DUM č. 15 v sadě

28. Inf-4 Jednoduchá hra Had ve Flashi (ActionScript)

Autor: Robert Havlásek

Datum: 18.05.2013

Ročník: 5AV

Anotace DUMu: Flash - teorie: Vícerozměrné pole. Asociativní pole. Asociativní pole v rámci objektu.\n

evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Vícerozměrné pole

Pole, jež jsme zaváděli v DUMu č. 7, jsme chápali jako konečnou posloupnost hodnot, staticky nebo dynamicky přiřazovaných. Řada začínala indexem 0. Pokud jsme uložili hodnotu do nějakého vyššího indexu, než byla velikost pole, prvky mezi starým koncem a právě ukládanou hodnotou se automaticky doplnily hodnotou undefined.

Pokud jsme potřebovali do jednoho prvku pole uložit víc hodnot, nebylo to dosud možné. Problém jsme občas trochu nešťastně obcházeli pomocí více polí – proč by například místo `zdix` a `zdiy` nemohlo být jen `zdi`? A proč bychom místo `zdix[5]=100;zdiy[5]=30;` nemohli psát `zdi[5]["x"]=100` a `zdi[5]["y"]=30` nebo třeba `zdi[5]=Array(100,30)`?
Pedagogická poznámka: Podobné srovnání by mělo studenty motivovat zavádět vícerozměrné pole; měli by si uvědomit, že je nepříjemné pro souřadnice zdí mít pole dvě, neboť nikdy nemá programátor úplnou jistotu, že zdix.length == zdiy.length…

Pokud připustíme, že prvkem pole může být opět pole (což Flash připouští), získáme pole více rozměrů. Lze je deklarovat buď jako prázdné jednorozměrné pole (a poté do něj přidávat) nebo je lze deklarovat rovnou i s hodnotami:
`var promenna:Array = Array(hodnota, hodnota, hodnota),`
kde místo `hodnota` lze použít opět `Array(hodnota, hodnota, hodnota, hodnota)`

Například:
```
var zdi:Array = Array( Array(0,0), Array(10,0), Array(2,0), false,
Array(40,0) );
```
Dotazovat se pak můžeme:
`trace(zdi[1])` … vypíše `10, 0`
`trace(zdi[1][0])` … vypíše `10`
`trace(zdi[3])` … vypíše `false`
`trace(zdi[3][0])` … vypíše `undefined`
`trace(zdi)` … vypíše `0,0,10,0,2,0,false,40,0` (sice položky vnitřně implementuje jako samostatná pole, ale ve výpisu se to nijak neprojeví)

Přiřazovat pak můžeme:
`zdi[1]=5` …
celé pole pak bude `Array( Array(0,0), 5, Array(2,0), false, Array(40,0) )`
`zdi[0][1]=7` …
celé pole pak bude `Array( Array(0,7), 5, Array(2,0), false, Array(40,0) )`
`zdi[3]="F"` …
celé pole pak bude `Array( Array(0,7), 5, Array(2,0), "F", Array(40,0) )`
Ovšem příkaz `zdi[3][2]="NE"` … se neprovede, protože položka s indexem 3 není polem. Museli bychom napsat příkazy:
`zdi[3]=Array();` …
celé pole pak bude `Array( Array(0,7), 5, Array(2,0), Array(), Array(40,0) )`
`zdi[3][2]="ANO"` …
celé pole pak bude `Array( Array(0,7), 5, Array(2,0), Array(undefined, undefined, "ANO"), Array(40,0) )`

Souhrnem: Kdekoliv očekáváme jakoukoliv hodnotu, můžeme použít pole, ale musíme to Flashi před prvním použitím říct – přiřazením Array() nebo Array(hodnota, hodnota).

Praktický úkol: Vyrobte pole 32×20 hodnot a naplňte je čísly 1 a 0 podle toho, jestli je prvek „u kraje“ pole nebo „uprostřed“. Vypište jednotlivé řádky.

Řešení:

```
var pole=Array();
// Prvni radek:
pole[0]=Array();
for (s=0;s<=31;s++) pole[0].push(1);
```

…první řádek (s indexem 0) řešíme zvlášť, je plný jedniček

```
// Druhy az predposledni radek:
for (r=1;r<=18;r++)
  {pole[r]=Array();
   pole[r].push(1);
   for (s=1;s<=30;s++) pole[r].push(0);
   pole[r].push(1);
   }
```

…vnější cyklus (pro r) udává index řádku. Pro každý řádek nejprve zapíšeme jedničku, pak 30 nul a nakonec jedničku

```
// Posledni radek:
pole[19]=Array();
for (s=0;s<=31;s++) pole[19].push(1);
```

…poslední řádek (s indexem 19) řešíme zvlášť, je plný jedniček

```
// Vypis pole:
for (r=0;r<=19;r++) trace(pole[r]);
```

Pedagogická poznámka: Studenti, máte-li dvojrozměrné pole, zvykněte si používat jako indexy proměnné r, s, které vám lépe připomenou, s kolikátým pracujete řádkem a sloupcem.

Program vypíše:

```
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

Pedagogická poznámka: Pokud bychom chtěli takto zdi, hada a jablka „kreslit do bitmapy ve formě dvojrozměrného pole“, znamenalo by to sice větší paměťovou náročnost, ale podstatně rychlejší vyhledávání (=odpověď na dotaz „Je na tomto místě zeď?“) Přeprogramovávat celý kód samozřejmě nebudeme… ☺

Asociativní pole

Asociativním polem myslíme pole, jež má za indexy jiné typy než čísla. Máme-li běžně zadefinované pole, můžeme při běhu programu běžně přistupovat k položkám s nečíselnými indexy, například:

```
var pole:Array=Array(10,20);
pole["barva"]="modra";
pole["viditelnost"]=50;
```

V tomto případě máme pod indexem 0 číslo 10, pod indexem 1 číslo 20, pod indexem "barva" text "modra" a pod indexem "viditelnost" číslo 50.
Nicméně nečíselné indexy Flash nevypíše pomocí trace celkového pole a dokonce je ani nepočítá do celkového počtu prvků v pole.length! *(Pedagogická poznámka: Chová se v tomto zvláštním případě rovněž zvláštně. Asi je to dáno snahou, aby cyklus od 0 do length-1 vypsat všechny očíslované prvky a nic víc. Studentům je třeba toto chování explicitně zdůraznit.)*
Rozšíříme-li předchozí příklad:

```
var pole:Array=Array(10, 20);
pole["barva"]="modra";
pole["viditelnost"]=50;
trace(pole);
```

…vypíše pouze očíslované prvky, tedy `10,20`

```
trace(pole.length);
```

…vypíše pouze 2, neboť dva prvky jsou očíslované

```
trace(pole["barva"]);
```

…vypíše hodnotu `modra`

Studenti, jak se tedy dostaneme k nečíselným indexům, jejichž jména jsme zapomněli nebo jsou nám neznámá? Ano, správně, pomocí cyklu for-in, viz DUM č. 9, strana 2.
S použitím dat z předchozího příkladu:

```
for (jmeno in pole) {for (jmeno in pole) {trace(jmeno+" = "+pole[jmeno]);}
```

…vypíše poslušně všechno, tedy

```
viditelnost = 50
barva = modra
1 = 20
0 = 10
```

Asociativní pole jako objekt

Druhou možností je chápat asociativní pole jako objekt. Jsou to de facto běžné objekty s vlastnostmi a jejich hodnotami, jen nemají metody.
Syntaxe při deklaraci je následující:

```
var jmeno_promenne:Object = {vlastnost:hodnota, vlastnost:hodnota};
```

Například: `var souradnice:Object = {x:160, y:100};`
Použití je pak: `souradnice["x"];`
Ani u objektu nefunguje vlastnost .length, ale funguje cyklus for-in.

Pedagogická poznámka: Někteří studenti mají tendenci v asociativním poli namísto souřadnice["x"] psát souřadnice[x], je nutné jim vysvětlit, že x není hodnota, ale (případně) jméno proměnné.

Rovněž běžným objektům (například těm umístěným na plochu) můžeme přidávat další vlastnosti, byť nemají nic společného s jejich vzhledem ani funkcí. Běžně tak činíme, když zavádíme lokální proměnnou v kódu nějakého objektu.
Typicky budeme v našem programu například zavádět a používat _root.hlava.zivoty a _root.hlava.body. Viz další DUM.