Cold Fusion MX 2.Rész

Kinek mi jut eszébe e szó hallatán? Nekem egy programozási nyelv, melyről eddig nem sokan hallottak kicsiny hazánkban. A Cold Fusion ahogy a cikk első részében is mondtam - egy webre szánt programozási nyelv, mellyel nagyon egyszerűen készíthetünk alkalmazásokat az internetre (vagy intranetre). Alkalmazás alatt értendő mondjuk egy portál, melyeket nap, mint nap is látogatunk.

Áttekintés

A cikk első részében megtárgyaltuk a Cold Fusion szintaktikáját, és azt is láttuk, hogy mennyire egyszerű, könnyen tanulható maga a nyelv, gondoljunk csak vissza az SQL lekérdezésekre. Mint ígértem, ez alkalommal megpróbálom bemutatni a CF-ben használt fontosabb függvényeket. Arra is ki fogunk térni, hogy hogyan hozzunk létre saját függvényeket, és hogy miként használjuk fel ezeket az alkalmazásainkban.

UDF Az meg mi?

Az UDF a User-Defined Function-nek a rövidítése, magyarul felhasználó által definiált függvény. Ez a szolgáltatás a Cold Fusion 5-ös verziójában jelent meg először, és természetesen a CF MX is támogatja. Az un. CFScript típusú kódokat a <cfscript> és a </cfscript> elemek közé kell elhelyeznünk.

```
<cfscript>
function HelloVilag() {
   var nev = "Hello Világ!";
   return nev;
}
</cfscript>

<cfoutput>#HelloVilag()#</cfoutput>
```

A kód első sorában a <cfscript>-tel jeleztük a fordító felé, hogy CFScript-et szeretnénk használni. A function után megadtuk a függvény nevét (HelloVilag), majd a var kulcsszóval deklaráltunk egy változót (nev). Ezek után visszatértünk a nev értékével, és a korábban megismert <cfoutput>-al megjelenítettük a képernyőn a változó tartalmát. A példából jól látható, hogy függvényeket a #FuggvenyNeve()# formában hívhatunk meg.

1 nyelv 2 szintaktika?

Ha azonban alaposabban szemügyre vesszük a forráskódot, két dolgot is tapasztalhatunk:

Először, a CFScript szintaktikája eléggé eltér az eddig megszokott CFML-étől (Cold Fusion Markup Language): az előbbi inkább a C-hez, míg az utóbbi a HTML-hez hasonlít. Néhány fejlesztőnek ez a kevert módszer kényelmetlen lehet.

Másodszor, a CFScript-ben nem helyezhettünk el Cold Fusion tagokat (pl. egy <cfoutput>), csak kifejezéseket, feltételes szerkezeteket, függvényeket stb. Például a képernyőre való íratás így festett a CF 5-ben függvényen belül CFScript nyelven, a writeoutput-on keresztül:

```
<cfscript>
function HelloVilag() {
         writeoutput("Hello Világ!");
}
</cfscript>

<cfoutput>#HelloVilag()#</cfoutput>
```

A Macromedia fejlesztői is rájöttek erre, hogy ez így nem nagyon lesz jó, éppen ezért megvalósították a <cffunction>-ös szerkezetet. Természetesen a Cold Fusion MX-ben, az új <cffunction> mellett továbbra is használhatjuk a <cfscript>-et. Visszatérve a Hello Világos példára, Cold Fusion MX-ben ez így is megvalósítható:

```
<cffunction name = "HelloVilag" hint = "Hello
Világ példaprogram" returntype = "string" output
= "true">
   <cfset nev = "Hello Világ!">
   <cfreturn nev>
</cffunction>

<cfoutput>#HelloVilag()#</cfoutput>
```

Tehát egy új függvény létrehozásakor meg kell adnunk a függvény nevét, a <cffunction> elem name attribútumával. Opcionális megjegyzést is fűzhetünk a függvényhez, erre a hint kulcsszó áll rendelkezésünkre. A hint után találunk egy returntype-ra keresztelt lehetőséget, melyből 13 típus áll rendelkezésünkre:

- Any (bármilyen)
- Array (tömb)
- Binary (bináris)
- Boolen (logikai)
- Numeric (szám)
- Date (dátum)
- Guid (globális egyedi azonosító)
- Query (lekérdezés)
- String (karaktersorozat)
- Struct (több változós struktúra)
- Uuid (univerzális egyedi azonosító)
- Variablename (változó)
- Void (nincs visszatérési érték)

Így pontosan meghatározhatjuk, hogy milyen típusú adatot fog visszaadni az adott függvényünk, és nem érhet minket váratlan meglepetés. Ez biztonsági okokból is jó megoldás, ugyanis ha nem olyan értéket kapunk vissza, amilyet előre meghatároztunk, akkor a fordító hibát jelez, és megszakad a program futása. Mivel a nev változó a példánkban egy karaktersorozatot (string), ezért a függvény visszatérési értéke is string lesz.

A <cfset> segítségével deklarálhatunk változókat, mint ahogy a CFScript-ben a var-ral. Az output 2 értéket vehet fel: true és false. Ezzel tulajdonképpen azt jelezzük a szerver felé, hogy ez a függvény a <cfoutput> között (true), vagy a <cfsilent> elemek között (false) fog megjelenni a későbbiekben.

CFLib.org Frissen, ropogósan

Ha fejlesztéseink során szükségünk lenne egy fontos funkcióra, és nem szeretnénk rajta órákat ülni, akkor csak látogassunk el a www.cflib.org weboldalra, ahonnan több mint 800 előre elkészített függvényt tölthetünk le! Az oldal Common Function Library Project keretében indult, s szinte mindent megtalálunk az e-mail cím ellenőrzéstől kezdve, a különböző titkosító algoritmusokig. Az oldalon lévő UDF-ket tetszés szerint módosíthatjuk, és felhasználhatjuk programjainkban.

Dátum

Jelenítsük meg a dátumot az alapértelmezett formázással!

```
<cfoutput>
   <p>
      Formázás nélkül: #Now()#
   </p>
   <p>
      Formázva: #DateFormat(Now())#,
#TimeFormat(Now())#
   </p>
</cfoutput>
```

Az aktuális dátumot és időt a Now() függvény adja vissza, melyet a Cold Fusion szervertől kér le. Lehetőségünk van ezt kedvünk szerint formázni; ebben a DateFormat(Datum()) és a TimeFormat(Ido()) segít.

A függvény-referenciát mindenki elérheti a Cold Fusion adminisztrátor (127.0.0.1/CFIDE/administrator/index.cfm) Documentation menüpontja alatt (CFML Reference).

Ciklusok

Nem létezik programozási nyelv ciklusok nélkül így van ez a Cold Fusion esetében is.
Írassuk ki a számokat 1-től 10-ig!

```
<cfloop index = "i" from = "1" to = "10">
   <cfoutput>
     #i#<br>
   </cfoutput>
</cfloop>
```

A <cfloop> elemmel hozhatunk létre ciklusokat, a ciklus nevét pedig az index attribútumával adhatjuk meg. A kezdőértéket a from, az utolsó számot a to jelenti. A ciklus minden lefutásakor kiíratjuk az i értékét és beszúrunk egy soremelést, hogy minden szám egymás alatt jelenjen meg. Végeredményül megkapjuk a számokat 1-től 10-ig.

Most annyit módosítsunk a kódon, hogy visszafelé (azaz csökkenő sorrendben) jelenjenek meg a számok:

```
<cfloop index = "i" from = "10" to = "1" step =
"-1">
   <cfoutput>
      #i#<br>
   </cfoutput>
</cfloop>
```

Itt a kezdő és az utolsó számot felcseréltük, és egy step tulajdonságot is beállítottunk. Ennek hatására a szám értéke mindig 1-el csökken, amíg el nem éri az 1-et.

Feltételes szerkezetek

Végrehajthatunk programrészeket egy feltétel teljesülése esetén is (pl. ha i = 8 akkor a ciklus álljon le):

```
<cfloop index = "i" from = "1" to = "10">
   <cfif i is "8">
      <cfbreak>
   </cfif>
   <cfoutput>
     #i#<br>
   </cfoutput>
</cfloop>
```

Ha az i értéke eléri a 8-at, akkor a <cfbreak> utasítással a ciklust megszakítjuk. Ennek hatására a számok kiírása 7-nél megszakad. Ezt feltételes ciklussal is megtehetjük:

```
<cfset i = 0>
<cfloop condition = "i LESS THAN OR EQUAL TO 7">
   <cfset i = i + 1>
   <cfoutput>
      #i#<br>
   </cfoutput>
</cfloop>
```

A második sorban egy feltételes ciklust hoztunk létre: ha i értéke kevesebb vagy egyenlő, mint 7, akkor álljon le. Az eredmény ugyanaz, mint az első példánál.

Tömbök

Nézzünk egy példát a tömbök használatára:

```
<cfset HelloVilagTomb = ArrayNew(1)>
<cfset temp = ArraySet(HelloVilagTomb, 1, 5,
"Hello Világ!")>
<cfset HelloVilagTomb[1] = "Hello Világ! 1.">
<cfset HelloVilagTomb[2] = "Hello Világ! 2.">
<cfset HelloVilagTomb[3] = "Hello Világ! 3.">
<cfset HelloVilagTomb[5] = "Hello Világ! 5.">

<cfloop index = "i" from = "1" to =
"#ArrayLen(HelloVilagTomb)# ">
   <cfoutput>
     #HelloVilagTomb[i]#<br>
   </cfoutput>
</cfloop>
```

Az első sorban létrehoztunk egy 1 dimenziós tömböt (több is lehet, maximum 3): ArrayNew(1), a másodikban megadtuk a tömb kezdő (1), utolsó elemét (5) és a tömb alapértelmezett elemének az értékét (Hello Világ). Ezek után beállítottuk az első, második, harmadik és utolsó elemek értékét, a negyedik elemet direkt nem deklaráltuk hogy lássuk, mi történik ilyenkor. Ezt követően egy index-es ciklussal kiírattuk a tömb elemeit. A 4. elemet nem deklaráltuk, így az alapértelmezett értéket veszi fel, azaz a Hello Világ-ot.

Hogyan tovább?

Rengetek lehetőség áll rendelkezésünkre, amikről még nem is beszéltünk. Például az UDF-ek formázása (megadhatjuk a sémát külső fájlként), külső függvények meghívása fájlokból stb. A cikk 3. részében egy adatbázis alapú vendégkönyvet fogunk elkészíteni.

Kiss-Tóth Marcell marcell@kiss-toth.hu