou *framework* de desenvolvimento.

Os *Design Patterns* surgiram com a necessidade dos desenvolvedores de resolver problemas de maneira eficaz. Christoper Alexander estabeleceu na década de 70 que os padrões de projeto devem ter como objetivo resolver um problema específico e, para isso, devem conter um nome, um exemplo, um contexto, um problema e uma solução.

Entre os padrões mais utilizados no desenvolvimento de software atualmente estão o *Factory*, com o objetivo de centralizar a criação de objetos, e o *Singleton*, que faz com que uma classe tenha somente uma instância ativa. Já o *Strategy* encapsula algoritmos e os representa através de uma *interface*. Outro padrão interessante é o *Template Method*, que define uma estrutura abstrata de classes com funcionamento parecido.

O MVC vai além do conceito de *Design Patterns*, pois define padrões para a arquitetura do software, como veremos nas próximas seções.

2. O que é MVC?

Antigamente as aplicações eram monolíticas, ou seja, não havia separação em camadas no desenvolvimento. Com o surgimento de linguagens de programação orientadas a Objetos, como C# e Java, surgiu também uma nova maneira de se desenvolver, separando em uma camada específica o código relativo à persistência. Com isso, um padrão antigo se popularizou, principalmente por ter sido implementado em alguns *frameworks*.

Esse padrão é o MVC, que foi descrito pela primeira vez em 1979 por Trygve Reenskaug nos laboratórios da Xerox, utilizando SmallTalk.

MVC é a sigla de *Model View Controller*, onde temos uma camada denominada Modelo, que são as regras de negócio do sistema, a camada da Visão, que é a *interface* entre usuário e o sistema, e os Controladores, que são utilizados para controlar o fluxo da aplicação.

Com o objetivo de separar as camadas, o MVC tem como resultado um código extremamente organizado, facilitando o desenvolvimento e a manutenção de sistemas.

Para que o MVC seja implementado é necessária a utilização de uma linguagem de programação com suporte à Orientação a Objetos. Portanto, com o melhor suporte do PHP 5 em relação à Orientação a Objetos, trabalhar com PHP 5 e MVC é uma tarefa simples e eficaz.

2.1. Vantagens de utilizar MVC

Ao desenvolver utilizando MVC, há várias vantagens em relação ao modelo estruturado.

- Reaproveitamento de código

Uma das maiores vantagens da Orientação a Objetos é o reaproveitamento de código, já que ao "modularizar" são criadas classes e funções que podem ser utilizadas em várias partes da aplicação.

- Facilidade de manutenção

Com uma boa estrutura de diretórios e arquivos relativamente pequenos, encontrar um trecho de código

para alterações se torna uma tarefa simples.

- Integração de equipes e/ou divisão de tarefas

Cada integrante da equipe pode trabalhar em uma camada específica. Por exemplo, um designer pode fazer a camada de apresentação, enquanto programadores trabalham nos controladores e regras de negócio.

- Camada de Persistência independente

Utilizando MVC, a aplicação não fica dependente de um banco de dados, pois a comunicação entre PHP e BD é feita em uma camada específica. A alteração do Banco de Dados utilizado pode ser feita sem afetar outras partes da aplicação. Além disso, há a possibilidade da utilização de uma biblioteca de BD para que a aplicação se torne portável a vários bancos.

- Implementação de segurança

Uma grande vantagem de utilizar controladores é a implementação de segurança, já que todo o fluxo da aplicação inicia e termina em um controlador. Além disso, há outras partes do sistema na qual a segurança pode ser feita.

- Facilidade na alteração da interface da aplicação

Para alterar a interface da aplicação não é necessário refazer o sistema, basta aplicar as alterações na camada de visão. Para facilitar ainda mais, pode ser implementada alguma biblioteca de *templates*.

Além disso, há outras vantagens como a padronização do código e o aumento da produtividade pela velocidade do desenvolvimento, entre outros pontos positivos.

2.1. Frameworks que implementam MVC

CakePHP

O CakePHP é um *framework* de desenvolvimento de software que utiliza PHP e proporciona uma extensa arquitetura para o desenvolvimento de sistemas com velocidade. Ele utiliza alguns *design patterns* como MVC e ORM. O objetivo do CakePHP é diminuir o custo do desenvolvimento ajudando os desenvolvedores a escrever códigos menores.

-> Versão Estável: 1.1.19.6305

-> Gratuito

-> Link: http://www.cakephp.org/

CodeIgniter

O CodeIgniter é um *framework* feito para desenvolvedores que necessitam de uma ferramenta simples e elegante para desenvolver aplicações Web completas. Uma das maiores vantagens de utilizá-lo é a alta performance que ele proporciona.

-> Versão Estável: 1.6.1

-> Gratuito

-> Link: http://www.codeigniter.com

PHP.MVC

O PHP.MVC foi desenvolvido com o objetivo principal de implementar o MVC com PHP utilizando um controlador principal. Teve como base o *framework* Jakarta Struts, inclusive implementando várias funcionalidades do Struts, como utilizar XML em sua configuração.

-> Versão Estável: 1.0

-> Gratuito

-> Link: http://www.phpmvc.net/

Symphony

O *Symphony* é um *framework* de desenvolvimento em PHP que proporciona uma arquitetura, componentes e ferramentas para desenvolvedores construir aplicações Web robustas e complexas mais rapidamente. O *Symphony* utiliza boas práticas de desenvolvimento Web integradas com bibliotecas de terceiros.

-> Versão Estável: 1.1

-> Gratuito

-> Link: http://www.symfony-project.org/

Zend Framework

Desenvolvido pela empresa criadora do PHP, o *Zend Framework* é baseado na simplicidade, boas práticas do desenvolvimento orientado à objetos e conta com uma base de código fortemente testado. Tem como objetivos desenvolver sistemas seguros, confiáveis e modernos.

-> Versão Estável: 1.5.1

-> Gratuito

-> Link: http://framework.zend.com/

PRADO

O PRADO é um *framework* baseado em componentes para o desenvolvimento de aplicações Web com PHP. PRADO é a sigla para ***P**HP **R**apid **A**pplication **D**evelopment **O**bject-oriented*, e conta com uma rica documentação e uma grande comunidade.

-> Versão Estável: 3.1.1

-> Gratuito

-> Link: http://www.pradosoft.com/

3. Model

Podemos dizer que a camada *Model* é o coração da aplicação. É nela que está a lógica da aplicação, responsável pelo que a mesma irá realizar. Nesta camada os dados são manipulados e armazenados. Para facilitar ainda mais, esta camada é separada em outras três par-

tes: A Entidade, que chamamos de *Bean*, a camada de Persistência, chamada de DAO e os arquivos de regras de negócio, que são as *Actions*. Veremos a seguir cada uma delas.

3.1. Bean

Implementando um conceito muito utilizado em Java, os *JavaBeans*, temos as entidades encapsuladas. A definição de *Bean* seria um componente de *software* que pode ser reutilizado várias vezes. São utilizados para encapsular vários atributos em um objeto. Na prática teremos um *Bean* para cada entidade do sistema.

A estrutura de um *Bean* é bastante simples, começando com a declaração da classe, que tem o nome da entidade, seguido da declaração dos atributos com o modificador de acesso *private*, e os métodos *setters* e *getters*. Para que a aplicação obtenha maior segurança, deve ser implementado o conceito de "Encapsulamento". Com isso, os atributos não podem ser acessados diretamente, já que estão declarados como *private*, podendo ser acessados somente pelos *setters* e *getters*. Um *setter* é um método que segue o padrão setAtributo(), e serve para definir o valor de um atributo. Já o *getter* é um método que segue o padrão getAtributo () e serve para obter o valor de um atributo, sempre que for necessário utilizar um atributo em um sistema, será esse o método utilizado.

Qualquer validação ou verificação a ser feita em um atributo deve ser implementada nos métodos *setters* e *getters*. É aí que questões de segurança devem ser implementadas, como forçar um atributo a ter um determinado tipo, evitando SQL *injection*, por exemplo.

Segue um exemplo de um *Bean*.

```
<?

class Produto
{
  private $id;
  private $nome;
  // ... outros atributos ...

  public function setId($id) {
    $this->id = (int)$id;
  }
  public function getId() {
    return $this->id;
  }
  public function setNome($nome) {
    $this->nome = $nome;
  }
  public function getNome() {
    return $this->nome;
  }
  // ... outros setters e getters ...
}
?>
```

Figura 1 – Exemplo de *Bean*

3.2. DAO

DAO é a sigla de *Data Access Object*, que pode ser traduzido como Objeto de Acesso a Dados. Ele é um objeto que provê uma interface abstrata para persistência de dados, ou seja, é um conjunto de funções que fazem a comunicação com os dados, seja em um banco de dados, um arquivo XML ou qualquer outra forma de armazenamento.

Para utilizar um SGBD, por exemplo, as consultas SQL serão escritas no DAO, ou então uma biblioteca de persistência pode ser implementada nessa camada para facilitar essa comunicação.

Uma das grandes vantagens de se utilizar uma camada de persistência independente é a portabilidade de banco de dados que o DAO oferece. Para fazer uma migração de banco de dados, ao invés de alterar toda a aplicação nos locais onde há comunicação com o banco, basta fazer as alterações nesta camada. Se uma biblioteca de persistência estiver sendo utilizada, essa alteração fica ainda mais simples, já que apenas o arquivo de conexão precisa ser alterado. Entre as bibliotecas mais utilizadas estão a EzPDO, o Propel e o PEAR:DB_DataObject.

Após implementado, o DAO pode ser utilizado sempre que for necessária a persistência de dados.

Segue um exemplo de DAO.

```
<?

class ProdutoDAO {
  public function listar()
  {

    $sql = "SELECT id, nome, valor
            FROM produto
            ORDER BY nome";

    $query = mysql_query($sql);

    $lista = array();

    $cont = 0;
    while ($linha = mysql_fetch_object
($query)) {
  $lista[$cont] = new Produto();
  $lista[$cont]->setId($linha->id));
  $lista[$cont]->setNome($linha>nome));
  $lista[$cont]->setValor($linha>valor));
  }
    return $lista;
  }
}
?>
```

Figura 2 – Exemplo de DAO

3.3. Actions

Os arquivos *Actions* são arquivos de Ação. É nele que se localizam as regras de negócio da aplicação. O arquivo de ação consiste em uma classe com o nome da ação a ser executada e um método *execute()*, que contém o

código que será executado.

Esses arquivos utilizam o *Bean* e o DAO para realizar persistência de dados, além de disponibilizar dados para a camada de Visão.

Segue um exemplo de um arquivo de Ação.

```
<?
class Listador {

  public function execute()
  {
    Conexao::conectar();

    $produtoDAO = new ProdutoDAO();
    $lista      = $produtoDAO->listar();

    $_REQUEST['lista'] = $lista;
    return "Listar";
  }

}
?>
```

Figura 3 – Exemplo de *Action*

4. View

A *View*, ou camada de visão, é a parte da aplicação que comunica e interage com o usuário final, ou seja, uma interface com ele. É nela que são implementadas a usabilidade, a interatividade e o *layout* visual, incluindo a linguagem de estilo (CSS), a linguagem de marcação (HTML) e a linguagem de programação cliente (*JavaScript*).

Quando o *Model* termina de fazer o processamento dos dados que serão apresentados ao usuário, envia-os ao *controller* juntamente com qual *view* será usada. Com isso, o *controller* joga o fluxo para a *view* que transforma esses dados em informação para o usuário.

É notável que dessa maneira a camada de visão quase não contém código PHP, apenas as variáveis geradas no *Model* e algum laço de repetição, em listagens. Mas podemos abstrair ainda mais a linguagem PHP da camada de visão utilizando *templates*.

Os *templates* são modelos, sem conteúdo, que possuem a apresentação visual de aplicações dinâmicas. Para suprir a pequena necessidade de linguagem não visual (PHP) da *view*, muitos sistemas de *templates* utilizam uma linguagem própria para manipular a exibição dos dados provindos do servidor. Como exemplo, pode-se citar o *Smarty*, o sistema de *templates* mais tradicional para PHP.

Independente das implementações contidas na *view*, é importante observarmos que com o MVC a camada de visão fica tão bem separada que pode ser desenvolvida inteiramente por um designer, que não entende de PHP. Ou, ainda, podemos desenvolver duas camadas de visão para uma mesma aplicação: a Web e a *Desktop* (utilizando PHP-GTK), tornando a aplicação ainda mais portável.

A boa usabilidade, implementada nessa camada, é essencial para o sucesso da aplicação. De nada adianta ter uma aplicação com um alto nível de processamento, se a apresentação não for muito usual e o usuário não conseguir utilizá-la. Para uma boa usabilidade devemos ter muita interatividade e transparência para o usuário, que podem ser proporcionadas pelo AJAX.

4.1 AJAX

O AJAX é o uso em conjunto de várias tecnologias promovendo uma técnica de requisição assíncrona muito difundida com a WEB 2.0. Antes de entender o que é Ájax, temos que compreender como funciona a requisição assíncrona em páginas WEB.

Temos por requisição assíncrona tudo aquilo que requisita dados do servidor sem a necessidade de toda a página ser recarregada. *Applet*, *Flash* e *Iframes* são exemplos de tecnologias que fazem requisição assíncrona. Para entender melhor esse conceito, observe os diagramas abaixo.

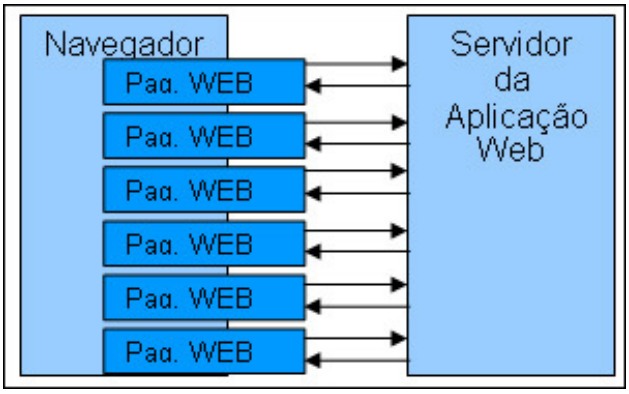


Figura 4 – Fluxo na Web sem requisição assíncrona

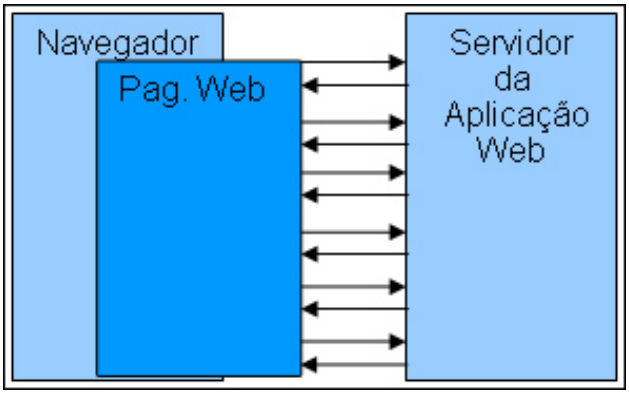


Figura 5 – Fluxo na Web com requisição assíncrona

Ao contrário do que é feito normalmente, quando utilizamos requisição assíncrona, uma mesma página consegue fazer várias requisições. Isso nos fornece diversas vantagens, como a redução do tráfego na rede e a maior transparência para o usuário, tornando o sistema semelhante a uma aplicação *Desktop*.

No AJAX a requisição é feita através do J*avascript* para um XML que se encontra no servidor. Quando usa-

mos AJAX com PHP, o XML é formado pelo PHP dinamicamente através de algum parâmetro que o *Javascript* passou na requisição. Integrando o AJAX com MVC teremos dois tipos de *view*: a *view* que solicita arquivos XML para o servidor e as *views* que constroem esses arquivos.

Para compreender como a requisição assíncrona com *Javascript* funciona, devemos entender os objetos XMLHttpRequest, responsáveis por ela. Infelizmente esse objeto ainda não está padronizado, sendo possível que em navegadores diferentes ele esteja em diferentes escopos. Para sanar essa incompatibilidade o seguinte código pode ser usado:

```
try
{
  // Firefox, Opera 8.0+, Safari, etc...
  xhr = new XMLHttpRequest();
}
catch (e)
{
  // Internet Explorer
  try
  {
    xhr = new ActiveXObject("Msxml2.XMLHTTP");
  }
  catch (e)
  {
    xhr = new ActiveXObject
("Microsoft.XMLHTTP");
  }
}
```

Figura 6 – *Javascript* para instanciar o objeto XMLHttpRequest

Dentre os métodos do objeto XMLHttpRequest os mais importantes são o *open*, responsável por configurar o metodo e a url da requisição e o *send*, responsável por enviar a requisição. Há tambem alguns atributos que meressem ser comentados: *readyState*, que representa o estado atual da requisição (variando de 0 – não inicializada – à 4 – concluída); *responseXML*, a resposta em XML; *responseText*, a resposta em texto; status, o código do status da URL requisitada e o evento *onreadystatechange*, que é chamado em cada mudança de estado do objeto. Abaixo, o exemplo mais básico de função para a requisição assíncrona:

```
function ajax(url, metodo, parametros) {

  xhr.onreadystatechange = function() {
    var divId = 'conteudo';
    if (xhr.readyState==4 && xhr.status==200)
{

      var div =  document.getElementById
(divId);
      var texto = xhr.responseText;
      div.innerHTML = texto;

    }
  }

  xhr.open(metodo, url, true);
  xhr.send(parametros);
}
```

```
//Exemplo de uso
ajax("pagina.php", "GET",
     "categoria=5&produto=1");
```

Figura 7 – Exemplo de função *JavaScript* para a requisição.

Observe no código acima que ao evento *onreadystatechange* é atribuída uma função que verifica se a requisição foi concluída e se a URL é válida. Se sim, coloca o conteúdo da requisição numa *tag* com id igual a 'conteudo'. Após definir o manipulador do evento, a função '*ajax*' configura a requisição com o método *open* e requisita com o método *send*, passando os parâmetros.

Apesar de AJAX utilizar XML, é comum chamarem de AJAX toda requisição assíncrona feita através do *JavaScript*, que pode ainda retornar HTML, Texto puro, Scripts ou dados em JSON. O exemplo trata um retorno em HTML ou Texto puro que é o mais simples. A resposta em *script* é uma *string* que pode ser transformada em código *JavaScript* através da função eval(). JSON (*JavaScript Object Notation*) é um formato para serialização de objetos em *JavaScript*. O PHP 5 possui nativamente funções que serializam/deserializam objetos dele em JSON (*json_encode*, *json_decode*). Com isso é fácil passar um objeto do PHP para o *JavaScript* de forma bem simples, utilizando o mínimo possível de banda. JSON têm se tornado uma excelente alternativa ao XML no AJAX.

Para facilitar o uso do AJAX e possibilitar outros recursos da WEB 2.0, podemos utilizar bibliotecas como o *jQuery*, o *MooTools* ou o *Prototype*, ou *frameworks* como o *Dojo*, o *Rico* ou o *extJS*.

5. Controller

O *Controller* é o controlador da aplicação, ele é o responsável por controlar o fluxo da aplicação e pode também implementar segurança.

Todas as requisições feitas ao sistema são iniciadas pelo controlador. O usuário acessa o controlador pela URL passando parâmetros, que serão utilizados para determinar o fluxo da aplicação.

Além disso, o fluxo da aplicação sempre será encerrado pelo controlador, por isso, ele se torna bastante útil para implementar segurança, como verificações de níveis de acesso e gravação de *logs* de acesso. Pode ser utilizado também para abrir e fechar uma conexão ao banco de dados, por exemplo.

```
<?
$module = $_GET['module'];
$acao   = $_GET['acao'];

$url = $_GET['module']. "/action/" . $_GET
['acao'] . ".php" ;
require_once($url) ;

$acao    = new $_GET['acao'];
$retorno = $acao->execute();
$view = $_GET['module'] . "/view/frm" . $re-
torno . ".php" ;
require_once($view) ;
?>
```

Figura 8 – Exemplo de Controlador

6. Estrutura de Diretórios

Na raiz da aplicação situa-se o controlador principal. Para cada entidade cria-se um diretório com o nome do mesmo. Dentro de cada entidade, há um arquivo com o nome da entidade, que é o arquivo *Bean*, e o arquivo DAO. Além disso, há um diretório *Actions*, onde se localizam os arquivos de Ação, e por fim o diretório v*iew*, onde são armazenados os arquivos de apresentação, que por padrão tem o prefixo *frm*.

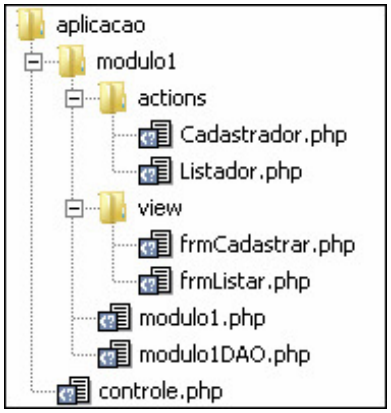
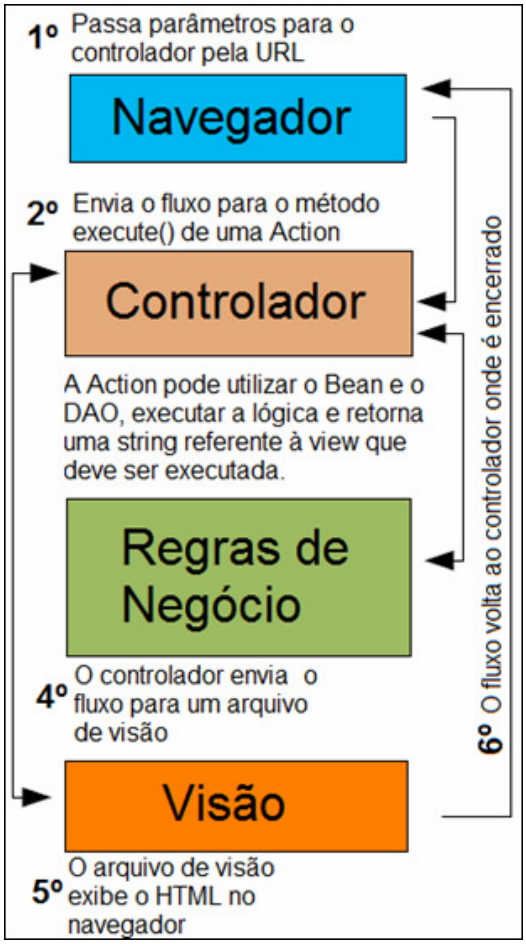


Figura 9 – Exemplo de estrutura de diretórios

7. Fluxo da Aplicação

Na figura seguinte, tem-se o fluxo completo dentre as camadas na execução normal da aplicação utilizando MVC.



Considerações Finais

Quando desenvolvemos para Web, temos um desafio maior em tornar a aplicação interativa e com o código organizado, sem misturar Linguagem de Banco de Dados, Linguagem de Programação e Linguagem de Marcação. Neste artigo mostramos uma maneira de usar conceitos e padrões que se dão muito bem sozinhos e melhor ainda juntos: MVC e AJAX com PHP. Detalhamos o necessário para criar uma aplicação modular que seja de fácil manutenção, agradável e interativa para o usuário, seguindo os padrões WEB 2.0.

Referências e links sugeridos

[PEAR] – http://pear.php.net/
[jQuery] – http://jquery.com/
[MooTools] – http://mootools.net/
[prototype] – http://www.prototypejs.org/
[dojo] – http://dojotoolkit.org/
[Rico] – http://openrico.org/
[extJS] – http://extjs.com/
[XMLHttpRequest] – www.w3.org/TR/XMLHttpRequest
[GOPHP] – http://www.gophp.com.br
[Wikipédia] – http://www.wikipedia.org.br

Almir Neto - almirneto@gmail.com
Almir Neto é desenvolvedor da HP Transportes e é um entusiasta do PHP + MVC. Foi palestrante no *PHP Conference Brasil 2007* além de ter ministrado palestras e mini-cursos no III FGSL e IV FLISOL-GO.

Otávio Calaça Xavier - otaviocx@gmail.com
Entusiasta do Desenvolvimento WEB 2.0, do Software Livre e de Padrões para Web, Otávio Calaça Xavier trabalha como analista desenvolvedor na Câmara de Dirigentes Lojistas de Goiânia. Ministrou o mini-curso "PHP Orientado a Objetos com AJAX" no IV FLISOL-GO e o "Introdução à Google Maps API" no FLISOL-DF 2008.

Almir e Otávio têm projetos em comum e já ministram palestras e mini-cursos em conjunto há algum tempo. Como exemplo, podemos citar o mini-curso "Introdução ao MVC com PHP 5 e Ajax" no Executive IT Meeting (IV FGSL).

Edições anteriores

edição 01 - janeiro/2007 (lançamento)

Edição especial de lançamento apresentando a revista à comunidade. Os artigos abordam temas relacionados a CMS's, Design Patterns, integração com SOA e sessões. No final, uma entrevista com Cristian Pedroso.

edição 02 - março/2007

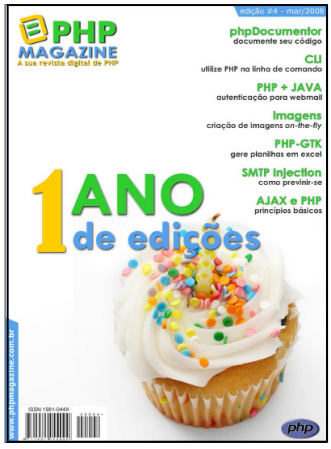
Segunda edição com um acervo maior de artigos. Nesta edição são apresentados alguns números e comentários da repercussão do projeto. Os artigos abordam desde CMS, camadas de persistência até uma avaliação de frameworks. Ao final, Via6 e Rec6 são apresentados como exemplos de cases de sucesso PHP.

edição 03 - junho/2007

A terceira edição surge com um pequeno atraso devido à cobertura de dois eventos. Nesta edição são apresentados alguns números e comentários da repercussão do 1° PHP Road Show e do FISL, em sua 8a edição. Os artigos abordam desde LCMS, segurança, sugestões de codificação até uma ferramenta para desenvolvimento de projetos PHP. A equipe apresenta um pequeno tour em três IDEs conhecidas da comunidade.

edição 04 - março/2008

Nesta edição, selecionamos 7 artigos para leitores de todos os níveis. Os artigos abordam documentação de código fonte, recursos de linha de comando, imagens e geração de planilhas em excel. Na categoria segurança, artigos sobre autenticação de webmail e SMTP Injection.



PHP MAGAZINE
ISSN 1981-044X

www.phpmagazine.org.br

Os números do FISL 9.0

Nesta seção apresentamos alguns números do fórum que a cada ano obtém mais expressão dentre as diversas comunidades de tecnologia.

O evento reuniu mais de 7,4 mil participantes de 21 países no Centro de Eventos da PUCRS, o maior público desde 2000, quando foi realizado o primeiro fisl. Nos três dias do evento houve debates, palestras, troca de informações e experiências, entre grupos de usuários, desenvolvedores, empresas públicas e privadas que compareceram como participantes, expositores ou patrocinadores. Foram 257 palestras, com nomes nacionais e internacionais, que lotaram todas as oito salas destinadas aos encontros.

Para a coordenação do fisl9.0, esta edição com público recorde mostra a importância que o software livre vem conquistando no mundo todo e como o compartilhamento de informações pode beneficiar a toda a sociedade. "Nosso desafio agora será organizar o evento do ano que vem, quando pretendemos marcar de forma ainda mais forte a presença do software livre na vida das pessoas, em função dos 10 anos do fisl", diz Sady Jacques, coordenador geral do Fórum.

A meta da Associação Software Livre.Org, organizadora do fisl, é chegar a 10 mil participantes em 2009. "Sabemos que é um grande desafio, mas estamos nos preparando para isso", afirma Jacques.

Além do público recorde, o fisl9.0 teve a participação de empresas como Província Marista; UOL; Terra; Telefônica; Google; Globo.com; Intel; Sun microsystems; Yahoo Brasil; Comitê Gestor da Internet no Brasil (CGI); Núcleo de Informação e Coordenação do Ponto br (NIC.br); Companhia de Processamento de Dados do Município de Porto Alegre (PROCEMPA); Companhia de Informática do Paraná (CELEPAR); Empresa de Tecnologia e Informações da Previdência Social (DATAPREV); Serviço de Processamento de Dados (SERPRO); Caixa Econômica Federal; Banco do Brasil; Secretaria de Educação a Distância (SEED); Ministério da Cultura; Ministério da Ciência e Tecnologia; Ministério da Educação; Ministério do Planejamento, Orçamento e Gestão; Propus; Locaweb; Serviço Nacional de Aprendizagem Comercial (SENAC); Diginet; iSafe; Kroma Informática; Logins; ProDesk; ArgoHost.net; Brasil Telecom; Powerlogic; Minuano; Cobra Tecnologia; Casa Brasil; Petrobrás; Correios; Ministérios das Comunicações; Ministério da Saúde; Instituto Nacional de Tecnologia da Informação (ITI); Governo do Paraná; Companhia de Processamento de Dados do Estado do Rio Grande do Sul (PROCERGS); Sindicato das Empresas de Informática do Rio Grande do Sul (SEPRORGS);

O fisl contou com apoio das empresas: Baguete; A Rede; Instituto Nokia de Tecnologia (INdT); Associação dos Jovens Empresários de Porto Alegre (AJE); Sindicato dos Bancários; Instituto Projetos Pesquisas; Tangalomango; e Centro de Processamento de Dados (CPD) da Universidade Federal do RS (UFRGS).
Participaram da Mostra de Soluções as empresas: Solis Cooperativa de Soluções Livres; Livraria Tempo Real Informática e Negócios; ZOPING; Trolltech ASA; Moradia e Cidadania; Education Technolog; Red Hat Brasil; Sociedade de Educação Ritter dos Reis; OpenS Tecnologia Ltda; Escola Alcides Maya; Linux Magazine; Unicamp; Universidade Federal de Pernambuco; ThreePointsWeb; NetScience; ParanaCidade; BrOffice.org; LM2 Consulting; PostgreSQL; DBSeller Serviços de Informática; Ory Soluctions Group; FAURGS; Centro de Empreendimentos do Instituto de Informática da UFRGS; CONEXUM; DFL; SOL7; Laboratório de Estudos Cognitivos – (LEC); e Projeto Germinar.

Vamos aos números

Total de participantes: 7.417

- Países: 21 (Alemanha, Argentina, Austrália, Bélgica, Bolívia, Brasil, Canadá, Chile, Cuba, Espanha, Estados Unidos, Finlândia, França, Holanda, India, Inglaterra, Noruega, Paraguai, Portugal, Suíça, Uruguai)
- Todos os estados brasileiros foram representados no fisl.
- Perfil dos participantes: 24% são estudantes e 63% são profissionais e empresários de TI;

Patrocinadores: 41

Mostra de Soluções: 28

Apoiadores: 8

- Destes, 58 são expositores

Submissões de propostas de palestras: 574

Palestras realizadas: 257

Palestrantes: 402

Caravanas: 59 (vindas de 12 estados brasileiros e de 2 outros países – Uruguai e Paraguai)

- Maior caravana: São Paulo, com 47 participantes
- Mais distante: Manaus ;

Grupos de Usuários: 48 (vindos de 8 estados brasileiros e de outros 5 países – Paraguai, Argentina, Uruguai, Bolívia e Chile);

Visitas a TV Software Livre: 3.175

Tráfego de upload da TV Software Livre: 70 Gb

Tráfego de download da TV Software Livre: 179 Gb

Conexões: 19 mil simultâneas

Tráfego de upload/download: 250 Gb

Visitas ao site do fisl durante o evento: 12.159

Arena de Programação Livre

- Fase Remota = 35 inscritos
- Qualifying = 32 presentes (8 equipes de 4 participantes)
- Insanifying = 18 classificados (6 equipes de 3 participantes)

Fonte
http://fisl.softwarelivre.org/9.0/www/node/532

Por Helton Ritter

9º Fórum de Software Livre e *muito PHP*

A linguagem de programação PHP marcou presença no último Fórum Internacional do Software Livre (FISL 9.0), entre os dias 17 e 19 de abril na PUC-RS, em Porto Alegre. Com diversos grupos de usuários e palestras sobre o assunto, o fórum reuniu mais de 7.000 pessoas, entre entusiastas, desenvolvedores, usuários, designers, expositores e voluntários. E, como não poderia deixar de ser, a PHP Magazine esteve lá e conferiu tudo de perto para você.

Ao todo foram três dias de evento que deixaram um desejo de "queremos mais em 2009". Muito bem organizada, as apresentações foram divididas em 14 trilhas, com uma delas chamada "Desenvolvimento: PHP", que contou com 5 palestras de pessoas ilustres da comunidade PHP do Brasil e do exterior.

Presente do início ao fim, a equipe PHP Magazine não podia deixar de registrar as palestras de nosso interesse específico. Agradecemos os autores pelo incentivo e colaboração para este resumo, bem como a colaboração de Fernando Fischer e Ubiratan de Carvalho. Infelizmente, não conseguimos acompanhar a palestra "Desenvolvendo portais com o Drupal: Estudo de caso dos portais dos Democratas". Fica aqui o nosso pedido de desculpas. A seguir, um apanhado dos principais pontos abordados em cada apresentação.

Large Scale PHP

Autor: Rasmus Lerdorf

Rasmus Lerdorf, que desenvolveu a primeira versão da linguagem de programação PHP, apresentou neste FISL um pouco da história do PHP, passando para a otimização de *scripts* e do ambiente do servidor em geral e por último focou em segurança, demonstrando de modo prático um ataque de XSS.

No estado vicioso de nosso trabalho acabamos não lembrando como as coisas começaram, reclamamos da falta de padronização das funções, da capacidade ou não da orientação a objetos, se deveriam ou não deixar de existir os "aliás" para as funções... Imagine que o início disso tudo foi lá por 1994 ou 1995. Naquela época a realidade era outra: outro público com acesso a Internet, outros *softwares* para acessar as páginas. A internet era texto e poucas responsabilidades com dinamicidade.

Com o passar do tempo as exigências aumentaram, a linguagem evoluiu muito e de modo descentralizado, dificultando várias normatizações e resultando em funções com as mais diversas responsabilidades. O incrível é que nem por isso a linguagem deixou de ser fácil. Hoje PHP é o *front-end* de 80-90% das páginas da Internet.

Com aplicações grandes de alta responsabilidade, iniciam-se análises antes inviáveis, começa-se a pensar em otimização máxima do *script*, em devolver o mínimo possível de HTML para o cliente, em analisar em detalhes e medir os tempos do tráfego HTTP e da resposta do servidor, do processamento de cada binário ou biblioteca que estiver envolvida. Aplicações de grande responsabilidade exigem cuidados muito específicos.

Para ajudar-nos existe o **YSlow**, que é um complemento para a extensão **FireBug** que roda no Mozilla Firefox. O YSlow foi desenvolvido pela Yahoo! para facilitar a análise e, com isso, saber em que pondo agir buscando o melhor tempo de resposta.

Precisamos de mudanças em nossos hábitos de programar. A simples substituição de "*require_once*" por "*require*" já representa uma melhora de performance, pois será um processamento a menos. Por isso, tenha um olhar clínico e faça da maneira mais simples possível. Evite o uso de *frameworks* que tendem a ser grandes e complexos quando não são necessários.

Serviços de alta responsabilidades ou apenas sistemas complexos tendem a ter variáveis em tão grande número que em algum momento ficam incontroláveis e neste descontrole pode-se abrir uma brecha na segurança. Por isso, "*never click on a link!*" (nunca clique em um link!), pois você internauta não sabe o que é processado no servidor e no seu navegador no momento daquele clique. Na realidade não há como garantir a segurança.

Quando se abre uma URL estamos sujeitos a *scripts* que agem nos *bugs* do nosso navegador. Estamos sujeitos a sermos usados para propagar *links* maliciosos. A solução novamente é simples: filtrar par garantir que os dados que estamos recebendo realmente são os que esperamos receber.

Lição final: Faça o mais performático que puder, garantindo a integridade do sistema e o máximo de segurança para o usuário. Rasmus deixou os slides no seguinte endereço: http://talks.php.net/show/fisl08 .

Duro de Errar 5.0 - Tratamento, Controle e Configuração de erros em PHP 5

Autor: Marcelio Leal

Marcelio Leal trouxe um assunto que ninguém quer deparar-se: os erros, as exceções e os *bugs* em nossos programas PHP, junto a isso algumas configurações e métodos do PHP para melhorar sua visualização e gerenciamento.

Apresentou-nos os nossos "direitos". O "direito 0" diz que devemos ter a liberdade de errar e conseguir entender o erro. Nesse ponto comentou-se que o PHP é muito bom, pois indica de forma clara qual foi o erro e onde que ele ocorreu.

No arquivo "php.ini" há algumas diretivas relacionadas a erros. Tais diretivas devem ser diferenciadas em ambiente de desenvolvimento, no ambiente de testes e no ambiente de produção. No ambiente de desenvolvimento mostra-se todos os erros, *warnigs* e *notices,* além das possíveis funções que depreciarão e incompatibilidades futuras. Temos assim uma riqueza de informações que nos auxilia a fazer o código da maneira mais perfeita possível. No ambiente de produção a recomendação é não mostrar erros e guardá-los em *logs*, ou tratá-los por funções especializadas.

A customização ainda depende da experiência do desenvolvedor. Um desenvolvedor com mais experiência pode configurar para que vejam apenas os erros que lhes são interessantes. Essa é uma flexibilidade do PHP que pode ser explorada.

O bloco "*try/catch*" foi reforçado, pois, embora seja de um uso muito simplificado, é pouco aplicado na prática. Sabemos que os erros da maneira original apresentam-se sem sentido algum para o usuário e, para piorar, pode se tornar até um problema de segurança, dependendo de como se apresenta o erro em ambiente de produção. Afinal, nada como poder reagir diante de uma situação de erro.

Há ainda a "*triger_error*" em que podemos criar erros em tempo de execução, com mensagens claras (objetivas em relação ao sistema) e tratar com "*set_error_handler*".

Fica clara a necessidade de testar por diversas vezes as rotinas, ou melhor, usar o próprio computador para testar a página. Nesse momento apresenta-se *o phpUnit* e o *Selenium*, ambas ferramentas que permitem automatizar testes.

Marcelio também disponibilizou os slides em http://marcelioleal.wordpress.com/2008/04/22/duro-de-errar-50/ .

Prevenindo XSS: Execute apenas o SEU código

Autor: Er Galvão Abbott

Um dos assuntos mais destacados no momento são ataques por XSS, ou seja, injeção de códigos dentro de nossa aplicação. Er Galvão Abbott, que já havia falando sobre segurança no FISL, este ano voltou a expor o assunto. Galvão também já publicou na 3ª edição da PHP Magazine, não deixe de conferir "Segurança no PHP – Os 6 requisitos mínimos".

Ataques de XSS são tentativas de embutir, ou seja, de fazer nossa página executar códigos (normalmente JavaScript e HTML) de forma intrusiva, causando ou danos visuais ou, na pior hipótese, fazendo processar comandos invisíveis ao usuário, mas que agem de modo obscuro.

Novamente a solução é simples: garantir-se de que os dados que recebemos ou enviamos estão no formato que esperamos que estejam. Tal regra vale para dados vindos por $_COOKIE, $_SESSION, $_POST, $_GET ou $_REQUEST. Também devem ser tratados os dados originados do banco de dados e que chegam até nossa aplicação

Costumamos nos preocupar com os dados que recebemos por parte do usuário, mas acabamos esquecendo de tratar os dados que são inseridos em nossa aplicação vindos do banco de dados e simplesmente exibidos. Se esses dados tiverem *scripts*, eles provavelmente serão executados e podem comprometer o sistema ou dados.

Segundo Galvão, a incapacidade técnica não é o único problema. O que ocorre é que o erro de segurança é identificado, mas não corrigido por falta de tempo e por menosprezarmos os danos que podem ser causados. Cabe nesse momento um pouco mais de compromisso nos trabalhos já finalizados por parte do desenvolvedor. Isso não é especifico para PHP, pois o problema é genérico.

Os slides estão no site do autor: http://www.galvao.eti.br em "keynotes".

Implementando Enterprise Patterns em PHP

Autor: Pablo Dall'Oglio

Pablo Dall'Oglio brincou que contava sua idade pela quantidade de participações na FISL e falou de um assunto que muito nos interessa: os *patterns* de acesso a dados. Em sua palestra foram comentados e demonstrados os principais modelos.

A atualidade corporativa requer que o desenvolvedor conheça alguns nomes que cercam a área. Conhecimento importante, sem dúvidas, pela objetividade que proporciona, uma vez que a regra de negócio não precisa

ser explicada, basta ser denominada, por já ser um padrão. Padrão este que, por vezes, usamos e não sabemos.

De forma simplificada, um padrão é criado quando uma determinada tarefa ou modo de resolvê-la é usado por mais de uma vez. Objetiva-se, então, que seja aplicável sempre que aquela situação ocorrer. *Design patterns* não é invenção da área de informática.

Os padrões demonstrados favorecem o uso do modelo MVC, separando a persistência da camada visual e dos controles. No caso usamos bancos de dados relacionais. Então, como fazer para que um padrão orientado a objeto seja usado em um banco de dados que não seja? A solução é colocar nas classes as regras de negócio e separar até nas classes de persistência o que diz respeito direto ao banco de dados e o que são regras. Neste momento entra o conceito de *gateways*.

Os *patterns* apresentados foram: *Table Data Gateway*, *Row Data Gateway*, *Active Record*, *Layer Supertype* e o *Repository Pattern.* Os padrões foram apresentados partindo-se das premissas mais simples até se chegar em um modelo aplicado que realmente provê flexibilidade e organização do código-fonte.

Os slides estão disponíveis no site do Pablo Pablo Dall'Oglio: http://www.pablo.blog.br/talks. Recomenda-se a leitura do livro do autor: "PHP Programando com Orientação a Objetos".

Conclusões

Com a migração de softwares desktop para internet, tentando aproveitar-se do "ambiente colaborativo", há cada vez mais aplicações com necessidade de alta performance, escalabilidade, disponibilidade e segurança. O site virou aplicação e a rede de pesquisa da ARPANET virou uma oportunidade para negócios.

Em geral, as palestras do fórum na área de PHP não trouxeram novidades, mas todas elas focaram na qualidade. Que no próximo FISL possamos ver os resultados destes esforços!

Helton Eduardo Ritter - heltonritter@gmail.com
Bacharelando em Sistemas de Informação pela Sociedade Educacional Três de Maio – SETREM, pela qual também é Técnico em informática formado em 2006 e funcionário desde julho do mesmo ano.
Moderador PHP do fórum ScriptBrasil, Helton é também colunista do Imasters e faz trabalhos como free-lancer.