Jana Bromová

Caché Server Pages

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta Katedra informatiky

Caché Server Pages

bakalářská práce

Autor: Jana Bromová

Vedoucí bakalářské práce: Mgr. Miloš Prokýšek

České Budějovice 2008

Název práce: Caché Server Pages
Autor: JANA BROMOVÁ
Katedra: Katedra Informatiky
Vedoucí diplomové práce: Mgr. Miloš Prokýšek
e-mail vedoucího: prokysek@pf.jcu.cz

Abstrakt:

Cílem této bakalářské práce je vytvořit uživatelskou příručku pro tvorbu interaktivních webových aplikací Caché Server Pages a soubor několika tutoriálů vhodných pro výuku. Základem pro práci a tvorbu CSP aplikací je znalost HTML, bez které se nelze obejít, a tato znalost se v této práci předpokládá.

Tištěná práce je doplněna o přílohu ve formě přenosného media CD, kde lze najít nejen soubor výukových tutoriálů, ale i uživatelskou příručku ve formátu PDF a všechny v ní použité příklady.

Klíčová slova: Příručka, tvorba, Caché, CSP, HTML, programování, příklady

Title: Caché Server Pages
Author: JANA BROMOVÁ
Department: Katedra Informatiky
Supervisit: Mgr. Miloš Prokýšek
Supervisor´s e-mail address: prokysek@pf.jcu.cz

Abstract:

The aim of this B.A. is to create a user´s guide for making interactive Caché Server Pages web applications and a file of a number of convenient tutorial methods. The basis for creating CSP applications and thus completing this B.A. is sufficient knowledge of HTML, which cannot be dispensed with.

The printed essay includes enclosure in a portable CD-form, where not only the file of a number of tutorail methods is to be found, but also involves a user´s guide in PDF data format and all the examples that have been used in this guide.

Keywords: guide, creation, Caché, CSP, HTML, programming, examples

Poděkování
Děkuji Mgr. Miloši Prokýškovi za informace, trpělivost, rady a materiály, které mi poskytl během vypracovávání bakalářské práce.

Prohlašuji, že jsem předloženou bakalářskou práci vypracovala samostatně a použila jen pramenů, které cituji a uvádím v seznamu použité literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Č. Budějovicích, 22.4.2008

Podpis

Obsah

Seznam obrázků

Seznam tabulek

Seznam příkladů

1.0 Úvod

Cílem této bakalářské práce je vytvořit uživatelskou příručku, která čtenáři přiblíží, co je to Caché a hlavně Caché Server Pages (CSP) a popíše způsoby, jak vytvořit CSP stránky. Příručka by měla být určena pro čtenáře, kteří nemají s CSP zkušenosti, ale umějí tvořit databáze pomocí Caché, nebo chtějí pouze pro tyto databáze vytvářet webové rozhraní. Předpokladem pro čtenáře je znalost HTML, CSS stylů a alespoň částečná znalost JavaScriptu. Čtenář se v této práci postupně dozví, co je Caché a Caché Server Pages, jak to vše funguje, naučí se základům jazyka Caché ObjectScript (bez jehož znalosti není možné programovat CSP stránky) a seznámí se se základními prvky CSP. Stěžejní kapitoly by měly být doplněny o vzorové příklady, které objasní popisovanou problematiku.

Druhá část této práce by měla být tvořena sadou výukových tutoriálů. Tyto tutoriály by měly popisovat, jak pomocí CSP vytvořit jednotlivé části webové stránky (jako jsou seznamy, tabulky, formuláře, aj.), které budou pracovat s daty uloženými v databázi. Vše bude doplněno o konkrétní příklady s poměrně podrobným popisem. Tyto příklady budou používat data ze vzorové databáze.

Celá práce by měla být doplněna o přílohu ve formě CD-ROM, kde budou k dispozici nejenom všechny zde použité příklady, vzorová databáze, všechny obrázky použité v této práci, HTML verze výukových tutoriálů a PDF soubor s touto prací.

2.0 Co je to Caché a Caché server pages

Caché je „postrelační“ databázová technologie nové generace kombinující objektovou databázi, vysoce výkonný jazyk SQL a rychlý přístup k vícerozměrným datům. Systém Caché je označován jako postrelační technologie, protože poskytuje takovou úroveň výkonu, rychlého programování, rozšiřitelnosti a snadného použití, jakou nelze relační technologií dosáhnout. Data jsou v ní popsána pouze jednou v jediném integrovaném slovníku dat a jsou okamžitě dostupná prostřednictvím všech přístupových metod. Caché v sobě zahrnuje i aplikační server s progresivními možnostmi objektového programování, schopnost snadné integrace se širokou paletou technologií a mimořádně výkonný databázový stroj s jedinečnou technologií vyrovnávacích pamětí.

V Caché lze navíc oproti tradičním databázím vyvíjet propracované webové aplikace využívající jako klienta prohlížeč – zahrnuje totiž bohaté prostředí pro vývoj takovýchto aplikací. Technologie CSP umožňuje rychlé vyvíjení a spouštění dynamicky generovaných webových stránek. Do databázových technologií lze přistupovat i s pomocí poměrně levného hardwaru a počet přístupů ve stejném čase není prakticky omezen. U aplikací nevyužívajících jako klienta prohlížeč je uživatelské rozhraní většinou naprogramováno s využitím některé z oblíbených technologií pro vývoj uživatelských rozhraní, např. Visual Basic, Delphi, Java nebo C++. Nejlepších výsledků (větší výkon, rychlejší programování, nejnižší cena za údržbu) bývá ovšem dosaženo, pokud je vše ostatní naprogramováno v jazycích systému Caché. Technologie Caché umožňuje spolupráci s ostatními technologiemi a podporuje prakticky všechny běžně používané vývojové nástroje.[1]

Caché bylo vyvinuto společností Intersystems. Tato společnost byla založena roku 1978 P. T. Ragonem. Jedná se o soukromou společnost s hlavním sídlem

[1] Podle: InterSystems Corporation. *Caché tech guide CZ: průvodce technologií databáze Caché* [online]. Praha, c2003 [cit. 2008-02-29]. Dostupný z WWW: <http://intersystems.cz/iarchive/printversion/cache/technology/Cache_Tech_Guide_CZ.pdf>.

v Cambridges ve státě Massachusetts v USA. V současné době má tato společnost pobočky v mnoha státech na všech kontinentech, včetně České republiky a zabývá se vývojem softwaru pro vývoj databázových aplikací a práci s těmito aplikacemi. V Americe se zaměřují zejména na informační systémy zdravotní péče. Mezi její hlavní produkty patří Caché (1997), Ensemble (2003), HealthShare (2006) a TrakCare (2007).

3.0 Programovací jazyky a CSP

Caché kromě databázového stroje obsahuje také nástroje pro psaní skriptů. To umožňuje psaní libovolných programů, které mohou manipulovat s daty, zpracovávat transakce, dynamicky ovlivňovat vzhled webu, atd. S trochou nadsázky by se dalo říci, že Caché přináší nový programovací jazyk, jako je například Java, C aj.

V současné době Caché obsahuje dva programovací jazyky. Prvním z těchto jazyků je Caché ObjectScript, druhým je Caché Basic. Z obou těchto jazyků vzniká po kompilaci stejný kód. Caché Basic byl implementován až později a na rozdíl od Caché ObjectScriptu, který přináší zcela nový pohled na psaní zdrojového kódu, má Caché Basic základy v VBScriptu. Programovat v Caché Basicu je tudíž téměř stejné jako programovat ve Visual Basicu.

Oba tyto jazyky lze dle potřeb kombinovat. Není sice možné v rámci jedné metody psát jeden řádek kódu v ObjectScriptu a druhý v Basicu, ale v rámci jedné třídy lze psát každou metodu v jiném jazyce. Dokonce se zde mohou objevit i metody napsané v Javě, kterou Caché také podporuje.

Nás ale bude zajímat použití skriptovacích jazyků a psaní skriptů na webových CSP stránkách. Tělo každého takového skriptu je ohraničeno počátečním a koncovým tagem **SCRIPT**. Počáteční tag je jakousi hlavičkou tohoto skriptu a specifikuje základní vlastnosti. Nejdůležitější z těchto vlastností je jazyk tohoto skriptu. Ten se nastavuje atributem **LANGUAGE**, který může nabývat hodnot jako například **cache**, **SQL**, **javascript**. Ostatní vlastnosti bychom mohli rozdělit do několika skupin, podle použitého jazyka. Blíže je tato problematika popsána v kapitole 5.5, která se věnuje přímo psaní metod na CSP stránkách.

3.1 Caché ObjectScript

Caché ObjectScript je speciální skriptovací jazyk vyvinutý pro **Caché**. Jedná se o **objektově orientovaný jazyk**, ale jeho velkou předností je možnost provázanosti s **datovým přístupem**. Vývojáři mohou na data pohlížet buďto jako na **objekty**, nebo jako na **relační tabulky** (kde mohou využít **jazyk SQL**), nebo jako na **vícerozměrná pole**.

Příkazy v **Caché ObjectScriptu** lze zkracovat na jednopísmenné zkratky. To může sice ze začátku přinést trochu zmatky, ale programátor, který si na tuto možnost zvykne, může ušetřit mnoho času a částečně i zpřehlednit strukturu psaného kódu.

Další výhodou **Caché ObjectScriptu** oproti jiným programovacím jazykům je možnost psaní samostatných **rutin**. Tyto rutiny lze používat bez toho, aby musel být vytvořen nějaký objekt. Toto je asi hlavní rozdíl mezi klasickou třídní metodou a rutinou.

Se základy jazyka Caché ObjectScript se blíže seznámíme v kapitole 4, která je věnována přímo tomuto jazyku.

3.2 Caché Basic

Alternativou ke Caché ObjectScriptu je jazyk Caché Basic, který byl do Caché implementován od verze 5 z důvodu oslovení programátorů ve **Visual Basicu**. Caché Basic vychází (stejně jako Visual Basic) z **VBScriptu** a navíc je obohacen o nástroje potřebné pro komunikaci s databázovou vrstvou Caché.

Caché Basic je přednostně určen pro vývoj internetových databázových aplikací.

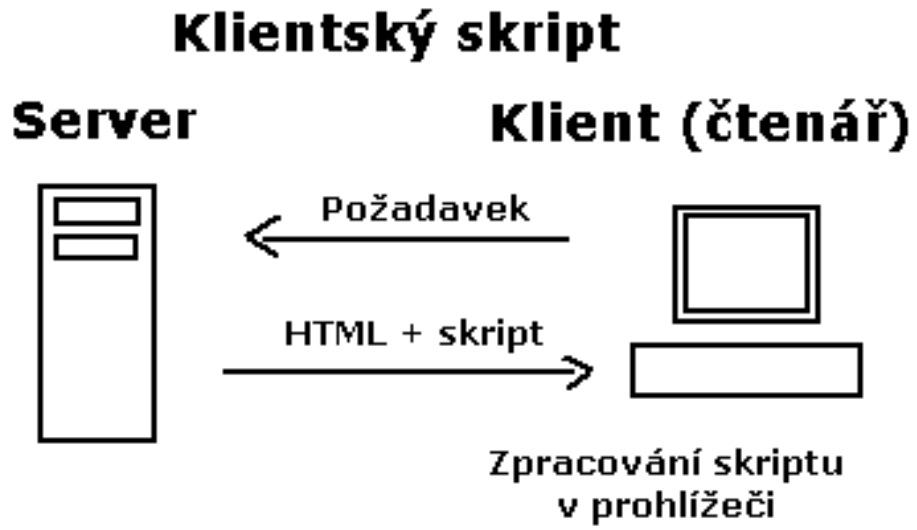
Tímto jazykem se zde ale zabývat nebudeme. S Basicem má zkušenosti téměř každý a proto naučit se v Caché programovat v Basicu bude téměř pro každého jednoduché. Postačí k tomu znalosti z Visual Basicu a případně nahlédnutí do dokumentace.

Caché Basic na rozdíl od Caché ObjectScriptu obsahuje spoustu rezervovaných slov a jeho syntaxe je stejná jako u Visual Basicu.

3.3 JavaScript

Při vývoji dynamických webových stránek vytvářených pomocí technologie CSP se setkáme ještě s jedním skriptovacím jazykem a tím je JavaScript. Je to programovací jazyk, který se používá na internetových stránkách. Zapisuje se přímo do HTML kódu, což je velká výhoda, protože je to jednoduché.

JavaScript je klientský skript. To znamená, že se program odesílá se stránkou na klienta (do prohlížeče) a teprve tam je vykonáván.[1] Jak vlastně funguje JavaScript a ostatní klientské skripty názorně popisuje následující obrázek 3.1.



Obr. 3.1 Jak funguje JavaScript[1]

Mezi základní rysy JavaScriptu patří to, že se jedná o interpretovaný (bez nutnosti kompilace) objektový programovací jazyk, který je case sensitivní. To znamená, že rozlišuje velká a malá písmena.

Nevýhodou je jeho omezená funkčnost. JavaScript funguje pouze v prohlížeči, kde ho uživatel může kdykoli zakázat.

JavaScript může být na CSP stránkách vygenerován i automaticky. Takovou funkci má například tag **CSP:SEARCH**, který generuje vyhledávací stránku. Při zobrazení zdrojového kódu stránky (obsahující tento tag) v prohlížeči, se na místě tohoto tagu zobrazí **JavaScript funkce**, která fyzicky zajišťuje vygenerování vyhledávací stránky. Blíže se tomuto budeme věnovat v kapitole 5, která je věnována CSP.

---

[1] JANOVSKÝ, Dušan. *Jakpsatweb: Úvod do JavaScriptu* [online]. Yuhů, Dušan Janovský, [2003], 14.1.2008 [cit. 2008-03-02]. Dostupný z WWW: <http://www.jakpsatweb.cz/javascript/javascript-uvod.html>. ISSN 1801-0458.

3.4 SQL

CSP stránky mohou také obsahovat skripty psané jazykem SQL, respektive tělo těchto skriptů tvoří **dynamické SQL dotazy**. SQL je standardizovaný dotazovací jazyk používaný pro práci s daty v relačních databázích, a jelikož na data v Caché je možné se mimo jiné dívat i jako na relační tabulky, je možné i v CSP stránkách používat dynamické dotazy generované pomocí jazyka SQL.

Zde se nejedná o klasické SQL. SQL implementované v Caché je obohaceno o určité rysy, jako například možnost použití tečkové notace pro přístup k vlastnostem objektů.

4.0 Základy jazyka Caché ObjectScript

Nyní se seznámíme se základy jazyka Caché ObjectScript. Omezíme se skutečně pouze na jeho úplné základy, jako je syntaxe a několik málo příkazů a funkcí, které můžeme využít při programování webových stránek. Ostatní aspekty tohoto jazyka již nebudou předmětem této práce, která se zabývá převážně tvorbou webových stránek.

S výrazy v Caché ObjectScriptu se na CSP stránkách můžeme setkat hned na několika místech. Tím prvním jsou těla skriptů, kde specifikujeme jazyk tohoto skriptu jako Cache. Druhým případem jsou výrazy přímo v kódu stránky. V tomto případě je snadno poznáme, jelikož tento výraz začíná znaky **#(** a končí **)#**.

Caché ObjectScript ale nemá využití pouze na CSP stránkách. Jeho hlavní využití je při psaní metod Caché tříd a samostatných rutin.

Na rozdíl od klasických programovacích jazyků, jako je například Java nebo C, zde není nutná **žádná deklarace proměnných**, **nepracuje se** zde **s datovými typy**. Probíhá zde pouze **implicitní datová konverze**, kdy je možné s řetězcem znaků pracovat jako s čísly. Také zde nejsou **žádná rezervovaná slova,** Caché ObjectScript je **case sensitiv.** To znamená, že pokud nějakou proměnnou nazveme například *Jmeno*, jedná se o jinou proměnnou, než je *jmeno* nebo *JMENO*. Jednotlivé **příkazy se nijak neukončují**, žádné středníky, konce řádek apod. Příkazy jsou odděleny pouze mezerou, a pokud se jedná o příkazy, které nemají žádné argumenty (jako například ELSE), je nutné je oddělit dvěma mezerami. Toto je řádkový způsob zápisu kódu. Kód lze také zapisovat metodou příkazové konstrukce, kdy příkazy tvoří bloky uzavřené ve svorkách {}, jako je obvyklé v ostatních programovacích jazycích.

Caché ObjectScript rozlišuje **dva typy dat**. Jedná se o data **dočasná** a **trvalá**. Dočasná data, neboli transientní, existují pouze v operační paměti. Oproti tomu trvalá neboli perzistentní data jsou trvale uložená v databázi.

4.1 Proměnné

Caché ObjectScript rozlišuje dva typy proměnných - lokální a globální. **Globální** proměnné od lokálních rozeznáme podle **prefixu ^** před názvem proměnné.

Pro globální proměnné se také využívá pojmenování **globály**. Dočasné proměnné jsou uložené pouze v paměti vyhrazené procesu. Tedy proměnná ”^x” a ”x” jsou dvě různé proměnné. K první z nich mají přístup všechny procesy, ke druhé jen proces, v němž je nastavena. V Caché ObjectScriptu existuje ještě třetí typ proměnných, jsou to systémové, neboli speciální proměnné.

4.1.1 Názvy proměnných

Pro délku názvu proměnných nejsou žádná omezení, pouze prvních 31 znaků musí být jedinečných, protože slouží k jednoznačné identifikaci proměnné.

První znak názvu může být písmeno, ať již velké nebo malé, nebo znak %. Další znaky mohou tvořit písmena nebo čísla. Písmena mohou být včetně národních znaků, jako jsou ˇ, ´, ¨ aj. Jediné, co nemůže název proměnné obsahovat, jsou interpunkční znaménka.

Caché ObjectScript rozlišuje v názvech proměnných velká a malá písmena, je case sensitiv. Pokud budeme mít tyto tři slova: rok, Rok a ROK, v Caché ObjectScriptu se bude jednat o tři rozdílné proměnné.

4.1.2 Definice proměnných

V Caché ObjectScriptu není nutná žádná deklarace proměnných, proměnné se automaticky vytvoří při jejich prvním použití. Pokud přesto chceme proměnné deklarovat, nebo dočasně zakrýt nějakou proměnnou, lze toho docílit použitím příkazu **New**.

Proměnné lze modifikovat třemi příkazy, jsou to **Set**, **Read**, **For**. Příkaz **Set** mění hodnotu proměnné, příkaz **Read** čte hodnotu proměnné ze vstupu a cyklus **For** vytváří proměnnou, která funguje jako počítadlo.

První z nich je příkaz **Set**. Tento příkaz vytvoří proměnnou a nastaví její hodnotu. Pokud je použit pro již vytvořenou proměnnou, změní její hodnotu. Jeden příkaz Set může být najednou použit i pro několik proměnných. Ty jsou potom od sebe oddělené čárkou.

Syntaxe příkazu Set:

Set *jmProm1*=“*hodnota1*“ **[,** *jmProm2*=“*hodnota2*“**]**

Příkaz **Read** není využitelný v CSP, proto se jím zde ani nebudeme zabývat. S cyklem **For** se blíže seznámíme v kapitole věnované cyklům.

4.1.3 Odstranění proměnných

Lokální proměnné jsou z paměti automaticky odstraněné při skončení procesu, ve kterém byly vytvořeny, nebo je možné je odstranit použitím příkazu **Kill**.

Jsou dvě možnosti, jak použít příkaz Kill. První z nich je, když za příkazem Kill následuje výčet proměnných, které mají být zrušeny. Druhým je použití příkazu Kill bez argumentů, kdy jsou zrušeny všechny lokální proměnné.

Syntaxe příkazu Kill:

Kill *jmProm1*, *jmProm2*

4.1.4 Reprezentace proměnných

V Caché ObjectScriptu neexistuje žádná deklarace a žádné datové typy. Všechny proměnné jsou implicitního datového typu a tím je **řetězec**. Interní reprezentace tohoto řetězce může být i logická nebo číselná. Jak bude řetězec reprezentován, záleží na druhu operace, jakou tato proměnná vznikla. Pokud se jednalo o výsledek logických operací, pak bude i interní reprezentace tohoto řetězce logická, pokud to je výsledek aritmetických operací, půjde o reprezentaci číselnou.

4.1.5 Speciální proměnné

Caché ObjectScript obsahuje množství předdefinovaných proměnných, tzv. speciálních proměnných. Většina těchto proměnných má danou hodnotu, nebo se automaticky mění a uživatel nebo programátor jejich hodnotu nemůže změnit. Názvy těchto proměnných začínají prefixem **$**.

Jednou z těchto speciálních proměnných je **$Horolog**, která poskytuje interní systémový čas. S touto proměnnou se setkáme téměř na každé CSP stránce. Hodnota této proměnné je nezáporné reálné číslo, kde jeho celá část reprezentuje datum a část za desetinou tečkou reprezentuje čas. Datum se počítá od 31.12.1840, toto datum

má hodnotu 0. Čas se počítá v sekundách od poslední půlnoci. Například hodnota proměnné **$Horolog** pro **2.3.2008 6:30** je **60878.23400**.

Použití speciálních proměnných je ukázáno na příkladu 5.2 Použití tagu CSP:IF.

4.2 Výrazy a operátory

Zapíšeme-li v matematice například **a + b**, hovoříme o **výrazu**. Ten má dva **operandy a** a **b** a jeden **operátor** +. Jedná se sice o velmi jednoduchý příklad, ale k vysvětlení jednotlivých pojmů, kterými se bude zabývat tato kapitola, postačuje.

Výraz je syntaktický element, který produkuje hodnotu. V **CSP** je výraz v Caché ObjectScriptu uzavřený mezi **#(** a **)#**. Výraz je tvořen atomy (operandy) a operátory. Atomem může být například proměnná (ať již lokální, globální nebo speciální), funkce, řetězce nebo další výrazy v závorkách, aj.

Výrazy mohou obsahovat libovolný počet mezer pouze tehdy, pokud výraz neobsahuje nějaký příkaz. Mezi tímto příkazem a jeho prvním argumentem musí být právě jedna mezera. Pokud se jedná o příkaz bez argumentů, musí za tímto příkazem následovat alespoň dvě mezery nebo konec řádky. Mezery nesmí obsahovat číselné literály (konstanty), názvy tříd, objektů, metod…

Zpracování výrazů:

1. Vyhodnocení výrazu na levé straně
2. Vyhodnocení výrazu na pravé straně
3. Provedení požadované operace

Operátory jsou symbolické znaky, které určují typ operace, jež má být provedena.[1]

Podle druhu operace rozlišujeme několik druhů operátorů. Jsou to operátory aritmetické, logické, řetězcové, porovnávací a nepřímý operátor. Jiné dělení operátorů je podle počtu operandů. Podle tohoto hlediska se rozlišují unární a binární operátory.

[1] Wolfgang KIRSTEN, Michael IHRINGER, Mathias KÜHN, Bernhard Röhrig. *Caché : Databáze postrelačního typu a tvorba aplikací*. Marek Kocan, Radek Havel, Jan Pupík. 1. vydání : CP Books, a.s., 2005. ISBN 80-251-0491-5.

Dalším dělením je pozice operátorů vzhledem k operandům. Takto rozlišujeme operátory prefixové a infixové. Operátory v jednotlivých případech zapisujeme v případě prefixových operátorů před operandy, nebo u infixových operátorů mezi operandy.

Operátory v Caché mají všechny stejnou prioritu a jsou vyhodnocovány podle zápisu zleva doprava. Jedinou výjimkou jsou unární operátory, které se vyhodnocují zprava doleva. Pokud je zapotřebí tuto prioritu změnit, musí se použít závorky, nebo přeskupit pořadí operandů do požadovaného pořadí.

4.2.1 Aritmetické výrazy

Jak již bylo řečeno, Caché ObjectScript je jazyk netypový a všechny proměnné jsou řetězce. Konverze datových typů se provádí automaticky podle syntaktického kontextu. To znamená, že například aritmetické operace prováděné s řetězcem vracejí číselné hodnoty. Tyto hodnoty jsou zvláštními typy řetězců.

Caché ObjectScript rozlišuje tři typy čísel, a to čísla celá, desetinná (s desetinou tečkou) a čísla exponenciální, která mají tvar mantisy a exponentu. Před číslem může být libovolný počet nul a znaménkových operátorů, přičemž každé mínus (-) mění znaménko výsledku a znaménka plus (+) se ignorují.

Použití těchto operátorů na CSP stránkách demonstruje příklad 5.6.

4.2.1.1 Unární operátory

Aritmetické unární operátory jsou pouze dva, a to **plus** (**+)** a **mínus** (-). Tyto znaménkové operátory ovlivňují číselnou interpretaci operandů.

Syntaxe:

<operátor> <operand>

Operátor	Význam	Výraz	Hodnota (číslo)
+	Kladný prefix	+“23kal5“	23
-	Záporný prefix	--“27“	27

Tab.4.1 Unární aritmetické operátory

4.2.1.2 Číselná interpretace řetězce

Pokud chceme s řetězcem pracovat jako s číslem, je nutné si vynutit jeho číselnou reprezentaci. Toho lze dosáhnout použitím unárního operátoru plus (+) před řetězcem. Tento řetězec by měl začínat čísly, pokud čísly nezačíná, jeho číselná reprezentace je 0. V opačném případě se čte řetězec zleva doprava, dokud se nenarazí na nečíselný znak. Samozřejmě se pokračuje i za desetinou tečkou, nebo mantisou.

Výraz	Hodnota (číslo)
+“23kal5“	23
--“27“	27
+“.5“	0.5
+“34E2“	3400
+“34e2“	3400
+“kl234“	0

Tab.4.2 Příklady číselné interpretace řetězce

4.2.1.3 Binární operátory

Caché ObjectScript kromě základních aritmetických operací, jako jsou sčítání (+), odčítání (-), násobení (*) a dělení (/), podporuje také celočíselné dělení (\), umocnění (**) a zbytek po celočíselném dělení - modulo (#).

Syntaxe:

<operand1> <operátor> <operand2>

Operátor	Význam	Výraz	Hodnota (číslo)
+	Sčítání	10+5	15
-	Odčítání	10-5	5
*	Násobení	2*6	12
/	Dělení	10/4	2,5
\	Celočíselné dělení	10\4	2
#	Modulo	11#4	3
**	Mocnina	2**3	8

Tab.4.3 Binární aritmetické operátory

4.2.1.4 Porovnávací operátory

Pro číselné porovnání Caché ObjectScript implementuje pouze dva operátory a to jsou **je větší** (>) a **je menší** (<).

Ještě před porovnáním se provede implicitní konverze operandů na číselné hodnoty, a pak může dojít k samotnému porovnání. Výsledkem tohoto porovnání je **logická 1** (pokud je výraz pravdivý) nebo **logická 0** (pokud je výraz nepravdivý).

Operátory **je rovno** a **není rovno** jsou v Caché ObjectScriptu pouze pro řetězcové porovnávání. Pokud je zapotřebí porovnat dvě čísla, musí se tomuto řetězci (který představuje daná čísla) vnutit číselnou interpretaci použitím znaménka + (kladný unární prefix) před řetězcem. Použití tohoto operátoru zajistí, že tento řetězec bude chápán jako číslo.

Syntaxe:

<operand> <operátor> <operand>

Syntaxe aritmetické rovnosti:

+<operand> = +<operand>

Operátor	Význam	Výraz	Hodnota (logická)
>	Je větší	7>5	1
<	Je menší	7<5	0
=	Je rovno	+“25“=+“2.5E1“	1

Tab.4.4 Porovnávací aritmetické operátory

4.2.2 Logické výrazy

Logické výrazy jsou takové operace, které jsou prováděny s logickými hodnotami **pravda** (1, TRUE) a **nepravda** (0, FALSE). Výsledkem těchto operací je opět logická hodnota. Základní unární logickou operací je **negace**, která dělá z logické pravdy nepravdu a naopak. Základními binárními operacemi jsou **konjunkce** (logické a, součin) a **disjunkce** (logické nebo, součet).

4.2.2.1 Unární operátory

Logický unární operátor je pouze jeden a tím je **negace**. Ta obrací logickou hodnotu daného výrazu. Použití dvojité negace nemá žádný vliv na hodnotu výsledku. Symbol pro negaci je **‘**.

Syntaxe:

‘<operand>

Operátor	Význam	Výraz	Hodnota (logická)
‘	Non, Negace	‘0 ‘1 ’’0	1 0 0

Tab.4.5 Unární logické operátory

4.2.2.2 Binární operátory

Binární logické operátory lze rozdělit do dvou skupin. První z nich jsou operátory s úplným vyhodnocením, kdy se oba operátory vyhodnotí, ať již je logická hodnota prvního z nich jakákoliv. Druhou z nich jsou operátory se zkráceným vyhodnocováním, kdy se po vyhodnocení prvního operátoru vyhodnotí, zda je toto postačující. Například pokud u logického součinu je hodnota prvního operandu nepravda, druhý už se nebude vyhodnocovat a rovnou se určí výsledek tohoto součinu jako nepravda.

Syntaxe:

<operand> <operátor> <operand>

Operátor	Význam
&	Logický součin- úplné vyhodnocení
|	Logický součet- úplné vyhodnocení
&&	Logický součin-zkrácené vyhodnocení
||	Logický součet-zkrácené vyhodnocení

Tab.4.6 Binární logické operátory

4.2.3 Řetězcové výrazy

Řetězec je základní a jediný datový typ Caché ObjectScriptu, pouze jeho vnitřní reprezentace může být logická nebo aritmetická. O těchto případech pojednávaly předchozí kapitoly.

Řetězec v Caché ObjectScriptu musí být zapsán v uvozovkách. Například "*Toto je řetězec.*"

S každým řetězcem (ať již je jeho vnitřní reprezentace jakákoli) je možné provádět porovnávací operace. Další operací, kterou je možné s řetězci provádět je jejich slučování.

4.2.3.1 Porovnávací operátory

Porovnávat řetězce je možné z mnoha různých hledisek. Je možné například zjišťovat, zda jsou dva řetězce **totožné**, nebo zda jeden řetězec **obsahuje** ten druhý. Další možností je lexikografické pořadí nebo setřídění. Těmito možnostmi se zde již nebudeme zabývat z důvodu malého využití v CSP stránkách. Pokud by někdo chtěl takové porovnání provádět, má k dispozici dokumentaci, kde je vše podrobně vysvětleno. Výsledkem všech těchto operací je logická hodnota, a to **logická 1**, pokud je výraz pravdivý, nebo **logická 0**, pokud je výraz nepravdivý.

Operátor **rovná se** (=) porovnává dva řetězce, zda jsou shodné. Pokud je zapotřebí porovnat dvě čísla, je nutné vnutit číselnou interpretaci řetězce použitím znaménka plus (+) před každým operandem.

Caché ObjectScript nemá žádný operátor pro **nerovnost**. Proto pokud je nutné zjistit nerovnost, použije se operátor **rovnosti** a výsledek se zneguje pomocí operátoru **negace**.

Operátor **obsažení [** testuje, zda pravý řetězec je součástí levého řetězce.

Syntaxe:

<operand> <operátor> <operand>

Syntaxe nerovnosti:

<operand> '= <operand> , nebo

'(<operand> = <operand>)

Syntaxe číselné rovnosti:

+<operand> = +<operand>

Operátor	Význam	Výraz	Hodnota (logická)
=	Rovnost	“abc“ = “den“ “abc“ ‘= “den“ ‘(“abc“ = “den“) +“3“= +“5“	0 1 1 0
[	Obsažení	“abc“ [“b“	1

Tab.4.7 Řetězcové porovnávací operátory

4.2.3.2 Operátor sloučení

Při práci s řetězci je také někdy zapotřebí ze dvou řetězců udělat jeden. K tomu slouží operátor sloučení. Znak pro tento operátor v Caché ObjectScriptu je podtržítko _.

Operátor	Význam	Výraz	Hodnota (řetězec)
_	sloučení	“abc“_“den“	“abcden“

Tab.4.8 Operátor sloučení

4.3 Příkazy a funkce

Názvy příkazů a funkcí v Caché ObjectScriptu jsou jednoduché, vycházejí z angličtiny a jejich názvy jsou většinou shodné s názvy v jiných programovacích jazycích. Na rozdíl od většiny jiných programovacích jazyků, příkazy v Caché ObjectScriptu nejsou ničím ukončené. Stačí udělat mezeru a psát hned další příkaz.

Rozdíl mezi příkazy a funkcemi je hlavně v tom, že příkazy slouží především k práci – vykonávají určitou činnost. Funkce mimo to, že provedou nějakou činnost jako příkazy, ještě vrátí nějaký výsledek. Některé funkce se mohou použít i jako příkaz. Vstupní data předáváme příkazům a funkcím ve formě argumentů. Funkce se od příkazů odlišují tím, že mají parametry zadané v závorkách, příkazy toto nevyžadují.

Caché ObjectScript obsahuje řadu **předdefinovaných funkcí**, například matematické funkce, funkce pro práci s řetězci, datem a časem a mnoho jiných. Na tyto funkce se lze odkazovat v každém výrazu. Volání takové funkce je nahrazeno hodnotou, která je vypočtena ze zadaných argumentů této funkce.

Názvy předdefinovaných funkcí začínají znakem dolaru **$** (uživatel si může také definovat své vlastní funkce, které začínají znakem $$). Argumenty funkcí jsou uzavřeny v závorkách ( ). Každá předdefinovaná funkce musí mít alespoň jeden argument. Jedinou výjimkou je funkce $ListBuild, která nemusí mít žádný argument. Argumentem funkce mohou být libovolné výrazy, ať již číselné, logické, nebo jiné. Konverze těchto argumentů probíhá automaticky. Pokud má být argumentem funkce například celé číslo, argument je automaticky interpretován jako celé číslo. Pokud máme například funkci, která má pět argumentů a chceme zadat pouze čtvrtý z nich, musí před hodnotou tohoto argumentu být tři čárky. To proto, aby bylo jasné, že se zadává čtvrtý argument.

V předchozí kapitole bylo řečeno, že Caché ObjectScript je jazyk rozlišující velká a malá písmena. Toto pravidlo je u názvů příkazů potlačeno. Každý programátor může psát názvy příkazů a funkcí, jak je zvyklý, ať již velkými písmeny, nebo malými.

V Caché ObjectScriptu existují dva typy příkazů, **příkazy s argumenty** a **příkazy bez argumentů**. Příkladem příkazu bez argumentů je příkaz Else. Některé příkazy se mohou vyskytovat s argumenty i bez nich, to jsou například příkazy Do, For, If, Quit. V případě **příkazu bez argumentů** musí za tímto příkazem následovat alespoň **dvě mezery**, nebo odřádkování. U **příkazů s argumenty** musí být mezi **názvem** tohoto **příkazu a** jeho **prvním argumentem právě jedna mezera**.

Pokud za příkazem následuje blok příkazů, musí být počáteční svorka na stejné řádce jako příkaz, za kterým následuje a od jeho posledního argumentu musí být tento začátek bloku oddělen právě jednou mezerou. Příkazy tohoto bloku mohou pokračovat na stejné řádce, nebo mohou začínat na další řádce. Ukončení tohoto bloku může být také na nové řádce nebo za posledním příkazem tohoto bloku.

V případě, že by měl být nějaký příkaz opakovaně použit, je možné se tomuto opakování vyhnout a za příkaz pouze napsat seznam argumentů oddělených čárkou.

Caché ObjectScript přináší **možnost zkracovat příkazy a funkce** na jednopísmenné (popřípadě dvoupísmenné z důvodu zajištění jedinečnosti zkratky). Tento zkrácený tvar je tvořen prvním nebo prvními dvěma písmeny. Tato možnost zkracování může být pro začátečníky poněkud brutální, ale zkušeným programátorům to přináší možnost zefektivnění a zpřehlednění kódu.

Téměř každý příkaz má i svou **podmíněnou podobu**. To znamená, že se provede jen v případě, že daná podmínka je splněna. To dává programátorovi možnost říci, kdy se má daný příkaz provést, aniž by byl použit příkaz pro větvení programu If. Podmínka se uvádí znakem dvojtečka. Podmínkou musí být logický výraz, kdy se po jeho splnění provede příkaz pro zadané argumenty. Následnou podmínku nelze využít u všech příkazů, mezi takové příkazy patří například If, Else, For.

Syntaxe příkazu s argumenty:

```
<příkaz><mezera><arg1>[, <arg2>]
```

Syntaxe příkazu bez argumentů:

```
<příkaz><mezera><mezera>
```

Syntaxe podmíněného příkazu:

```
<příkaz>:<podmínka><mezera><arg1>[, <arg2>]
```

Syntaxe předdefinované funkce:

```
$<jméno funkce>(< arg1>[, <arg2>…])
```

Syntaxe příkazu s argumenty, za nímž následuje blok příkazů:

```
<příkaz><mezera><arg1>[, <arg2>] {
    Blok příkazů
}
```

Příkazy v Caché ObjectScriptu lze rozdělit do několika skupin. Jsou to příkazy pro řízení běhu programu (neboli řídící struktury), příkazy pro práci s proměnnými, příkazy vykonávající vstupně-výstupní operace a ostatní příkazy.

Nyní si některé z příkazů blíže vysvětlíme - opět se omezíme pouze na příkazy, které jsou často používané na CSP stránkách.

4.3.1 Řídící struktury

Řídící struktury jsou takové příkazy a funkce, které ovlivňují pořadí provádění příkazů. Mezi příkazy ovlivňující chod programu patří jednoduchá podmínka, cykly aj.

4.3.1.1 If, ElseIf, Else

Jedná se o podmíněný příkaz, který se provede pouze tehdy, je-li splněna podmínka.

Syntaxe:

If <podmínka> [, <podmínka> …] { blok příkazů 1 }
ElseIf <podmínka> [, <podmínka> …] { blok příkazů 2 }
Else { blok příkazů 3 }

Popis syntaxe:

Jsou-li(If) podmínky splněny (=True), **potom proveď** blok příkazů 1. **Pokud nejsou splněny první podmínky (u If), zjisti, zda jsou splněny další a jsou-li (ElseIf)** splněny **potom proveď** blok příkazů 2, nejsou-li splněny žádné podmínky, **proveď (Else)** blok příkazů 3.

Podmínky u každého příkazu se vyhodnocují zleva doprava. Pokud je nějaká vyhodnocena jako nepravda, okamžitě se přistupuje k dalšímu příkazu, aniž by se ostatní podmínky vůbec testovaly.

Větve **ElseIf** a **Else** nejsou povinné. Tento složený příkaz tudíž může být tvořen pouze větví **If**, nebo větví **If a Else,** anebo větví **If a ElseIf**. Větve ElseIf a Else nelze použít bez předchozího použití větve If - nejsou to nezávislá příkazová slova.

4.3.1.2 $Select, $Case

Tyto funkce slouží jako několikanásobné použití podmíněného příkazu. Rozdíl mezi funkcí **$Case** a **$Select** je v tom, zda je u všech podmíněných příkazů stejný výraz v podmínce s pokaždé jinou výstupní hodnotou, nebo zda se pokaždé testuje jiná podmínka.

Syntaxe $Select:

$Select(<logický výraz1>:<výraz1>[,<logický výraz2>:<výraz2>…])

Popis syntaxe $Select:

Argumenty funkce $Select tvoří uspořádané dvojice logického výrazu a libovolného výrazu oddělených dvojtečkou (:). Tyto argumenty se prochází zleva doprava a pokud je splněn některý z logických výrazů, provede se výraz následující. Pokud ani jeden z logických výrazů není splněn, je vygenerována chyba.

Syntaxe $Case:

$Case(<výraz>,<literál1>:<výraz1>[,<literál2>:<výraz2>…],:<výchozí>)

Popis syntaxe $Case:

Nejprve se vyhodnotí výraz. Pokud se literál shoduje s výrazem, výstupem této funkce bude hodnota výrazu1. Pokud se žádný s literálů neshoduje s výrazem, vrátí se hodnota výchozího výrazu.

4.3.1.3 Cyklus: For

Cyklus For se používá v situacích, kdy je nutné, aby se nějaký kód opakovaně provedl, a počet opakování je předem znám.

Syntaxe:

For <formální parametr> {blok příkazů}

Popis syntaxe:

Blok příkazů se provede tolikrát, jak je specifikováno v části formální parametr. Přičemž tato část může mít čtyři rozdílné varianty:

Syntaxe 1. varianta:

For <proměnná>=<výraz1 >[, <výraz2> …] {blok příkazů}

Popis syntaxe:

V tomto případě je formální parametr tvořen výčtem výrazů. Tělo cyklu se provede pro nějakou proměnnou, které je postupně přiřazena hodnota každého příkazu z výčtu. Tzn., že poprvé se blok příkazů provede pro proměnnou, jejíž hodnota se nastaví na hodnotu výrazu 1, poté pro hodnotu výraz 2, atd. Počet opakování cyklu je dán počtem výrazů ve výčtu.

Syntaxe 2. varianta:

For <proměnná>=<čís. výraz >:<čís. výraz>:<čís. výraz> {blok příkazů}

Popis syntaxe:

Tato varianta je obvyklá v ostatních programovacích jazycích, kdy první číselný výraz reprezentuje počáteční hodnotu, druhý krok (inkrementaci) a třetí konečnou hodnotu.

Syntaxe 3. Varianta:

For <proměnná>=<čís. výraz >:<čís. výraz> {blok příkazů}

Popis syntaxe:

Tato varianta vychází z předchozí, opět se specifikuje počáteční hodnota a krok, konečná hodnota není určena. Jedná se tedy o nekonečnou smyčku, která se opustí příkazem Quit, který musí být obsažen v bloku příkazů.

Syntaxe 4. varianta:

For {blok příkazů}

Popis syntaxe:

Varianta bez argumentů. Jedná se o nekonečný cyklus, jehož ukončení závisí na použití příkazu Quit někde v jeho těle.

4.3.1.4 Cyklus: While

Cyklus While je cyklus řízený podmínkou na začátku. To znamená, že tělo cyklu tvořené nějakým blokem příkazů se provede pouze tehdy, jsou-li splněné počáteční podmínky (výrazy za příkazovým slovem while).

Syntaxe:

While <výraz> [, <výraz> …] {blok příkazů}

Popis syntaxe:

Výrazy u každého příkazu se vyhodnocují zleva doprava. Pokud je nějaký vyhodnocen jako nepravda, okamžitě se opouští cyklus, aniž by se ostatní podmínky vůbec testovaly. Jinak se provede blok příkazů tvořící tělo cyklu.

4.3.1.5 Cyklus: Do-While

Cyklus Do-While je řízen podmínkou na konci. Tzn., že blok příkazů tvořící tělo cyklu se provede alespoň jednou, neboť výrazy určující zda se provede opakování cyklu, se testují až po průchodu tělem cyklu.

Syntaxe:

Do {blok příkazů} While <výraz> [, <výraz> …]

Popis syntaxe:

Nejprve se jednou provede blok příkazů tvořící tělo cyklu, potom se vyhodnocují výrazy. Tyto výrazy se vyhodnocují zleva doprava. Pokud je nějaký vyhodnocen jako nepravda, okamžitě se opouští cyklus, aniž by se ostatní podmínky vůbec testovaly a další průchod cyklem se již neprovede. V opačném případě, když jsou všechny výrazy vyhodnoceny jako pravda, se opět přeskočí na začátek bloku příkazů, ten se provede a poté se znova testují výrazy ovlivňující opakování cyklu.

4.3.1.6 Quit, Continue

Tyto příkazy slouží k opuštění cyklu. Příkaz **Quit** ukončuje celý cyklus. Příkaz **Continue** ukončuje pouze aktuální smyčku a pokračuje se dalším průchodem cyklu.

Tyto příkazy nemají žádné atributy, a tudíž po tomto příkazu musí následovat buďto dvě mezery, nebo konec řádku.

4.3.2 Práce s proměnnými

Caché ObjectScript obsahuje také několik příkazů, které pracují a ovládají proměnné. Jsou to příkazy **Set**, **New** a **Kill**.

4.3.2.1 Set

Tento příkaz vytvoří proměnnou a nastaví její hodnotu. Pokud je použit pro již vytvořenou proměnnou, změní její hodnotu. Jeden příkaz Set může být najednou použit i pro několik proměnných. Ty jsou potom od sebe oddělené čárkou.

Syntaxe příkazu Set:

Set *jmProm1*=“*hodnota1*“**[,** *jmProm2*=“*hodnota2*“**]**

4.3.2.2 Kill

Příkaz Kill odstraňuje proměnné, ať již lokální nebo globální, zadané v seznamu argumentů.

Jsou dvě možnosti, jak použít příkaz Kill. První z nich je, když za příkazem Kill následuje výčet proměnných, které mají být zrušeny. Druhým je použití tohoto příkazu bez argumentů, kdy jsou zrušeny všechny lokální proměnné.

Syntaxe příkazu Kill:

Kill *jmProm1[, jmProm2]*

4.3.2.3 New

Příkaz New slouží k deklaraci proměnných. Ta sice v Caché ObjectScriptu není nutná, ale pokud je nutné například v těle cyklu překrýt lokální proměnnou, tak toho lze docílit použitím právě tohoto příkazu.

Syntaxe příkazu New:

New *jmProm1[, jmProm2]*

4.3.3 Práce s řetězci

Caché ObjectScript obsahuje několik předdefinovaných funkcí pro práci s řetězci. Mezi tyto funkce patří například **$Length**, **$Find**, **$ZConvert** aj.

Funkce **$Length** zjišťuje délku řetězce, funkce **$Find** vyhledává v řetězci zadaném jako první argument podřetězec zadaný jako druhý argument a funkce **$ZConvert** slouží k převodu řetězce na malá nebo na velká písmena. Tato funkce má dva argumenty, prvním je řetězec, který má být převeden a druhým je mód převodu. To může být písmeno U pro převod na velká písmena, nebo znak L pro převod na malá písmena.

4.3.4 Práce s datem a časem

Součástí předdefinovaných funkcí v Caché ObjectSriptu je také skupina funkcí pracujících s datem a časem. Tyto funkce vytvářejí a zobrazují tvar data a času z jeho interní reprezentace, nebo naopak zadané datum a čas převádějí na jeho interní reprezentaci. Jedním z argumentů těchto funkcí je formát data, jak má být zobrazen.

Základem pro práci s datem a časem je znalost interního data - to uchovává speciální proměnná **$Horolog** (viz kap 4.1.5 Speciální proměnné).

Použití funkcí pro práci s datem je ukázáno na příkladu 5.5 Použití tagu CSP:IF.

4.3.4.1 $ZDate, $ZDateH

Funkce **$ZDate** převádí systémové datum do formátu, které udává druhý argument této funkce. Prvním argumentem je hodnota data v systémovém tvaru. Jako tento argument lze použít speciální proměnnou **$Horolog** a funkce $ZDate potom vrací aktuální datum.

Názvy měsíců vychází z angličtiny - to lze také změnit použitím třetího argumentu této funkce, kdy hodnotou tohoto argumentu je seznam názvů měsíců.

Toto ani zdaleka nejsou všechny možné argumenty, které může funkce $ZDate mít, například čtvrtým argumentem může být formát roku (zda se mají zobrazovat pouze poslední dvě číslice, nebo všechny čtyři).

Syntaxe $ZDate:

$ZDATE(hdate, dformat, monthlist, yearopt, startwin, endwin, mindate, maxdate, erropt)

Argument	Popis
HDATE	Hodnota interního data
DFORMAT	Formát data (viz tab. 4.10)
MONTHLIST	Seznam měsíců
YEARORP	Formát roku

Tab.4.9 Hlavní argumenty funkce $ZDate

Následující tabulka zobrazuje všechny možné formáty data, které je možné použít. První sloupeček v této tabulce udává hodnotu druhého argumentu funkce $ZDate.

Hodnota	Formát data	Příklad (5. 2. 07)	Poznámka
0	DD Mmm RRRR	05 Feb 2007	
1	MM/DD/RRRR	02/05/2007	
2	DD Mmm RRRR	05 Feb 2007	
3	RRRR-MM-DD	2007-02-05	
4	DD/MM/RRRR	20/02/2007	
5	Mmm D, RRRR	Feb 5,2007	
6	Mmm D RRRR	Feb 5 2007	
7	Mmm DD RRRR	Feb 05 2007	
8	RRRRMMDD	20070205	
9	Měsíc D,RRRR	February 5, 2007	
10	Číslo dne v týdnu	1	Neděle je 0
11	Zkratka dne v týdnu	Mon	Vychází z angličtiny
12	Den v týdnu	Monday	Vychází z angličtiny

Tab.4.10 Formát data funkce $ZDate

Příklad 4.1: Ukázka použití funkce $ZDate

$ZDate($Horolog,0,“Leden Únor Březen Duben Květen Červen Červenec Srpen Září Říjen Listopad Prosinec“)

Popis: Pokud by dnes bylo například 4. Března 2008, výstupem této funkce by bylo: 04 březen 2008.

K funkci $ZDate existuje také inverzní funkce **$ZDateH**, která naopak převádí uživatelem zadané datum na jeho interní reprezentaci. Tato funkce má podobné argumenty, jako funkce $ZDate.

Syntaxe $ZDateH:

$ZDATEH(date, dformat, monthlist, yearopt, startwin, endwin, mindate, maxdate, erropt)

4.3.4.2 $ZDateTime, $ZDateTimeH

Pokud je zapotřebí, aby se zobrazilo datum i čas, funkce $ZDate nám moc nepomůže. K tomu slouží funkce **$ZDateTime**, která zobrazuje datum a čas v požadovaném formátu. Samozřejmě i tato funkce má k sobě inverzní a tou je **$ZDateTimeH**.

Syntaxe $ZDateTime:

$ZDATETIME(hdatetime, dformat, tformat, precision, monthlist, yearopt, startwin, endwin, mindate, maxdate, erropt)

Argument	Popis
HDATETIME	Hodnota interního data a času
DFORMAT	Formát data (viz tab. 4.10)
TFORMAT	Formát času (viz tab. 4.12)
MONTHLIST	Seznam měsíců
YEARORP	Formát roku

Tab.4.11 Hlavní argumenty funkce $ZDateTime

Jaké jsou možnosti formátování data, jsme si již ukázali v tabulce 4.10. Jak již bylo řečeno fnkce $ZDateTime zobrazuje kromě data i čas a také jeho zápis je možné ovlivnit. Jaké jsou možnosti jeho zobrazení, shrnuje následující tabulka.

Hodnota	Formát data	Příklad (15:36:7)	Poznámka
1	HH:MM:SS	15:36:07	24 hodinový formát
2	HH:MM	15:36	24 hodinový formát
3	HH:MM:SS [AM/PM]	2007-02-05	12 hodinový formát
4	HH:MM [AM/PM]	20/02/2007	12 hodinový formát

Tab.4.12 Formát času funkce $ZDateTime

4.3.4.3 $ZTime, $ZTimeH

Poslední zde probíranou funkcí pro práci s datem a časem je funkce **$ZTime**, která zobrazuje pouze čas. Dalo by se říci, že funkce $ZDateTime se dá rozdělit na dvě funkce a to $ZDate (zobrazující datum) a $ZTime (zobrazující čas). Také funkce $ZTime má k sobě inverzní a tou je **$ZTimeH**.

Syntaxe $ZTime:

$ZTIME(htime, tformat, precision, erropt)

Argument	Popis
HTIME	Hodnota interního času
TFORMAT	Formát času (viz tab. 4.13)

Tab.4.13 Hlavní argumenty funkce $ZTime

5.0 Caché server pages

Technologie CSP je součástí Caché od verze 4. Základem této technologie jsou nástroje pro tvorbu webových stránek. Tyto stránky mohou být aplikačním rozhraním pro Caché databázi.

5.1 Úvod do technologie CSP

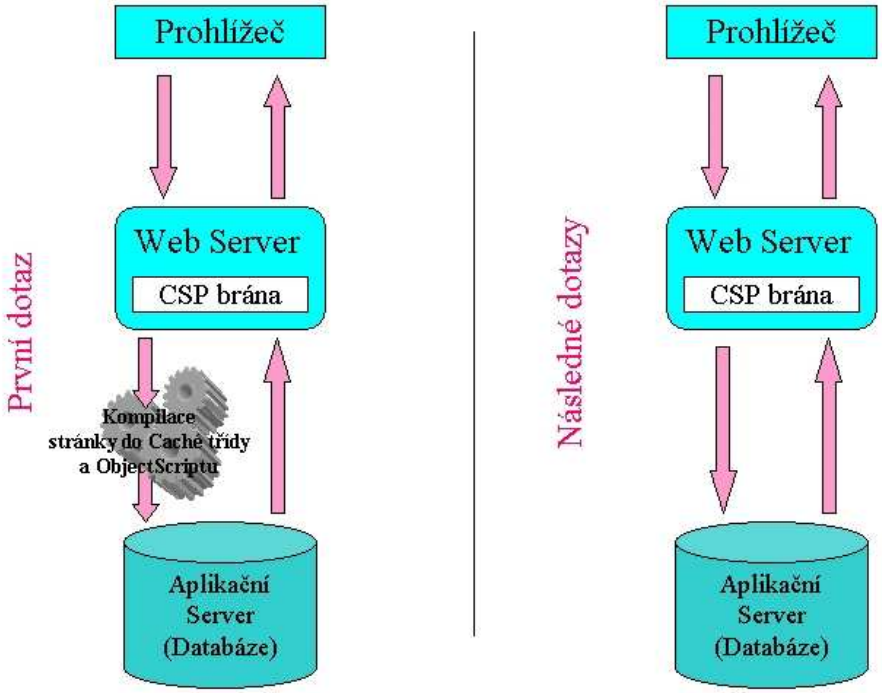
Technologie CSP je součástí Caché pro tvorbu dynamických webových aplikací. Základem CSP je soubor předdefinovaných tagů pro tvorbu aplikační logiky webových stránek. CSP neobsahuje nástroje pro tvorbu grafického prostředí, k tomuto účelu je v současné době na trhu velké množství programů.

CSP pracuje na mnoha webových serverech, jako například Apache, Microsoft aj. CSP je podporováno také všemi nejobvyklejšími prohlížeči, jako je Internet Explorer, Netscape, Mozilla Firefox a mnoho dalších.

5.2 Cesta od napsání kódu až ke klientovi

Vše začíná napsáním kódu celé aplikace. Tato aplikace je uložena na databázovém serveru. Klient se k datům na tomto serveru dostane prostřednictvím požadavku klienta odeslaného na webový server, na kterém musí být nainstalována CSP brána. Ta zajišťuje přesměrování dotazů na databázový server a po jejich vyhodnocení opět zpět ke klientovi. V klientském prohlížeči se objeví vždy pouze čisté HTML.

Na podobném principu pracují také ASP.NET nebo JSP.



Obr. 5.1 Použití CSP stránek[1]

Popis: Klient dá prostřednictvím prohlížeče požadavek na CSP stránku. Webový server, na kterém je nainstalována CSP brána, přesměruje tento požadavek na databázový server Caché. Zde se tento dotaz zpracuje a vytvoří se odpovídající HTML kód, který je vrácen zpět na webový server a přeposlán do prohlížeče.

Při prvním spuštění aplikace v klientově prohlížeči je celá aplikace zkompilována. Při kompilaci je aplikace převedena do Caché tříd, kdy jazykem těchto tříd je Caché ObjectScript. Další akce, které klient chce, aby byly vykonány, se již vyhodnocují prostřednictvím zkompilovaného kódu. Vyhodnocení těchto požadavků je již mnohem rychlejší.

[1] KUTÁČ, Daniel. *Caché Studio 5* [online]. InterSystems CZ, 2002 [cit. 2008-03-15]. Dostupný z WWW: <http://www.intersystems.cz/iarchive/printversion/cache_studio/cs5/cs5.html>.

5.3 CSP stránky a třídy

Vytvářet aplikace technologií CSP je možné dvěma způsoby. Prvním z nich je vytváření HTML souborů obohacených o speciální prvky jazyka CSP. Tomuto způsobu budeme říkat **vývoj pomocí značek**. Druhý je **vývoj psaním kódu**, kdy se vytváří přímo Caché třídy.

Vývoj pomocí značek

Značkovací vývoj je realizovaný pomocí CSP souborů. Tyto soubory jsou tvořeny klasickým HTML a speciálními CSP tagy a skripty, které jsou vyhodnocovány na straně serveru.

Vytvářet CSP soubory je možné v Caché Studiu, kde se automaticky vygeneruje pro nový dokument kostra HTML souboru. Zde jsou také barevně rozlišeny jednotlivé komponenty CSP stránek. Tzn., že prostý text má jinou barvu, než HTML tagy, nebo než CSP tagy aj.

CSP soubory lze také vytvářet v libovolném HTML editoru, jako je například Macromedia Dreamweaver, kam se nechá také importovat zvláštní modul pro vkládání CSP tagů.

CSP soubory Caché automaticky převádí na Caché třídy, které jsou následně překládány do spustitelného kódu.

Tento způsob vývoje je oblíbenější u web designerů. Pro vývoj složitých aplikací je lepší využít druhou variantu vývoje.

Vývoj psaním kódu

Kódový vývoj je založen na vytváření Caché tříd odvozených od třídy **%CSP.Page**. Kód, který má být zobrazen na výsledné webové stránce musí být součástí metody **OnPage()**.

Tomuto způsobu se zde věnovat nebudeme, protože k tomuto způsobu tvorby aplikací je nutná lepší znalost definice Caché tříd a definice jejich metod.

5.4 CSP tagy

Caché obsahuje několik předdefinovaných CSP tagů pro vytváření dynamických webových stránek. Tyto tagy jsou definované pomocí XML a jejich použití je naprosto stejné jako u klasických HTML tagů.

Kromě předdefinovaných tagů si může programátor definovat také své vlastní tagy, které budou splňovat jeho specifické požadavky.

Zde si některé z těchto předdefinovaných tagů představíme a seznámíme se s jejich funkcí a využitím.

Tag	Popis
<CSP:CLASS>	Určuje podrobnosti o třídě vygenerované pro CSP stránku.
<CSP:COMMENT>	Vymezuje CSP komentář.
<CSP:CONTENT>	Specifikuje obsahový typ CSP stránky.
<CSP:CONTINUE>	Přeskočí na další iteraci bloku CSP:WHILE nebo CSP:LOOP.
<CSP:ELSE>	Stanoví hranice ELSE bloku.
<CSP:ELSEIF>	Stanoví hranice ELSEIF bloku.
<CSP:IF>	Stanoví hranice IF bloku.
<CSP:INCLUDE>	Určuje run - time obsah CSP souboru.
<CSP:LOOP>	Vymezuje tělo FOR cyklu.
<CSP:NEW>	Generujte příkaz Caché ObjectScriptu NEW.
<CSP:OBJECT>	Vytváří nový objekt (instanci).
<CSP:PARAMETER>	Specifikuje hodnoty parametrů třídy pro vygenerovanou CSP třídu.
<CSP:QUERY>	Definuje a provádí předdeklarovaný třídní dotaz (definovaný v dané třídě).
<CSP:QUIT>	Ukončuje <CSP:WHILE> nebo <CSP:LOOP> blok.
<CSP:SEARCH>	Definuje JavaScript funkci která vyvolává CSP Search Page.
<CSP:SECTION>	Přesměrujte výstup k specifikované části dokumentu.
<CSP:SQLQUERY>	Spustí dynamický SQL dotaz
<CSP:WHILE>	Vymezuje tělo WHILE cyklu.

Tab.5.1 Přehled hlavních csp tagů

Kromě těchto hlavních tagů, uvedených v tabulce 5.1, nabízí Caché také kolekci tagů pro vytváření šablon.

Caché CSP tagy lze rozdělit do několika skupin podle účelu na tagy základní, řídící, dotazovací atd.

5.4.1 Základní tagy

Základem jazyka CSP je několik tagů, které zajišťují propojení CSP stránky s databází a tím i s databázovým severem.

Mezi základní tagy patří **CSP:OBJECT** a **CSP:CLASS**.

5.4.1.1 CSP:OBJECT

Tag **SCP:OBJECT** realizuje umístění Caché objektu (instance Caché třídy) na CSP stránce.

Syntaxe:

<CSP:OBJECT NAME="*jm_prom***" CLASSNAME="***jm_třídy***" OBJID="***hodnota_ID***">**

Vše si vysvětlíme na následujícím velmi jednoduchém příkladu. Tento příklad je pouze demonstrativní, složitější konstrukce s využitím tagu CSP:OBJECT jsou v souboru vypis.csp, který najdeme na přiloženém CD spolu s ostatními příklady.

Příklad 5.1: Ukázka použití tagu CSP: OBJECT

```
<csp:object name="osoba" classname="BP.kontakt" objid='1'>
  <b>#(osoba.Prijmeni)# #(osoba.Jmeno)#</b>
```

Na první řádce příkladu máme definovaný tag **CSP:OBJECT** s nějakými atributy. Tím říkáme, že chceme, aby byl na tuto stránku umístěn Caché objekt, se kterým budeme dále pracovat. O jaký objekt se jedná, nám specifikují hodnoty atributů. K tomuto účelu slouží atributy **CLASSNAME** a **OBJID**. Dále potřebujeme vědět, jak můžeme s tímto objektem pracovat a odkazovat se na něj. To zajišťuje atribut **NAME**.

V našem případě chceme na CSP stránku umístit již vytvořený objekt, který je instancí třídy *Kontakt* z balíčku *BP*. To použijeme jako hodnotu atributu **CLASSNAME**. Jednoznačnou identifikací tohoto objektu je jeho vlastnost ID. Tu využijeme v atributu **OBJID**. Chceme vytvořit objekt, který má ID rovno *1*.

Samotné vytvoření objektu je realizováno prostřednictvím volání metody **%OpenId** s parametrem, který určuje atribut OBJID.

Pro jistotu zde uvedeme tabulku doposud vytvořených objektů- instancí třídy *BP.Kontakt*, abychom měli představu, jaký objekt chceme na CSP stránce umístit.

%ID	jmeno	prijmeni	datumNar	…
1	**Marie**	**Kusá**	**26/04/1973**	
2	Karel	Bláha	15/02/1946	
3	Radim	Novotný	25/01/1981	
4	Karel	Novák	15/03/1949	
5	Ladislav	Vohnout	12/06/1963	
6	Pavel	Kramář	16/05/1974	

Tab.5.2 Tabulka vlastností doposud vytvořených objektů třídy BP.Kontakt

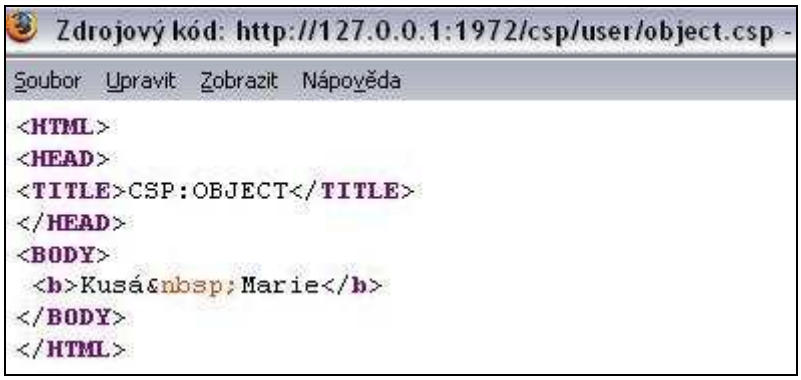
Nyní máme vytvořený objekt, se kterým chceme dále na CSP stránce pracovat. To je možné prostřednictvím lokální proměnné, kterou specifikuje atribut **NAME**. V našem případě je to *osoba*.

Nyní máme na stránce umístěný objekt. Následující řádka kódu realizuje výpis vlastností tohoto objektu. Tento výpis je proveden substitucí hodnot, kterou zajistí výrazy v Caché ObjectScriptu (blíže viz kapitola 5.6). Kterou vlastnost a pro který objekt chceme vypsat, zajišťuje tento výraz, který je tvořen názvem objektu a vlastnosti oddělené tečkou.

Výsledná webová stránka vypadá následovně:

Obr. 5.2 Výřez webové stránky vygenerované z příkladu 5.1

Pokud si necháme vygenerovat zdrojový kód, zjistíme, že je zde pouze čisté HTML, server vykonal požadované operace a vygeneroval výslednou stránku.



Obr. 5.3 Zdrojový kód stránky z obr 5.2

Dalším typickým příkladem použití tagu **CSP:OBJECT** je v souvislosti s formuláři. V tomto případě je vytvořený objekt svázaný s elementy HTML formuláře. Jednotlivé elementy tohoto formuláře pak musí obsahovat atribut **CSPBIND**, který propojuje tyto elementy s vlastnostmi objektu.

Příklad 5.2: Ukázka použití tagu CSP: OBJECT s formulářem

```
<csp:object name="polozka" classname="BP.kontakt"
        OBJID=#(%request.Get("OBJID"))#>
<form name="form" cspbind="polozka" cspjs="All" onsubmit='return
        form_validate();'>
  <table cellpadding="3">
    <tr>
      <td>ID:</td>
      <td><input type="text" name="sys_Id" cspbind="%Id()" readonly></td>
    </tr><tr>
      <td>*Jméno:</td>
      <td><input type="text" name="Jm" cspbind="Jmeno" csprequired></td>
    </tr><tr>
      <td>*Příjmení:</td>
      <td><input type="text" name="Prijmeni" cspbind="Prijmeni"
          csprequired></td>
    </tr>
…
```

Použitím tagu CSP:OBJECT společně s formulářem říkáme, že chceme, aby tento formulář odkazovat na instance třídy specifikované tagem CSP:OBJECT. Formulář poté může i instance dané třídy vytvářet. Jak již bylo řečeno výše, jednotlivé prvky tohoto formuláře musí mít nastaveny hodnoty atributu CSPBIND. Těmito hodnotami jsou vlastnosti objektů třídy specifikované atributem

CLASSNAME tagu CSP:OBJECT. Také hlavička formuláře musí mít nastaven atribut CSPBIND. Hodnotou tohoto atributu je nyní jméno proměnné odkazující na objekty (hodnota atributu NAME tagu CSP:OBJECT).

Hodnotou atributu OBJID tagu CSP:OBJECT tentokrát není číslo, ale funkce, která získá ID zobrazené instance třídy.

Tag CSP:OBJECT má tři atributy. Všechny tyto atributy jsme použili v obou zde zmíněných příkladech. Následující tabulka tyto atributy shrnuje:

Atribut	Popis	Hodnota
CLASSNAME	Jméno třídy, jejíž instanci chceme vytvořit	Platný název třídy (včetně balíčku)
NAME	Jméno lokální proměnné používané pro odvolávání se na vytvořený objekt	Řetězec
OBJID	Hodnota identifikátoru objektu, který má být otevřen	Řetězec

Tab.5.3 Atributy tagu CSP: OBJECT

5.4.1.2 CSP:CLASS

CSP:CLASS tag určuje charakteristiky třídy která je vygenerovaná při zkompilování CSP stránky. Lze takto například specifikovat stránku jako soukromou - privátní , nebo stránku zakódovat…

Syntaxe:

<**CSP:CLASS [***seznam atributů viz tab. 5.4***]**>

Příklad 5.3: Ukázka použití tagu CSP: CLASS- vytvoření soukromé stránky

```
<body>
<csp:class private="1">
<p>soukromá stránka </p>
</body>
```

Tento příklad demonstruje, jak jednoduše se vytvoří soukromá stránka. V kontextu tohoto příkladu to je vcelku zbytečné, ale pro demonstraci postačující. Mnohem lepším příkladem je například internetový obchod, kde po přihlášení lze nakupovat na svůj účet, nebo emailová schránka aj.

Na rozdíl od veřejných stránek, které jsou dostupné odkudkoli (mají stálou URL adresu), na soukromé stránky je přístup možný pouze prostřednictvím odkazu z jiné CSP stránky. Jejich URL adresa je zakódovaná a platí pouze pro jeden přístup.

Příklad 5.4: Ukázka použití tagu CSP: CLASS- odkaz na soukromou stránku

```
<body>
  <a href="private.csp">vstup na soukromou stránku</a>
</body>
```

Pokud klikneme na tento odkaz, zobrazí se nám soukromá stránka, jejíž adresa se vždy automaticky vygeneruje a pokaždé je jiná. To je zajištěno pomocí CSP tokenu, což je náhodně vygenerovaný identifikátor aktuální relace.

Obr. 5.4 Různé URL adresy soukromé stránky

Následující tabulka uvádí všechny možné atributy tagu CSP:CLASS.

Atribut	Popis	Hodnota
DOMAIN	Výchozí hodnota zprávy doméně ve vlastnosti DOMAIN objektu %CSP:Response	Řetězec
ENCODED	Parametry dotazů mohou být zašifrovány (0 nejsou, 1 jsou, 2jsou a nezašifrované budou odstraněny	"0", "1" nebo "2"
ERRORPAGE	Jméno chybové stránky, která má být v případě problémů vyvolaná	Řetězec
EXPIRES	Výchozí hodnota vlastnosti EXPIRES objektu %CSP:Response	Řetězec
IMPORT	Čárkou oddělený seznam importovaných balíčků	Řetězec
INCLUDES	Čárkou oddělený seznam vložených souborů	Řetězec

Atribut	Popis	Hodnota
PRIVATE	Specifikování, zda bude daná stránka veřejná (PUBLIC) nebo soukromá (PRIVATE)	Logická “1“ nebo “0“
SUPER	Čárkou oddělený seznam rodičovských tříd (první musí být %CSP.Page)	Řetězec

Tab.5.4 Atributy tagu CSP:CLASS

5.4.2 Řídící tagy

Stejně jako v ostatních programovacích jazycích, i na CSP stránkách můžeme požadovat, aby byl nějaký úsek kódu proveden opakovaně - proto také CSP obsahuje sadu řídících tagů. Mezi tyto tagy patří jednoduchá podmínka, realizovaná pomocí tagu **CSP:IF**, cykly **CSP:WHILE** a **CSP:LOOP** aj.

5.4.2.1 CSP:IF, CSP:ELSE, CSP:ELSEIF

Jedná se o strukturovaný podmíněný příkaz, který se provede pouze tehdy, je-li splněna podmínka.

Tag CSP:IF reprezentuje jednoduchou podmínku. Chceme-li vícenásobně větvit kód, použijeme uvnitř jeho těla tag CSP:ELSEIF, který specifikuje další podmínku. Pokud není žádná podmínka splněna, provede se blok začínající tagem CSP:ELSE.

Syntaxe:

<CSP:IF CONDITION=“*podmínka 1***“>***Blok 1*

[<CSP:ELSEIF CONDITION=”*podmínka 2***”>***Blok 2***]**

[<CSP:ELSE>*Blok 3***]**

</CSP:IF>

Popis syntaxe:

Je-li(If) podmínka splněna (=True), **potom proveď** Blok 1. **Pokud není splněna první podmínka, zjisti, zda je splněna další a je-li (ElseIf)** splněna, **potom proveď** Blok 2, nejsou-li splněny žádné podmínky, **proveď (Else)** Blok 3. Ukonči **podmíněný příkaz.**

Všimněme si v syntaxi, že klíčová slova CSP:ELSEIF a CSP:ELSE nejsou povinné. Příkaz tedy může také vypadat takto:

<CSP:IF CONDITION="*podmínka***">** *Blok 1* **</CSP:IF>**

Pokud je splněna podmínka, **splň** Blok 1, **jinak** (není-li splněna) pokračuj dál.

Více si vysvětlíme v popisu příkladu 5.5, který dle aktuálního data vypíše, o jaký den v týdnu se jedná.

Příklad 5.5: Ukázka použití tagu CSP:IF

```
<BODY>
<h2 align=center>Dnes je:
<CSP:IF condition="$ZDate($Horolog,10)=0">neděle
<CSP:ELSEIF condition="$ZDate($Horolog,10)=1">pondělí
<CSP:ELSEIF condition="$ZDate($Horolog,10)=2">úterý
<CSP:ELSEIF condition="$ZDate($Horolog,10)=3">středa
<CSP:ELSEIF condition="$ZDate($Horolog,10)=4">čtvrtek
<CSP:ELSEIF condition="$ZDate($Horolog,10)=5">pátek
<CSP:ELSE>sobota
</CSP:IF>
#($ZDate($Horolog,4))#.
</h2>
</BODY>
```

Tento příklad zobrazuje na webové stránce datum, např.: **Dnes je sobota 5/12/2006**.

Tělo tagu **CSP:IF** se zobrazí, pokud bude **podmínka** splněna (**TRUE**). Tuto podmínku specifikuje atribut **CONDITION**. To je také jediný atribut tohoto tagu a je povinný. Vyhodnocení podmínky provádí server.

Uvnitř těla máme další tagy **CSP:ELSEIF** a **CSP:ELSE**, které specifikují dodatečné průběhy výpočtu. Tyto tagy se nemohou objevit samostatně, pouze v těle CSP:IF.

Pokud není splněna podmínka CSP:IF, prochází se tělo a pokud se nalezne tag CSP:ELSEIF, vyhodnotí se podmínka u tohoto tagu.

V našem případě to znamená, že pokud hodnota výrazu **$ZDate($Horolog,10)** bude rovna například 3 (to odpovídá středě), budou se postupně testovat podmínky. Nejprve u CSP:IF, poté u CSP:ELSEIF, dokud se nenalezne ta správná, která bude vyhodnocena jako pravdivá. V našem příkladě je to třetí použití tagu CSP:ELSEIF.

Provede se kód přímo následující za tímto tagem. Konec odpovídajícího bloku specifikuje použití dalšího tagu CSP:ELSEIF a přeskočí se za celý blok této struktury, ohraničené koncovým tagem CSP:IF.

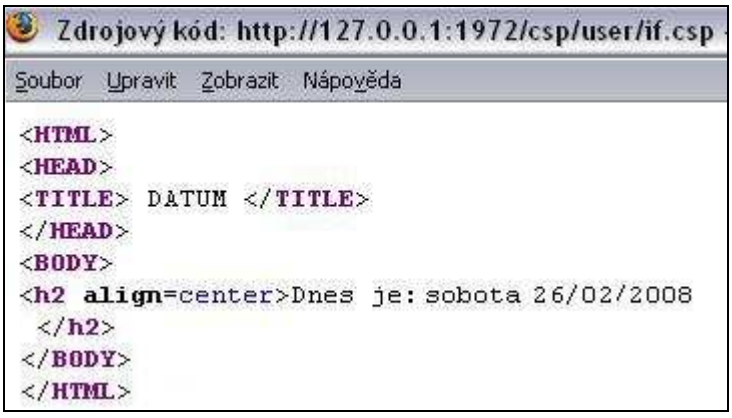
Takto by vypadal průchod touto strukturou, pokud by dnes byla středa. Jak by ale vypadala, pokud by dnes byla sobota? Odpověď je jednoduchá, vyhodnotily by se všechny podmínky jako nepravdivé (**FALSE**) a provedl by se blok ohraničený tagem **CSP:ELSE** a koncovým tagem **CSP:IF**.

Na tomto příkladu je také vidět, jak je možné použít speciální proměnnou **$Horolog** (blíže vysvětleno v kapitole 4.1.5 Speciální proměnné) a funkce **$ZDate** (blíže vysvětleno v kapitole 4.3.4 Práce s datem a časem).

Výsledná webová stránka vypadá následovně:

Obr. 5.5 Výřez webové stránky vygenerované z příkladu 5.5

Pokud si necháme vygenerovat zdrojový kód, zjistíme, že je zde pouze čisté HTML, server vykonal požadované operace a vygeneroval výslednou stránku.



Obr. 5.6 Zdrojový kód stránky z obr 5.5

Jak vyplývá z příkladu 5.5, tag CSP:IF a CSP:ELSEIF mají jediný atribut. Tím je CONDITION, který je u obou tagů povinný.

Tag	Atribut	Popis	Hodnota
CSP:IF, CSP:ELSEIF	*CONDITION*	Výraz ohodnocený za běhu - při kompilaci.	Výraz vyhodnocovaný na straně serveru

Tab.5.5 Atributy tagu CSP:IF a CSP:ELSEIF

5.4.2.2 CSP:LOOP

Tag **CSP:LOOP** opakovaně vykonává obsah (tělo cyklu) dokud počítadlo nedosáhne dané koncové hodnoty. Tento cyklus tudíž použijeme v případech, kdy předem známe, kolikrát se má daný cyklus provést.

Syntaxe:

<CSP:LOOP COUNTER="*jméno počítadla***" [FROM="***počátek***"]**
[STEP="*krok***"] [TO="***konec***"]>**
Tělo cyklu
</CSP:LOOP>

Popis syntaxe:

1. Jméno počítadla přiřaď hodnotu *počátek*.
2. Porovnej hodnotu počítadla (*jméno počítadla*) a *konec*
   a. *Jméno počítadla*>= *konec* pak skonči
   b. jinak proveď tělo cyklu.
3. *Jméno počítadla* inkrementuj o *krok*
4. Pokračuj opět ad 2

Příklad 5.6: Ukázka použití tagu CSP: LOOP, CSP:CONTINUE a CSP:QUIT

```
<BODY>
<CSP:LOOP counter="i" from="1" step="1" to="20">
  <CSP:IF CONDITION="(i*i)>300">
    <p>Druhá mocnina čísel větších než <b>#(i-1)#</b> je větší než 300.</p>
    <CSP:QUIT>
  <CSP:ELSE>
    <p>Druhá mocnina čísla b>#(i)#</b je <b>#(i*i)#.</b>
  </CSP:IF>
```

```
  <CSP:IF CONDITION="(i**3)>1000">
    Třetí mocnina čísla <b>#(i)#</b je větší než 1.000.
    <CSP:CONTINUE>
  <CSP:ELSE>
    Třetí mocnina čísla <b>#(i)#</b> je <b>#(i**3)#.</b>
  </CSP:IF>
  </p>
</CSP:LOOP>
</BODY>
```

Výsledná HTML stránka vypočítává druhou a třetí mocninu čísel do 20. To je zajištěno **loop** cyklem. Tělo tohoto cyklu je ohraničeno počátečním a koncovým tagem **CSP:LOOP**. Hlavička cyklu je tvořena čtyřmi atributy- **COUNTER**, **FROM**, **STEP** a **TO**.

Atribut **COUNTER** je povinný a specifikuje jméno počítadla, s nímž je možné v těle cyklu pracovat, ale nelze zde jeho hodnotu změnit - ta se inkrementuje pouze v hlavičce.

Atribut **FROM** určuje počáteční hodnotu. V našem případě je nastaven na hodnotu 1 - to je jeho implicitní hodnota a tudíž bychom ho klidně mohly pro tentokrát vypustit.

Atribut **STEP** určuje krok, neboli o kolik se bude počítadlo při každém průchodu cyklem inkrementovat. V našem případě je i u tohoto atributu hodnota nastavena na implicitní a jeho použití je zde tudíž zbytečné.

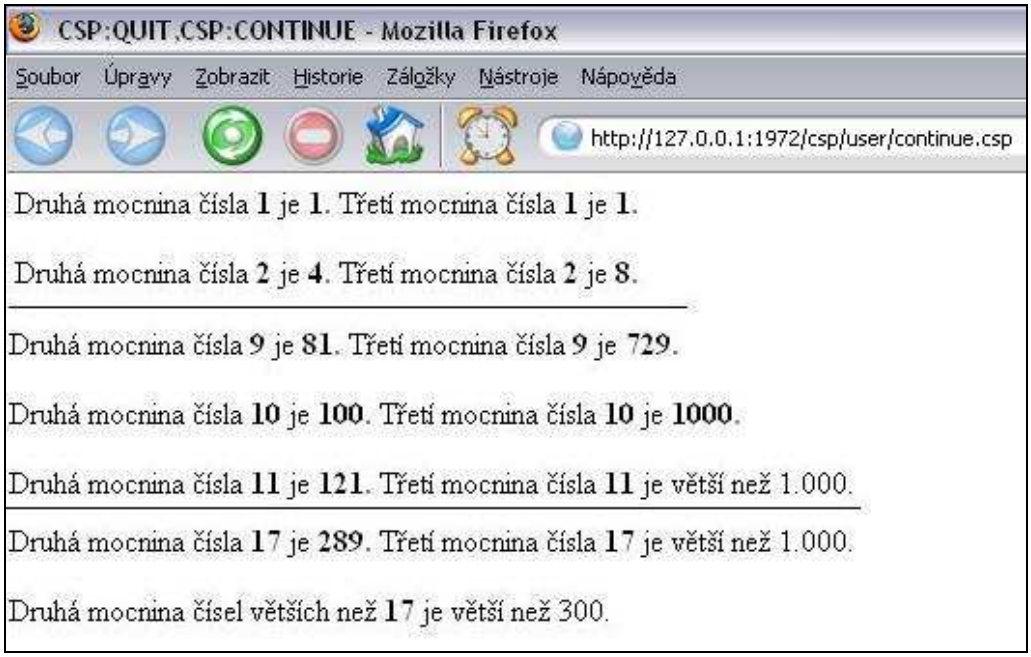
Atribut **TO** určuje horní mez počítadla. Také tento atribut má svou implicitní hodnotu nastavenou na 1, kterou jsme jeho použitím potlačili a vnutili mu námi požadovanou hodnotu 20.

Uvnitř tohoto cyklu jsou **dvě podmínky**. První obsahuje tag **CSP:QUIT**. Pokud je splněna podmínka, celý cyklus je ukončen. V opačném případě se vypočítá druhá mocnina daného čísla.

Druhá podmínka obsahuje tag **CSP:CONTINUE**. Pokud je podmínka splněna, je aktuální průchod cyklem ukončen a pokračuje se dalším průchodem. V opačném případě se vypíše třetí mocnina daného čísla.

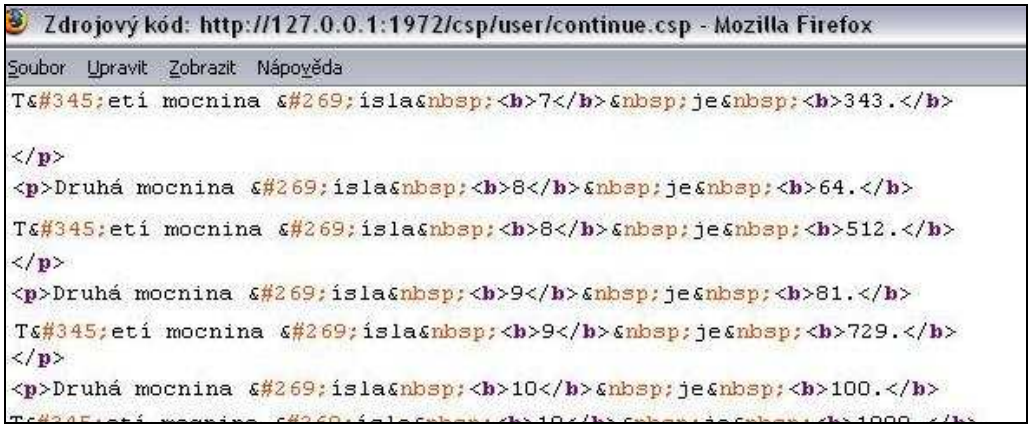
Tento příklad také demonstruje použití aritmetických operátorů (blíže vysvětleno v kapitole 4.2.1 Aritmetické operátory) a tagů reprezentující příkazy skoku- **CSP:CONTINUE** a **CSP:QUIT** (kapitola 5.4.2.4).

Výsledná webová stránka vypadá následovně (zde je zobrazena pouze její část, vodorovné čáry naznačují, kde došlo k vypuštění některých řádek):



Obr. 5.7 Výřez webové stránkya vygenerované z příkladu 5.6

Pokud si necháme vygenerovat zdrojový kód, zjistíme, že je zde pouze čisté HTML. Server vykonal požadované operace a vygeneroval výslednou stránku. Na obrázku níže je zobrazen pouze její fragment.



Obr. 5.8 Zdrojový kód stránky z obr 5.7

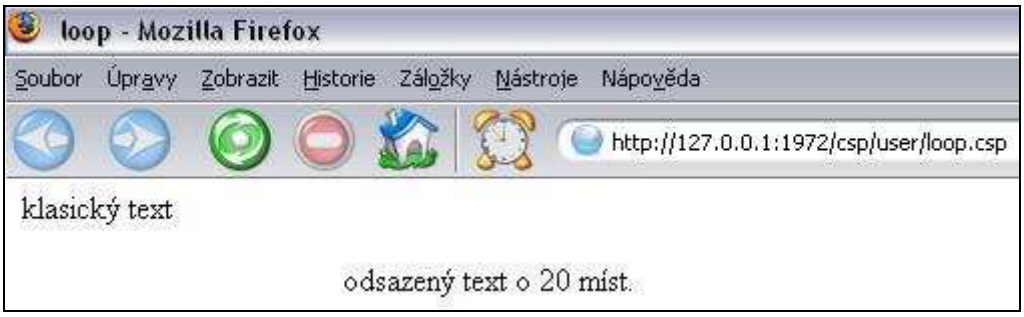
Další možností použití tagu CSP:LOOP je vytvoření odsazeného textu nebo bílých mezer v textu.

Příklad 5.7: Ukázka použití tagu CSP:LOOP

```
<BODY>
<p>klasický text</p>
<p>
<CSP:LOOP counter="i" to="20">
 
</CSP:LOOP>
odsazený text o 20 míst.</p>
</BODY>
```

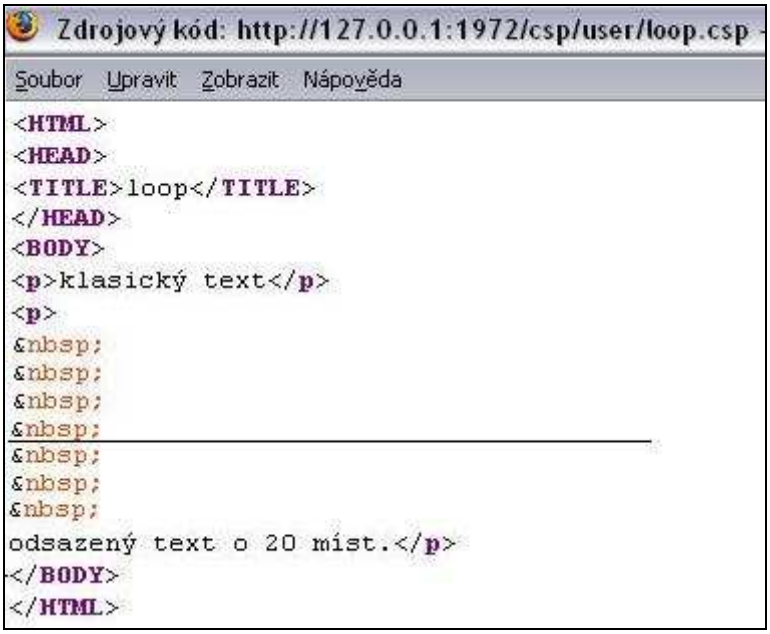
Výsledná HTML stránka zobrazuje klasický text odstavce v prvním odstavci. V druhém odstavci využíváme cyklu s předepsaným počtem opakování prostřednictvím tagu **CSP:LOOP**, díky němuž jsme počátek odstavce odsadily o 20 míst.

Tělo cyklu je tvořeno jedinou řádkou kódu a to je symbol pro bílou mezeru.



Obr. 5.9 Výřez webové stránky vygenerované z příkladu 5.7

Také zde zdrojový kód obsahuje pouze čisté HTML:



Obr. 5.10 Zdrojový kód stránky z obr 5.9

Následující tabulka shrnuje všechny atributy tagu CSP:LOOP. Atribut **COUNTER** je povinný, ostatní atributy jsou nepovinné a proto mají nastavenou svou implicitní hodnotu. Pro všechny tyto tři atributy (**FROM**, **STEP** a **TO**) je implicitní hodnota **1**.

Atribut	Popis	Hodnota
COUNTER	Povinný. Jméno proměnné počítadla používané tímto cyklem	Platný název proměnné
FROM	Počáteční hodnota čítače cyklu.	Číslo nebo výraz
STEP	O kolik se změní hodnota čítače při každém průchodu cyklem.	Číslo nebo výraz
TO	Konečná hodnota čítače cyklu.	Číslo nebo výraz

Tab.5.6 Atributy tagu CSP:LOOP

5.4.2.3 CSP:WHILE

Tag **CSP:WHILE** reprezentuje cyklus s podmínkou na začátku. Technologií CSP jsou implementovány dvě možnosti jeho použití. První z nich je klasické, jak ho známe z klasických programovacích jazyků. Druhá z nich je cyklus nad ukazatelem, který je specifikován atributem **CURSOR**.

Nyní se seznámíme s první variantou použití CSP:WHILE:

Syntaxe s podmínkou:

<**CSP:WHILE [CONDITION=”***výraz***”] [COUNTER=”***jméno***”]**>

Tělo cyklu

</**CSP:WHILE**>

Popis syntaxe:

Jedná se o cyklus s podmínkou na začátku. Tělo cyklu se opakovaně provádí, dokud je podmínka vyhodnocena jako pravdivá (TRUE). Podmínku specifikuje hodnota atributu **CONDITION** a vyhodnocení této podmínky provádí server.

Tag CSP:WHILE může mít i další atributy a to atribut počítadlo (**COUNTER**), který uchovává informaci o počtu provedených průchodů cyklem. Pokud chceme s takovouto informací pracovat, musíme počítadlo definovat. Tato proměnná není automaticky vytvořená.

Příklad 5.8: Ukázka použití tagu CSP:WHILE

```
<BODY>
  <CSP:WHILE COUNTER="i" CONDITION="i**2<300">
    <p>Druhá mocnina čísla<b>#(i)#</b> je <b>#(i*i)#.</b>
    <CSP:IF CONDITION="(i**3)>1000">
      Třetí mocnina čísla <b>#(i)#</b> je větší než 1.000.
      <CSP:CONTINUE>
    <CSP:ELSE>
      Třetí mocnina čísla <b>#(i)#</b> je<b>#(i*i*i)#.</b>
    </CSP:IF>
    </p>
  </CSP:WHILE>
</BODY>
```

Tento příklad má téměř stejnou funkčnost, jako příklad 5.6 z kapitoly věnované tagu CSP:LOOP.

Nejprve definujeme **WHILE cyklus** a počítadlem *i*. Podmínkou (**CONDITION**) pro provedení cyklu je, aby druhá mocnina počítadla byla menší než *300*.

Při prvním vyhodnocování podmínky je hodnota počítadla *0*. Ihned poté, co se vejde do těla cyklu, hodnota počítadla se inkrementuje. Z toho také vyplývá

rozdíl od předchozího příkladu, kdy se vypsalo pouze *17* čísel. Nyní se vypíše čísel *18*, protože poslední hodnota počítadla, se kterou se ještě podmínka vyhodnotí jako pravdivá je *17*. Tato hodnota se ještě inkrementuje v těle tohoto cyklu, kde se již tedy pracuje s hodnotou *18*.

Pokud bychom chtěli, aby byla funkčnost obou příkladů stejná, musel by se výraz v podmínce změnit na *(i-1)**2<300*.

Výsledná webová stránka i její kód je téměř totožná s příkladem 5.6, jen je zde o jeden záznam více.

Nyní se podíváme na druhou variantu použití tagu CSP:

Syntaxe s CURSOR:

<SCRIPT LANGUAGE="SQL" CURSOR=" *jméno SQL dotazu***">**
SQL dotaz
</SCRIPT>
<CSP:WHILE [CURSOR="*jméno SQL dotazu***"]**
[INTO="*seznam používaných proměnných***"]**
[COUNTER="*jméno***"] >**
Tělo cyklu
</CSP:WHILE>

Popis syntaxe:

Jedná se o cyklus, který simuluje průchod SQL tabulkou vytvořenou pomocí serverové metody psané v SQL (**SCRIPT LANGUAGE="SQL" CURSOR="***jméno SQL dotazu***"**).

Atribut **CURSOR** nám specifikuje jméno dotazu, se kterým se bude pracovat, proto hodnota atributu CURSOR v hlavičce metody a hlavičce WHILE cyklu musí být stejná.

Atribut **INTO** nám říká se kterými sloupci (z vygenerované tabulky) se bude pracovat. Tyto sloupce musí být také součástí SELECT klauzule SQL dotazu.

I v této variantě může mít tag CSP:WHILE atribut počítadlo (**COUNTER**), který uchovává informaci o počtu provedených průchodů cyklem.

Příklad 5.9: Ukázka použití tagu CSP:WHILE s CURSOR

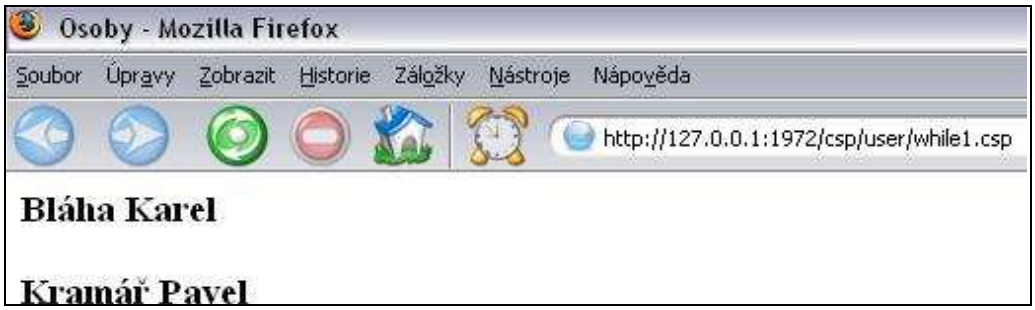
```
<BODY>
  <script language="SQL" cursor="dotaz">
      Select prijmeni,jmeno From SQLUser.BP.kontakt
  </script>
  <CSP:WHILE cursor="dotaz" into="prijmeni,jmeno">
      <H3>#( prijmeni)# #( jmeno)#</H3>
  </CSP:WHILE>
</BODY>
```

Jak již bylo řečeno v popisu syntaxe, použití tagu CSP:WHILE musí předcházet definice SQL dotazu prostřednictvím skriptu. Jazykem tohoto skriptu musí být **SQL**, což specifikuje atribut **LANGUAGE**. Dalším atributem, který musí tento skript mít, je **CURSOR**. Hodnotou tohoto atributu je jméno SQL dotazu, které je použito i v hlavičce WHILE cyklu. V našem příkladě je to *dotaz*. Tělo skriptu je tvořeno SQL dotazem, který vybírá některé položky z tabulky *kontakt*. Blíže se metodám (skriptům) na CSP stránkám věnuje kapitola 5.5.

Hlavička **CSP:WHILE** tagu má atributy **CURSOR** a **INTO**. Hodnoty těchto atributů vychází z hodnot uvedených v metodě *dotaz*. Hodnota atributu **CURSOR** musí odpovídat hodnotě téhož atributu z hlavičky skriptu. Hodnota atributu **INTO** je dána sloupci, které v těle metody získáváme v tamním SQL dotazu.

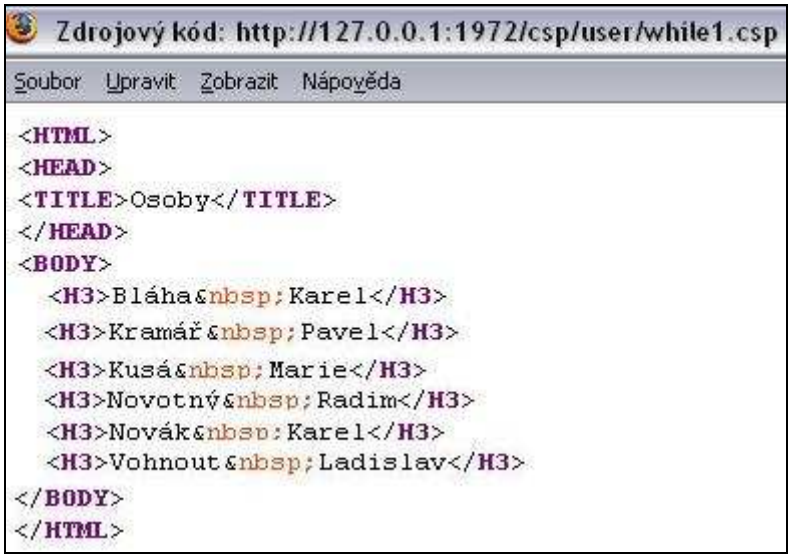
Tělo while cyklu nám specifikuje zobrazení detailů o jednotlivých kontaktech. To je realizováno prostřednictvím položek, které jsme specifikovali v atributu INTO. Přístup je prostřednictvím výrazu v Caché ObjectScriptu (blíže viz kapitola 5.6). Tento výraz je tvořen pouze názvem této položky.

Výsledná webová stránka má následující vzhled:



Obr. 5.11 Výřez webové stránky vygenerované z příkladu 5.9

Také zde zdrojový kód vygenerovaný v prohlížeči obsahuje pouze čisté HTML:



Obr. 5.12 Zdrojový kód stránky z obr 5.11

Přepis předchozího příkladu bez použití atributu CURSOR:

Příklad 5.10: Přepis př. 5.9 bez použití atr. CURSOR

```
<BODY>
  <script language="SQL" name="os">
      Select prijmeni,jmeno From SQLUser.BP.kontakt
  </script>
  <CSP:WHILE condition="os.Next()">
      <H3>#(os.Data("prijmeni"))# #(os.Data("jmeno"))#</H3>
  </CSP:WHILE>
</BODY>
```

Také zde je nejprve definován skript, jehož jazyk (**LANGUAGE**) je **SQL**. Na rozdíl od předchozího příkladu hlavička tohoto skriptu neobsahuje atribut CURSOR. Tento atribut zde byl nahrazen atributem **NAME**. Hodnotou atributu NAME je jméno tohoto skriptu, v našem případě je to *os*. Tělo skriptu je tvořeno stejným SQL dotazem jako v předchozím příkladě.

Dalo by se říci, že tento skript vygeneruje tabulku. Přístup k jednotlivým položkám této tabulky je realizován pomocí metody **Data**, jejímž parametrem je název vlastnosti. Na další řádku v tabulce se přeskočí pomocí metody ***NEXT*** bez parametrů.

Za skriptem následuje WHILE cyklus. Hlavička **CSP:WHILE** tagu má jediný atribut a to **CONDITION**. Hodnotou tohoto atributu je podmínka, která určuje, zda má vygenerovaná tabulka další řádek. Dokud má tabulka další řádku, vypisují se informace o kontaktu.

Webová stránka i její zdrojový kód jsou totožné s předchozím příkladem.

Dalším příkladem použití tagu CSP:WHILE je v kapitole 5.4.3 věnované dotazovacím tagům. Jedná se o příklady 5.11 a 5.12. Oba tyto příklady jsou modifikací příkladu 5.6.

Následující tabulka shrnuje všechny možné atributy tagu CSP:WHILE:

Atribut	Popis	Hodnota
CONDITION	Serverový výraz vyhodnocený po každé iteraci. Pokud je výraz vyhodnocen jako pravdivý (TRUE), cyklus se opakuje znovu.	Serverový výraz
COUNTER	Jméno lokální proměnné, které je automaticky zvýšeno po každé iteraci.	Platný název proměnné
CURSOR	Pouze pokud tělo cyklu obsahuje SQL. Jedná se o SQL ukazovatel, který uchovává informace pro další iteraci.	Platný název SQL ukazatele
INTO	Pouze pokud tělo cyklu obsahuje SQL. Specifikuje proměnné z SQL tabulky, které se budou používat.	Čárkou oddělený seznam proměnných

Tab.5.7 Atributy tagu CSP:WHILE

Atribut **COUNTER (**neboli počítadlo) je dobrovolný, a pokud se použije, vytvoří se lokální proměnná, která se při každé iteraci inkrementuje. Před prvním průchodem se hodnota proměnné počítadlo nastaví na 0 a na začátku každého průchodu cyklem se její hodnota o 1 zvýší.

5.4.2.4 CSP:CONTINUE, CSP:QUIT

Oba tyto CSP tagy lze použít uvnitř těla tagu CSP:LOOP i CSP:WHILE. Tyto tagy ovlivňují provádění cyklu. Tag CSP:QUIT ukončuje nejvnitřnější neuzavřenou smyčku a okamžitě opouští cyklus. Tag CSP:CONTINUE skáče na

konec nejvnitřnější neuzavřené smyčky a tím vynutí další iteraci. V případě použití CSP:CONTINUE nedochází k ukončení cyklu.

Syntaxe CSP:CONTINUE:

<CSP:CONTINUE>

Syntaxe CSP:QUIT:

<CSP:QUIT>

Příklad použití viz kapitola 5.4.2.2 CSP:LOOP.

5.4.3 Dotazovací tagy

Dotazovací tagy realizují předdefinované nebo dynamické databázové dotazy. Jaký typ dotazu bude vykonán, závisí na použitém tagu a nastavení jeho atributů.

Všechny tyto tagy vytvářejí objekt **%ResultSet**, který reprezentuje tabulku výsledků vygenerovanou daným dotazem.

Do této skupiny tagů patří **CSP:QUERY** a **CSP:SQLQUERY**. Rozdíl mezi těmito tagy je v tom, že **CSP:QUERY** vyvolává předdefinovaný databázový SQL dotaz, který je součástí definice nějaké Caché třídy. Tag **CSP:SQLQUERY** je tvořen dvojicí tagů (počátečním a koncovým) a tělo tohoto tagu obsahuje požadovaný SQL dotaz. Tento tag realizuje dynamický databázový dotaz.

Funkčnost obou těchto tagů si ukážeme na příkladu z kapitoly 5.4.2.3 věnované tagu CSP:WHILE. Tento příklad ukazuje, jak je možné vypsat některá data uložená v databázi kontaktů. Příklad poněkud zjednodušíme a necháme si vypsat pouze jména osob uložených v databázi. To bude zcela postačující pro demonstraci použití dotazovacích tagů a následnou práci s vytvořeným objektem %ResultSet.

5.4.3.1 CSP:QUERY

Tag CSP:QUERY je jednou z nejdůležitějších komponent jazyka CSP. Jeho použití na CSP stránce realizuje předdefinovaný databázový dotaz. Tato realizace se skládá z několika dílčích kroků. Nejprve se provede připojení k vybrané třídě Caché a vykoná se zadaný dotaz. Nakonec je množina výsledků tohoto dotazu předána zpět

CSP stránce. Tato množina výsledků je reprezentována objektem **%ResultSet**. Přístup k tomuto objektu je možný pomocí serverové proměnné, jejíž jméno je specifikované atributem **NAME**.

Syntaxe:

<**CSP:QUERY** *seznam_atributů*>

Příklad 5.11: Ukázka použití tagu CSP: QUERY

```
Třída BP.Kontakt
Query vsichni() As %SQLQuery(CONTAINID = 1) {
 SELECT * FROM kontakt ORDER BY prijmeni, jmeno
}

CSP stránka
<BODY>
 <CSP:QUERY NAME="dotaz" CLASSNAME="BP.kontakt"
          QUERYNAME="vsichni">
 <CSP:WHILE CONDITION=dotaz.Next()>
  <H3>#(dotaz.Get("prijmeni"))# #(dotaz.Get("jmeno"))#</H3>
  …
 </CSP:WHILE>
</BODY>
```

V příkladu 5.11 máme nejprve zobrazený fragment kódu třídy Kontakt obsahující SQL dotaz, který budeme vyvolávat.

Na CSP stránce máme nejprve definovaný tag **CSP:QUERY**, který vytvoří objekt **%ResultSet**, jehož název je specifikován atributem **NAME**. V našem případě je to *dotaz*. Pro vytvoření tohoto objektu je nutné ještě říci, realizací jakého dotazu se má vytvořit - například prostřednictvím dotazu *vsichni*, který je definován ve třídě *kontakt*. Tato třída je součástí balíčku *BP*. Jméno tohoto dotazu určuje atribut **QUERYNAME**. Hodnota tohoto atributu musí být název existujícího dotazu definovaného ve třídě specifikované atributem **CLASSNAME**. Z předchozí věty vyplývá, že hodnotou atributu CLASSNAME je název třídy obsahující dotaz, který má být vykonán. Z důvodu jednoznačnosti je nutné, aby byl název třídy doplněn i o jméno balíčku, ve které je daná třída.

Objekt **%ResultSet** reprezentuje tabulku výsledků dotazu specifikovaného tagem CSP:QUERY. V našem případě tato tabulka vypadá následovně:

%ID	jmeno	prijmeni	datumNar	vek	…
2	Karel	bláha	15/02/1946	62	
6	Pavel	Kramář	16/05/1974	33	
1	Marie	Kusá	26/04/1973	34	
3	Radim	Novotný	25/01/1981	27	
4	Karel	Novák	15/03/1949	58	
5	Ladislav	Vohnout	12/06/1963	44	

Tab.5.8 Tabulka výsledků reprezentující objekt %ResultSet z příkladu 5.11

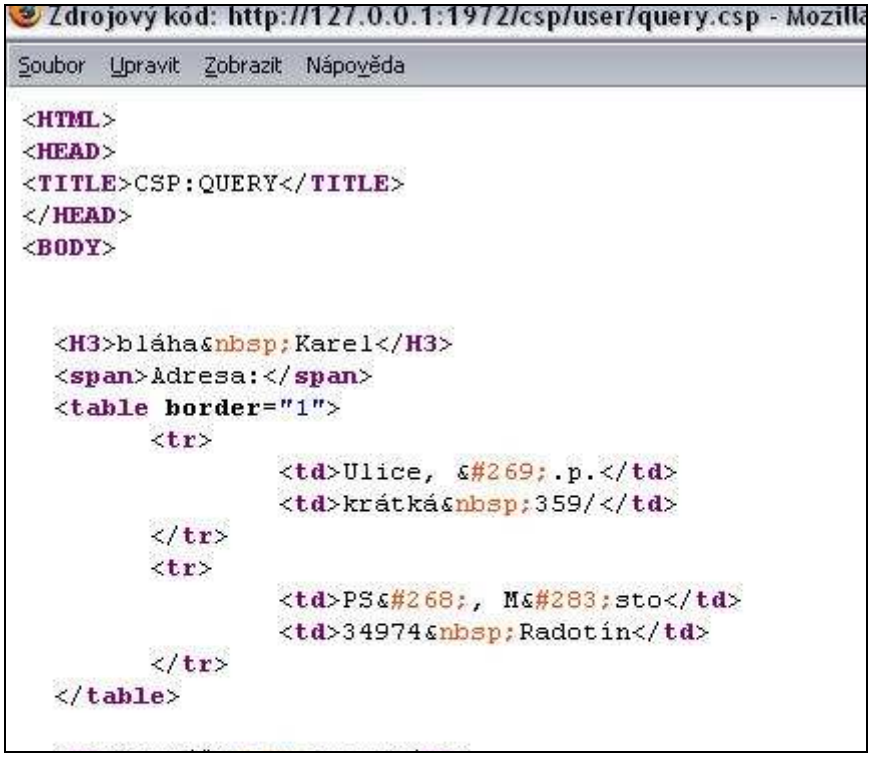
K jednotlivým položkám této tabulky se přistupuje pomocí metody *GET*, jejímž parametrem je název vlastnosti. Na další řádku v tabulce se přeskočí pomocí metody *NEXT* bez parametrů.

V našem příkladě za definicí tagu CSP:QUERY následuje WHILE cyklus, který pracuje s objektem %ResultSet. Dokud má tabulka další řádku, vypisují se informace o kontaktu.

Výsledná webová stránka vypadá následovně:

Obr. 5.13 Výřez webové stránky vygenerované z příkladu 5.11

Pokud si necháme vygenerovat zdrojový kód, zjistíme, že je zde pouze čisté HTML - server vykonal požadované operace a vygeneroval výslednou stránku.



Obr. 5.14 Zdrojový kód stránky z obr 5.13

Tento příklad demonstruje základní použití tohoto tagu. V příkladu samozřejmě nejsou použity všechny možné atributy. Ty jsou shrnuty v následující tabulce obsahující všechny atributy tagu CSP:QUERY.

Atribut	Popis	Hodnota
CLASSNAME	Jméno třídy, ve které je požadovaný dotaz	Platný název třídy (včetně balíčku)
MODE	Mód dotazu	LOGICAL, ODBC, DISPLAY nebo SYSTEM
NAME	Jméno lokální proměnné používané pro odvolávání se na objekt vytvořený tímto dotazem	Řetězec
P1, P2…	Hodnota parametru	Řetězec
QUERYNAME	Jméno třídního dotazu, který má být spuštěn	Platný název dotazu

Tab.5.9 Atributy tagu CSP: QUERY

5.4.3.2 CSP:SQLQUERY

Na rozdíl od předešlého tagu tag **CSP:SQLQUERY** nevyvolává dotaz definovaný v nějaké Caché třídě, ale tento dotaz je přímo jeho součástí. Jedná se o dynamický dotaz definovaný SQL příkazem v jeho těle. Výsledky našeho dotazu jsou uloženy v objektu **%ResultSet**. Práce s tímto objektem je možná pomocí serverové proměnné, jejíž jméno je specifikované atributem **NAME**.

Tento tag je identický s metodou **<SCRIPT LANGUAGE="SQL">**, jak je původně použito v příkladu 5.9. Jediný rozdíl je v tom, že tag **CSP:SQLQUERY** je možné použít pro přesměrování dotazu k externí databázi přes Caché SQL bránu.

Syntaxe:

<CSP:SQLQUERY [*seznam atributů* **]>** *SQL dotaz* **</CSP:SQLQUERY>**

Popis syntaxe:

Tag CSP:SQLQUERY může mít větší množství atributů, některé z nich jsou uvedeny v tabulce níže (viz. Tab. 5.11 Atributy tagu CSP:SQLQUERY). Tento tag je párový. Jeho tělo je tvořeno SQL dotazem.

Vše si blíže vysvětlíme na přepisu předchozího příkladu 5.11.

Příklad 5.12: Ukázka použití tagu CSP:SQLQUERY

```
<BODY>
  <CSP:SQLQUERY NAME="dotaz">
    Select * From SQLUser.BP.kontakt
  </CSP:SQLQUERY>
  <CSP:WHILE CONDITION=dotaz.Next()>
    <H3>#(dotaz.Get("prijmeni"))# #(dotaz.Get("jmeno"))#</H3>
    <span>Adresa:</span>
    <table border="1">
      ...
    </table>
  </CSP:WHILE>
</BODY>
```

V příkladu 5.12 máme nejprve definovaný tag **CSP:SQLQUERY**, který vytvoří objekt **%ResultSet**, jehož název je specifikován atributem **NAME**.

V našem případě je to *dotaz*. Pro vytvoření tohoto objektu je nutné ještě v těle tohoto tagu specifikovat SQL dotaz, jehož realizací se daný objekt vytvoří.

Jak je vidět na příkladu 5.12, jedná se o klasický SQL dotaz. Pouze v jeho WHERE klauzuli musíme přesně napsat, v jakém balíčku se daná tabulka nachází. Tentokrát nehovoříme o třídě, ale o tabulce, neboť zde používáme relační přístup k datům. Proto je zde také použito kromě názvu balíčku ještě úložiště tabulek. V našem případě to je *SQLUser*, za kterým následuje název balíčku a samozřejmě i název tabulky.

Od předchozího příkladu se tabulka výsledků reprezentovaná objektem **%ResultSet** liší pouze v pořadí jednotlivých záznamů, protože jsme tentokrát v SQL dotazu vynechali ORDER BY klauzuli. Pořadí záznamů je určeno hodnotou %ID.

%ID	jmeno	prijmeni	datumNar	vek	…
1	Marie	Kusá	26/04/1973	34	
2	Karel	Bláha	15/02/1946	62	
3	Radim	Novotný	25/01/1981	27	
4	Karel	Novák	15/03/1949	58	
5	Ladislav	Vohnout	12/06/1963	44	
6	Pavel	Kramář	16/05/1974	33	

Tab.5.10 Tabulka výsledků reprezentující objekt %ResultSet z příkladu 5.10

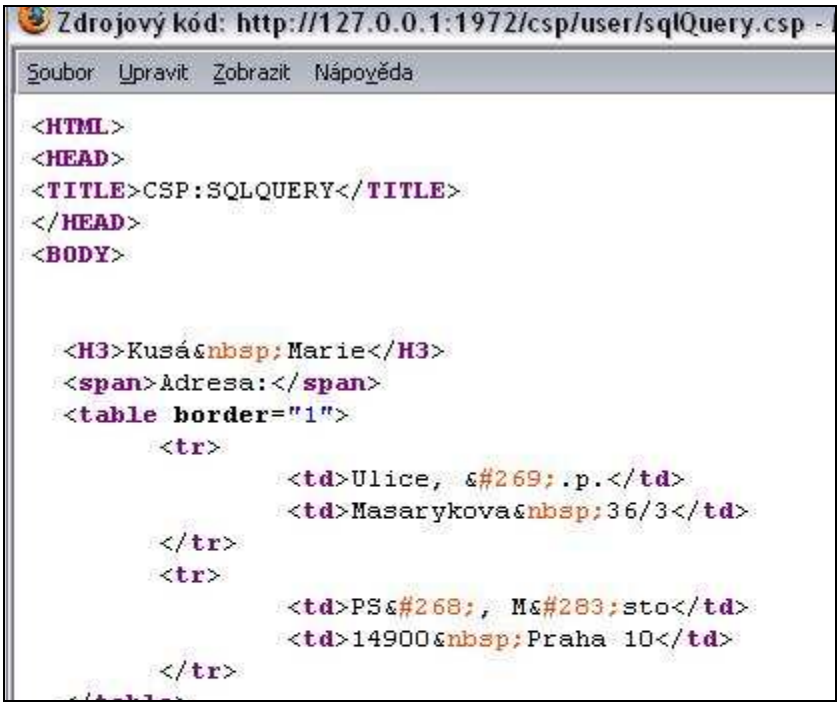
Přístup k jednotlivým položkám je stejný jako v předchozím příkladě pomocí metod GET a NEXT.

Opět za definicí tagu CSP:SQLQUERY následuje WHILE cyklus. Tento cyklus je v obou příkladech naprosto stejný.

Výsledná webová stránka vypadá následovně:

Obr. 5.15 Výřez webové stránky vygenerované z příkladu 5.12

Také zdrojový kód vygenerovaný prohlížečem se od předchozího příkladu příliš neliší. Pouze pořadí záznamů je zde jiné:



Obr. 5.16 Zdrojový kód stránky z obr 5.15

Následující tabulka opět zobrazuje všechny možné atributy tohoto tagu.

Atribut	Popis	Hodnota
DATASOURCE	Specifikuje jméno externího datového zdroje, který bude pro tento dotaz používán	Řetězec
MODE	Mód dotazu	LOGICAL, ODBC, DISPLAY nebo SYSTEM
NAME	Jméno lokální proměnné používané pro odvolávání se na objekt vytvořený tímto dotazem	Řetězec
P1, P2…	Hodnota parametru (budou nahrazeny za běhu v jeho **WHERE** klauzuli, kde je znak **?**, který se nahradí za požadované parametry)	Řetězec
PASSWORD	Dobrovolné heslo pro externí datový zdroj	Řetězec
USERNAME	Dobrovolné uživatelské jméno pro externí datový zdroj	Řetězec

Tab.5.11 Atributy tagu CSP:SQLQUERY

Pokud specifikujeme nepovinný atribut DATASOURCE, dojde k přesměrování dotazu k externí databázi používající Caché relační bránu (**Caché Relational Gateway**).

5.4.4 Vyhledávání

Caché server pages obsahuje také tag, jehož pomocí lze vytvořit vyhledávací stránku, která vyhledává objekty podle zadaných kritérií.

5.4.4.1 CSP:SEARCH

Tag CSP:SEARCH vytváří JavaScript funkci, která vytvoří vyhledávací stránku, kde je možné podle zadaných kritérií vyhledávat data z databáze. Toto hledání je realizováno prostřednictví SQL dotazu, který také specifikuje tento tag.

Vyhledávací stránka podle dat zadaných uživatelem vykoná SQL dotaz a jeho výsledky zobrazí v tabulce.

Tag CSP:SEARCH se nejčastěji používá ve spojení s HTML formulářem, kde je tlačítko, které vyvolá vyhledávací funkci a prostřednictví té i vyhledávací stránku. Pokud si necháme vyhledat nějaká data, pak poklepem na vybraný řádek v tabulce výsledků se tyto informace o objektu zobrazí ve formuláři na stránce, která vyvolala vyhledávací stránku.

Syntaxe:

<CSP:SEARCH [*seznam atributů***] >**

Příklad 5.13: Ukázka tagu CSP:SEARCH

```
<csp:search
  name="form_search"
  onselect="update"

  classname="BP.kontakt"
  where="Prijmeni As Příjmení,Jmeno As Křestní jméno,DN As Datum
    narození"
  predicates="select,%startswith,="
  select="Prijmeni,Jmeno,DN,Vek,bydliste.Mesto As město"
  order="Prijmeni,Jmeno"

  caption="vyhledávání kontaktů"
  options="popup,predicates,sortbox">
```

Náš vzorový příklad je tvořen pouze hlavičkou tagu CSP:SEARCH. Co tento tag dělá, již víme a nyní si povíme něco o jeho atributech. Tyto atributy si rozdělíme do tří skupin (v příkladě oddělených volnou řádkou) - základní atributy, atributy ovlivňující vyhledávací dotaz a atributy specifikující rysy vyhledávací stránky.

Základní atributy

Mezi základní atributy patří **NAME** a **ONSELECT**.

Atribut **NAME** určuje jméno vyhledávací JavaScript funkce. Jedná se o nepovinný atribut. My jsme ho definovali a jméno naší vyhledávací funkce je *form_search*.

Druhým zde zmíněným atributem je **ONSELECT**, který udává, co se má provést za akci, pokud klient poklepe na vybraný řádek v tabulce výsledků.

V našem případě se vybraný objekt má zobrazit ve formuláři. K tomuto účelu je definována funkce *update*.

Specifikace vyhledávacího dotazu

Mezi atributy ovlivňující vyhledávací dotaz patří například **CLASSNAME**, **WHERE**, **SELECT**, **ORDER**, **PREDICATES** aj.

U těchto atributů by si měl dát každý programátor pozor, protože jejich hodnoty vycházejí z objektového přístupu (například hodnotou atributu CLASSNAME je jméno třídy, **nikoli** jméno SQL tabulky).

Vyhledávací dotaz má následující obecný tvar:

SELECT *seznam_položek_SQL_tabulky _odpovídajících_hodnotě_atributu_SELECT*
FROM *jméno_SQL_tabulky_odpovídající_hodnotě_atr_CLASSNAME*
WHERE *seznam_položek_SQL_tabulky_odpovídajících_hodnotě_atr_WHERE*
ORDER BY *seznam_položek_SQL_tabulky_odpovídajících_hodnotě_atr_ORDER*

Příklad 5.14: Vyhledávací dotaz vygenerovaný z příkladu 5.13

```
SELECT ID, prijmeni, jmeno, datumNar, vek, bydliste_mesto
  FROM BP.Kontakt
  ORDER BY prijmeni, jmeno
```

Jako první máme uveden atribut **CLASSNAME**. Tento atribut je povinný a určuje jméno třídy, v jejíž instancích se bude vyhledávání provádět. My chceme vyhledávat v instancích třídy *Kontakt*, která je součástí balíčku *BP*. Pokud tento atribut převedeme do vyhledávacího dotazu, bude odpovídat FROM klauzuli.

Za atributem CLASSNAME následuje **WHERE**. Jeho hodnotou je seznam položek, které specifikují pole vstupů, podle kterých se dá vyhledávat. Ve vyhledávacím dotazu to odpovídá WHERE klauzuli. Tento atribut není povinný. Pokud bychom ho v hlavičce CSP:SEARCH tagu neuvedli, vyhledávání by bylo možné pouze prostřednictvím vlastnosti ObjectID. Tento atribut má ještě jednu funkci a to, pokud nebude zadaný atribut SELECT, jeho hodnota bude stejná jako hodnota tohoto atributu. Tzn. Seznam vlastností objekt specifikovaný atributem WHERE se použije také v SELECT klauzuli vyhledávacího dotazu.

Vlastnosti, které jsou součástí seznamu určující atribut WHERE a SELECT, je možné přejmenovat. K tomuto účelu slouží alias formule. Také tento aspekt jazyka CSP je použit ve vzorovém příkladě:

WHERE=“jmeno As Jméno, prijmeni As Příjmení, DN As Datum narození“

S atributem WHERE souvisí i atribut **PREDICATES**. Tento atribut určuje způsoby porovnání (zda se bude vyhledávat na základě porovnání řetězců, nebo podle počátečních znaků, aj.). Jeho hodnota je tvořena seznamem operátorů porovnání, jak je uvádí i následující tabulka 5.12 operátory porovnání. Jedná se o nepovinný atribut. V případě, že nebude definován, použijí se implicitní operátory.

Například v našem příkladě chceme, aby se příjmení vybíralo, jméno by mělo být porovnáváno podle počátečních znaků a datum narození by se mělo shodovat.

WHERE=“Prijmeni,Jmeno,DN“ PREDICATES=“ select,%startswith, =“

Operátor	Popis	Omezení
%STARTSWITH	Všechny hodnoty začínající zadanými znaky.	Pouze pro řetězce
=	Všechny hodnoty rovnající se zadanému vstupu	
<>	Všechny hodnoty nerovnající se zadanému vstupu	
>	Všechny hodnoty, které jsou větší než zadaný vstup	
<	Všechny hodnoty, které jsou menší než zadaný vstup	
BETWEEN	Všechny hodnoty mezi hodnotami na vstupu. Oddělovačem je ampersand (&)	
CONTAINS	Všechny hodnoty obsahující zadaný vstup.	Pouze pro řetězce
SELECT	Všechny hodnoty shodné s vybraným vstupem, vstup se nezapisuje, ale vybírá se ze seznamu	

Tab.5.12 Operátory porovnání

Dále v našem příkladě máme použit atribut **SELECT**. Tento atribut určí, které vlastnosti objektu se budou zobrazovat v tabulce výsledků hledání. Hodnota tohoto atributu je tvořena seznamem vlastností objektu. Také tento atribut ovlivňuje vyhledávací dotaz a to jeho SELECT klauzuli.

Posledním zde zmíněným atributem, ovlivňující vyhledávací dotaz, je **ORDER**. Tento atribut určuje setřídění údajů v tabulce výsledků. Ve vyhledávacím dotazu

to odpovídá ORDER BY klauzuli. Hodnotou tohoto atributu je opět seznam vlastností objektů.

Deklarace rysů vyhledávací stránky

Tag **CSP:SEARCH** má implementovanou i řadu atributů, které ovlivňují vzhled a chování vyhledávací stránky. Mezi tyto atributy patří **OPTIONS**, **CAPTION**, **FEATURES** a **MAXROWS**. V našem příkladě jsme se omezili pouze na použití těch nejzákladnějších a to **CAPTION** a **OPTION**.

Atribut **CAPTION** zobrazuje na vyhledávací stránce nadpis. Ten je zobrazen na začátku (na 1. řádku) vyhledávací stránky.

Atribut **OPTION** umisťuje na vyhledávací stránku další ovládací prvky. Jaké to jsou, zobrazuje následující tabulka:

Operátor	Popis
CLEARBTN	Přidává na vyhledávací stránku tlačítko vymazávající klientem zadané hodnoty.
DISPLAY	Zobrazuje na vyhledávací stránce hlavičku tabulky výsledků hledání.
LEAVEOPEN	Tento operátor je vhodné použít společně s operátorem popup, kdy vyhledávací stránka nebude po výběru položky v tabulce výsledků uzavřena.
POPUP	Vyhledávací stránka je zobrazena jako překrývající okno. Lze ovládat velikost a zjev překrývající stránky použitím atributu FEATURES.
PREDICATES	Použití tohoto operátoru dává klientovi možnost změnit typ porovnání. Mezi popiskem a polem pro vstupní data je zobrazen combobox s operátory porovnání
SHOWCOUNT	Zobrazí počet položek splňující vyhledávací kritéria (počet řádků v tabulce výsledků hledání).
SORTBOX	Zobrazí na vyhledávací stránce kolekci radiobotton, které dovolí klientovi určit, jak a podle čeho se mají výsledky hledání třídit.

Tab.5.13 Operátory volby

Obr. 5.17 Vyhledávací stránka

A jak vlastně vypadá JavaScript funkce, kterou vytvoří tento příklad?

```
<script language=JavaScript>
<!--
function form_search()
{
   var values = '';
   var url =
'_CSP.PageLookup.cls?CSPToken=uaJrALko4N/tTOf71wI7_7zow/oXWVbIgNo1ivXh9MxvBjcI
w94pwlYptlGzkq1b5Nz66TlQgV_VK3AVVEE9Wu/kz9wKczqOf/MKqTyMBqmVqoFxu44HH9FES0LCtX
5a63GMQsthnHe6dwwQZ5nJEbZUQB9rAqwFJg9U5joY4MBzsYvOrmZHy2BthcwJGkkFs7zOGH2JxosM
Ee4YYj7HEL6C2fVPbae7tn4s0wtnASYTbG2ecPuxGJjqEc/f5c19bIc6Yn5VCQAoyrTgHDom32OgpL
6KxTu8QdmXCESdX66nKXz6aM_f_gSF6N_hApnAMc5THFLIKpO76sCwJwdtgBPmqmPqVhcO4KjeegdX
7mFrt1Icivf4ZXIRzto9q9/XU3M/VM0nz9wwXz6TkeBGpNiMsPs1JB6LWatcmEbQe876u9MJjl75Cl
suBoOvKS4H4hmaI7g09Tvpo3zB9tVOFQ--&CSPCHD=4553430794271699534' + values;
   self.cspPopupWindow =
window.open(url,'cspPopup','width=400,height=400,status,scrollbars,resizable')
;
   self.cspPopupWindow.focus();
}
// -->
</script>
```

Obr. 5.18 JavaScript funkce vygenerované tagem CSP:SEARCH z příkladu 5.13

Již máme specifikovaný kompletní CSP:SERCH tag a tím i vyhledávací funkci. Nyní si ukážeme, jak je možné tuto funkci použít. Nejčastějším příkladem je vyvolání vyhledávací stránky z formuláře obsahují tlačítko realizující vyvolání této funkce.

Příklad 5.15: Formulář obsahující tlačítko vyvolávací vyhledávací funkci

```
<form name="form" cspbind="polozka" cspjs="All" onsubmit='return
form_validate();'>
…
<input type="button" name="btnSearch" value="Hledej"
onclick='form_search();'>
</form>
```

V následující tabulce jsou uvedeny všechny možné atributy tagu CSP:SEARCH, i ty které jsme v příkladu nepoužili.

Atribut	Popis	Hodnota
CAPTION	Nepovinný atribut, titulek vyhledávací stránky.	Řetězec
CLASSNAME	Povinný atribut, jméno třídy (ne tabulky), jejíž instance se mají vyhledávat.	Řetězec
FEATURES	Čárkou oddělený seznam parametrů a jejich hodnot funkce windows.open. Tato funkce ovlivňuje vzhled a velikost vyhledávacího okna.	Řetězec
FORM	Jméno formuláře spojeného s vyhledáváním. Používá se pro aktualizaci propojeného formuláře na stránce, která vyvolala vyhledávací stránku.	Řetězec
IDNAME	Nepoužívá se, jméno ID položky hledaného objektu.	Řetězec
MAXROWS	Specifikuje nejvyšší možný počet řádků ve vyhledávací tabulce. Implicitně je **100.**	Číslo
NAME	Jméno vyhledávací funkce. Implicitně je nastavena na "**search**".	Platný název funkce javascriptu
OBJID	Hodnota objektového identifikátoru používaného pro identifikaci objektu na úvodní stránce. Používá se pro aktualizaci propojeného formuláře na stránce, která vyvolala vyhledávací stránku.	Řetězec
OBJIDATTR	Jméno atributu objektového identifikátoru propojeného formuláře. Výchozí hodnota je "OBJID".	Řetězec
ONSELECT	Nepovinný atribut, jméno funkce javascriptu (na stránce vyvolávající	Název funkce javascriptu

	vyhledávací stránku), která je volána pro položku vybranou na vyhledávací stránce (např.:update)	
OPTIONS	Čárkou oddělený seznam možností vyhledávacího okna.	Řetězec tvořený některými z těchto hodnot: “popup“, “clearbtn“, “display“, “predicates“, “sortbox“ a “leaveopen“
ORDER	Čárkou oddělený seznam položek, podle nichž dojde k setřídění výsledků hledání.	Řetězec
PREDICATES	Čárkou oddělený seznam operátorů porovnání	Řetězec
SELECT	Čárkou oddělený seznam položek zobrazených v tabulce zobrazující výsledky hledání, pokud není specifikovaný atribut WHERE, jsou tyto položky použity jako výběrový seznam (podle čeho lze vyhledávat)	Řetězec
SHOWSQL	Zobrazit SQL používané pro vyhledávání (má využití při ladění programu)	“0“ nebo “1“
STARTVALUE	Čárkou oddělený seznam položek, jejichž počáteční hodnota by se měla převést z formuláře.	Řetězec
TARGET	Určuje, kde se zobrazí vyhledávací stránka. V novém okně, nebo při použití rámu, v které části okna.	Řetězec
WHERE	Čárkou oddělený seznam položek k vyhledávání	Řetězec

Tab.5.14 Atributy tagu CSP:SEARCH

5.4.5 Ostatní tagy

K ostatním tagům patří například CSP komentáře (CSP:COMMENT), tag pro import souboru (CSP:INCLUDE) aj.

5.4.5.1 CSP:COMMENT

Jedná se o zvláštní typ komentářů, které nejsou dále nikde zveřejňovány.

Klasické HTML komentáře jsou spolu se zbytkem obsahu CSP stránek poslány do klientského prohlížeče. Tag **CSP:COMMENT** specifikuje oproti tomu interní komentáře.

Syntaxe:

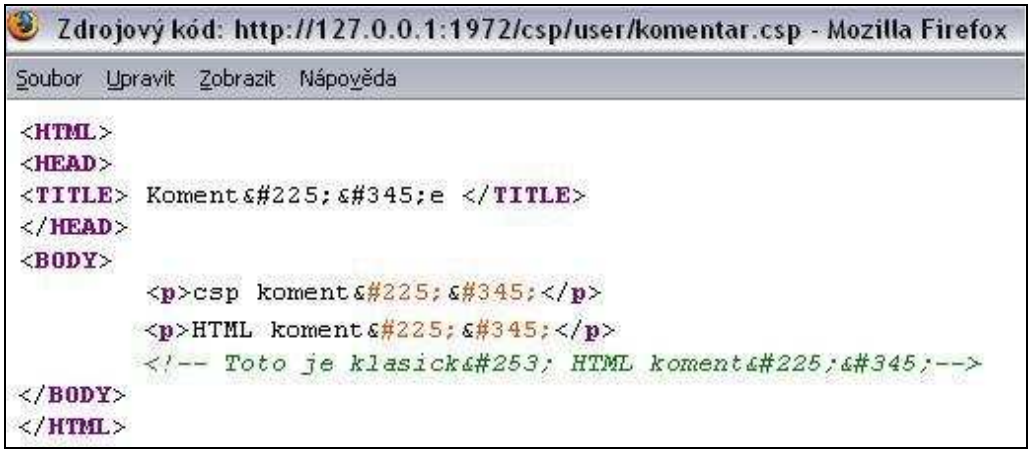
<CSP:COMMENT> *tělo komentáře* **</CSP:COMMENT>**

Příklad 5.16: Ukázka použití tagu CSP:COMMENT

```
<BODY>
  <p>CSP komentář</p>
  <CSP:COMMENT>Toto je interní CSP komentář</CSP:COMMENT>
  <p>HTML komentář</p>
  <!--Toto je klasický HTML komentář --!>
</BODY>
```

Na tomto příkladě není co vysvětlovat, pohled na webovou stránku a hlavně její zdrojový kód mluví za vše…

Obr. 5.19 Výřez webové stránky vygenerované z příkladu 5.16



Obr. 5.20 Zdrojový kód stránky z obr 5.19

Tento tag nemá žádné atributy.

5.4.5.2 CSP:CONTENT

Tag **CSP:CONTENT** specifikuje standardní obsahové charakteristické rysy vygenerované stránky, jako jsou obsahový typ a znaková sada.

Obsahový typ

Standardně obsahový typ pro CSP stránku je "text/html". Toto nastavení můžeme změnit použitím atributu **TYPE** tagu **CSP:CONTENT**.

Příklady obsahových typů: **text/xml**, **application/xml**, **text/html**, **text/css**, **image/jpeg**, **video/mpeg** aj.

Znaková sada

Také lze nastavit znakovou sadu, kterou bude CSP stránka používat. Toho dosáhneme použitím atributu **CHARSET**. K tomuto nastavení dochází při kompilaci CSP stránky.

Příklady znakových sad: **Windows-1250** (pro češtinu, používaná v českých Windows), **ISO-8859-2** (pro češtinu, např. v Linuxu), **UTF-8**, **ISO-8859-1** (západoevropská latinka), **ISO-8859-5** (cyrilice) aj.

Syntaxe:

<CSP:CONTENT [CHARSET="*znaková sada***"]**
[NOCHARSETCONVERT="*0/1***"] [TYPE="***obsahový typ***"]>**

Na každé CSP stránce by se tento tag měl objevit maximálně jednou.

Použití tohoto tagu je velmi důležité například při vytváření CSP stránky zajišťující export do XML.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<csp:content type="text/xml">

<script language="sql" name="dotaz">
    SELECT ID FROM SQLUSER.BP.kontakt
    ORDER BY prijmeni
</script>

<kontakty>
<csp:while condition="dotaz.Next()">
    <script language="Cache" runat="Server">
        set kontakt=##class(BP.kontakt).%OpenId(dotaz.Data("ID"))
        Do kontakt.XMLExport()
    </script>
</csp:while>
</kontakty>
```

Obr. 5.21 Použití tagu CSP:CONTENT

1. řádek je standardní hlavička XML dokumentu. Na dalším řádku je použití tagu **CSP:CONTENT**, který specifikuje obsah tohoto dokumentu. Na výstupu se nám má zobrazovat XML dokument, proto je zde také jako obsahový typ použito **text/xml**.

Na následujících řádcích je hlavička a tělo metody. Tato metoda se jmenuje *dotaz* a jako jazyk je zde použito *SQL*. Fakticky se jedná o jednoduchý SQL dotaz pro výběr všech **ID** z tabulky **kontakt**. Tyto ID jsou řazeny za sebou podle příjmení v abecedním pořadí.

Na další řádce je hlavička **while** cyklu, který se bude opakovat, dokud neprojdeme celou tabulku vygenerovanou metodou dotaz. To zajišťuje atribut **condition** a hlavně jeho hodnota **dotaz.Next()**. Podrobnější vysvětlení while cyklu viz kapitola 5.1.5.

Tělo while cyklu je tvořeno metodou, která fakticky provádí výpis jednotlivých xml elementů a jejich hodnot. Nejprve se do proměnné kontakt načte aktuální údaj a poté se pomocí funkce **XMLExport()** vypíše.

Následující tabulka shrnuje všechny atributy tagu CSP:CONTENT:

Atribut	Popis	Hodnota
CHARSET	Specifikuje výchozí znakovou sadu	Řetězec
NOCHARSETCONVERT	Zakazuje konverzi znakové sady	“0“ nebo “1“
TYPE	Specifikuje výchozí obsahový typ	Řetězec

Tab.5.15 Atributy tagu CSP:CONTENT

5.4.5.3 CSP:INCLUDE

Tag **CSP:INCLUDE** importuje další CSP stránku nebo soubor. Pokud nastane problém s importem, pak se vyvolá chybová stránka zobrazující nastalé chyby.

Syntaxe:

<**CSP:INCLUDE PAGE="***importovaná stránka***">**

Tag **CSP:INCLUDE** lze použít také k importu jednotlivých komponent, které chceme, aby výsledná stránka obsahovala.

Použití tagu **CSP:INCLUDE** je velmi výhodné pro zvětšení přehlednosti kódu, kdy jeden soubor můžeme rozdělit po jednotlivých částech. Například skripty můžeme dát zvlášť do souboru, nebo delší text, aj.

Příklad 5.17: Ukázka použití tagu CSP:INCLUDE

Soubor tab1.csp:

```
<table>
  <tr><td>…</td><td>…</td></tr>
  <tr><td>…</td><td>…</td></tr>
</table>
```

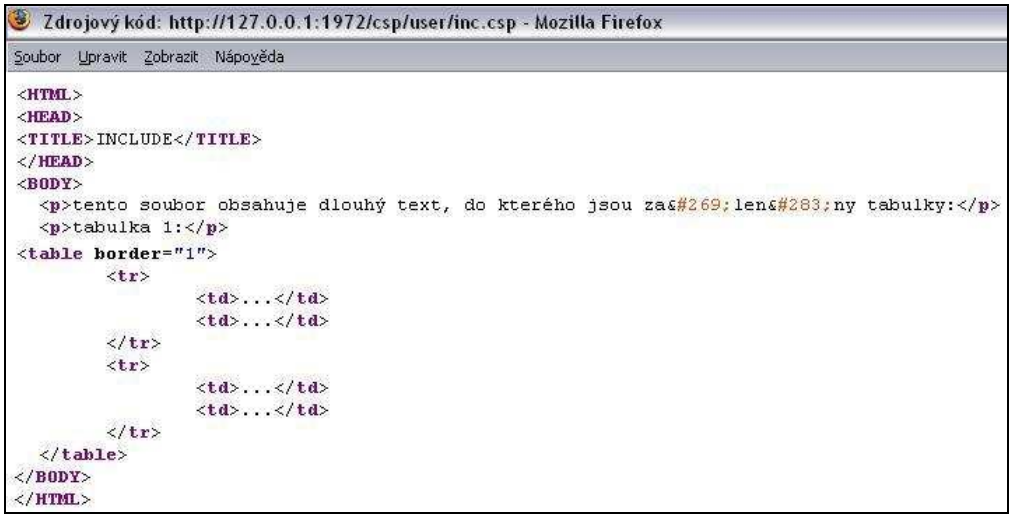
Importující soubor:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<p>tento soubor obsahuje dlouhý text, do kterého jsou začleněny tabulky</p>
<p>Tabulka 1:</p>
<CSP:INCLUDE PAGE="tab1.csp">
  …
</BODY>
</HTML>
```

Webová stránka zobrazí text i tabulky:

Obr. 5.22 Výřez webové stránky vygenerované z příkladu 5.17

Zda se kód skutečně importoval, dokazuje zdrojový kód této stránky:



Obr. 5.23 Zdrojový kód stránky z obr 5.22

Tag CSP:INCLUDE má jediný atribut, jak ukazuje i následující tabulka.

Atribut	Popis	Hodnota
PAGE	CSP stránka nebo soubor, který má být importován na toto místo	Řetězec

Tab.5.16 Atributy tagu CSP:INCLUDE

5.4.5.4 CSP:SECTION

Pomocí tagu **CSP:SECTION** lze určit, že kód uvnitř tohoto tagu se bude na klientském počítači zobrazovat na programátorem definovaném místě.

Příklad 5.18: Ukázka použití tagu CSP:SECTION

```
<HTML>
<HEAD>
</HEAD>
<BODY>
  <CSP:SECTION NAME="PREHTML">
    <!-- Komentář zobrazený před počátečním tagem HTML--!>
  </CSP:SECTION>
 <CSP:SECTION NAME="POSTHTML">
    <!-- Komentář zobrazený za koncovým tagem HTML--!>
  </CSP:SECTION>
  <CSP:SECTION NAME="BODY">
    <!-- Komentář zobrazený v těle HTML dokumentu --!>
  </CSP:SECTION>
  <CSP:SECTION NAME="HEAD">
    <!-- Komentář zobrazený v hlavičce--!>
  </CSP:SECTION>
</BODY>
</HTML>
```

Tělo souboru je tvořeno tagy **CSP:SECTION**. Tento tag je párový. Kód napsaný mezi jeho počátečním a koncovým tagem se klientovi zobrazí na pozici v dokumentu, která je specifikována atributem **NAME**. Tento atribut může nabývat čtyř hodnot. Jsou to **PREHTML**, **POSTHTML**, **BODY** a **HEAD**. Použitím první z jmenovaných hodnot říkáme, že chceme, aby se obsah tohoto tagu zobrazil ještě před počátečním tagem HTML.

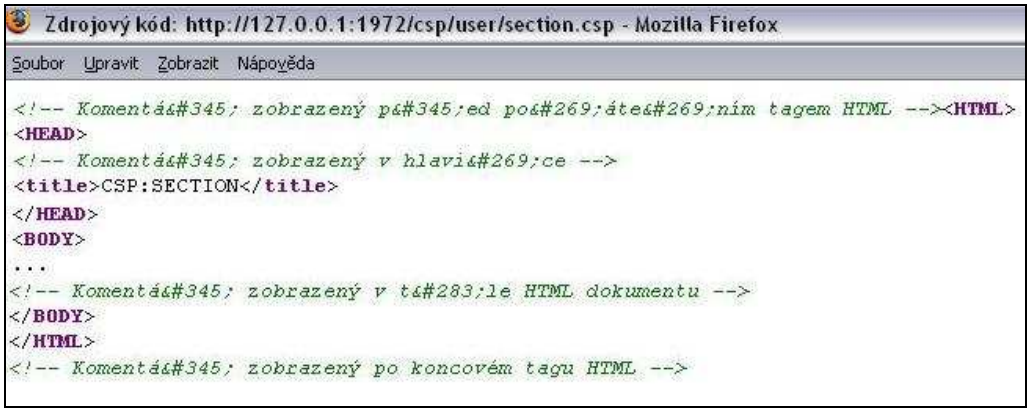
Další hodnotou je **POSTHTML**, která je opakem předchozí a obsah tohoto tagu bude zobrazen za koncovým tagem **HTML**.

Další možností je **BODY**. Použitím této hodnoty určujeme, že se kód v těle CSP:SECTION zobrazí v těle dokumentu.

Poslední hodnotou je **HEAD**. V takovém případě se kód zobrazí v hlavičce dokumentu.

Všechny tyto možnosti také zobrazuje příklad 5.18.

Zobrazovat zde webovou stránku je zbytečné, jediné co ukazuje výsledek je zdrojový kód zobrazený v prohlížeči:



Obr. 5.24 Zdojový kód vygenerovaný prohlížečem z příkladu 5.18

Všechny možné atributy tagu CSP:SECTION shrnuje následující tabulka:

Atribut	Popis	Hodnota
BLOCK	Relativní pozice v sekci (výchozí je 0, záporná hodnota= na začátku sekce, kladná hodnota = na konci sekce)	Číslo
NAME	Jméno sekce	"PREHTML", "HEAD", "BODY" nebo "POSTHTML"

Tab.5.17 Atributy tagu CSP:SECTION

5.4.6 Vlastní tagy - pravidla

Technologie CSP také poskytuje nástroje potřebné pro definování vlastních tagů. Stejně jako vestavěné tagy, i tyto se definují pomocí XML. K tomuto účelu je definována sada XML značek, jejichž názvy začínají písmeny **CSR**.

K vytvoření nového pravidla je nutné definovat několik bloků. Prvním z nich je definice názvu pravidla, další mohou být atributy, akce (která se provede při použití tohoto pravidla na CSP stránce), aj.

Následující příklad je definicí jednoduchého pravidla pro zobrazení údajů o poslední aktualizaci stránky:

Příklad 5.19: Definice vlastního pravidla

```
<CSR:RULE name="aktualizace" match="akt" empty>
 <CSR:ACTION>
   <p>
     Poslední aktualizace byla provedena
        ##($ZDate($Horolog,4))## v ##($ZTime($Horolog,2))##.
   </p>
 </CSR:ACTION>
</CSR:RULE>
```

Příklad 5.20: Použití pravidla z předchozího příkladu

```
<body>
 <akt>
</body>
```

Obr. 5.25 Výřez webové stránky, kde je použio vlastní pravidlo

Zda se kód skutečně importoval, dokazuje zdrojový kód této stránky:

Obr. 5.26 Zdrojový kód stránky z obr 5.25

Následující tabulka zobrazuje přehled základních XML značek pro definici vlastních tagů (tyto značky jsou postačující pro definici jednoduchých pravidel, ostatní viz dokumentace):

XML značka	Popis
CSR:ACTION	Párový tag, tělo tvoří popis akcí, které se mají při použití pravidla provést
CSR:ATTRIBUTE	Nepárový, definice atributů
CSR:RULE	Hlavní párový tag, ohraničuje celé pravidlo, definuje název pravidla

Tab.5.18 Základní XML značky pro definici vlastních pravidel

5.5 Metody (skripty)

Součástí CSP stránek jsou také skripty. Ty slouží k dynamickému generování obsahu.

Syntaxe:

<script language="*jazyk***" [***další atributy***]>**

Tělo skriptu

</script>

Skripty můžeme rozdělit do několika skupin podle jejich jazyka (**LANGUAGE**). To je také hlavní atribut každého skriptu, ten určuje jejich další atributy. Tímto jazykem může být například **CACHE**, **SQL**, **JAVASCRIPT, BASIC, VBScript** aj.

5.5.1 Caché skript

Nejprve se seznámíme se skripty, jejichž jazyk je Caché. V takovém případě je kód těla skriptu napsán jazykem Caché ObjectScript. V kapitole 5.4 jsme se již s takovými skripty setkali.

Příklad 5.21: Použití caché skriptu vyhodnoceného kompilátorem (přepis př. 5.19)

```
<script language="cache" runat="compiler">
  $ZDate($Horolog,4)
</script>
```

Příklad 5.22: Použití caché skriptu vyhodnoceného serverem

```
<script language="cache" runat="server">
    $ZDate($Horolog,4)
</script>
```

Tyto dva příklady jsou téměř totožné, tělo tvoří stejný kód. Jediný rozdíl je v hodnotě atributu **RUNAT**, který určuje dobu jejich vyhodnocení.

První skript se má vyhodnotit již při kompilaci. Tělo skriptu je tvořeno pouze funkcí, která vrací aktuální datum. Tento skript nám zobrazí datum kompilace, neboli datum poslední aktualizace stánky, na které bude tento skript volán.

Oproti tomu druhý skript je vyhodnocován serverem. To znamená, že se zobrazí aktuální datum, kdy jsme si tuto stránku otevřeli.

V těle skriptu můžeme použít příkaz **WRITE**. Tento příkaz zajistí, že se kód za ním následující vypíše na webové stránce. V takovém případě musí být skript napsán na pozici na stránce, kde chceme, aby se použil.

Příklad 5.23: Použití caché skriptu obsahující příkaz WRITE

```
<script language="cache" runat="server">
      Write $ZDate($Horolog,4)
</script>
```

Následující tabulka shrnuje atributy Caché skriptu:

Atribut	Popis	Hodnota
METHOD	Jméno metody, která je definovaná v těle skriptu	Řetězec
RUNAT	Určení místa vyhodnocení skriptu	"server" nebo "compiler"

Tab.5.19 Atributy Caché skriptu

Pokud v hlavičce skriptu použijeme atribut **METHOD**, znamená to, že tento skript definuje metodu CSP třídy. Takto vytvořená metoda se na CSP stránce volá pomocí tečkové notace. To znamená, že samotnému názvu metody předcházejí dvě tečky. Volání metody je samozřejmě výraz Caché ObjectScriptu a musí být uzavřen v **#(** a **)#.**

5.5.2 SQL skript

Dalším jazykem skriptů na CSP stránkách je SQL. SQL skripty realizují dynamické databázové dotazy.

Také s tímto typem skriptů jsme se již setkali v kapitole 5.4:

Příklad 5.24: Skript z příkladu 5.9 (použití tagu CSP:WHILE)

```
<script language="SQL" cursor="dotaz">
    Select prijmeni,jmeno From SQLUser.BP.kontakt
</script>
```

Příklad 5.25: Skript z příkladu 5.10 (použití tagu CSP:WHILE)

```
<script language="SQL" name="os">
    Select prijmeni,jmeno From SQLUser.BP.kontakt
</script>
```

Oby tyto skripty realizují klasický SQL dotaz. Přístup k výsledkům tohoto dotazu je možný prostřednictvím ukazatele nebo prostřednictvím názvu skriptu.

V příkladě 5.24 je použit ukazatel. To poznáme podle toho, že hlavička skriptu obsahuje atribut CURSOR, který specifikuje jméno ukazatele. Jak se s tímto ukazatelem pracuje, demonstruje příklad 5.9, kde je také tento skript použit.

Oproti tomu druhý zde zmíněný příklad demonstruje přístup přes jméno skriptu. Zde hlavička skriptu obsahuje atribut NAME, který určuje jeho jméno. Práci s tímto skriptem a přístup k výsledkům SQL dotazu ukazuje příklad 5.10.

Ve výše uvedených příkladech nejsou použity všechny možné atributy SQL skriptu, proto zde máme následující tabulku, která všechny atributy shrnuje:

Atribut	Popis	Hodnota
CURSOR	Jméno SQL ukazovatele	Řetězec
MODE	Mód běhu dotazu	"LOGICAL", "ODBC", "DISPLAY", nebo "SYSTEM"
NAME	Jméno lokální proměnné odkazující na objekt %ResultSet (pro dynamické SQL)	Řetězec
Pn	Hodnoty parametrů	Řetězec

Tab.5.20 Atributy SQL skriptu

5.5.3 JavaScript

Posledním zde zmíněným jazykem skriptů na CSP stránkách je JavaScript. S tímto skriptem jsme se také již setkali a to v kapitole věnované vyhledávání a formulářům. Zde byl jednak přímo definován skript obsahující funkci update a krom toho zde bylo také řečeno, že tag CSP:SEARCH vytváří JavaScript.

Atribut	Popis	Hodnota
CHARSET	Specifikuje znakovou sadu (kódování externího skriptu)	Řetězec
DEFER	Odložení provedení skriptu	Řetězec
SRC	Cesta k externímu skriptu	Řetězec
TYPE	Mime typ obsahu skriptu (např. text/javascript)	Řetězec

Tab.5.21 Atributy JavaScript skriptu

5.6 Výrazy Caché ObjectScriptu

Součástí téměř každé CSP stránky jsou výrazy v Caché ObjectScriptu. Tyto výrazy mohou být vyhodnoceny serverem nebo již při kompilaci (při vytvoření Caché tříd). Jedná se o substituci hodnot, kdy tyto výrazy jsou nahrazovány jejich aktuálními hodnotami.

Syntaxe serverových výrazů:

#(*výraz_Caché_ObjectScriptu***)#**

Syntaxe výrazů vyhodnocených při kompilaci:

##(*výraz_Caché_ObjectScriptu***)##**

Prostřednictvím výrazů v Caché ObjectScriptu můžeme přistupovat k proměnným, objektům, pracovat s funkcemi…

S těmito výrazy se můžeme setkat téměř kdekoli na CSP stránkách, mohou být součástí URL adresy, mohou to být parametry HTML nebo CSP tagů atd.

Nyní si připomeneme některé serverové výrazy, které jsme použili v příkladech v kapitole 5.4 CSP tagy.

Příklad 5.26: Příklady Caché ObjectScript výrazů vyhodnocovaných serverem

```
#($ZDate($Horolog,4))#          z příkladu 5.2 CSP:IF
#(i**1)#                        z příkladu 5.3 CSP:LOOP
#(os.Data(“prijmeni”))#         z příkladu 5.7 CSP:WHILE (bez CURSOR)
#(dotaz.Get("prijmeni"))#       z příkladu 5.8 CSP:QUERY
```

To byly serverové výrazy. V kapitole 5.4 jsme se setkali také s výrazy vyhodnocovanými kompilátorem:

Příklad 5.27: Příklady Caché ObjectScript výrazů vyhodnocovaných kompilátorem

```
##($ZDate($Horolog,4)##          z příkladu 5.13 vlastní pravidla
##($ZTime($Horolog,2)##          z příkladu 5.13 vlastní pravidla
```

Pomocí jednoduchého výrazu lze také vytvořit počítadlo přístupů na webové stránce:

Příklad 5.28: Vytvoření počítadla přístupů

```
#($Increment(^pocet))#
```

V předchozích příkladech jsme pracovali s lokálními proměnnými nebo objekty. V tomto příkladě jsme vytvořili globální proměnnou **^pocet**,

která se při každém načtení stránky inkrementuje pomocí funkce **$Increment**. Tato proměnná je trvale uložena v paměti.

5.7 Alternativy klasickým stránkám (WAP, XML)

CSP technologie umožňuje nejen psát klasické webové stránky a aplikace, podporuje též ostatní standardy používané v internetové a mobilní komunikaci. Těmi hlavními jsou WAP a XML.

Rozdíl proti psaní klasických (D)HTML stránek v CSP a uvedenými spočívá pro vývojáře pouze v tom, že soubor tvoří pomocí WML značek (pro WAP) nebo libovolných (ale samozřejmě předem dohodnutých) XML značek.

Pro správnou interpretaci výsledného kódu musí ještě uvést příslušný MIME typ stránky, například pomocí tagu **<csp:content type="text/xml">** (zde pro XML výstup). [1]

[1] KUTÁČ, Daniel. *Caché Studio 5* [online]. InterSystems CZ, 2002 [cit. 2008-03-15]. Dostupný z WWW: <http://www.intersystems.cz/iarchive/printversion/cache_studio/cs5/cs5.html>.

6.0 Shrnutí a závěr

Cílem práce bylo vytvořit uživatelskou příručku, v teoretické části, a sadu výukových tutoriálů vhodných pro samostudium, v části praktické.

Teoretická část je rozdělena do čtyř kapitol. První z nich (druhá v práci) je věnována seznámení s Caché a technologií Caché Server Pages. Čtenář se zde dozví, co přináší Caché, jaké jsou jeho přednosti a možnosti. Také je zde základní seznámení s technologií CSP. Je zde také zmínka o firmě Intersystems, která je tvůrcem tohoto všeho.

Další kapitola čtenáře seznámí se základními programovacími jazyky, které je možné při vývoji aplikací pomocí technologie CSP využít. Většinu těchto programovacích jazyků již čtenář určitě zná, a proto také není nutné se zde těmto jazykům nějak více věnovat. Jediný, ne příliš známý, programovací jazyk je Caché ObjectScript, kterému se blíže věnuje následující kapitola a představuje čtenáři základní rysy tohoto skriptovacího jazyka. Tato kapitola se omezuje pouze na ty rysy, které jsou při vývoji aplikací pomocí technologie CSP využitelné.

Následuje poslední kapitola teoretické části, která je v této části stěžejní a věnuje se technologii CSP. Je zde popis základních prvků a komponentám jazyka CSP. Čtenář se seznámí s prvky, se kterými je možné se na CSP stránkách setkat. Důkladně je zde rozpracována část věnující se jednotlivým tagů, které jsou základem jazyka. Tato kapitola je také nejrozsáhlejší kvůli množství příkladů, které slouží k vysvětlení právě probírané problematiky.

To byla teoretická část. Tato část by měla sloužit jako uživatelská příručka a je vhodná pro každého čtenáře, který má již s programováním nějaké zkušenosti. Vytvoření této příručky bylo z důvodu nedostatku zdrojů vcelku obtížné. Na českém trhu je pouze jediná kniha (Caché Databáze postrelačního typu a tvorba aplikací), kde je problematice CSP věnováno pouze 30 stran (celá kniha má cca 400 stran).

Nyní se možná ptáte, kde vlastně tedy je praktická část? Odpověď je jednoduchá. Praktická část je tvořena sadou výukových tutoriálů a z důvodu lepší dostupnosti a čitelnosti jsou z této práce vyjmuty a jsou koncipovány e-learningový materiál (webové stránky). Praktická část- výukové tutoriály je rozdělena do deseti tematických celků, které na sebe navzájem navazují. Nejprve je zde seznámení

s vývojovým prostředím a práce v něm. Následující dělení vychází z komponent, které jsou obvyklé na klasických webových stránkách. Setkáme se zde například se seznamy, tabulkami, formuláři aj.

Využitelnost těchto tutoriálů v praxi jsem si vyzkoušela na několika studentech, kterým jsem je poskytla a poté, co si je prošli, jsem se jich ptala, jaký z toho mají pocit, zda si myslí, že jsou schopni naprogramovat jednoduché stránky technologií CSP. Poté jsem jim dala zkušební příklad, který všichni bez větších obtíží zvládli. Z tohoto důvodu si také myslím, že jsem cíle práce splnila.

Seznam použitých zkratek

tzn.	To znamená
tzv.	Tak zvané
např.	Například
fce	Funkce
popř.	Popřípadě
č.	Číslo
obr.	Obrázek

Seznam použitých pojmů

C++	Objektově orientovaný programovací jazyk.
Caché	Postrelační databáze firmy Intersystems.
Caché Basic	Alternativa ke Caché ObjectScriptu založený na VBScriptu.
Caché ObjectScript	Objektově orientovaný programovací jazyk Caché.
CSP	[Caché server Pages]– technologie Caché, která dynamicky generuje HTML stránky pro vývoj web aplikací.
Java	Objektově orientovaný programovací jazyk.
XML	Jazyk, určený pro výměnu dat mezi aplikacemi a pro publikování dokumentů.
Rutina	Jedná se o modul obsahující kód psaný skriptovacím jazykem (Caché ObjectScript nebo Caché Basic) uložený pod jedinečným názvem.
Obsahový typ	Označuje typ média (např. audio, video, aplikace, text…). Skládá se z typu a podtypu. Obsahový typ informuje příjemce o obsahu zprávy.

Seznam použitých zdrojů

Internet:

[1] http://www.intersystems.cz/cache/

[2] http://www.fi.muni.cz/~kripac/PV136/holoubek.pdf

[3] http://platinum.intersystems.com/csp/docbook/DocBook.UI.Page.cls

[4] http://kryl.info/clanek/72-cache-prvni-dotyk

[5] http://www.cache.cz/iarchive/printversion/cache/technology/Caché_Tech_Guide_CZ.pdf

[6] http://www.intersystems.cz/education/cache_studio/index.htm

Literatura:

[7] **Caché Databáze postrelačního typu a tvorba aplikací**, Kirsten Wolfgang, Ihringer Michael, Kühn Mathias, Röhrig Bernhard, Computer Press, 2005, ISBN: 80-251-0491-5, CD-ROM

Příloha 1

Přílohy a jejich obsah:

CD-ROM:

- bakalářská práce ve formátu PDF
- příklady a obrázky použité v práci
- e-learningový tutoriál (webové stránky)