A háromrétegű szerkezet (2. rész)

Szerzőnk további tanácsokat ad a háromrétegű felépítés használatához.

A múlt hónapban megkezdtük az ismerkedést a webes alkalmazásainkhoz illeszkedő háromrétegű felépítéssel. Egy középső objektumréteg segítségével különválasztjuk az adatbázis-kiszolgálót magától a webalkalmazástól, ezzel jelentősen leegyszerűsíthetjük az alkalmazás ügymenetét. Ezenkívül az alkalmazás és az adatbázis közötti réteg lehetővé teszi számunkra, hogy ugyanezt a középső réteget nem webes alkalmazásainkban is felhasználhassuk, sőt, az adatbázist anélkül módosíthatjuk, hogy ennek bármiféle hatása lenne az alkalmazásra.
Múlt havi írásom végére felépítettünk egy egyszerű középső réteget, mely képes kapcsolatot tartani a PostgreSQL adatbázisban létrehozott emberek és találkozók táblával. Most az ezen objektumok felhasználásával elkészíthető webes alkalmazásokkal ismerkedünk meg. Látni fogjuk, hogy a webes réteg soha nem éri el közvetlenül az adatbázist.

A webes alkalmazásréteg

Egy tökéletes világban a webes alkalmazásréteget bármely nyelven elkészíthetnénk, és a középső réteggel egy általánosan elfogadott protokoll biztosítaná a kapcsolattartást. A világ azonban nem elég fejlett ehhez, így a webes alkalmazásréteg kiválasztásakor a használt objektumréteg lesz a döntő.
Objektumainkat Perlben hoztuk létre, így e nyelv használatával kell kialakítanunk az alkalmazásréteget is. A CGI-programokkal kapcsolatos nehézségek elkerüléséért – mivel jelen esetben az Apache mod_perl-je sem hozna elegendő teljesítménytöbbletet – a már ismertetett Perl-alapú sablon- és alkalmazásfejlesztő környezetet, a Masont (Linuxvilág 2000. november, 59. oldal) fogjuk használni.
A Mason-elemek szükséges esetben a Perl-eljárásokba fordítódnak, ezek pedig újabb fordítással Perl opkódokká alakulnak. Ezeket az Apache mod_perl modulja gyorstárazza, és így sokkal gyorsabban elindíthatók, mintha CGI-t használnánk.

Személy hozzáadása

Első webalkalmazásos példánkban az adatbázist egy új személy bejegyzésével bővítjük. Ehhez két Mason-elemre van szükségünk: az egyik egy HTML kérdőív (ez lehet statikus is), a másik pedig magát a műveletet végzi el. A feladathoz a középső réteg emberek objektumát használjuk, mely az adatbázishoz kapcsolódva megkísérli új sor létrehozását. E két elem egyszerű változatát a *listán* is láthatjuk. Az összes lista túl sok helyet foglalna el, így azokat az ➲ ftp://ftp.ssc.com/pub/lj/listings/isue82 címről tölthetjük le.
A HTML kérdőív (add-person-form.html) a név–érték párjait az add-person.html-nek adja át, mely egy emberek nevű példányt készít belőlük, majd a new_person eljárás meghívásával új személy bejegyzését hozza létre:

```
my $success = $people->new_person
       (first_name =>$first_name,
       last_name => $las_name,
       country => $country,
       email => $email);
```

Ha $success igaz, akkor egy új személy került az adatbázisba, a $people->new_person-nak átadott értékekkel. Egyébként a meghívás sikertelen volt.
Ez azonban túl egyszerű módszer annak eldöntésére, hogy a művelet lezajlott-e vagy sem: a felhasználók számára nem egy igen/nem típusú visszajelzést célszerű adnunk. Sokkal ötletesebb megoldás, ha a látogatóval közöljük, hol rontotta el, és ennek megfelelően kijavíthatja a hibát. Ha az adatbázis belső hibája ugyanazt a hibaüzenetet juttatja el a felhasználóhoz, mintha egy újabb személy beillesztését kísérelte volna meg egy már használt levélcímre, akkor ezt nagyon nehéz kiküszöbölni. Ezért webes alkalmazásunk a bevitt adatokat még a középső réteghez való továbbítás előtt ellenőrzi. Minél több ellenőrző eljárást helyezünk el itt, és a rendszer minél több hibaüzenet megjelenítésére képes, annál jobb.
Az add-person.html összetevőnk két egyszerű ellenőrzést végez el. A Mason <%args> szakasza használatával minden adatot kötelező megadni. Ha egy kérdőív az adatait az add-person.html felé továbbítja, azok addig nem kerülnek feldolgozásra, míg a felhasználó az összes adatot be nem írja. Az értékeiket az add-person.html számára átadó HTML kérdőíveknek az összes elemet továbbítaniuk kell, különben a Mason megtagadja a kérelem teljesítését, és hibaüzenetet jelenít meg. Ezt nem az adatokat hiányosan beíró felhasználók fogják látni, hanem nálunk jelenik meg akkor, ha valamelyik <input> tagot kihagyjuk.
Amint a Mason-elemek működnek, biztosak lehetünk abban, hogy megkaptuk a megfelelő név–érték párokat. De vajon érvényes adatokat tartalmaznak-e? Az add-person.html fájl legelején lévő parancs segítségével ellenőriztük, hogy a $people->new_person meghívásához használt négy érték nem üres. Ha ezek bármelyike hiányozna, a látogatót felkérjük az adat begépelésére.
Azt is ellenőriztük, hogy a levélcímek érvényesek-e. A *listán* látható kifejezés nem illeszkedik minden levélcímre, de ehhez a példához megfelel. A szabálytalan levélcímmel próbálkozó felhasználók hibajelzést kapnak, ebből megtudhatják, hogy mit kell módosítaniuk.
Miután biztosak lettünk abban, hogy a kapott értékek értelmezhetők, meghívhatjuk a $people->new_person-t. Figyeljük meg, hogy az add-person.html mindezt anélkül teszi, hogy az adatbázissal közvetlenül kapcsolatot tartana. A DBI kétségtelenül nagy részt vállal a $people->new_person minden egyes meghívásában, de mindez a színfalak mögött történik, és Mason-elemeinknek nem kell foglalkozniuk ezekkel. Ha a people objektumot kijavítottuk, akkor biztos, hogy nem fogunk SQL-hibákba ütközni.

Az adatok változtatása

Miután áttekintettük, hogy miként illeszthetünk új személyt az adatbázisba a people objektum felhasználásával, próbálkozzunk valami bonyolultabbal: az update_first_name eljárással módosítsuk egy személy keresztnevét (a példák az ➲ ftp://ftp.ssc.com/pub/lj/listings/issue82/ címen érhetőek el, a fontosabb részeket a *3.* és *4. listában* találjuk). Ezt az eljárást csak akkor hívhatjuk meg, ha már kiválasztottuk a személyt, tehát a kérdőívünknek ezt lehetővé kell tennie.
Bár csábító lehetőség, hogy a személyek kiválasztását a név vagy a levélcím begépelésével oldjuk meg, ennek ellenére e módszert – a félre-

Lista add-person-form.html

```
<!-- -*- mmm-classes: mason -*- -->
<HTML>

<Head><Title>Add a new person</Title></Head>

<Body>
<H1>Add a new person</H1>

<Form method="POST" action="add-person.html">

<table>

  <tr>
    <td>First name</td>
    <td><input type="text" name="first_name"></td>
  </tr>

  <tr>
    <td>Last name</td>
    <td><input type="text" name="last_name"></td>
  </tr>

  <tr>
    <td>Address</td>
    <td><input type="text" name="address1"></td>
  </tr>

  <tr>
    <td>Address (line 2)</td>
    <td><input type="text" name="address2"></td>
  </tr>

  <tr>
    <td>E-mail address</td>
    <td><input type="text" name="email"></td>
  </tr>

  <tr>
    <td>City</td>
    <td><input type="text" name="city"></td>
  </tr>

  <tr>
    <td>State</td>
    <td><input type="text" name="state"></td>
  </tr>

  <tr>
    <td>Postal code</td>
    <td><input type="text" name="postal_code"></td>
  </tr>

  <tr>
    <td>Country</td>
    <td><input type="text" name="country"></td>
  </tr>

  <tr>
    <td>Comments</td>
    <td><input type="text" name="comments"></td>
  </tr>

</table>

    <input type="submit" value="Add this person">

</Form>
</Body>

</HTML>
```

gépelésekből adódó hibák elkerülése végett – nem ajánlom. A kiválasztást egy <select> lista segítségével végezzük inkább el, így kizárható annak lehetősége, hogy egy felhasználó olyan személy levélcímét, vagy bármely más adatát írja be, aki nem is szerepel az adatbázisban. A módosítani kívánt keresztnévhez tartozó személyt mindenképpen valamilyen egyedi adat segítségével kell azonosítanunk, de máris megjegyzem, hogy kicsit személytelennek tűnik, ha egyszerűen egy levélcímmel kell dolgoznunk. Megoldásként azt találtam ki, hogy az emberek objektumában (People.pm) új eljárást hozok létre (get_names_and_addresses, lásd az *5. listát* a már említett FTP-címen). Ez egy tömbhivatkozásokból álló listával tér vissza, melynek minden eleme névből és levélcímből áll. Az előbbit egyedi azonosítóként (az <option> tagban a „value” kapcsoló értékeként), az utóbbit pedig a megjelenítéshez használhatjuk. Átnézzük a levélcímeket és egy, az alábbihoz hasonló <select> listát készítünk belőlük:

```
<select name="email">
% # ftfutunk a neveken és az e-mail címeken és
kiírjuk azokat.
% foreach my $info (@names_and_addresses) {
  <option value="<% $info->[1] %>"><% $info->[0]
%>
% }
</select>
```

A felhasználók a többi személyi adatot is hasonló módon szerkeszthetik. Ha a személy azonosítására szolgáló adat valóban egyedi, akkor a személyhez tartozó bármelyik adatot egy hasonló kérdőívvel módosíthatjuk.

Találkozó hozzáadása

Miután áttekintettük, hogy miként használhatjuk a people objektumot az adatbázis tábláinak közvetett módosítására, térjünk rá a találkozók kezelésére, ezeket az appointments elem végzi. Ez az objektum lehetővé teszi, hogy adott személlyel, adott időpontban esedékes találkozót bejegyezzünk.
Ehhez ismét két elemre lesz szükségünk. Az első egy HTML kérdőívet készít, (add-appointment.html, a *6. lista* a már említett FTP-címen) ennek segítségével a felhasználók új találkozót jegyezhetnek be a rendszerbe. A személy kiválasztása egy <select> listáról történik. (Megjegyzés: ha ez „éles” feladat lenne, akkor a <select> listát külön egységként valósítanám meg, és más egységre bíznám, hogy a címjegyzékben lévő adatokat felkínálja.) Emellett tudnunk kell a találkozó kezdési és befejezési időpontját. Ismét a listából választást javasolnám, hiszen így elkerülhetjük a különböző dátumalakokból eredő félreértéseket.
Az itt látható Mason-kód a hónap, nap és év megadásához szükséges három <select> listát állítja elő. A @months és a @years előre meghatározásával a kódot olvashatóbbá tehetjük, és a rendszert könnyen felkészíthetjük a későbbi változtatásokra:

```
<select name="begin_month">
% foreach my $month (@months) {
        <option value="<% $month %>"><% $month %>
% }
</select>
<select name="begin_day">
% foreach my $day (1 .. 31) {
        <option value="<% $day %>"><% $day %>
% }
</select>
<select name="begin_year">
% foreach my $year (@years) {
        <option value="<% $year %>"><% $year %>
% }
</select>
```

A második összetevő (add-appointments.html; a *7. lista*) lehetővé teszi, hogy új bejegyzéssel bővítsük a találkozók naptárát. Egy <%args> szakaszban ellenőrzi, hogy az add-appointment-form.html-ből minden szükséges név–érték párt továbbítottunk, ezt követően pedig ugyanazokat az alapvető ellenőrzéseket hajtjuk végre, melyeket a többi elem esetében már bemutattunk.

Ez most tényleg három réteg?

Az eddigiekben láthattuk, hogy milyen egyszerűen készíthetünk háromrétegű webalkalmazásokat. Jogos a kérdés: a bemutatott példa valóban háromrétegű?

E szerkezet a régebbi modell, az „ügyfél–kiszolgáló” felépítés hiányosságainak hatására jött létre. A jelenlegi adatbázisok és webkiszolgálók tökéletes példái az ügyfél–kiszolgáló szerkezetnek. Ahogy ez a kifejezés két számítógépre utal, a háromrétegű szerkezet három különálló számítógépet jelent, tehát minden rétege egy-egy gépen található. Elmondhatjuk, hogy a példában vizsgált háromrétegű alkalmazást valóban három réteg alkotja, ezek önálló feladatokat végeztek, és lehetővé tették, hogy az alkalmazás és az adatbázis egy „középső réteg” közvetítésével tartson kapcsolatot egymással. Azonban az alkalmazás- és a középső réteg ugyanazon a gépen fut, és ezek elkülönítésére nincs is igazán lehetőség. Ha a webalkalmazásra nagyon sok kérelem kiszolgálásának feladata hárul, akkor egyszerűen egy vagy több azonos felépítésű Apache-ot futtató gépet kell illesztenünk a rendszerbe, de az alkalmazás- és a középső réteg objektumait nem helyezhetjük el külön gépeken.

Reményeim szerint sikerült bemutatnom a háromrétegű szerkezet előnyeit a szabványos API-kat igénylő programozó szemszögéből. A vázolt példa azonban nem tekinthető az elmélet tökéletes megvalósításának. Ehhez ugyanis távoli eljáráshívásokra (RPC) lenne szükségünk, melyek segítségével az egyik számítógép a másik gép valamelyik programelemét indíthatja el. Az RPC sokkal egyszerűbben használható, mióta megjelent a SOAP (Single Object Access Protocol, azaz egyszerű objektumelérési protokoll), de ez újabb gondokat szült, ráadásul újabb adatátviteli protokollal is meg kell barátkoznunk.

Míg a rétegek értelmezéséről elmélkedünk, talán figyelembe kellene vennünk azt is, hogy az itt bemutatott alkalmazás valóban három réteget használ, csak ebben a megfogalmazásban máshogy határozzuk meg a „réteg” fogalmát. Az „adatbázis, középső réteg, webkiszolgáló” helyett az „adatbázis, böngésző, webkiszolgáló” hármast is lehetne három rétegnek tekinteni – csak az a baj, hogy ezt bárki elmondhatja magáról, aki valaha írt már adatbázissal kapcsolatot tartó CGI-programot.

Sőt, ha már itt tartunk, újabb rétegeket is játékba lehetne hozni, csak hogy tovább bonyolódjon a helyzet. Mi van például az adatbázis tárolt eljárásaival, triggereivel és nézeteivel? Bár a tárolt eljárások nem tekinthetők fizikai rétegeknek, azért jól jönnek, ha egy adatbázissal dolgozó objektum- vagy alkalmazásréteget fejlesztünk. Meg merem kockáztatni, hogy a tárolt eljárások még jobbak is a középső rétegnél, hiszen közvetlenül az adatbázisban futnak le, és előre le vannak fordítva, tehát általában gyorsak.

A kódot futtathatjuk a webböngészőn is, például a JavaScripttel. Általában minden ügyfelem figyelmét felhívom arra, hogy ne használja ezt a hibáktól hemzsegő, megbízhatatlan és egyáltalán nem biztonságos nyelvet, ennek ellenére a böngészőkön kizárólag a JavaScript segítségével futtathatunk könnyedén programokat.

Ha webalkalmazásunk kapcsolati adatbázist, tárolt eljárásokat, középső réteget, webalkalmazás-réteget és ügyféloldali JavaScriptet is használ, akkor hány rétegünk is van? Valószínűleg még mindig három, de az igazság az, hogy teljesen mindegy, hogy minek nevezzük ezeket. Végtére is a tökéletes megoldás mindig az, ha a feladatnak leginkább megfelelő rendszert választjuk, és a jövőbeli bővítésekre is felkészülünk a tervezéskor. Így a rendszer anélkül is tökéletesen működik, hogy a legfrissebb, legmenőbb elnevezésekkel kellene dobálóznunk.

A háromrétegű szerkezet gondjai

Egyszerű példánk áttekintése után hadd szóljak a háromrétegű szerkezettel kapcsolatos gondokról. Nem állítom, hogy a rendszer alapjaiban véve rossz, de tudnunk kell, hogy nem maga a tökély. A legtöbb megoldáshoz hasonlóan csupán bizonyos körülmények között válik be igazán. Sok esetben, ha a tervezést és a megvalósítást külön munkatársakra bízzuk, az sokat könnyíthet a feladat nehézségén – ezt a webalapú határidőnapló rendszer esetében is láthattuk. Egy ember is képes megírni a szükséges kódot, de ketten hamarabb végeznek, ehhez azonban állandó kapcsolat kell.

Mint bármely mérnöki terv esetében, itt is engedményeket kell tennünk. A háromrétegű szerkezetnél az idővel lesznek bajok, hiszen egy ilyen rendszer megtervezése sokkal több időt emészt fel.

Bár a munkamegosztás megkönnyíti az egyes részterületek elkülönítését és kipróbálását, a „főpróbát”, azaz a teljes rendszer ellenőrzését eléggé megnehezíti. Ha mindenki egyetlen nyilvánosságra hozott és általánosan elfogadott API-t használna, akkor a dolog sokkal egyszerűbb lenne. Az előírás és a megvalósítás között azonban mindig ott tátong a szakadék, és teljes próba során ez általában ki is derül. Minél több réteg szerepel a tervben, annál összetettebb feladat a kipróbálás és az ellenőrzés.

Végül az adatbázissal való kapcsolatot biztosító középső réteg létrehozása is igen nehéz feladat. Az SQL sem tökéletes nyelv, viszont kevés paranccsal nagyon sokféle lekérdezést végezhetünk el.

Az SQL eltávolítása a Mason elemei közül, valamint egy adott API használatának kényszerítése azt eredményezi, hogy a programozó az adatbázis lehetőségeinek csak egy részét lesz képes kihasználni. Egy-egy újabb lehetőség iránti igény felmerülésekor a középső réteget kell bővítenünk. A programozó számára óriási segítség, hogy egy HTML sablonba (például egy Mason-elembe) közvetlenül beleépítheti az adatbázis-lekérdezést. Ezt a szabadságot nagy hiba lenne megvonni tőlük.

Összegzés

A nagy és összetett webalkalmazásokat fejlesztő programozók egyre célszerűbbnek vélik a régi ügyfél–kiszolgáló rendszer leváltását a háromrétegű szerkezetre. Ez ugyanis gyakran nagyon megkönnyíti a rendszergazdák és a programozók életét. A legfontosabb azonban az, hogy mindig valós szükségleteinknek megfelelően válasszunk.

Reuven M. Lerner
(reuven@lerner.co.il) egy izraeli web- és internet-tanácsadó cég tulajdonosa és vezetője. A Kovácsműhely rovat honlapja a ➲ http://www.lerner.co.il/atf/ címen érhető el.

linuxvilág

A magyar Linux-barátok magazinja.

www.linuxvilag.hu

Kiskapu Kft., 1081 Budapest, Népszínház u. 29. Telefon: 477-0443, Fax: 303-1619