DBPHP: Biblioteca para Comunicação entre o Sistema e a Base de Dados

Instituição: Universidade Federal do Rio Grande do Sul
Autores: Fernando Henrique Canto, Augusto Dias Pereira dos Santos
Tipo: Trabalho técnico consolidado
Área: Sistemas de Informação: Padrões de desenvolvimento

INTRODUÇÃO

Através deste trabalho, apresentaremos uma biblioteca desenvolvida pelo CPD da UFRGS para controlar a comunicação feita entre os sistemas desenvolvidos e o banco de dados da Universidade. A idéia era criar uma camada separando o desenvolvedor de sistemas das funções de acesso ao banco de dados, utilizando os conceitos de orientação a objetos, ferramentas e artifícios da linguagem PHP como expressões regulares, e idéias inspiradas em outras linguagens de programação e ambientes de desenvolvimento, como Delphi e ASP.

A biblioteca é utilizada hoje em diversos sistemas, e mostrou-se extremamente útil e importante para a segurança dos dados, a padronização, a legibilidade e a organização do código. Nosso interesse é de divulgar, não apenas a nossa capacidade de produzir uma ferramenta própria, mas também a importância de criar soluções para problemas críticos do desenvolvimento de software que garantem mais segurança, e também a oportunidade de repensar a maneira como os sistemas são produzidos, compartilhando assim idéias e soluções entre as instituições e os desenvolvedores.

MOTIVAÇÃO

A idéia para o desenvolvimento da biblioteca surgiu em 2005, quando os sistemas para a Web da Universidade começaram a ser desenvolvidos na linguagem PHP, e não mais na linguagem ASP. Nesta migração, começamos a perceber as particularidades da linguagem, que resultam em algumas deficiências no código. Mas além disso, procuramos estudar também os benefícios que a linguagem traz com sua sintaxe e com seus recursos, que poderiam ajudar-nos a encontrar soluções de mais alto nível, para o uso dos próprios programadores.

A linguagem PHP disponibiliza bibliotecas específicas para cada sistema de banco de dados, que consistem principalmente em uma série de funções a serem usadas pelos desenvolvedores em seus *scripts*. Várias dessas funções geram, como valor de retorno, variáveis de tipo específico para a integração do sistema com a base de dados. Por exemplo: a conexão com o banco e o resultado de uma consulta SQL são guardados em variáveis, e fica a cargo do programador manipulá-las.

Como resultado, a programação tende a se tornar desorganizada, à medida que as transações com o banco ficam dispersas em múltiplas variáveis e se confundem com a lógica do programa. Além disso, o programador se torna encarregado de fazer, em seu próprio código a integração com o sistema de banco de dados específico que está sendo usado. No caso de uma migração para outro SGBD, a readaptação do código pode se tornar demorada e suscetível a erros, pois embora os nomes dos comandos tendem a ser semelhantes (e.g. *mysql_query()*, *pgsql_query()*, *mssql_query()*, etc.), algumas dessas funções podem não estar disponíveis para um determinado banco de dados.

A consequência mais grave, porém, é de deixar a cargo do programador a montagem dos comandos SQL, que é feita nos *scripts* através da montagem de *strings*, enviadas para o banco através de uma conexão. O problema é que, na maioria dos casos, essas consultas são feitas de acordo com dados enviados por formulários HTML, seja por opções feitas pelo usuário, ou por dados digitados no teclado. Isso abre uma comunicação direta entre o usuário final e o banco de dados, e para evitar possíveis inconsistências ou ataques, os dados recebidos devem ser devidamente tratados.

Quando se tem uma base diversa de programadores, porém, esse problema torna-se ainda maior. Cada programador pode ser motivado a criar sua própria solução, muitas vezes incompatível, ou até incompreensível pelos outros programadores. Essa falta de padrões gera desorganização, dificulta o processo de manutenção, e torna os sistemas pouco confiáveis, pois algumas dessas soluções podem conter brechas e falhas, inclusive em porções críticas do código. Existe um tipo de ataque, denominado *SQL injection*, que se baseia em procurar e explorar exatamente esse tipo de brecha. Muitas vezes, uma simples tela de *login* pode ser usada para esses fins, e a existência dessas falhas pode se dar por maus hábitos de programação, falta de padrões, ou apenas por descuido.

Como desenvolvedores, sentíamos a necessidade de possuir um idioma comum de programação, que permitisse que esses problemas fossem resolvidos de forma padronizada e centralizada entre os diversos sistemas, mas que também fosse natural aos nossos hábitos de programação e pudesse atender às nossas necessidades mais urgentes.

DO ASP PARA O PHP

Os primeiros sistemas para a Web desenvolvidos no CPD utilizavam o ASP, linguagem originária do VB Script da Microsoft. Essa linguagem possui um recurso nativo de integração ao banco de dados, em que a conexão com a base de dados é representada em um objeto, e cada consulta SQL é encapsulada em um objeto separado. Por ser um recurso nativo da linguagem, esse artefato se tornou um padrão bastante natural e de fácil compreensão, além de ajudar a tornar o código mais limpo, reduzindo o número de variáveis e embutindo nelas as operações sobre as consultas (percorrer, buscar valor do campo, verificar fim de arquivo).

A perda desse recurso foi percebida na migração para o PHP. À medida que os *scripts* se tornavam longos e complexos, a dependência em funções isoladas, argumentos e variáveis soltas, muitas vezes tornava o código confuso. Rapidamente chegamos à conclusão de que o uso de classes e objetos para envolver as consultas ajudaria a tornar o código mais conciso.

DO DELPHI PARA O PHP

A nossa maior preocupação deu-se, porém, com a montagem das consultas em si. Percebemos a falta de um recurso na linguagem PHP que é presente no ambiente Delphi com a linguagem Object Pascal, que era utilizado para desenvolver os sistemas corporativos da Universidade até a adoção do modelo orientado à Web.

No Delphi, é possível definir as consultas estaticamente nos formulários que compõem o sistema, e os dados externos são anexados às consultas no momento em que elas são disparadas. Isso é feito com o uso de parâmetros, demarcados no texto da consulta com identificadores precedidos de dois pontos (:). Devido ao fato da linguagem Object Pascal ser fortemente tipada, o programador não precisa de preocupar com detalhes de formatação e tratamento de dados ao inserir os dados nos parâmetros, pois esse tratamento é feito pelo próprio sistema. Isso garante segurança às consultas e simplifica o código-fonte.

No PHP, é complicado implementar esse tipo de facilidade sem usar funções e recursos mais avançados. Percebemos que a necessidade mais urgente era de ter uma camada entre os dados e as consultas em si, que fosse responsável pelo tratamento e pela validação dos dados a serem enviados à base de dados, de forma que o usuário não fique encarregado de nenhum tratamento em especial, garantindo assim proteção à base de dados, e tornando o processo de desenvolvimento e manutenção mais produtivo, menos desgastante e mais intuitivo.

O PROJETO

O projeto e a implementação inicial foi iniciada junto com os desenvolvedores Lúcio Iglezias Pacheco e Renato Krause Gonçalves Júnior. A idéia básica era de, através do modelo de orientação a objetos da linguagem PHP, desenvolver uma classe que fosse responsável por montar e efetuar as consultas, usando um modelo inspirado no Delphi de inserção de parâmetros. Esse modelo permitiria também que a lógica de execução das consultas e recuperação de dados fosse encapsulada na classe, por sua vez baseado no modelo do ASP.

Nossa experiência com as dificuldades comuns enfrentadas nos processos de desenvolvimento e manutenção dos sistemas, bem como o conhecimento das demandas mais comuns dos usuários da Universidade, nos permitiram definir as prioridades mais altas e os recursos essenciais da biblioteca. A modelagem e a implementação da biblioteca seriam baseados na ordem assim estabelecida. Esses foram os primeiros passos no desenvolvimento da biblioteca hoje denominada DBPHP.

O recurso de parâmetros foi planejado para seguir o padrão do Delphi, em que o sinal de dois pontos (:) identifica um parâmetro, consistido por um nome escolhido pelo programador. Esse nome seria utilizado pelo método de atribuição para associar o parâmetro com o dado correspondente, e realizar a substituição no comando final. Esse mecanismo, além de ser a demanda mais urgente, seria também a parte mais desafiadora e mais crítica da implementação final.

A IMPLEMENTAÇÃO

Devido ao modelo de variáveis não-tipadas da linguagem PHP, algumas adaptações ao modelo do Delphi tiveram que ser feitas. O *tipo* de cada parâmetro deve ser identificado na própria consulta, pois exigir que o tipo seja inferido pela variável recebida pode levar a um código extremamente obscuro e suscetível a erros, além de provocar complicações no tratamento de dados mais complexos, como data e hora. Os tipos de dados foram definidos de acordo com os tipos de registro presentes nos bancos de dados, e.g. texto, número (inteiro e decimal), data e hora.

Para a identificação dos parâmetros e de seus tipos, e também para a validação dos dados da aplicação, utilizamos expressões regulares, um recurso já extensamente utilizado na análise léxica de interpretadores e compiladores, baseado na teoria de linguagens formais, e disponível em uma grande quantidade de linguagens. As expressões regulares foram a resposta definitiva para um problema cuja solução poderia demorar meses para ser implementada. O reconhecimento de padrões na consulta é praticamente automático, e o processo de substituição de dados tem uma lógica extremamente simples.

A validação dos dados é feita de forma individual para cada tipo de dados, cada um com regras bem definidas. Por exemplo: o tratamento de números evita que qualquer caractere não numérico seja incorporado na consulta; o tratamento de texto elimina a confusão entre apóstrofes que delimitam o *string* na sintaxe SQL e os apóstrofes presentes no texto enviado pelo usuário; o tratamento de data faz automaticamente a conversão da data em formato brasileiro (dd/mm/aaaa), recebido por padrão, para o formato americano (mm/dd/aaaa) reconhecido pelo banco de dados, além de recusar datas inválidas (e.g. 31/04/2009 ou 29/02/2010).

Todas essas regras, logicamente, precisam estar de acordo com as regras léxicas e sintáticas do banco de dados em questão. É importante ressaltar que o padrão ANSI da linguagem SQL não é inteiramente adotado na maioria dos bancos de dados disponíveis, portanto as diferenças entre as sintaxes podem exigir adaptações a essas regras. No caso de uma migração de banco, o uso dessa ferramenta pode facilitar enormemente o processo de adaptação dos códigos, que deve ser feita manualmente em cada *script* caso não haja uma ferramenta do tipo.

A biblioteca procura também evitar a necessidade de uma sintaxe longa, repetitiva e confusa. Para isso, o programador pode fazer a substituição de vários parâmetros de uma vez só, utilizando *arrays* como dicionários de dados. Essa prática se torna natural para o modelo da Web, no desenvolvimento de um formulário, por exemplo. Todos os dados contidos em um formulário (delimitado pela tag FORM em HTML) são enviados para o script PHP na forma de um vetor, denominado *$_GET* ou *$_POST*, dependendo do método escolhido. Este mesmo *array* pode ser fornecido diretamente para o método da classe, sem que o programador precise manipular cada dado individualmente.

Seguindo essa demanda, foram criados os métodos que permitem executar e percorrer consultas, recuperar os dados do registro atualmente selecionado, testar condição de fim de arquivo, entre outros. A própria nomenclatura desses métodos tornam a leitura do código mais limpa e clara, sem a margem para dúvidas e erros que a sintaxe baseada em funções nativas do PHP tendem a gerar. Isso também cria a noção de uma consulta como uma entidade individual e independente, o que facilita o entendimento do código-fonte e permite uma detecção de erros mais rápida.

Devido ao escopo que a biblioteca ia obtendo, decidiu-se separá-la em duas classes: a classe *consulta* representa exatamente um comando SQL e seus resultados, e a classe *db* representa a conexão com o banco de dados. Foi necessário fazer essa comparação para não criar uma confusão entre conceitos que, embora estejam intimamente relacionados, são completamente distintos.

A IMPLANTAÇÃO

Começamos a desenvolver sistemas utilizando a biblioteca DBPHP, em ambiente de teste, assim que determinamos que ela estava robusta o suficiente para atender nossas necessidades. Essa fase de testes ajudou muito a detectar dificuldades e encontrar melhorias que mantivessem a idéia e o propósito original, mas que aumentariam sua facilidade de entendimento e utilização.

A biblioteca apresentou reduzidíssimos problemas ao ser implantada no ambiente de produção, e os sistemas antigos começaram a também ser adaptados. Devido a natureza inobstrusiva da biblioteca, era perfeitamente possível adaptar um sistema aos poucos, *script* por *script*, sem afetar o funcionamento das partes antigas.

Nos sistemas novos, os desenvolvedores logo sentiram as vantagens da nova sintaxe e do paradigma orientado a objetos trazidos pela biblioteca. Embora alguns “atalhos” de programação tiveram de ser erradicados, a clareza do código-fonte trouxe grandes benefícios para os processos de manutenção. Além disso, o fim desses “atalhos” foi também compensado pela eliminação da necessidade de soluções individuais, algumas vezes idiossincráticas, para os problemas de validação de dados e montagem de consultas, o que eliminou também muitos erros em tempo de execução, percebidos pelos usuários finais.

Outro exemplo dos benefícios surgiu quando, em 2007, o CPD mudou sua plataforma de banco de dados do Sybase para o MS SQL Server. O uso da biblioteca minimizou, e em alguns casos anulou, os esforços de adaptação do código.

O uso contínuo da DBPHP permitiu a descoberta de novos recursos e novas idéias, e a biblioteca continua em evolução. Os esforços se voltaram para tornar a sintaxe ainda mais intuitiva e mais amigável, evitando redundâncias e argumentos desnecessários. Os recursos oferecidos pela linguagem, e até mesmo a teoria e os conceitos da orientação a objetos, permitem um maior entendimento do uso da biblioteca e de seu funcionamento interno.

CONCLUSÃO

O saldo da desenvolvimento e da utilização da biblioteca DBPHP foi extremamente positivo. Não só obtivemos uma ferramenta simples e robusta para solucionar uma questão crítica do desenvolvimento de sistemas, mas também encontramos um idioma próprio, originado das dificuldades e dos anseios que tínhamos de acordo com as demandas da Universidade. Possuímos nela uma possibilidade de suprir essas peculiaridades com soluções cada vez mais abrangentes e naturais, de forma a garantir cada vez mais segurança, tanto para os dados, quanto para as aplicações que os manipulam. O usuário obtém um sistema mais confiável, e o programador obtém um código mais claro e compreensível.

Outra questão importante, também, é a importância da biblioteca não no nível prático, mas no nível conceitual. A separação dos dados da aplicação final é algo que merece grandes considerações, por permitir que os desenvolvedores e os projetistas enxerguem o sistema em níveis de abstração mais altos. A complexidade dos sistemas e dos processos automatizados exigem que os sistemas sejam vistos como módulos e processos desacoplados e individuais, e não apenas como fluxos e processamento de dados inter-relacionados e misturados. A biblioteca DBPHP oferece uma visão diferenciada da relação entre o programa e o dado, e dá a possibilidade de cada vez mais expandir e aprofundar essa visão.

Entendemos a biblioteca como uma contribuição à comunidade das IFES, não apenas como uma ferramenta a ser utilizada, mas como uma idéia a ser estudada. Esperamos que este trabalho sirva como o início de uma colaboração e uma troca de idéias mútua entre as instituições, em prol de um ambiente mais seguro e mais produtivo.

REFERÊNCIAS

Manual do PHP. Disponível em http://www.php.net/manual/pt_BR/.