# Vysoká škola ekonomická v Praze

**Fakulta informatiky a statistiky**
**Katedra informačních technologií**



Student : **Borek Bernard**
Vedoucí bakalářské práce : **ing. Tomáš Brabec**
Recenzent bakalářské práce : **ing. Radek Holis**


TÉMA BAKALÁŘSKÉ PRÁCE

# Využití objektových technologií při vývoji webových aplikací: Srovnání PHP s ASP.NET

**ROK : 2004**

## Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne 20. srpna 2004

……

Borek Bernard

## Poděkování

Za mnoho podnětných připomínek bych rád poděkoval panu ing. Tomáši Brabcovi, vedoucímu práce.

# Abstrakt (česky)

Náplní práce je komplexní srovnání dvou rozšířených technologií pro vývoj webových aplikací, skriptovacího PHP 5 s objektovým ASP.NET 1.1. Cílem textu je podat dostatečné podkladové informace pro výběr adekvátní technologie pro daný projekt.

Text je členěn do třech hlavních oddílů. V prvním jsou uvedeny důvody pro výběr tématu, teoretická východiska práce, cíle textu a metody použité při jeho vypracovávání.

Druhý oddíl je nosným pilířem práce. Nejprve jsou stručně představeny obě technologie a zbytek oddílu je pak věnován konkrétním oblastem srovnání. Nejprve jsou probrány základní záležitosti jako např. podporované jazyky nebo způsob zápisu serverového kódu, hodně místa je poté věnováno syntaktickým záležitostem, ve kterých je PHP srovnáváno s jazykem C#. Probrány jsou jak základy syntaxe, tak jazykové konstrukty podporující objektově orientované programování. Následující kapitola se potom věnuje tomu, jak technologie řeší typické úlohy při vývoji webových aplikací, jako např. komunikaci s databázemi, překonávání nestavovosti HTTP apod. V následující kapitole jsou probrány další podstatné vlastnosti technologií, jako např. výkonnost nebo bezpečnost, a závěr oddílu je věnován dalším souvisejícím záležitostem – konkrétně vývojovým prostředím, dokumentaci a podpoře komunit.

Třetí oddíl sumarizuje zjištěné závěry, vyzdvihuje nejdůležitější učiněné poznatky a snaží se odpovědět na otázku, v jakých scénářích by mělo dostat přednost ASP.NET před PHP a naopak. Nechybí ani shrnutí oblastí, které se do této práce nevešly a které mohou sloužit jako námět pro další zkoumání.

# Summary (English)

This document contains a complex comparison of two famous web development technologies—scripting PHP 5 and object ASP.NET 1.1. The goal of the text is to provide sufficient information to be able to choose adequate technology for a given project.

The text is divided into three main sections. The first one contains reasons for choosing the subject, theoretical basis of the work and also goals of the text and methods used for its elaboration.

The second section is a storey post of the work. Both technologies are briefly introduced and the rest of the section is dedicated to concrete comparison areas. Firstly, the basics are examined, e.g. supported languages or the manner of writing server code. Much space is then dedicated to syntactical stuff in which PHP is compared to C#. Both syntactical basics and language constructs supporting object oriented programming are examined. Next chapter sticks at how are common web development tasks, such as communication with databases or stateless HTTP overcome, addressed in both technologies. Next chapter analyses another important technologies' features such as performance or security. At the end of the section, there are another related things examined. To be concrete, it's going on development environments, documentation and community support.

The third section summarizes conclusions, emphasises the most important findings and tries to answer the question, in which scenarios is ASP.NET preferable to PHP and vice versa. Also a summary of areas which couldn't be deeper analysed and which can serve as a suggestion of the following research can be found here.

# Obsah

## Oddíl první: ÚVOD


**1 Vymezení tématu práce ..... 1**
**2 Pojem webové aplikace ..... 1**
**3 Prostředky vývoje webových aplikací ..... 2**
3.1 CGI ..... 2
3.2 Skriptovací technologie ..... 3
3.3 Objektové technologie ..... 4
3.4 Upozornění ..... 4
**4 Náplň hlavní části ..... 4**
**5 Cíle práce ..... 5**


## Oddíl druhý: PHP versus ASP.NET


**6 Stručné představení PHP ..... 6**
6.1 Něco málo z historie ..... 6
6.2 Důvody obliby ..... 6
**7 Stručné představení ASP.NET ..... 7**
7.1 Co to vlastně ASP.NET je ..... 7
7.2 Verze ..... 8
**8 Jaké verze budou srovnávány ..... 8**
**9 Typologie webových projektů ..... 9**
**10 Srovnání ..... 9**
10.1 Základy ..... 9
10.1.1 Technologické pozadí ..... 9
10.1.2 Podporované jazyky ..... 9
10.1.3 Způsob zápisu serverového kódu ..... 10
10.1.3.1 PHP a serverové vsuvky ..... 10
10.1.3.2 ASP.NET a serverové ovládací prvky ..... 10
10.1.3.3 Shrnutí ..... 15
10.2 Syntaxe ..... 15
10.2.1 Jaké jazyky budou srovnávány ..... 15
10.2.2 Základní prvky syntaxe ..... 15
10.2.3 Podpora OOP ..... 17
10.2.3.1 Zapouzdření ..... 17
10.2.3.2 Dědičnost ..... 19
10.2.3.3 Skládání ..... 19
10.2.3.4 Polymorfismus ..... 20
10.2.3.5 Znovupoužitelnost ..... 22
10.2.3.6 Výjimky ..... 22
10.2.3.7 Organizace dostupných funkcí ..... 23
10.2.3.8 Shrnutí ..... 23
10.2.4 Shrnutí jazykového srovnání ..... 24
10.3 Řešení typických úloh ..... 24
10.3.1 Databáze a jiné datové zdroje ..... 24
10.3.2 Podpora XML ..... 26
10.3.3 Řízení stavů ..... 27

10.3.4 Webové služby ..... 28
10.3.4.1 Ujasnění ..... 28
10.3.4.2 Lehký úvod do webových služeb ..... 28
10.3.4.3 Webové služby v PHP ..... 28
10.3.4.4 Webové služby v ASP.NET ..... 29
10.3.4.5 Shrnutí ..... 30
10.4 Další vlastnosti technologií ..... 31
10.4.1 Konfigurace ..... 31
10.4.2 Výkonnost ..... 31
10.4.3 Podpora platforem ..... 32
10.4.4 Bezpečnost ..... 33
10.4.5 Rozšiřitelnost ..... 34
10.4.6 Cena ..... 34
10.4.7 Produktivita ..... 35
10.5 Jiné související záležitosti ..... 36
10.5.1 Vývojová prostředí ..... 36
10.5.2 Dokumentace, komunity ..... 37

## Oddíl třetí: ZÁVĚR

**11 Sumarizace výsledků ..... 39**
**12 Náměty na další práci ..... 40**
12.1 Měření výkonnosti ..... 40
12.2 Produktivita ..... 41
12.3 SEO ..... 41

**Zdroje ..... 42**
I. Hlavní zdroje ..... 42
II. Články a jiné „menší“ zdroje ..... 42
III. Odkazy ..... 43
**Terminologický slovník ..... 45**
**Příloha: Jiná srovnání PHP s ASP.NET ..... 49**

# Oddíl první
# ÚVOD

## 1 Vymezení tématu práce

Masivní rozšíření Internetu vyneslo do popředí zájmu webové aplikace a rovněž nastolilo otázku, jaké technologie pro jejich vývoj používat. Velmi dlouho byly hlavními rivaly PHP a ASP, ASP již ale není vyvíjeno a bylo nahrazeno novějším ASP.NET. Právě srovnání PHP (verze 5) s ASP.NET (verze 1.1) je tématem této práce.

Hlavním smyslem textu je podat informace, které by pro určitý projekt umožnily z těchto dvou technologií vybrat vhodnější vývojovou a běhovou platformu. Proč ale právě PHP a ASP.NET, když zde existují i jiné rozšířené a kvalitní platformy jako např. JSP nebo Python? Rozhodně ne proto, že bych zvolené technologie považoval za absolutně nejlepší, spíše se jedná o mé subjektivní rozhodnutí. Pokusím se stručně shrnout, co mě k výběru motivovalo.

Zaprvé to je značná popularita a rozšířenost jak PHP, tak ASP.NET. Mnoho projektů si tedy bude vybírat právě mezi těmito dvěma platformami.

Druhým motivem je jejich fundamentální odlišnost. PHP je typickým reprezentantem koncepčně starší skriptovací technologie, zatímco ASP.NET je od základů vybudováno jako moderní objektová platforma, která mění zažitý způsob vytváření webových aplikací. Je tedy zajímavé, jak se nový přístup srovná se starým, jaký je jeho přínos a jaké má nedostatky.

Pro ASP.NET s PHP jsem se také rozhodl proto, že se mi nepodařilo najít žádné seriózní srovnání, které by pro tyto dvě technologie už bylo vypracované. Přestože je otázka porovnání PHP s ASP.NET častým hostem v různých diskuzích a bývá námětem mnoha kratších článků, většinou se jedná o texty, které někdy postrádají objektivitu, jindy komplexnost a obecně si z nich většinou nelze udělat nějaký ucelenější názor.

Mou pozornost také upoutala zbrusu nová pátá verze PHP, novinka po dlouhých čtyřech letech (PHP 4 bylo vydáno v květnu roku 2000). Přináší řadu vylepšení, takže je potřeba staré závěry různých srovnání revidovat.

Když už bylo uvedeno, jaké hlavní motivy mě vedly k výběru tématu, zbývá ještě zodpovědět otázku, zda je vůbec relevantní srovnávat něco tak odlišného, jako je PHP a ASP.NET. To je často zpochybňováno, podle mého názoru však není odlišnost jako odlišnost. Je pravda, že „uvnitř“ jsou obě technologie úplně jiné, na druhou stranu se používají ke stejným věcem, např. „udělat e-obchod“. Společný účel je podle mého názoru to, co srovnání posvěcuje.

Více o náplni hlavní části a o jejích cílech bude uvedeno až po ujasnění některých základních východisek této práce.

## 2 Pojem webové aplikace

Termín *webová aplikace* je dnes používán velmi frekventovaně, přesto bude dobré ujasnit, v jakém významu bude v této práci používán.

Definic lze najít mnoho. V dokumentaci k Microsoft Windows 2000 Serveru je uvedeno, že „webová aplikace je softwarový program, který jako svůj základní komunikační protokol používá HTTP a uživateli

doručuje data v jazyce HTML.“ ([1]) Tato definice byla převzata i do jiných glosářů, jak lze snadno nahlédnout, pokud v Googlu necháme vyhledat „define:web application“. Wikipedie[1] praví: „Webová aplikace je aplikace doručená uživateli z webového serveru pomocí služby WWW.“ ([2]) Aplikace je přitom definována jako program nebo skupina programů, které mají koncovému uživateli poskytovat nějakou funkcionalitu ([3]). Firma Sun zase pro svou platformu J2EE používá následující definici ([4]): „Webová aplikace je aplikace napsaná pro Internet, ať už pomocí javovských technologií, jako jsou např. JavaServer Pages a servlety, nebo jako aplikace vytvořené nejavovskými technologiemi, jako jsou např. CGI nebo Perl.“

Jaký je smysl předchozího odstavce? Ukazuje, jak nejednotný je pohled na pojem webové aplikace a jak je obtížné vyhnout se sporným tvrzením. Tak např. první definice vyžaduje, aby výsledným formátem bylo HTML, což je zjevně příliš přísná podmínka, druhá definice nepřímo tvrdí, že HTML stránka je „aplikací“, zatímco definice Sunu z nějakého důvodu zdůrazňuje rozdělení všech technologií na javovské a nejavovské.

Dalo by se uvádět mnoho dalších definic, pro účely této práce se však bude hodit vymezení vlastní. To bude vycházet z jednoduché úvahy: slovo „webová“ je odvozeno od služby WWW. Tato služba je také definována mnoha různými způsoby, častým společným jmenovatelem je však koncové zařízení pro procházení jejího obsahu, kterým je *webový prohlížeč*. Pro účely této práce tedy jednou z podmínek bude zobrazitelnost výstupu ve webovém prohlížeči.[2]

Pracovní definice bude vypadat následovně: Webová aplikace je aplikace, která

- je postavena na modelu klient/server
- pro komunikaci mezi klientem a serverem používá protokol HTTP
- rozhraním pro koncového uživatele je webový prohlížeč (výjimečně wapový nebo jiný hypertextový prohlížeč)
- požadavky klienta zpracovává webový server (někdy nazývaný HTTP server)

Definice nijak neupravuje počet nebo strukturu dalších vrstev aplikace a v podstatě se stará jen o způsob komunikace s uživatelem. Webová aplikace tedy může být vícevrstvá.

Co se týče konstrukce, je takto postavená definice poměrně přísná. Výhodou je, že celkem jasně vymezuje, která aplikace bude v této práci považována za webovou a která nikoliv, může se však zdát, že je příliš omezující. Např. aplikace komunikující přes Internet pomocí protokolu HTTP ještě nemusí být webovou aplikací, ať už nesplňuje třetí nebo čtvrtou podmínku. Myslím ale, že připuštěná obecnost je do určité míry subjektivní záležitostí. Koneckonců, v praxi je docela jedno, jestli aplikaci nazveme webovou nebo třeba internetovou. Smyslem této kapitoly bylo pouze naznačit, jak bude webová aplikace v této práci vnímána.

# 3 Prostředky vývoje webových aplikací

Hlavní část této práce se věnuje srovnání dvou zcela konkrétních technologií – PHP a ASP.NET – věřím však, že neuškodí, bude-li problematika zasazena do širšího kontextu. Tato kapitolka by také měla ujasnit, jak se práce staví k dalšímu pojmu ze zadání, k pojmu „objektová technologie“.

Způsobů, jak se lze na prostředky vývoje webových aplikací dívat, je mnoho, což je dáno velkou rozmanitostí použitelných technologií. Pro účely této práce bude asi nejvhodnější chronologický, respektive „evoluční“ pohled.

## 3.1 CGI

Historie webových aplikací začíná v roce 1993, kdy bylo pro webový server NCSA HTTPd vyvinuto *CGI* (*Common Gateway Interface*, [5]). Je to rozhraní definující komunikaci mezi webovým serverem a externí

[1] Wikipedie je „živá“ encyklopedie, která v průběhu času svůj obsah může měnit (zkvalitňovat, upřesňovat). Je tedy možné, že zde uvedená definice ztratí časem na platnosti.

[2] Tato podmínka je dokonce v terminologickém slovníku Katedry informačních technologií podmínkou postačující, viz http://www.cssi.cz/all_terminologie.asp?kod=264&strana=5&volba=n.

aplikací, označovanou za CGI skript. Celé to funguje asi tak, že klient zažádá o určitou stránku, web server zjistí, že tento požadavek má zpracovávat CGI skript, a pomocí standardního rozhraní CGI předá případné vstupní parametry od klienta příslušnému programu. Tento program svůj výstup předává zpátky webovému serveru opět pomocí rozhraní CGI. Web server případně přidá chybějící HTTP hlavičky a vše pošle zpátky klientovi. Obslužný program může být jak kompilovaná aplikace napsaná třeba v jazyce C, tak interpretovaný skript napsaný např. v Perlu. Jedinou podmínkou, kterou musí CGI skript splňovat, je schopnost komunikovat s web serverem.

CGI ve své původní verzi mělo jednu zásadní nevýhodu – bylo málo výkonné. Při každém požadavku se program musel znovu zavádět do paměti, což bylo problémem při velkém množství nebo frekvenci požadavků (tento problém řeší o něco mladší FastCGI). Za nevýhodu lze rovněž považovat nepřehlednost kódu, což může být u velkých projektů dost vážný problém.

CGI ale mělo i jednu pokrokovou vlastnost. Tím, že se jednalo pouze o definici rozhraní, byla umožněna nezávislost na programovacím jazyce. Tento rys je svým způsobem ojedinělý a podobný „komfort" nabízí až o mnoho let později technologie .NET[3].

## 3.2 Skriptovací technologie

Hlavní nevýhodou CGI skriptů tedy zůstala především jejich obtížná tvorba – chybějící infrastruktura a málo přehledný zdrojový kód. Tyto nešvary se snaží řešit technologie, které využívají serverem vkládané vsuvky. Ať už se jedná o staré SSI (Server Side Includes) nebo dnes rozšířené PHP či ASP[4], princip je vždy stejný: do HTML jsou vloženy určité části kódu, které nejsou odesílány klientovi, ale nejdříve se nějakým způsobem zpracují na serveru.

Jednotlivé konkrétní technologie se samozřejmě mohou dost výrazně lišit, přesto však lze vypozorovat určité společné výhody, které těmto technologiím v dnešním dnech zajišťují značnou popularitu. Především se jedná o velmi jednoduchý způsob zápisu kódu. Vývojář se vůbec nemusí starat o nějaké předávání požadavku ve vstupním streamu a o generování streamu výstupního, naopak velmi jednoduchým a transparentním způsobem píše jak HTML kód, tak určitou funkčnost v místech, kde je potřeba výstup generovat dynamicky.

Další výhodou těchto technologií je také nenáročnost z administračního pohledu. Např. instalace PHP je otázkou přesunutí několika málo souborů a pár konfiguračních zásahů do web serveru (nehledě na to, že existuje mnoho balíků, které obsahují webový server – nejčastěji Apache – spolu s PHP tak, aby bylo vše předpřipravené k okamžitému používání).

Poslední podstatnou výhodu vidím v tom, že jsou dnes tyto technologie velmi rozšířené. Ovládají podstatnou část „trhu" a to je samo o sobě výhodou. Začínající vývojář velmi pravděpodobně slyšel o PHP nebo o ASP (třeba i podle přípon dokumentů, jejichž cestu zadává do adresního řádku prohlížeče), ale rovněž velmi pravděpodobně neměl tu čest zaslechnout něco o Pythonu nebo o JSP. Rozšířené technologie mají automaticky velkou podporu komunit – na internetu lze najít velké množství už hotových skriptů nebo cenných informací, které by jinak bylo těžké shánět.

Tyto technologie, které zde pracovně označím jako *skriptovací*, však mají i své nevýhody. Tou první je obtížný vývoj větších projektů (v některých případech i projektů menších). Tato nevýhoda má dvě hlavní složky. Tou první je chybějící pokročilá infrastruktura. (Námitkou může být, že např. infrastruktura PHP je velmi bohatá. Ano, srovnáme-li ji např. s CGI skripty, ne, srovnáme-li ji třeba s JSP nebo s ASP.NET. Sice se jedná o můj subjektivní názor, nicméně jsem přesvědčen, že sadu modulů PHP nelze porovnávat např. s .NET Frameworkem.) Druhou složkou je problém označovaný jako „spaghetti code", kdy je v jednom zdrojovém souboru promícháno HTML se skriptem. Právě uvedené neduhy nejsou ničím jiným než druhou stranou jedné mince: skriptovací technologie nemůžou být zároveň subtilní a zároveň využívat nějaký pokročilý framework, stejně jako nelze mít dokonalou kontrolu nad každým znakem odesílaného HTML a zároveň se vyvarovat serverových vsuvek nebo obtížně čitelného CGI skriptu.

[3] Ve skutečnosti .NET podporuje pouze omezenou sadu programovacích jazyků, na výběr jich však je přes 30 a používat lze jak populární jazyky typu Pascal nebo Perl, tak jazyky méně rozšířené, jako např. Fortran, Haskell nebo Ruby. Přehled jazyků viz http://www.jasonbock.net/dotnetlanguages.html.

[4] Nezaměňovat ASP s ASP.NET! Jedná se o dvě naprosto odlišné technologie.

Další nevýhodu spatřuji ve slabší podpoře objektově orientovaného přístupu k vývoji webových aplikací. Paušalizovat toto tvrzení je sice balancováním na tenkém ledě, protože míra „objektovosti“ je obtížně měřitelná a navíc ne všechny skriptovací technologie musí tímto neduhem trpět, ale ve srovnání s poslední skupinou technologií, ke které se dostaneme za chvíli, jsou skutečně skriptovací technologie méně objektové. „Objektovostí“ samozřejmě není myšlena pouze možnost vytvářet v daném jazyce třídy, spíše jde o celkovou koncepci technologie, tj. nakolik naplňuje paradigma OOP, jestli jsou standardní funkce přístupné pouze strukturovaně nebo jestli jsou součástí objektů apod. Objektovost však nekončí u syntaxe. Jde o celkový přístup k problémům – od návrhu aplikace až po její implementaci. Zjednodušeně by se dalo říct, že technologie je natolik objektová, nakolik podporuje objektové myšlení svých vývojářů.

## 3.3 Objektové technologie

Výše uvedené problémy skriptovacích technologií se snaží řešit poslední skupina technologií, jejíž hlavní představitelé jsou dnes JSP (JavaServer Pages) a ASP.NET (Active Server Pages .NET). Od skriptovacích technologií se liší poměrně zásadně hned v několika věcech. Jednak používají vlastní pokročilou infrastrukturu převzatou z technologie, která má širší záběr než jen podporu vývoje webových aplikací (u JSP to je Java, u ASP.NET je to .NET Framework). To má pro vývoj několik příjemných důsledků – k dispozici jsou totiž automaticky některé pokročilé nástroje např. pro šifrování, pro práci s XML, pro podporu transakcí apod. Také skriptovací technologie mohou obsahovat stejnou podporu, není to však zaručeno a některé věci je nutno řešit rozšířeními třetích stran. Vlastní infrastruktura znamená i další výhody, ale nebudeme předbíhat, na všechno bude čas v hlavní části této práce.

Jak již bylo řečeno, poslední skupina technologií byla vytvořena především proto, aby odstranila nedostatky skriptovacích technologií a naplno využila skutečnosti, že se může opřít o technologicky silný framework ve svém pozadí. Bylo by tedy obtížné pokoušet se v tuto chvíli nějak sumarizovat výhody a nevýhody této poslední skupiny, přesto je zde však ještě něco, co nelze opomenout: tyto technologie podporují vývoj webových aplikací objektovým způsobem. To je pro ně jednak charakteristické a jednak odlišující od skriptovacích technologií, proto tyto technologie nazveme technologiemi *objektovými*.

## 3.4 Upozornění

Ačkoliv je třetí kapitola zaměřena obecněji, práce si v žádném případě neklade za cíl porovnat skriptovací a objektové technologie jako takové (nehledě na to, že výše naznačené dělení je vyloženě pracovního charakteru). Práce zůstává zaměřena na dva zcela konkrétní zástupce, na PHP a na ASP.NET, učiněné závěry tedy nelze generalizovat.

# 4 Náplň hlavní části

Hlavní část práce bude pokusem o komplexní srovnání PHP s ASP.NET. Rozsah práce je přeci jen limitován, takže se nelze do podrobností věnovat všemu. Práce je zaměřena především na:

- syntaktické záležitosti s důrazem na objektové principy
- způsoby řešení typických problémů vývoje webových aplikací
- jiné podstatné vlastnosti technologií (výkonnost, bezpečnost apod.)

Práce byla dokončena dne 22. srpna 2004, všechny uvedené informace tedy platí k tomuto datu.

Co se použitých metod týče, většinou jsem postupoval tak, že jsem nastudoval, jak nějakou problémovou oblast řeší PHP a potom jsem totéž udělal z pohledu ASP.NET. Pokud se jednalo o věci, které bylo možno prakticky vyzkoušet (např. různé syntaktické vlastnosti), učinil jsem tak a výsledky jsem shrnul do textu, často s praktickou ukázkou. Pokud možnost praktického prověření nebyla (např. u srovnání bezpečnosti), snažil jsem se najít zdroje, které už nějaké srovnání v dané oblasti provedly nebo které by se mohly stát dobrým podkladovým materiálem.

Bohužel v práci nezbývá místo na podrobnější seznámení se s technologiemi samotnými. Musím tedy upozornit, že jejich základní znalost je tichým předpokladem, bez kterého možná bude čtenář v některých místech tápat, ačkoliv se budu snažit nejdůležitější věci stručně popsat. Naštěstí je základní zvládnutí PHP

otázkou několika desítek minut a se základy ASP.NET, potažmo .NET Frameworku, se lze seznámit během několika jednotek až desítek hodin.[5]

# 5 Cíle práce

Hlavním cílem a smyslem práce je podat čtenáři informace, které by mu pomohly orientovat se ve shodných a rozdílných rysech obou technologií. Tyto informace se mohou hodit např. při výběru adekvátní technologie pro určitý projekt.

Hypotéza je taková, že modernější objektová technologie ASP.NET je „lepší" než skriptovací technologie PHP postavená na starších základech. Formulace bez ošemetného slovíčka „lepší" by vypadala asi tak, že ASP.NET je pro vývoj webových aplikací vhodnější než PHP. Součástí cíle je také stanovit podmínky, za kterých se (možná) ukáže, že hypotéza platí.

[5] Pro PHP na internetu existuje nepřeberné množství seriálů, velmi dobře jsou však zpracovány také úvodní kapitoly manuálu. Ten je k dispozici na adrese http://www.php.net/manual/en/ (nebo …/cs/ pro češtinu). Pro seznámení se s .NET Frameworkem doporučuji knihu Dalibora Kačmáře „Programujeme .NET aplikace ve Visual Studiu .NET" ([.NET+VS]) nebo QuickStart Tutorial na webu asp.net (celé URL je http://asp.net/Tutorials/quickstart.aspx) – upozorňuji, že to zas tak „quick" není.

# Oddíl druhý

# PHP VERSUS ASP.NET

## 6 Stručné představení PHP

### 6.1 Něco málo z historie

PHP je rekurzivní zkratkou pro PHP: Hypertext Preprocessor (kde původní význam zkratky PHP je Personal Home Page). Jedná se o Open Source skriptovací technologii, jejíž historie je pěkně popsána na webové stránce http://cz.php.net/history/, ze které vybírám několik zajímavých okamžiků.

Základy PHP položil Rasmus Lerdorf v roce 1995, kdy pro své potřeby sepsal několik Perlovských skriptů, které nazval Personal Home Page Set. Jak rostla potřeba pro nové a nové funkce, PHP/FI (jak se technologie až do verze 2.0 jmenovala) se rozrůstalo, bylo přepsáno do jazyka C, ale ještě v roce 1997 bylo jen velkým projektem jednoho člověka. Podle odhadů bylo PHP v té době nainstalováno na 50 000 doménách, což představovalo asi jedno procento všech domén Internetu.

PHP velmi podobné tomu, které známe dnes, vzniklo ke konci roku 1997 jako kompletní přepsání PHP/FI 2.0 pány Andi Gutmansem a Zeevem Suraskim. Rasmus Lerdorf se k této iniciativě přidal, v půlce roku 1998 vzniklo PHP 3.0 a vývoj PHP/FI byl definitivně ukončen. Hlavními přínosy verze 3 byla dobrá infrastruktura pro mnoho databází a různých API, rozšiřitelnost a výkonnější a konzistentnější syntaxe s podporou OOP. Za své největší slávy bylo PHP 3.0 nainstalováno na stovkách tisíc webových serverů, což podle odhadů činilo 10% webových serverů Internetu.

PHP 3.0 bylo už poměrně dobrým nástrojem pro mnoho typových situací, nebylo ale navrženo tak, aby komplexní situace zvládalo efektivně. To bylo hlavní motivací pro PHP 4.0, jehož základem bylo nové jádro napsané Andi Gutmansem a Zeevem Suraskim. Toto jádro nese název, který byl inspirován jejich křestními jmény – Zend Engine. Kromě zvýšené výkonnosti přineslo PHP 4 několik dalších podstatných změn, jako např. HTTP sessions, podporu pro mnoho web serverů apod. PHP 4, současná nejrozšířenější verze, je nainstalována na několika milionech web serverů, odhadem na 20% doménách internetu.

Nejnovějším přírůstkem je PHP 5.0, které bylo vydáno 13. července 2004. Opět přineslo několik vylepšení, ale jedno je mimořádného významu: PHP 5 výrazně zlepšuje objektově orientovanou syntaxi. Přibyly výjimky, konstruktory, destruktory a mnoho dalších jazykových rozšíření. Mezi další podstatná rozšíření patří např. přepracovaná podpora XML, podpora COMu a technologie .NET, lepší podpora MySQL a další ([6]).

Ještě malá poznámka: PHP je jak technologií samotnou, tak jejím skriptovacím jazykem. Většinou ale budeme o PHP mluvit jako o technologii.

### 6.2 Důvody obliby

To, že je PHP velmi oblíbenou technologií pro vývoj webových aplikací, je beze sporu (viz např. http://www.php.net/usage.php). Jaké jsou důvody této obliby?

Tím prvním je asi velmi jednoduchá syntaxe. Při prvním setkání s jazykem hraje nezanedbatelnou roli to, jak jednoduše lze napsat aplikaci „Hello world“, a nutno uznat, že `<?php echo „Hello world“; ?>` vypadá velmi slibně.

Druhým důvodem bude pravděpodobně sama skutečnost, že je PHP tak známé. Má nálepku nejrozšířenější technologie pro vývoj webových aplikací, a i když v něm někdo třeba žádný projekt nevyvíjí, s vysokou pravděpodobností si o něm něco přečetl nebo napsal nějaký jednoduchý skriptík. Mnoho lidí určitě začínalo vyzkoušením a pak už u něj zůstalo.

Poslední vážný důvod vidím v tom, že PHP je nejen zadarmo a navíc Open Source, ale současně jsou dobře dostupné hostingy – např. jeden český hosting nabízí 50 MB s podporou PHP a MySQL zcela zdarma. Konkurenční technologie zdaleka podobné podmínky nenabízejí (respektive je nenabízejí hostingové firmy).

Pro někoho může být důležité také to, že na internetu lze sehnat mnoho „all-in-one“ balíčků, které z desktopu během pár sekund udělají kompletní vývojové prostředí s nainstalovaným Apachem a správně nakonfigurovaným PHP. Začátečník tak může své první „Ahoj světe“ vidět téměř okamžitě, což může být silnou pozitivní motivací.

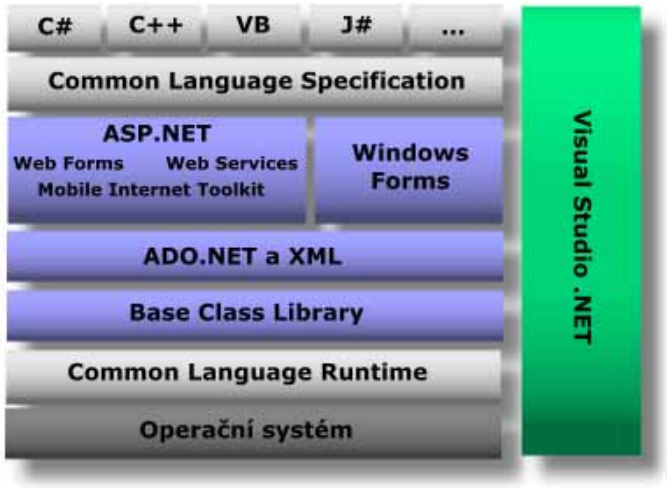
# 7 Stručné představení ASP.NET

## 7.1 Co to vlastně ASP.NET je

ASP.NET je nejnovější technologií firmy Microsoft určenou pro vývoj webových aplikací.

První verze ASP (Active Server Pages) byla vydána na konci roku 1996 a kromě podobného data vydání toho měla s PHP společného opravdu hodně. Koneckonců, jak bylo uvedeno v předchozích kapitolách, PHP i ASP spadají do stejné kategorie, zde pracovně nazvané jako skriptovací technologie.

Koncem devadesátých let minulého století začal Microsoft pracovat na zcela nové platformě, kterou pojmenoval *.NET*. Je to „sada softwarových technologií společnosti Microsoft pro propojování světa informací, lidí, systémů a zařízení“ ([7]; zní to trochu příliš marketingově, ale v podstatě to je pravda, alespoň co se záměru týče). Základem je platforma nazvaná *.NET Framework*, jejíž struktura je nejlépe vidět z obrázku.




**Obrázek 1 – Architektura .NET Frameworku**

Předmětem této kapitoly rozhodně není popis .NET Frameworku, z obrázku je však velmi pěkně vidět, co to vlastně ASP.NET je. Není to nějaká samostatná technologie pro tvorbu webových aplikací, je integrální součástí nové platformy a staví na stejných základech jako její doplněk – Windows Forms (část frameworku pro tvorbu tlustých klientů, ne nutně pro operační systém Windows). ASP.NET tedy není jen další verzí ASP, ve skutečnosti se jedná o zcela novou technologii.

Obrázek ukazuje tři základní stavební kameny ASP.NET: *Web Forms*, *Web Services* a *Mobile Internet Toolkit*. To všechno jsou velmi důležité věci, které dávají tušit sílu ASP.NET. Web Forms slouží k vytváření webových stránek pomocí komponent podobně, jako je tomu zvykem u „okénkových" aplikací. Důležitou součástí je rovněž koncept serverových ovládacích prků, kterým se práce věnuje níže.

To, že jsou do ASP.NET integrovány webové služby, je velkým plusem. Jak bylo řečeno, celý .NET je inspirován snahou usnadnit vývoj distribuovaných systémů, a webové služby jsou jasnou cestou, kudy se pravděpodobně integrace heterogenních systémů bude ubírat.

Mobile Internet Toolkit je sada nástrojů pro generování výstupu pro mobilní zařízení. Opět je to něco, co velmi podstatně usnadňuje vývoj. O způsobu generování výstupu však bude řeč dále, takže to nyní v tichosti opustíme.

## 7.2 Verze

ASP.NET nelze stáhnout a nainstalovat samostatně (vždy je součástí .NET Frameworku), proto jsou verze číslovány shodně s celým frameworkem.

Verze 1.0 a 1.1 jsou v současnosti jediné stabilní a na verzi 2.0 Microsoft v současnosti pracuje (nyní, v srpnu 2004, je ke stažení první beta verze, SDK zatím k dispozici samostatně není – pouze jako součást beta verzí vývojových nástrojů). Vlastnosti prvních verzí budou předmětem zkoumání o něco níže, zde bude uvedena jen ochutnávka novinek v druhé verzi.

Ta bude plně zpětně kompatibilní s verzí 1.1, přesto přinese několik velmi podstatných změn. Bude obsahovat podporu šablon stránek (tzv. Master Pages; bude tedy jednodušší udržet společný obsah nebo design na všech stránkách), výrazně zjednoduší data binding[6], který je už v současné verzi chloubou ASP.NET, pomocí nových ovládacích prvků[7]. Druhá verze dále přinese podporu skinovatelnosti stránek, kolem 50 nových ovládacích prvků, nové rozhraní pro administraci, podporu úplné prekompilace a distribuce bez zdrojového kódu, nový model separace HTML a skriptu a další. Na závěr jedna velmi zajímavá novinka: „odlehčený callback". V současnosti klient interaguje s webovým serverem pomocí plného callbacku, což je v podstatě odeslání všech dat na server a jejich opětovné přijetí, i když se změní třeba jen jedna hodnota. Client Callback Manager umožní odeslání pouze malého množství dat protokolem XMLHTTP (ten zvládá např. Internet Explorer) a jejich opětovné přijetí, aniž by muselo dojít k překreslení stránky. Modelovou situací je např. zadání PSČ a zjištění města.

ASP.NET 2.0 přinese mnoho dalších novinek. O některých se lze podrobněji dočíst na stránce http://msdn.microsoft.com/msdnmag/issues/04/06/ASPNET20Overview/default.aspx.

# 8 Jaké verze budou srovnávány

Jak je ze stručného představení jednotlivých technologií vidět, každá verze se od té minulé velmi podstatně liší. Je tedy důležité stanovit, které verze budou srovnávány.

Možná by bylo nejspravedlivější srovnávat ty verze, které byly vydány ve stejné době (alespoň zhruba, data vydání se nikdy nekryla). Potom bychom museli srovnávat PHP 4 s prvním ASP.NET. Minulost však nikdy není tak zajímavá jako přítomnost nebo budoucnost, proto se s PHP verze 4 rozloučíme a práce se zaměří na zbrusu nové PHP 5, které skoro jistě v blízké budoucnosti starší verzi vytlačí.

S ASP.NET to bude těžší, protože verze 1 je sice už trochu zastaralá, nicméně finální vydání verze druhé je zatím poměrně daleko (možná v prvním pololetí 2005, možná ještě později). Ačkoliv tedy z dlouhodobého pohledu asi bude větším soupeřem pro PHP 5 právě ASP.NET 2.0, v této práci bude srovnáváno s ASP.NET 1.1.

---

[6] *Data binding* je oficiální termín pro vázání dat z obecných datových zdrojů (databáze, XML, Excel, …) k ovládacím prvkům (viz následující poznámka pod čarou). Občas se překládá jako vázání dat, osobně mám více v oblibě anglický originál.

[7] Anglický termín „control" se do češtiny překládá jen obtížně, občas jako „ovládací prvek" (což je zavádějící) nebo přímočaře jako „kontrol" (což je zase nanejvýš nejasný pojem pro někoho, kdo s ním ještě není sžitý). „Control" v podstatě vyjadřuje určitý objekt nebo komponentu, např. tlačítko, komponentu připojení k databázi, časovač apod. V této práci bude používán překlad „ovládací prvek", který je upřednostňován např. na serveru Interval.cz.

# 9 Typologie webových projektů

Pro účely práce se může hodit stanovení základních typů webových projektů. Klíčovým rozlišovacím kriteriem je rozsah projektu, na čemž bude záviset mnoho hodnocení. Typy tedy budou následující:

1. Osobní projekty (někdy nazývané jako „hobby“ projekty). Ty se budou vyznačovat malým rozsahem, budou vytvářeny jednotlivcem nebo maximálně malou skupinkou dobrovolníků, typicky se bude jednat o nekomerční projekty nebo o projekty s malým obratem, aplikace bude hostovaná na nějakém veřejně dostupném hostingu (neplaceném nebo levném) a návštěvnost nebude nijak horentní. Příkladem jsou osobní stránky.
2. Projekty středního rozsahu. Budou vytvářeny menší firmou nebo komunitou, většinou se už bude jednat o nějakým způsobem komerční projekty (webová aplikace si na sebe musí vydělat ať už přímo – zpoplatněny jsou přímo služby nabízené na webu – nebo nepřímo, např. zvýšením prodeje jistého zboží). Aplikace bude buďto hostována jako v případě osobních projektů (i když už pravděpodobně ve výše tarifikovaném programu) nebo bude využito vlastního serveru. U těchto projektů také začne být kladen důraz na bezpečnost. Návštěvnost bude celkem slušná a příkladem mohou být např. internetové obchody.
3. „Velké“ projekty. Většinou se bude jednat o enterprise aplikace vytvářené velkým týmem. Kromě bezpečnosti bude požadován výkon aplikace a schopnost integrace heterogenních prvků systému. Příkladem mohou být velké intranety nebo různé portály.

# 10 Srovnání

## 10.1 Základy

### 10.1.1 Technologické pozadí

Srovnání obou technologií začneme jejich technologickým pozadím, tedy tím, o co se mohou opřít. Vše podstatné už vlastně bylo napsáno v představení technologií, proto jen krátké shrnutí: PHP je subtilní samostatná technologie, která pro vývojáře poskytuje poměrně dobrou infrastrukturu, sama se ale o nic většího neopírá. Naproti tomu je ASP.NET součástí rozsáhlého .NET Frameworku a samostatně ani existovat nemůže.

Oba přístupy mají něco do sebe. Speciálně vytvořené PHP je přímočarou platformou, která bez okolků poskytuje účelové nástroje pro vývoj webových aplikací. Také proto bývá PHP označováno jako „jednoduché“.

ASP.NET je naopak součástí víceúčelové platformy. Obrázek 1 na straně 7 ukazuje, že ASP.NET využívá úplně stejné knihovny funkcí (Base Class Library) jako Windows Forms, společná je také vrstva pro přístup k datům (v obrázku označena jako „ADO.NET a XML“, podrobnosti v kapitole 10.3.1 na straně 24). Tato integrace na jednu stranu zaručuje konzistenci s ostatními oblastmi vývoje, na druhou stranu je ASP.NET tak trochu spoutáno pravidly, které platí pro celý framework. Rozsáhlost .NET Frameworku je také nepříjemnou vlastností pro začátečníky – ASP.NET působí při prvním setkání složitým dojmem (v kontrastu s jednoduchostí PHP).

Ačkoliv tedy každý přístup má svá pro a proti, výhodou PHP je, že se ho lze kvůli jeho jednoduchosti rychle naučit.

### 10.1.2 Podporované jazyky

PHP není jen platformou pro vývoj webových aplikací, ale i skriptovacím jazykem této platformy, a to jazykem jediným. Existuje sice možnost integrace s Javou ([8]) nebo modul pro Python ([9]), nicméně nelze říct, že by PHP obecně podporovalo více jazyků.

Naproti tomu je celý .NET Framework jazykově neutrální – lze používat libovolný jazyk, který vyhovuje určitým pravidlům (Common Language Specification). ASP.NET 1.1 obsahuje přímou podporu pro Visu-

al Basic .NET, C# a JScript.NET, jinak je možno využívat z nabídky více než 25 jazyků ([10]). Dokonce je teoreticky možné jako jazyk ASP.NET používat samotné PHP, jistý pokus lze sledovat na adrese http://php-sharp.sourceforge.net/index.php.

Jazyková neutralita je nádherná vlastnost, která má mnoho příjemných důsledků. První je samozřejmě možnost volby. Vývojář ve Visual Basicu se nemusí učit nějaký nový jazyk, který je už od pohledu zcela odlišný, vývojář v Javě má okamžitě k dispozici J#, což je jazyk se shodnou syntaxí, vývojář v C++ může svůj oblíbený jazyk okamžitě začít používat atd. atd.

Tím, že se zdrojový kód kompiluje do MSIL (Microsoft Intermediate Language, podrobnosti v kapitole 10.4.2 na straně 31), je ale jazyková neutralita dotažena k dokonalosti. Není nejmenší problém napsat některé třídy třeba v COBOLu, jiné v Pascalu a přitom využívat funkčnosti tříd napsaných v C#.

Co se týče podpory jazyků, je ASP.NET nepochybně lepší.

### 10.1.3 Způsob zápisu serverového kódu

#### 10.1.3.1 PHP a serverové vsuvky

PHP se zapisuje pomocí klasických serverových vsuvek, což jsou určité části zdrojového kódu oddělené určitými značkami, které nebudou odeslány klientovi, ale nejdříve budou zpracovány na straně serveru. Typickým příkladem je např. následující fragment:

```
<p>Statický text</p>
<?php echo "<p>Dynamicky generovaný text</p>\n"; ?>
<p>Tady pokračuje HTML kód ...</p>
```

Výstupem je následující HTML kód:

```
<p>Statický text</p>
<p>Dynamicky generovaný text</p>
<p>Tady pokračuje HTML kód ...</p>
```

Jak vidno, oblast uzavřená v <?php a ?> byla transformována v čisté HTML. PHP umožňuje několik alternativních zápisů těchto vsuvek – nejkratší je <? a ?>, zápis použitý v příkladu je však v jedné věci elegantní. Vezměme dokument podle standardu XHTML:

```
<?xml version="1.0" encoding="windows-1250" ?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"
>

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>Demo</title>
</head>

<body>
  <p>Statický text</p>
  <?php echo "<p>Dynamicky generovaný text</p>\n"; ?>
  <p>Tady pokračuje HTML kód ...</p>
</body>
</html>
```

Bude to dokonale validní XML dokument, protože syntaxe <?xxx je v XML vyhrazena pro instrukce zpracování (processing instructions) a PHP je v tomto případě oním obslužným programem. Ostatní syntaxe nemají podle specifikace XML žádný význam ([11]).

#### 10.1.3.2 ASP.NET a serverové ovládací prvky

ASP.NET nabízí hned několik způsobů zápisu. Ten první vychází ještě ze starého ASP a velmi se podobá zápisu v PHP (až na to, že chybí elegantní souvislost s XML). Příklad:

```
<body>
  <% Response.Write(now()) %>
</body>
```

Kromě tohoto zápisu, který zůstal zachován především kvůli zpětné kompatibilitě s ASP 3.0, nabízí ASP.NET zcela nový přístup, tzv. *serverové ovládací prvky* (*server controls*). To je natolik důležitý koncept, že se mu budeme chvíli věnovat.

Jak bylo vidět na obrázku 1, důležitou součástí ASP.NET jsou tzv. Web Forms. Snahou Microsoftu bylo co nejvíce usnadnit vývoj webových aplikací. Jeff Prosise píše ([Prog.NET]): „ … Výzva číslo jedna, se kterou se dnes potýkají vývojáři? ‚Napiš software tak, aby běžel na webu'. Bohužel, webové programování není jednoduché. Napsat aplikace jako Microsoft Word nebo Microsoft Excel je dnes dobře zvládnutá oblast, napsat aplikace jako eBay nebo Amazon.com není. Ba co víc, současný webový vývoj je prováděn s nástroji a technologiemi první generace, které mají víc společného s Dartmouth BASICem šedesátých let než s moderními platformami a vývojovými prostředími, na které jsou vývojáři zvyklí. Odpověď Microsoftu na tento stav webového programování dneška je programovací model druhé generace nazvaný Web Forms." To jsou poměrně silná slova, pojďme se tedy podívat, co Web Forms nabízejí.

Web Forms používají před chvílí zmíněné serverové ovládací prvky (server controls). To jsou jakési entity, ze kterých se ASP.NET webová stránka skládá. Mohou to být jak vizuální komponenty (např. text, obrázky, tabulky, ale také komplexní záležitosti jako třeba kalendář) tak nevizuální komponenty (typickým příkladem je připojení k databázi nebo objekt DataSet, se kterým se setkáme později). Zapisují se jako *XML tagy*, které umějí *automaticky renderovat svůj HTML obraz* před odesláním klientovi a které dokáží *vyvolávat události na straně serveru*. To jsou ve zkratce nejdůležitější vlastnosti, které už na první pohled vypadají slibně a které ani z dálky nepřipomínají přístup ala PHP. V zásadě se serverové ovládací prvky dělí do několika skupin:

- HTML Controls, což jsou v podstatě HTML tagy s přidaným atributem `runat="server"`
- Web Controls, což jsou předpřipravené prvky, z nichž některé mají svůj obraz v HTML (viz např. `<asp:Label>` a `<span>`) a některé jsou komplexnějšího charakteru (např. `<asp:Calendar>`)
- User Controls a Custom Controls, což jsou uživatelsky vytvářené ovládací prvky. Rozdíly mezi nimi lze vyčíst např. z dokumentace ([12])

Občas jsou uváděny ještě další skupiny, jako např. Validation Controls, ale to už jsou detaily.

**Renderování výstupu (plusy a mínusy)**

První významnou vlastností tedy je, že serverové ovládací prvky generují HTML kód samy. Jak tomu rozumět? Vývojář nepíše výstup HTML přímo, jako je tomu např. v ASP nebo v PHP, pouze pomocí XML tagu určuje, že by se na daném místě měl vyskytnout určitý výstup. Jako příklad poslouží jednoduchý výpis textu.

```
...
<body>
  <form runat="server">
    <asp:Label id="Label1"
               runat="server"
               forecolor="Blue">Ahoj</asp:Label>
  </form>
</body>
...
```

Tag `<form runat="server">` vysvětluje název Web Forms – skutečně se jedná o HTML formuláře, které mají nový atribut `runat="server"`, který ASP.NET říká, že tento element má být zpracován na straně serveru (malá poznámka: neexistuje nic jako `runat="client"`).

Uvnitř formuláře není žádný serverový kód, pouze XML element `<asp:Label>` s několika atributy. Tím prvním je atribut `id`, který element jednoznačně identifikuje. Má velmi podstatný význam, jak bude vidno o kousek níže. Samozřejmě nesmí chybět `runat="server"` a potom už následují atributy specifické pro serverový ovládací prvek Label (příklad nastavuje barvu textu).

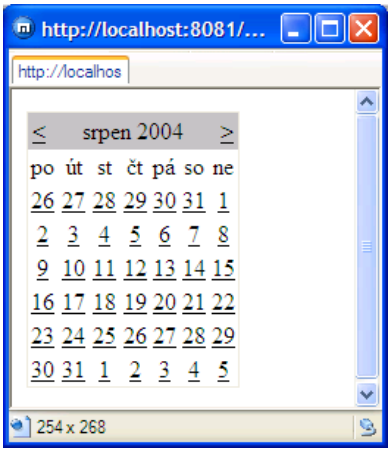
Udělal jsem malý pokus, aby bylo vidět, jak ASP.NET pracuje. Na pomoc jsem si vzal prohlížeč Mozilla Firefox (0.9.2 EN) s rozšířením PrefBar (2.3.0 RC3), který umí uživatelsky nastavovat různé user agenty (user agent je řetězec odesílaný v HTTP požadavku, který identifikuje prohlížeč). O stránku jsem simulovaně zažádal nejdříve jako Internet Explorer 6 a potom jako Netscape Navigator 4.7. Zde je výsledek, na který se serverový ovládací prvek `<asp:Label>` transformoval:

| | **Internet Explorer 6** | **Netscape Navigator 4.7** |
|---|---|---|
| Výstup | Ahoj | Ahoj |
| Zdrojový kód | `<span id="Label1" style="color:Blue;">Ahoj</span>` | `<span id="Label1"><font color="Blue">Ahoj</font></span>` |

Jak vidno, uživateli Internet Exploreru 6 se zobrazil stejný výsledek jako uživateli Netscape Navigatoru 4.7. Zdrojový kód je však jiný. Jak je to možné? ASP.NET „ví“, že Internet Explorer 6 je prohlížeč, který má poměrně slušnou podporu kaskádových stylů a že může HTML výstup pro tento prohlížeč vyrenderovat za použití inline stylu. Zároveň ví, že Netscape řady 4 je starý prohlížeč s mizernou podporou CSS a proto radši zvolí výstup, který se v Netscape zobrazí správně, ač je nucen použít zavrhnutý element `<font>`.

Tento přístup má podle mého názoru skutečně obrovský potenciál. Proč se totiž omezovat pouze na různé verze prohlížečů, když je zdrojový kód ASP.NET napsán zcela obecně? Pokud o stránku požádá např. wapový prohlížeč mobilního telefonu, není nejmenší důvod, proč by ASP.NET nemohlo vygenerovat WAP verzi ovládacího prvku Label.

Důležité také je, že serverové ovládací prvky mohou být skutečně komplexními komponentami. Příkladem je prvek `<asp:Calendar>`, který do stránky vloží plně funkční kalendář, který umožňuje kliknutím vybírat dny nebo přecházet mezi měsíci. Jednoduchý kód `<asp:Calendar id="Calendar1" runat="server" />` vygeneruje následující stránku:




**Obrázek 2 – Zobrazení prvku Calendar**

Nejedná se o žádný applet nebo o ActiveX, je to prosté HTML v kombinaci s JavaScriptem. Vytvoření podobné stránky v PHP by jednak bylo velmi pracné (HTML kódu a JavaScriptu je zde opravdu hodně), jednak je prvek Calendar plnohodnotnou komponentou – velmi jednoduše mu tedy lze nastavovat různé vlastnosti, např. kalendář, který zobrazuje dny od neděle místo od pondělí lze vygenerovat přidáním atributu `FirstDayOfWeek="Sunday"`.

Tento princip fungování však může vyvolávat také určité rozpaky. Vývojáři nejsou obvykle rádi, když nad něčím ztratí kontrolu (viz např. velký rozruch kolem principu Garbage Collection, který znemožňuje kontrolu nad přesným časem zániku objektu v paměti). V PHP mám naprosto plnou kontrolu nad každým znakem, který je zasílán jako odpověď. Tím pádem mám také plnou kontrolu nad formátováním zdrojového kódu, což může být důležité pro přehlednost v případě, že někdo bude zdrojový kód studovat.

Navíc, současné ASP.NET je pro webové puristy zapovězenou záležitostí, protože negeneruje validní XHTML stránky.

Co na to ASP.NET? K dispozici jsou tři možnosti. Tou první je ASP 3.0-kompatibilní zápis pomocí `Response.Write()`. To je prakticky ekvivalentní zápisu v PHP, přišli jsme však o všechny výhody serverových ovládacích prvků. Stejně tomu bude, použijeme-li druhé možné řešení – serverový ovládací prvek `<asp:Literal>`, který na výstup zapisuje vše tak, jak to najde ve zdrojovém souboru. Třetí možností je vytvoření nových nebo předefinování stávajících ovládacích prvků. Právě předefinování je asi nejpřijatelnější, i když stále velmi pracnou cestou. Pokud se někomu nelíbí, že `<asp:Label>` generuje `<span>`, je možno si vytvořit prvek MyLabel a nechat ho generovat cokoliv si bude přát. Stejně je možno vytvořit např. prvek MyTable odvozený (hle, dědičnost!) od standardního prvku Table a přidat mu atributy, které umožní dokonalou kontrolu nad formou generovaného HTML.

ASP.NET tedy umožňuje jisté řešení pro náročné. Na druhou stranu je potřeba uznat, že ve vygenerované stránce vždy zůstanou některé tagy, které prostě ASP.NET potřebuje a které má ve své moci. Příkladem je nevzhledný řádek

```
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTU3ODAzNTQ4MDs7Pk+pOJPcsiaMiIzzI64IDBfug4E9" />
```

který ASP.NET používá k účelům, které budou popsány později. To je potřeba přijmout jako určitou daň za komfort, který se nám výměnou dostane (i tohoto řádku se lze v extrémním případě zbavit – většinou by to však bylo proti logice věci).

Problém s validitou stránek opět není principiálním problémem ASP.NET, jedná se o konkrétní problém verze 1.0 a 1.1. V současnosti lze validitu řešit děděním od standardních serverových ovládacích prvků a předefinováním jejich výstupu, stále však platí, že se jedná o pracný postup. Microsoft nikdy příliš dobře nepodporoval webové standardy, proto je příjemným překvapením, že ASP.NET 2.0 bude podporovat generování stránek také podle standardu XHTML 1.1 Strict ([13]).

Mým pocitem je, že nová syntaxe zápisu v sobě skrývá velký potenciál, určitou daní je však ztráta dokonalé kontroly nad generovaným výstupem (přizpůsobit si ASP.NET (skoro) k obrazu svému je sice principiálně možné, ale náročné).[8]

### Událostmi řízené programování a oddělení HTML kódu od skriptu

Druhou podstatnou vlastností serverových ovládacích prvků je jejich schopnost vyvolávat události na straně serveru.

Předchozí příklad rozšíříme o tlačítko, jehož stisknutí změní původní text. Zdrojový kód vypadá následovně:

```
<%@ Page Language="C#" %>
<script runat="server">

  void ZmenitText(object sender, System.EventArgs e)
  {
    Label1.Text = "Změněný text";
  }

</script>
<html>
<head>
  <title>Demonstrace událostí</title>
</head>
<body>
  <form runat="server">
    <asp:Label id="Label1"
               runat="server"
               forecolor="Blue">Ahoj</asp:Label>
    <br />
    <asp:Button id="Button1"
```

[8] Úhledný a správně napsaný kód je dnes bohužel k vidění pouze u malých („hobby“) projektů. Google nebo Amazon.com se ani neobtěžují se „zbytečnostmi“ typu DOCTYPE.

```
                runat="server"
                text="Změnit text"
                onclick="ZmenitText" />
  </form>
</body>
</html>
```

V úvodu (v tzv. Page direktivě) jsme stanovili, že chceme používat jazyk C# (výchozím jazykem je Visual Basic .NET), a v elementu `<script runat="server">` (opět `runat="server"`) píšeme obslužný kód události `ZmenitText` stejně, jako bychom to psali v nějaké okénkové aplikaci. Za povšimnutí stojí, že je okamžitě přístupný objekt `Label1` – deklarativní zápis pomocí XML syntaxe je tedy automaticky namapován na objektový model. Velmi důležité je ale především to, že v hlavní části HTML dokumentu není vůbec žádná serverová vsuvka typu `<% %>` nebo `<?php ?>`.

Koncept oddělení skriptu od HTML kódu je doveden prakticky k dokonalosti, využijeme-li vlastnost zvanou *code behind*. Ta umožňuje zcela oddělit všechen serverový kód od HTML. Nejvýmluvnější asi bude opět malá ukázka – předělaný předchozí příklad.

V souboru aspx (přípona, podle které web server pozná, že má zpracování předat ASP.NET) nezůstane vůbec žádný kód, pouze v direktivě `Page` určíme, že tato stránka dědí od třídy `CodeBehindPage` a pomocí atributu `Src` sdělíme, kde se najde zdrojový soubor (tento atribut lze vynechat, pokud třídu nejdřív ručně zkompilujeme do DLL knihovny a tu umístíme do podadresáře bin):

```
<%@ Page Language="C#" Inherits="CodeBehindPage" Src="CodeBehindPage.cs"
%>
```

(Zbytek souboru zůstane beze změny, akorát pochopitelně zmizí element `<script>`.)

Třída samotná je velmi jednoduchá:

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;

public class CodeBehindPage : Page
{

  protected Label Label1;

  public void ZmenitText(object sender, EventArgs e)
  {
    Label1.Text = "Změněný text";
  }
}
```

Důležité je, že dědí od třídy `System.Web.UI.Page` a že třída obsahuje členskou proměnnou `Label1`, která umožňuje objektový přístup k popisku.[9]

Je tedy vidět, že všechen kód lze díky novému XML zápisu velmi jednoduše oddělit od HTML a že ve výsledku je psaní serverového kódu daleko více plnohodnotným programováním než pouhým skriptováním.

Jazyk C# verze 2.0 bude navíc podporovat tzv. částečné třídy (*partial classes*)[10], což ještě dále umožní oddělení např. kódu automaticky generovaného vývojovým prostředím a kódu, který skutečně píše člověk.

Na závěr jsem si nechal třešničku na dortu. Microsoft často proklamuje, že výhodou jejich řešení je velká konzistence jejich produktů – na straně koncového uživatele to údajně má být známé a přívětivé uživatelské rozhraní MS aplikací, na straně vývoje konzistentní programovací techniky. Zdá se, že v éře .NET to začíná být pravda, alespoň co se vývoje týče. Podíváme-li se na prezentační vrstvu „Avalon" budoucích Windows (známých pod kódovým označením Longhorn), zjistíme, že tento UI framework používá syntaxi velmi podobnou ASP.NET – konkrétně se jedná o speciální XML formát s názvem XAML (XML App-

[9] Pojem *členská proměnná* je odvozen z anglického *member variable*. Slovo *member*, *člen*, je souhrnným pojmenováním pro metody a pro atributy třídy.
[10] Částečné třídy se deklarují pomocí klíčového slova `partial` před klíčovým slovem `class` a umožňují mít jednu třídu zapsanou ve více zdrojových souborech. Podrobnosti viz např. http://msdn.microsoft.com/msdnmag/issues/04/05/C20/default.aspx.

lication Markup Language). Také události jsou k ovládacím prvkům přiřazovány deklarativně pomocí atributů. Kdo se tedy dnes seznámí s ASP.NET, bude daleko snáze pronikat do vývoje uživatelského rozhraní příštích Windows.

#### 10.1.3.3 Shrnutí

Viděli jsme tedy různé přístupy zápisu serverového kódu v PHP a v ASP.NET. PHP používá model serverových vsuvek, který je velmi intuitivní (na místo, kde má být výstup generován dynamicky, je vložen příslušný kód) a umožňuje dokonalou kontrolu nad odesílaným HTML. Serverové vsuvky bohužel mají své problémy: lze jen špatně oddělit prezentační vrstvu od aplikační logiky a kód je hůře přehledný („spaghetti code“).

ASP.NET tyto problémy řeší pomocí serverových ovládacích prvků. Zdrojový kód stránky zůstává velmi dobře čitelným XML-like dokumentem a obslužný kód je zcela odsunut buďto do jiné části zdrojového souboru nebo do souboru externího. Microsoft tomuto modelu natolik věří, že na něm postavil UI framework příštích Windows.

## 10.2 Syntaxe

Předchozí řádky byly věnovány způsobu integrace skriptového kódu s HTML kódem, nyní přišel čas na syntaxi jako takovou.

### 10.2.1 Jaké jazyky budou srovnávány

Jak bylo napsáno výše, PHP je jak prostředím pro běh webových aplikací, tak skriptovacím jazykem v tomto prostředí. ASP.NET je však jazykově neutrální, což jinými slovy znamená, že lze používat libovolný jazyk, pro který existuje kompilátor do MSIL (Microsoft Intermediate Language). Jaký jazyk má tedy ASP.NET reprezentovat? Nebo by snad mělo být PHP srovnáno přímo s MSIL?

To je jistě zajímavá myšlenka, protože pro schopnosti všech .NET jazyků je určující, nakolik je schopný samotný MSIL. Tomuto přístupu se však vyhneme z jednoho prostého důvodu: proč by PHP, v němž se kód normálně píše, mělo být srovnáváno s jakýmsi asemblerovým jazykem, ve kterém nikdo nikdy psát nebude? Navíc, MSIL má některé vlastnosti, které nejsou .NET jazykům vlastní – např. v MSIL lze program napsat strukturovaně ([14]), zatímco .NET jazyky vyžadují objektovost ([15]).

Pro účely této práce jsem se rozhodl pro jazyk C#. Důvody jsou následující:

- PHP i C# jsou jazyky z rodiny jazyků C. Jejich syntaxe je tak v lecčems podobná.
- C# je jazyk velmi oblíbený. Ačkoliv Microsoft z nějakého důvodu mírně upřednostňuje Visual Basic .NET (např. je primárním jazykem ASP.NET), sám své produkty vyvíjí v jazyku C# a obliba je také jistě zapříčiněna i velkou podobností s všeobecně uznávanou Javou.
- C# je jazyk moderní a velmi kvalitní – zajímavým počtením je třeba jeho srovnání s Javou v dokumentu [C#vsJava] (není třeba děsit se toho, že článek napsal člověk z Microsoftu, pan Kirk Radeck ke srovnání přistoupil seriózně).
- C# byl stvořen přímo pro platformu .NET

### 10.2.2 Základní prvky syntaxe

Jelikož jsou oba srovnávané jazyky z rodiny jazyků C, mají toho dost společného, alespoň na první pohled.

Oba jazyky jsou case-sensitive, tj. citlivé na velikost písmen. Názvy `var` a `VAR` určují dvě různé proměnné, zatímco např. ve Visual Basicu .NET (je příkladem case-insensitive jazyka) budou tyto dva názvy představovat jedinou proměnnou. Přesto je citlivost na velikost písmen v PHP mírnější, protože názvy funkcí už case-sensitive nejsou (`print` a `PRInT` jsou stejné funkce). ASP.NET něco jako funkce vůbec nezná, protože používá Base Class Library, kde jsou třídy a jejich metody pochopitelně case-sensitive.

Oba jazyky mají skoro stejná pravidla pro vytváření názvů proměnných. V zásadě může být proměnná tvořena písmeny, čísly a podtržítkem, přičemž nesmí začínat číslem. Oba jazyky v názvech proměnných

podporují diakritiku, není tedy problém mít např. proměnnou `DélkaKřídla`, i když to není vůbec doporučeníhodné.

Oba jazyky se podobají také v základních konstrukcích, např. v podmínkách nebo v cyklech. Podobnost dokumentuje následující minisrovnání:

| PHP | C# |
|---|---|
| `if ($udelamZkousku) {`<br>`  for ($i = 0; $i < 10; $i++) {`<br>`  print("Hip hip hurá");`<br>`  }`<br>`}` | `if (udelamZkousku) {`<br>`  for (int i = 0; i < 10; i++) {`<br>`  Response.Write("Hip hip hurá");`<br>`  }`<br>`}` |

(poznámka: znak dolaru není součástí názvu proměnné, pouze určuje začátek identifikátoru proměnné; jedná se o konvenci z Unixových systémů)

Oba uvedené výstupy se na první pohled liší jen velmi málo, přesto už na tomto kraťoučkém výpisu lze předvést základní odlišnosti.

Zaprvé, C# a obecně všechny .NET jazyky jsou přísně typové. To znamená, že každá proměnná musí mít explicitně stanovený nějaký typ (v .NETu je společný typový systém důležitou součástí architektury, viz např. [16]). PHP je na rozdíl od C# jazykem netypovým, není tedy nutno uvádět, jakého typu proměnná je, její typ bude určen až za běhu. Aby tedy ukázka v jazyku C# byla korektní, musela být někde dříve deklarována proměnná `udelamZkousku` typu `bool`, zatímco v PHP se o nějaké deklarování proměnné starat nemusíme (ačkoliv použití neinicializované proměnné v podmínce samozřejmě postrádá jakýkoliv smysl). Druhým příkladem v ukázce je for-cyklus, kde v jazyku C# musí být deklarováno, že proměnná `i` je typu `int` (synonymum pro třídu `System.Int32` .NET Frameworku), zatímco v PHP můžeme používat `$i` rovnou, typ bude dosazen až za běhu.

Netypové PHP tedy umožňuje „jednodušší“ zápis kódu, kdy se není potřeba starat o typy proměnných. Mezi velké nevýhody ale na druhou stranu patří chybějící typová bezpečnost. Porovnáme-li řetězec „10“ s číslem 10, výsledek bude true, což je v drtivé většině případů neočekávaný výsledek. Tuto situaci lze řešit buďto funkcí `is_{type}` (např. `is_int`), což je téměř nepoužitelné kvůli přílišnému zesložitění kódu, nebo pomocí operátoru `===`, který nejen že porovnává hodnoty proměnných, ale i jejich typ. Mohlo by se tedy zdát, že tímto operátorem je celý problém vyřešen, přesto zde však zůstává několik dalších věcí.

Tou první je okamžité zvýšení nároků na vývojáře. Pro začátečníky je dostatečným oříškem zvyknout si na rozdíl mezi `=` a `==` (první operátor je operátorem přiřazení, druhý operátorem porovnání), nyní přichází na řadu ještě jakýsi operátor s třemi rovnítky. Netvrdím, že by byl operátorem nějak záludným, ale klidně bych se vsadil, že se najdou lidé, kteří o něm nemají ani zdání. Velkou předností PHP je jeho jednoduchost a ta je tímto novým operátorem poněkud poskvrněna.

Dalším problémem netypového PHP je z principu obtížná podpora ve vývojových prostředích. Tak například Zend Studio, jedno z nejlepších IDE pro PHP, v automatickém dokončování neuvádí, jaký typ určitá funkce vrací. V definici funkce skutečně syntaxe žádnou deklaraci návratového typu nepovoluje, nebylo by ale možné funkci zkoumat, najít příkaz `return` a podle něj poradit? Bohužel nešlo, zde je důvod:

```
function vratitTyp($jakyTyp) {
  if ($jakyTyp == "bool")
    return true;
  if ($jakyTyp == "int")
    return 1;
}
```

V PHP totiž není problém v rámci jedné funkce vracet několik různých typů. Někomu to může připadat jako flexibilita, někomu jako anarchie, nicméně se jedná o naprosto korektní vlastnost netypového PHP. Zde uvedený příklad přitom není nějaký teoretický konstrukt – stačí se podívat na často používanou funk-

ci `strpos()` ([17]), která při nalezení vzorku na první pozici vrací nulu, zatímco pokud ho nenajde, vrátí false. Vývojář pak musí mít velmi jasno v tom, jestli použít operátor `==` nebo `===`.

V souvislosti se slabým typováním v PHP bývá uváděn další problém – neinicializované proměnné. Typový jazyk tímto neduhem z principu trpět nemůže, neznamená to však, že by neinicializované proměnné byly přímým důsledkem netypového jazyka, jedná se spíš o konkrétní nedostatek PHP.

Jako příklad poslouží následující kód:

```
function slozitaFunkce() {
  // ...
  return true;
}

$slozitaFunkceProbehlaUspesne = slozitaFunkce();
if ($slozitaFunceProbehlaUspesne) {
  // ...
} else {
  echo „Funkce neproběhla úspěšně“;
}
```

Výstupem skriptu bude pro vývojáře dost nepříjemná hláška, že funkce neproběhla úspěšně. Pustí se tedy do zkoumání obsahu své složité funkce a po několika hodinách testování a ladění nejspíše s velkou radostí zjistí, že se jednoduše přepsal v if-podmínce, když ve jménu proměnné `slozitaFunkceProbehlaUspesne` zapomněl na první „k“.

Řešení jsou v podstatě tři. Tím prvním je používání nějakého lepšího textového editoru, který umí automaticky doplňovat dříve použité názvy proměnných. Druhou možností je používání určitých analyzátorů kódu, které mohou na podezřelé konstrukce upozornit – zatím neinicializovaná proměnná v podmínce určitě bez povšimnutí nezůstane. Takový analyzátor kódu je např. součástí vývojových prostředí Zend Studio nebo NuSphere PhpED. Třetí možností je správné nastavení sekce *Error handling and logging* v souboru php.ini – pomocí direktivy `error_reporting` je možno zapnout zobrazování chyb úrovně `E_NOTICE`, které jsou při použití neinicializovaných proměnných generovány. Zde je však jednoduchost jazyka tak trochu vykoupena nutností starat se o administrační záležitosti.

Doposud byly uváděny především nevýhody slabě typového PHP, tento jazyk má však i svá kouzla. Je totiž velmi pružný a některé programátorské úkony lze řešit velmi snadno – jinými slovy nezapře, že má velmi blízko k Perlu. Typickým příkladem jsou pole, která už snad ani pružnější být nemohou. Prvním indexem může být číslo a hodnotou třeba string, druhým indexem může být naopak string a hodnotou nějaké další pole, jednoduše řečeno indexy i hodnoty v rámci jednoho pole můžou být skoro libovolných typů. Hlavní devizou však zůstává možnost řešit problémy spojené s typem objektu až v době běhu programu, nikoli během návrhu.

Jaký je tedy závěr? PHP je jednoduchý flexibilní jazyk, kterého se nemusí bát nikdo a který umožňuje stručný zápis kódu. Trpí však některými neduhy, které jazyk C# nemá. Osobně se také domnívám, že „profesionální“ programátoři jsou zvyklejší na typové jazyky (Java, C++, Visual Basic, …), takže jim může C# připadat přirozenější.

## 10.2.3 Podpora OOP

### 10.2.3.1 Zapouzdření

Zapouzdření je základním principem objektově orientovaného programování. Objekt zveřejňuje pouze rozhraní pro komunikaci s ním, zatímco vnitřní implementace zůstává skrytá a nedostupná.

ASP.NET (a tedy všechny .NET jazyky) i PHP podporují vytváření tříd a objektů. Syntaxe je opět velmi podobná:

| PHP | C# |
|---|---|
| `class MojeTrida {`<br>`  private $promenna;`<br>`  // další proměnné` | `class MojeTrida {`<br>`  private string promenna;`<br>`  // další proměnné` |

```
  function funkce1() {
    // kód
  }
  // další funkce
}
```

```
  void funkce1() {
    // kód
  }
  // další funkce
}
```

Ve svém základním tvaru se tedy PHP a C# opět liší pouze tím, že C# používá typy (`string`, pseudotyp `void`), zatímco PHP nikoliv. Pro zapouzdření je však důležité řízení přístupu k členům třídy.

Jazyk C# podporuje následující modifikátory přístupu: `public`, `protected internal`, `protected`, `internal` a `private` (seřazeno podle stupně omezení vzestupně). `Public` přístup neomezuje, `protected internal` znamená `protected` or `internal`, `protected` znamená omezení na danou třídu a její potomky, `internal` omezení na danou assembly[11] a `private` pouze na danou třídu. Jak vidno, jazyk C# (lépe řečeno celý .NET Framework) podporuje celou řadu modifikátorů přístupu, navíc princip zapouzdření podporuje tím, že pokud není žádný modifikátor u člena třídy uveden, je implicitně `private`.

Co se týče PHP, syntaxe vytváření tříd je podporována už od pradávné verze 3.0. Až v nové verzi 5.0 se ale objevují modifikátory přístupu `public`, `protected` a `private`, které **musí** být před proměnnou uvedeny. Význam je prakticky stejný jako v případě jazyka C# (modifikátor ala `internal` neexistuje, protože PHP nezná nic jako nějaké programové celky, komponenty). Malým problémem je zpětná kompatibilita. Jelikož modifikátory přístupu byly přidány až v páté verzi, je nutno se nějak vypořádat s klíčovým slovem `var`, které bylo dříve používáno pro označení členské proměnné třídy. Aby většina aplikací po přechodu z PHP 4 na PHP 5 fungovala, klíčové slovo `var` automaticky znamená ekvivalent modifikátoru `public`, což jde trochu proti principu zapouzdření (implicitní `private` v C# je určitě vhodnější).

Jazyk C# má jedno velice příjemné syntaktické zjednodušení, kterým jsou tzv. *vlastnosti* (*properties*):

PHP

```
class Vlastnosti {
  private $vlastnost = 1;

  public function getVlastnost() {
    return $this->vlastnost;
  }

  public function setVlastnost($val) {
    $this->vlastnost = $val;
  }
}


$objVlastnosti = new Vlastnosti();
$val = $objVlastnosti->getVlastnost();
$val++;
$objVlastnosti->setVlastnost($val);
```

C#

```
class Vlastnosti {
  private int vlastnost = 1;

  public int Vlastnost {
    get { return vlastnost; }
    set { vlastnost = value; }
  }
}



void spustitDemo()
{
  Vlastnosti objVlastnosti =
              new Vlastnosti();
  objVlastnosti.Vlastnost++;
}
```

Za povšimnutí stojí především řádek, kde se atribut `vlastnost` inkrementuje o jedničku. PHP, které se v tomto případě drží konvencí Javy (pro přístup k členské proměnné používá speciální metody nazývané *getters* a *setters*, které se tvoří podle konvence `getJmenoPromenne` a `setJmenoPromenne`), musí nejdříve pomocí metody `getVlastnost` získat hodnotu proměnné `vlastnost`, inkrementovat ji a následně uložit zpět[12]. Jazyk C# při použití vlastností umožňuje jednoduše použít lépe čitelnou syntaxi `Vlastnost++`.

[11] Assembly je logická programová jednotka, komponenta. Trochu by se dala přirovnat k javovskému JAR archivu, který také sdružuje určitou funkčnost a sám sebe popisuje pomocí manifestu , existují zde ale určité odlišnosti, např. JAR je fyzicky jeden soubor, zatímco assembly je pouze logická jednotka (i když se také může jednat o jediný soubor), rozdílů by se samozřejmě našlo víc (třeba rozdíl v přístupu k verzování). Koncept assembly je v .NET Frameworku důležitý, více se lze dozvědět např. z .NET Framework Developers Guide ([18]).

[12] Uvedený kód PHP by šel přepsat do podoby `$objVlastnosti->setVlastnost($objVlastnosti->getVlastnost() + 1)`, která je sice kratší, ale o něco méně přehledná.

Můžeme tedy shrnout, že princip zapouzdření obě technologie podporují dobře. Jazyk C# má o dva modifikátory přístupu víc (`internal` a `protected internal`), které PHP nemůže mít, protože se nejedná o komponentovou technologii. Vlastnosti (properties) jazyka C# navíc zjednodušují zápis a podle mého názoru dobře odpovídají intuitivní představě o tom, že objekt má určité *vlastnosti* a umí provádět určité činnosti.

#### 10.2.3.2 Dědičnost

Dědičnost je schopnost definovat určité třídy jako předky a jiné jako potomky, přičemž potomci po svých rodičích dědí jak členské proměnné, tak metody.

PHP se v tomto ohledu s jazykem C# podobají jako vejce vejci. Prvním významným společným rysem je podpora pouze jednoduché dědičnosti (mnohonásobnou dědičnost podporuje např. C++). Jednoduchá dědičnost je v současnosti jednoznačným trendem – kromě zde uváděných jazyků ji podporuje další „velký" jazyk, Java. Ta uvedla ve známost takzvaná *rozhraní* (*interface*), která mají vícenásobnou dědičnost zastoupit. Rozhraní je jakýsi předpis třídy, který určuje, jaké metody má mít třída, která bude toto rozhraní implementovat. Daná třída pak může implementovat libovolný počet rozhraní. Syntaxe i chování v Javě, v PHP i v C# se prakticky neliší.

PHP zná klíčové slovo `final`, které je ekvivalentem klíčového slova `sealed` v jazyce C#. Od takto označené třídy již nelze dědit.

Oba jazyky rovněž znají koncept abstraktních tříd. Hlavním znakem abstraktních tříd je, že nemohou být instanciovány, jejich účelem je tedy sloužit jako třídy předků. Mohou (ale nemusejí) obsahovat abstraktní metody, což jsou metody bez implementace. Potomek potom tyto metody musí buďto implementovat, nebo musí být sám abstraktní třídou.

V podpoře dědičnosti se jazyky neliší prakticky vůbec.

#### 10.2.3.3 Skládání

Skládání (composition) je jakýmsi doplňkem dědičnosti. Dědičnost by měla být použita tam, kde třída B *je* třídou A (student je osobou), skládání tehdy, pokud třída B *obsahuje* třídu A (auto obsahuje motor).

Skládání umožňuje znovupoužít dříve napsaný kód tím, že se členskou proměnnou třídy B stane objekt třídy A. Tím jsou ve třídě B dostupní členové třídy A (samozřejmě s ohledem na modifikátory přístupu, viz zapouzdření). Způsob použití nejlépe demonstruje malá ukázka:

PHP

```
class A {
  public function secti($a, $b) {
    return $a + $b;
  }
}

class B {
  public $a;

  public function __construct() {
    $this->a = new A();
  }

  public function secti($a, $b) {
    return $this->a->secti($a, $b);
  }
}

$b = new B();
echo $b->secti(1,2);
```

C#

```
class A {
  public int secti(int a, int b) {
    return a + b;
  }
}

class B {
  public A a;

  public B() {
    this.a = new A();
  }

  public int secti(int a, int b) {
    return this.a.secti(a,b);
  }
}

B b = new B();
vypis(b.secti(1,2));
```

Také skládání se tedy v obou jazycích realizuje téměř přesně stejně.

#### 10.2.3.4 Polymorfismus

Důležitým znakem objektově orientovaného přístupu je polymorfismus. Koncept polymorfismu je na rozdíl od předchozích značně obtížný a jeho podrobný rozbor by jistě bohatě vystačil na samostatnou práci. Nechme však stranou složitou teorii[13] a spokojme se s tím, že v běžném programování je polymorfismus většinou ztotožňován s vlastností, která umožňuje na hierarchicky odlišných stupních dědičnosti používat metody stejného názvu (lépe řečeno stejné signatury). Typickým příkladem je třída `Zvire`, která má metodu `kdoJsi()`, která vypisuje identifikaci zvířete. Základní třída bude např. vypisovat „Jsem kdovíjaké zvíře". Polymorfismus nám umožňuje vytvořit odvozené třídy `Pes` a `Kocka`, které budou mít rovněž metodu `kdoJsi()` (a nikoliv dvě speciální metody typu `kdoJsiPse` a `kdoJsiKocko`). Obě třídy si mohou tuto metodu implementovat podle svého a to nám umožňuje elegantně vytvořit malou zvířecí seznamku, když vytvoříme nějakou metodu, která ve svém parametru bude přijímat obecné zvíře. Jelikož podtyp je vždy zároveň svým nadtypem (tzv. ISA vztah, kočka *je* („is a") zvíře, pes *je* zvíře), můžeme této metodě předávat klidně kočku nebo psa. Ba co víc, pokud se nám naše zvířecí parta rozroste ještě o ježka, metodu vůbec není potřeba měnit, protože počítá s generickým typem `Zvire`. Následuje ukázka implementace:

PHP

```
class Zvire {
  function kdoJsi() {
    echo "Jsem kdovíjaké zvíře";
  }
}

class Pes extends Zvire {
  function kdoJsi() {
    echo "Jsem hodný pes";
  }
}

class Kocka extends Zvire {
  function kdoJsi() {
    echo "Jsem proradná kočka";
  }
}

function predstavSe($zvire) {
  $zvire->kdoJsi();
}

$zvire = new Zvire();
$kocka = new Kocka();
$pes   = new Pes();

predstavSe($zvire);
predstavSe($kocka);
predstavSe($pes);
```

C#

```
class Zvire {
  public virtual void kdoJsi() {
    vypis("Jsem kdovíjaké zvíře");
  }
}

class Pes : Zvire {
  public override void kdoJsi() {
    vypis("Jsem hodný pes");
  }
}

class Kocka : Zvire {
  public override void kdoJsi() {
    vypis("Jsem proradná kočka");
  }
}

public void predstavSe(Zvire zvire) {
  zvire.kdoJsi();
}

Zvire zvire = new Zvire();
Zvire kocka = new Kocka();
Zvire pes   = new Pes();

predstavSe(zvire);
predstavSe(kocka);
predstavSe(pes);
```

(Poznámka: Zatímco kód PHP lze překopírovat a jednoduše spustit, ukázku v C# tvoří poskládané fragmenty kódu uspořádané tak, aby svým umístěním odpovídaly PHP. Celý zdrojový kód by byl o něco složitější.)

Výpisem bude v obou případech nejdříve představení obecného zvířete a potom už představení kočky a psa, přestože ve funkci `predstavSe()` je metoda `kdoJsi()` volána na objektu typu `Zvire`. Takto pojatý polymorfismus bývá označován jako *pozdní vazba* (*late binding*), kdy se o konkrétní metodě, která bude nakonec vyvolána, rozhoduje až za běhu, kdy jsou konečně známy správné typy.

Pozornému oku nemohlo ujít, že kód v PHP je daleko přímočařejší. Mám na mysli především to, že v jazyku C# se muselo poměrně umně žonglovat s klíčovými slovy `virtual` (u základní třídy) a `override`

[13] Představu o komplexnosti problematiky si lze udělat např. pročtením odpovídajícího tématu Wikipedie ([19]).

(u tříd odvozených). To je dáno architekturou jazyka. PHP se chová podobně jako Java a všechny metody jsou implicitně virtuální. Dokonce neexistuje způsob, jak metodu deklarovat tak, aby virtuální nebyla. C# se chová opačně – všechny metody implicitně nejsou virtuální a právě proto bylo nutno použít speciální klíčová slova, která metody donutila k virtuálnímu chování. Otázka, zda k věci správně přistupuje PHP nebo C#, je sice velmi zajímavá, ale zároveň velmi těžko zodpověditelná. Je záležitostí poměrně hlubokých úvah, proto vřele doporučuji přečíst rozhovor s hlavním architektem jazyka C# Andersem Hejlsbergem ([20]). Zajímavé myšlenky lze také vyčíst z poněkolikáté citovaného srovnání C# a Javy ([C#vsJava], str. 36). Jako hlavní nevýhodu přístupu C# vidí Kirk Radeck větší náročnost designu aplikace (tvůrce základní třídy musí přemýšlet, zda má metoda být virtuální či nikoliv, a pokud se rozhodne špatně, může to mít velmi nepříjemné důsledky pro tvůrce odvozené třídy), výhodou je zase naopak příznivý dopad na výkon aplikace, protože non-virtuální metody jsou řešeny pomocí brzké vazby (early binding) při kompilaci, což je efektivnější než pozdní vazba probíhající až za běhu.

S konceptem polymorfismu rovněž souvisí další prostředek objektově orientovaných jazyků – *přetěžování* (*overloading*). Jde o to, že např. pro sečtení dvou čísel nemusíme mít dvě metody `sectiCelaCisla(int a, int b)` a `sectiDesetinnaCisla(double a, double b)`, ale prostě vytvoříme dvě metody `secti`, přičemž první bude přijímat parametry typu `int` a druhá typu `double`.

Přesně podle tohoto vzoru funguje C#, který přetěžování plně podporuje, zatímco PHP ho kvůli své dynamicky typované podstatě nepodporuje, respektive nepotřebuje. V jistých případech lze podobné chování velmi jednoduše nasimulovat, viz příklad:

| PHP | C# |
|---|---|
| `private static function secti($a, $b) {`<br>`  return $a + $b;`<br>`}` | `static int secti(int a, int b) {`<br>`  return a + b;`<br>`}`<br><br>`static double secti(double a, double b)`<br>`{`<br>`  return a + b;`<br>`}` |

V tomto případě opět výrazně sympatičtěji působí PHP, které umožňuje kratší a intuitivnější zápis. Podobně lze nasimulovat i situaci, kdy funkce jednoho jména budou mít různý počet argumentů, např. přidáme metodu `secti`, která ve svém třetím parametru bude přijímat formátovací řetězec a bude vracet součet ve formě adekvátně naformátovaného stringu (např. číslo zaokrouhlené na dvě desetinná místa apod.). V jazyce C# jednoduše vytvoříme novou přetíženou funkci, v PHP, které umožňuje vytvořit pouze jedinou funkci daného jména, přidáme třetí nepovinný parametr (hlavička funkce bude vypadat např. takto: `private static function secti($a, $b, $format = NULL)`). Již zde se vynořují některé problémy.

Zaprvé, v PHP se funkce `secti` stává složitější a složitější – např. po přidání třetího parametru bude muset obsahovat větvení podle toho, zda je tento parametr `NULL` či nikoliv. Čím více přetížených metod tedy musí PHP v jediné metodě simulovat, tím logicky vzrůstá její složitost. Argumenty navíc nemusejí zůstat tak jednoduché – představme si například, že metoda `secti` bude sloužit pro sčítání dvou chemických látek nebo dvou matic.

Zadruhé, PHP při simulaci většího množství přetížených metod ztrácí na flexibilitě. Pokud bychom například z nějakého důvodu chtěli, aby formátovací řetězec v metodě `secti` nebyl argumentem třetím, ale prvním, dostaneme se okamžitě do problémů, protože první dva argumenty budou vyhrazeny pro čísla.

Dalším aspektem je přetěžování operátorů, což je poměrně důležité syntaktické zjednodušení. Z velkých jazyků, které přetěžování operátorů podporují, jmenujme C++, Visual Basic .NET a C#, velkým zástupcem jazyka nepodporujícího tuto vlastnost je Java a právě PHP. Přetěžování operátorů umožňuje předchozí příklad v jazyce C# přepsat do elegantní podoby, kde se budou celá čísla sčítat stylem 1+2, desetinná stylem 1.3 + 8.8, vektory stylem (3;-1) + (8;0) atd. PHP bude muset používat prefixovou notaci `secti(prvek1, prvek2)`.

Přetěžování je tedy v jazyce C# plně podporováno, zatímco v PHP může být pouze simulováno. V některých případech dokonce kratším a „hezčím“ zápisem, nicméně s rostoucím rozsahem aplikace se projeví výhody typového přístupu jazyka C#.

(Poznámka na závěr: Přetěžování není některými jazykovými puristy vůbec považováno za polymorfní metodiku, protože se podle nich jedná pouze o syntaktickou konvenci. Tvrdí, že polymorfismus je pouze simulován stejným jménem metody, nicméně zde není používán princip pozdní vazby, proto se v pravém slova smyslu o polymorfismus nejedná.)

#### 10.2.3.5 Znovupoužitelnost

Znovupoužitelnost není principem objektově orientovaného přístupu, je však jedním z jeho důležitých důsledků. Realizuje se pomocí dědičnosti (10.2.3.2) nebo pomocí skládání (10.2.3.3).

Jak bylo uvedeno o něco výše, dědičnost i skládání jsou v PHP a v C# (a v celém ASP.NET) podporovány velmi podobně. Proč tedy znovu rozpitvávat znovupoužitelnost? Podle mého názoru se v tomto ohledu přeci jen jednotlivé technologie liší. Prvním důvodem je, že objektově orientovaný přístup je v PHP pouze alternativou, zatímco v ASP.NET je naprostým základem a jinak než pomocí OOP ani aplikace vyvíjet nelze. Kromě toho, často proklamovanou silnou stránkou PHP je právě jeho jednoduchost, proto se nelze divit, že mnoho aplikací bude vytvořeno bez použití OOP. Ačkoliv tedy zůstává pravdou, že míra znovupoužitelnosti silně závisí na zkušenosti a dovednostech vývojáře, přeci jen ASP.NET ke znovupoužitelnosti kódu přispívá podstatně větší měrou.

Dalším „donucovacím prostředkem“ ASP.NET (v tom nejlepším slova smyslu) jsou serverové ovládací prvky (kapitola 10.1.3.2, strana 10) a především možnost tvorby uživatelských ovládacích prvků. Budeme-li tvořit např. stránku, kde bude vyžadováno přihlášení pomocí uživatelského jména a hesla, vytvoříme si nový serverový ovládací prvek, skládající se ze dvou textových polí, možná nějakých popisků a tlačítka. Tento nový ovládací prvek pak bude fungovat naprosto stejně jako všechny ostatní, např. budeme moci do kódu jednoduše zapsat `<myControls:MyLoginControl runat="server" passwordchar="|" backgroundcolor="red" />`, tento nový prvek můžeme přidat do vývojového prostředí a vizuálně ho do stránky umísťovat, můžeme jeho vlastnosti nastavovat přes dialog Properties apod. Nejen že ovládací prvek `MyLoginControl` můžeme jednoduše používat na více místech projektu, můžeme ho pomocí dědičnosti velmi jednoduše rozšiřovat (pokud budeme např. chtít přidat zaškrtávátko „Zapamatovat heslo“).

Podle mého názoru tedy ASP.NET celou svou objektovou koncepcí i pomocí serverových ovládacích prvků velmi dobře podporuje znovupoužitelnost kódu, i když kód bude vždy jen tak znovupoužitelný, jak to umožní jeho autor – jinými slovy, i v PHP lze napsat krásně znovupoužitelný kód stejně jako jde v ASP.NET napsat kód zcela nepoužitelný.

#### 10.2.3.6 Výjimky

Výjimky umožňují reagovat na chybové stavy velmi přehledným a konzistentním způsobem, navíc s využitím objektového přístupu (výjimky jsou objekty). Nesouvisejí sice přímo s objektově orientovaným paradigmatem, jsou však většinou s OOP spojovány.

PHP i C# používají prakticky stejný zápis – známou strukturu try/catch (v jazyku C# najdeme navíc blok finally). Oba jazyky definují třídu s názvem `Exception` (v ASP.NET se tato třída nachází ve jmenném prostoru `System`), která je předkem všem ostatním výjimkám. ASP.NET to přísně vyžaduje, v PHP je to naopak pouze doporučený postup, lze si ale vytvořit vlastní hierarchii výjimek, která nebude mít s třídou `Exception` nic společného.

V čem jsou tedy rozdíly? První je v tom, že C# na rozdíl od PHP podporuje blok finally. Ten proběhne vždy, ať už výjimka nastane či nikoliv, což je velmi důležité, pokud se v bloku try pracuje s nějakými zdroji, které je vhodné v každém případě zavřít (třeba připojení k databázi). V PHP lze finally simulovat tak, že se příslušný kód napíše za try/catch. Přesto to není příliš čisté řešení, nehledě na to, že se zákonitě vyskytnou problémy, pokud budeme např. výjimku propagovat dál (klíčové slovo `throw` v bloku catch).

Druhým rozdílem je celkový přístup k výjimkám. V ASP.NET jsou výjimky základním prostředkem zpracování chybových stavů, zatímco v PHP se jedná pouze o nový jazykový konstrukt. Znamená to, že pokud v ASP.NET selže pokus o připojení k databázi, bude vyvolána výjimka, zatímco v PHP bude vyvolána chyba (error) ([21]). V PHP je nutno výjimky vyvolávat explicitně pomocí klíčového slova `throw`. Toto

chování se asi jen tak nezmění, protože vychází z podstaty PHP. Jednak je potřeba zachovat zpětnou kompatibilitu (kdyby např. funkce `mysql_connect` změnila své chování a začala by vyhazovat výjimku, mělo by to vliv na drtivé množství webových aplikací), jednak by bylo proti naturelu PHP, kdyby své vývojáře tlačilo do objektového programování.

#### 10.2.3.7 Organizace dostupných funkcí

Jak PHP, tak ASP.NET nabízejí předdefinované funkce pro řešení běžných problémů. Dostupné jsou knihovny pro práci s obrázky, pro zjednodušení uploadu na server, pro pokročilou práci s textem atd. Těchto funkcí je k dispozici velké množství (několik stovek až několik tisíc), proto je důležitá otázka, jak jsou funkce organizovány, uspořádány.

ASP.NET tento problém řeší zavedením tzv. *jmenných prostorů* (*namespaces*), což je v podstatě ekvivalent balíčků (packages) v Javě. Jmenné prostory umožňují logické seskupování tříd podle nabízené funkčnosti. Tak např. třídy pro podporu práce s XML jsou ve jmenném prostoru `System.Xml`, třídy pro práci s datovými zdroji ve jmenném prostoru `System.Data` apod. Jmenné prostory lze navíc uspořádat hierarchicky, takže např. třídy pro práci s ODBC datovými zdroji budou umístěny ve jmenném prostoru `System.Data` jako `System.Data.Odbc`. Hierarchické uspořádání není dokonalé (protože hierarchie může být vytvořena pouze podle jednoho kriteria), nicméně nevím o tom, že by pro uspořádání tříd bylo vymyšleno něco lepšího.

Důležité je, že veškerá funkčnost v ASP.NET je poskytována pomocí metod určitých tříd. Nenalezneme zde proto funkci typu `echo` z PHP, vždy to musí být něco jako `Response.Write` (objekt tečka metoda).

Naproti tomu PHP svou funkčnost zpřístupňuje pomocí lineárně uspořádaných funkcí, které nejsou součástí objektů. Podle mého názoru to není příliš šťastné řešení z několika důvodů.

Tím prvním je malá intuitivnost při hledání funkce. Funkce jsou seřazeny abecedně bez jakékoliv hierarchické organizace. Jediným podpůrným prostředkem je tedy manuál, který je naštěstí v případě PHP poměrně přehledný. Přesto zde zůstávají některé problémy, jako třeba že se řešení organizace funkcí přesouvá z vývojářského týmu na tým dokumentační (je na nich, zda funkce pro MySQL a PostgreSQL budou pod nějakou zastřešující kapitolou nebo jestli to budou dvě samostatné nezávislé kapitoly) nebo že je obtížnější dokumentaci lokalizovat („String functions“ bylo do české verze přeloženo jako „Funkce pro práci s řetězci“, což je jazykově úplně jiné uspořádání, navíc tato kapitola zůstává kapitolou stodevátou, takže je v české verzi manuálu zcela nemožné orientovat se podle abecedy).

Druhým problémem je nekonzistence v názvech funkcí. Do jisté míry je zaběhlým způsobem zápisu `oblast_nejaka_funkce()`. Např. všechny funkce týkající se PostgreSQL začínají prefixem `pg_`, všechny funkce pro PDF prefixem `pdf_` apod. (Mimochodem, funkce pro práci s PDF demonstrují, že ne vždy nabízí rozsáhlý .NET Framework lepší podporu – ten totiž funkce pro práci s PDF vůbec neobsahuje.) Potom ale existují skupiny funkcí, kde panuje naprostá anarchie. Stačí se podívat na funkce pro práci s řetězci ([22]), kde neexistují prakticky žádné konvence, někdy jsou dokonce v rozporu. Např. funkce `str_split()` obsahuje podtržítko, zatímco funkce `strcasecmp()` žádná podtržítka neobsahuje.

PHP tedy logické organizaci funkcí moc nepřispívá, a ačkoliv v dokumentaci k projektu PEAR ([23]) už určitá pravidla pro pojmenovávání funkcí existují, názvy starých funkcí musejí být kvůli kompatibilitě zachovány. Tak jako tak, hierarchické jmenné prostory se mi zdají být lepším řešením.

#### 10.2.3.8 Shrnutí

PHP ve své nové verzi 5 udělalo úžasný skok kupředu, co se podpory objektového programování týče. Zároveň však zůstala možnost psát webové aplikace „starým dobrým“ skriptovacím způsobem, což z PHP dělá vskutku univerzální nástroj. ASP.NET takto univerzální není, naproti tomu na syntaktické úrovni podporuje vývoj aplikací „plně objektovým“ způsobem. Nejde jen o velmi dobrou podporu objektově orientované syntaxe, ale také o koncept znovupoužitelných serverových ovládacích prvků a o objektové zpřístupnění funkcí v .NET Frameworku.

### 10.2.4 Shrnutí jazykového srovnání

Exkurze po syntaktických zákoutích jazyků C# a PHP je u konce a zbývá shrnout, co bylo k vidění. PHP se ukázalo jako pružný a jednoduchý jazyk, který ve své páté verzi dokonce kromě jednoduchého skriptování dobře podporuje vývoj objektovým způsobem. Tato univerzálnost může být pro vývojáře velmi lákavá. ASP.NET, zde zastoupené jazykem C#, je naproti tomu přísně objektové, což klade na vývojáře poněkud větší vstupní nároky. Některé jednoduché úlohy také nelze provádět s přímočarostí PHP. Výměnou za to všechno nabízí špičkovou podporu OOP se všemi spojenými výhodami.

## 10.3 Řešení typických úloh

Zatím byla popsána pouze syntaxe jazyků, velmi důležité je však také to, jaké prostředky nabízí ta která technologie pro řešení typických problémů při vývoji webových aplikací. Tomu bude věnována tato kapitola.

### 10.3.1 Databáze a jiné datové zdroje

Komunikace s databázemi a s jinými datovými zdroji je snad vůbec nejdůležitější prací webových aplikací. Proto mají pro tuto oblast jak PHP, tak ASP.NET velmi dobrou podporu. Koncepce je však trochu jiná.

PHP pro databáze nabízí speciální knihovny. Základní distribuce tak rovnou obsahuje podporu např. pro MySQL[14], PostgreSQL, Oracle, Microsoft SQL Server, Firebird/Interbase atd. Samozřejmě zde najdeme také modul pro ODBC a podporován je i zajímavý projekt SQLite (podrobnosti viz http://sqlite.org/). Zde je typický kód pro připojení k MySQL[15]:

```
<?php
/* Connecting, selecting database */
$link = mysql_connect("mysql_host", "mysql_user", "mysql_password")
   or die("Could not connect : " . mysql_error());
echo "Connected successfully";
mysql_select_db("my_database") or die("Could not select database");

/* Performing SQL query */
$query = "SELECT * FROM my_table";
$result = mysql_query($query) or die("Query failed : " . mysql_error());

/* Printing results in HTML */
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
   echo "\t<tr>\n";
   foreach ($line as $col_value) {
       echo "\t\t<td>$col_value</td>\n";
   }
   echo "\t</tr>\n";
}
echo "</table>\n";

/* Free resultset */
mysql_free_result($result);

/* Closing connection */
mysql_close($link);
?>
```

Z této ukázky jsou vidět dvě důležité věci: manipulace s databází probíhá strukturovaným voláním funkcí, přičemž reakcí na chybu je příkaz `die` (ukončení skriptu; syntaxe „připoj se or die“ je ekvivalentem „if (nepodařilo se připojit) then die“). Druhou důležitou věcí je výpis výsledků do tabulky ve while-cyklu, kdy se v každé iteraci serverový kurzor posouvá na další řádek výsledků. V rámci jednoho řádku jsou pak pomocí cyklu `foreach` vypsány hodnoty všech sloupců.

ASP.NET k datovým zdrojům přistupuje zcela jinak, a sice přes technologii ADO.NET (fyzicky je reprezentována třídami ve jmenném prostoru `System.Data`, její postavení v rámci .NET Frameworku viz

[14] Ačkoliv je PHP často s MySQL spojováno, není tato databáze nijak upřednostňována.
[15] Převzato z manuálu, http://www.php.net/manual/en/ref.mysql.php

obrázek 1 na straně 7). Otázkou je, k čemu další databázové API, když zde máme ODBC, OLE DB, DAO, ADO, RDO a další. Zjednodušeně proto, že žádná ze stávajících technologií není příliš dobře připravena na bezstavovou éru protokolu HTTP. Mezi další důležité cíle návrhu patří vylepšená podpora XML a snaha oddělit připojení k datům od manipulace s nimi ([24]). Asi nejlepší bude malá ukázka (pro jednoduchost bude vynecháno ošetřování možných výjimek).

Nejdříve situace, kdy máme kontakty uloženy v následujícím XML souboru (Kontakty.xml):

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<kontakty>
  <kontakt>
     <jmeno>Pavel Matlásek</jmeno>
     <email>matlas@nekde.com</email>
  </kontakt>
  <kontakt>
     <jmeno>Martina Králová</jmeno>
     <email>martinka@emmail.cz</email>
  </kontakt>
</kontakty>
```

Do souboru aspx umístíme následující tag:

```
...
<form runat="server">
  <asp:DataGrid id="MyDataGrid" runat="server" />
</form>
...
```

DataGrid je serverový ovládací prvek datové mřížky, hodí se tudíž k vypsání 2D dat. Zbývá už jen doplnit kousek kódu do události `Page_Load` (ta se automaticky spouští při načtení stránky).

```
private void Page_Load(object sender, EventArgs e)
{
  DataSet ds = new DataSet();
  ds.ReadXml(Server.MapPath("Kontakty.xml"));
  MyDataGrid.DataSource = ds;
  MyDataGrid.DataBind();
}
```

Nejdříve je vytvořen objekt DataSet. To je v ADO.NET naprosto klíčový objekt, který si lze představit jako in-memory databázi. Může obsahovat tabulky, vazby mezi tabulkami, integritní omezení apod. (více se lze dočíst v dokumentaci, [25]). Pomocí funkce `ReadXml` do něj načteme celý soubor, datové mřížce určíme, že má jako zdroj dat používat právě naplněný DataSet a poslední příkaz už je jen příkazem vynucujícím svázání dat definované o řádek výše. Nikde se tedy nevyskytuje žádný kód, který by iteroval přes všechny řádky a sloupce výsledku.

Druhá verze tohoto příkladu bude využívat Microsoft SQL Server 2000 (tento databázový server je samozřejmě upřednostňovaný, nicméně ADO.NET podporuje širokou škálu jiných databází, např. Oracle, MySQL a mnoho dalších, pro které existuje nativní nebo OLE DB datový poskytovatel). ADO.NET sice stále podporuje tzv. data readers, což jsou v podstatě serverové kurzory podobné těm v PHP, náš příklad však využije novější a pokročilejší techniku využívající tzv. *datové adaptéry* (*data adapters*). Ti jsou prostředníkem mezi databází a objektem DataSet a velmi zjednodušeně řečeno fungují tak, že je DataSet využívá k provádění nízkoúrovňových příkazů. Např. při aktualizaci dat v databázi DataSet pouze požádá o aktualizaci a je na datovém adaptéru, aby vykonal patřičný SQL příkaz.

Kód stránky samotné (soubor aspx) zůstává nezměněn, mění se pouze kód metody `Page_Load`:

```
SqlDataAdapter adapter = new SqlDataAdapter ("select * from kontakt",
    "server=localhost;database=test;uid=uzivatel;pwd=neprustrelneheslo");
DataSet ds = new DataSet();
adapter.Fill(ds);
MyDataGrid.DataSource = ds;
MyDataGrid.DataBind();
```

Tato verze se tedy od té předchozí liší jen velmi málo, ačkoliv předtím byla data získávána ze souboru na disku, zatímco teď z relační databáze. Tolik malá ochutnávka toho, jakou abstrakci ADO.NET nabízí a jakým způsobem jsou data prezentována uživateli (žádné iterace, používá se model vázání dat k ovládacím prvkům).

ADO.NET je poměrně šikovným API, jehož podstatnou vlastností je mimo jiné podpora přístupu k datům v nespojitém prostředí. Nejnázorněji to je vidět při tvorbě obyčejného tlustého klienta (i tam se používají stejné objekty DataAdapter a DataSet), kde dojde k jednorázovému načtení dat do objektu DataSet, uživatel se pak klidně může odpojit, hodinu provádět úpravu dat (přidávání, mazání nebo modifikaci záznamů) a nakonec vše zase jednorázově aktualizovat v databázi. V ASP.NET to už tak názorné není, nicméně i zde tento princip funguje. Už na naší krátké ukázce je vidět, že komunikace s databází se odehraje v rámci jednoho příkazu: `adapter.Fill(ds)`. Ten spojení otevře, získá data a bez okolků zase uzavře. Až potom je s daty dále manipulováno.

Obě technologie tedy podporují připojení k mnoha zdrojům dat. PHP tak činí svým typicky přímočarým způsobem, kdy jsou pro jednotlivé databáze napsány konkrétní knihovny. ASP.NET k problematice přistupuje komplexněji za využití technologie ADO.NET, která se jednak snaží lépe podporovat nespojité prostředí Internetu, jednak chce poskytnout konzistentní přístup k fyzicky odlišným datovým zdrojům.

### 10.3.2 Podpora XML

XML (eXtensible Markup Language) se během posledních let stal všeobecně rozšířeným a uznávaným standardem pro výměnu informací a komunikaci mezi různými systémy. Každá moderní technologie by tedy měla obsahovat kvalitní podporu tohoto formátu.

Celý .NET je na distribuované systémy zaměřen, není tedy divu, že na XML nebylo zapomenuto. Právě naopak, celý jmenný prostor `System.Xml` obsahuje velké množství tříd pro poměrně komfortní práci s XML. Podporovány jsou standardy XML 1.0 včetně DTD, XML jmenné prostory, XML schémata, výrazy XPath, XSLT transformace a DOM level 1 a 2 ([26]).

Podpora XML však nekončí u tříd jmenného prostoru `System.Xml`, ale jak bylo vidět v předchozí kapitolce, lze například velmi jednoduše načíst XML soubor do DataSetu pomocí jeho funkce `ReadXml`.

Také pokud se podíváme i na jiné věci, než na konkrétní jmenné prostory a třídy, lze říci, že celý .NET Framework se s XML velmi kamarádí. Příkladem je integrace XML webových služeb, zápis komentářů v XML, různé konfigurační soubory v XML apod.

PHP ve verzi 5 výrazně předělalo podporu XML, možná by se dalo říct, že XML konečně začalo plně podporovat. Verze 4 uměla ke XML dokumentům přistupovat pouze přes rozhraní SAX (Simple API for XML), což je sice efektivní, ale málo komfortní. Podpora DOM zůstala až do posledního vydání čtyřkové verze označena za „experimentální“ a nutno říci, že právem, protože několikrát změnila své API a obsahovala chyby.

PHP 5 staví na úplně jiné knihovně, a sice na libxml2 (pocházející z projektu GNOME). Ta kromě rozhraní SAX konečně plně podporuje DOM podle standardu W3C. Dále je podporován dotazovací jazyk XPath, validování proti DTD, XML Schema a RelaxNG. Posledním důležitým podporovaným standardem jsou XSLT transformace.

Pro zjednodušení práce vývojáře byla do nové verze PHP přidána šikovná knihovna SimpleXML. Příkladem pro vypsání ukázkového souboru z předchozí kapitoly je následující kód:

```
$sxe = simplexml_load_file("Kontakty.xml");
foreach ($sxe->kontakt as $kontakt) {
  echo $kontakt->jmeno;
  echo $kontakt->email;
}
```

Zde PHP jednoznačně boduje – jednodušeji to už snad ani zapsáno být nemůže. Rozšíření SimpleXML navíc podporuje XPath, takže by nebyl problém napsat `foreach($sxe->xpath(,kontakty/kontakt‘) as $kontakt)` atd. SimpleXML je navíc plně kompatibilní s DOM, takže mezi těmito dvěma „pohledy“

lze jednoduše „přepínat“. Jelikož je obojí postaveno na libxml2, nebude citelná ani ztráta výkonnosti, protože se struktura nemusí interně kopírovat.[16]

Můžeme tedy shrnout, že ASP.NET i PHP podporují XML na velmi dobré úrovni. PHP nabízí jednoduché rozhraní v podobně rozšíření SimpleXML, ASP.NET zase vyniká zapracováním XML do celé architektury .NET Frameworku.

### 10.3.3 Řízení stavů

Protokol HTTP byl navržen jako bezstavový, což je pro reálné webové aplikace problém. Typickým příkladem je třeba e-obchod, který umožňuje uživateli během prohlížení katalogu rovnou přidávat jednotlivé položky do virtuálního nákupního košíku. Jak má ale aplikace vědět, že požadavek na stránku nabízející videokazety zaslal stejný uživatel, který si na předchozí stránce do košíku přidal videorekordér?

Řešení je několik. V zásadě je lze rozdělit na ta, která problém řeší na straně klienta, a na ta, která problém řeší na straně serveru.

Řešení postavená především na klientovi jsou obecně cookies (cookie je textová informace posílaná v hlavičce HTTP, která může být uložena na straně klienta a při příštím přístupu na web opět odeslána serveru), skrytá formulářová pole a query string (část URL za otazníkem). To jsou však metody nesouvisející ani tak s PHP nebo s ASP.NET, jako spíš s protokolem HTTP, jazykem HTML a zdatností prohlížečů. PHP i ASP.NET tyto metody samozřejmě podporují, ale nejedná se o „jejich“ řešení.

ASP.NET má v této oblasti jednu „vymoženost“ navíc – serverové ovládací prvky totiž samy o sobě dokáží udržet svůj stav mezi jednotlivými požadavky, přičemž používají řešení postavené na klientovi. V kapitole 10.1.3.2 (strana 10) jsme se setkali se skrytým polem `__VIEWSTATE` s jediným atributem `value`, který právě obsahuje kódovaný stav všech ovládacích prvků formuláře. Jedná se tedy o elegantní využití známé metody vylepšené o dvě zásadní věci: zaprvé je stav přenášen relativně bezpečně (hodnota je kódována, komprimována a vše uložené je v Unicodu) a zadruhé je toto skryté pole generováno automaticky, což usnadňuje vývoj. Viewstate si ale stále zachovává hlavní nevýhodu skrytých polí – lze jej používat jen pro relativně malá data.

Nejrozšířenějším řešením na straně serveru jsou tzv. *sessions*, *relace*. Jejich obecný princip je takový, že danému uživateli je přidělen jednoznačný identifikátor zvaný session id. Ten je uložen buďto v cookie, je propagován v URL nebo ve skrytém poli formuláře. Informace jsou tedy ukládány na serveru, nikoliv v počítači uživatele, a s každým požadavkem klienta se přenáší pouze session id[17].

Relace v PHP fungují prakticky přesně podle zde uvedených obecných pravidel. Pokud jsou v klientském prohlížeči povoleny cookies, je session id přenášeno jimi (jde o obecně preferovaný přístup, protože je nejbezpečnější, viz [SessFix]). Pokud cookies povoleny nejsou, je session id zakomponováno do query stringu (např. http://website.stat/page.php?SID=1234). Pomocí konfiguračních direktiv lze toto chování upravit, např. lze vynutit pouze využívání cookies, takže pokud je prohlížeč nepodporuje (respektive je má zakázané), nebudou relace fungovat.

Velmi podobně fungují relace v ASP.NET ([28]). Dokonce syntaxe je s PHP dost podobná:

| PHP | C# |
|---|---|
| `$_SESSION['promenna'] = "hodnota";`<br>`$obsah = $_SESSION['promenna'];` | `Session["promenna"] = "hodnota";`<br>`string obsah;`<br>`if (Session["promenna"] != null)`<br>`  obsah = (string) Session["promenna"];` |

V obou jazycích se tedy jedná o speciální pole. PHP jako obvykle umožňuje kratší zápis (v jazyku C# je totiž `Session[„promenna“]` objekt, je tedy nejprve nutno otestovat, zda tento objekt není `null` a až potom získat hodnotu, kterou je navíc potřeba přetypovat).

[16] Pěkným článkem o podpoře XML v PHP 5, ze kterého byla přejata většina zde uvedených informací, je [27].
[17] I tento identifikátor je však stále lákavým terčem útočníků. Otázka bezpečnosti sessions je nad rámec této práce, zajímavé čtení lze nalézt v dokumentu [SessFix].

ASP.NET oproti PHP nabízí některé užitečné vlastnosti, co se relací týče. Tou první je jejich snadná a zároveň poměrně mocná konfigurovatelnost. To však spíše souvisí s celkovým přístupem ke konfiguraci, což bude probráno o něco níže. Druhou výhodou je vestavěná podpora pro ukládání relací do databáze – opět přes jednoduché konfigurační rozhraní (fyzicky se jedná o XML soubor, viz dále). To může být v některých scénářích (zvláště větších projektů) poměrně důležité.

ASP.NET nabízí navíc ještě něco, co se jmenuje *application state*. Jedná se vlastně o globální proměnné celé webové aplikace. Zápis je podobný jako u sessions: `Application["promenna"] = "hodnota"`.

Obě dvě technologie tedy obsahují snadno použitelné prostředky pro překonání problémů nestavovosti protokolu HTTP. Podle mého názoru nabízí ASP.NET o něco lepší komfort, ať už kvůli jednoduchému deklarativnímu nastavení chování relací nebo díky integraci podpory ukládání relací do databází (což přímo souvisí se škálovatelností). Zajímavou pomůckou pro jednodušší případy je rovněž viewstate, čímž ASP.NET celkem šikovně simuluje prostředí vývoje klasických aplikací – nejen že serverové prvky reagují na události, ale navíc si automaticky pamatují svůj stav.

## 10.3.4 Webové služby

### 10.3.4.1 Ujasnění

V úvodu (konkrétně v kapitole 2) bylo uvedeno, že se tato práce věnuje aplikacím, které staví na modelu klient/server, koncovým klientem je webový prohlížeč, serverem je webový server a pro komunikaci mezi nimi je použit protokol HTTP. Podle pracovní definice tedy webová služba není webovou aplikací, protože jejím koncovým klientem není webový prohlížeč, ale jiná aplikace. To může vyvolat rozpaky nad zařazením této kapitoly. Jejím smyslem však není popis vytváření webových služeb jako samostatných programových celků, ale spíše dostupnost nástrojů pro práci s nimi v rámci nějaké webové aplikace.

### 10.3.4.2 Lehký úvod do webových služeb

Webové služby jsou jasným trendem integrace heterogenních systémů. Zjednodušeně se jedná o nějakou funkčnost poskytovanou na určité URL adrese – oproti normálním objektům se tedy liší v tom, že zapouzdřují jak data a funkce, tak i využité výpočetní zdroje (funkce je prováděna na hardwaru příslušného poskytovatele webové služby). Dva základní protokoly, na kterých webové služby stojí, jsou HTTP a SOAP (použití těchto protokolů je nejběžnější, není však nezbytně nutné). SOAP je zkratkou pro Simple Object Access Protocol a v podstatě se jedná o XML formát, který definuje, jak mají vypadat zprávy, které si systémy vyměňují, jak mají být kódovány standardní i uživatelské datové typy a jak mají vypadat vzdálená volání metod a odpovědi na ně. Dalším důležitým pojmem webových služeb je WSDL (Web Services Description Language), což je opět XML formát, který popisuje, jak službu používat (tedy např. jak se jmenují zveřejněné metody, jaké jsou typy a pořadí argumentů apod.). Existuje několik dalších témat souvisejících s webovými službami, bez nich se však tato práce obejde.

### 10.3.4.3 Webové služby v PHP

PHP 4 nabízelo několik modulů podporujících práci s protokolem SOAP, všechny ale byly vytvořeny v PHP jako takovém s nepříjemnými výkonnostními důsledky. Verze 5 přináší nový modul, který je napsán v jazyku C (je tedy výrazně rychlejší) a je oficiální součástí distribuce.

Server (webovou službu) lze vytvořit pomocí následujícího kódu:

```
<?php
function secti($a, $b) {
  return $a + $b;
}

$server = new SoapServer("kalkulacka.wsdl");
$server->addFunction("secti");
$server->handle();
?>
```

Vytvořili jsme jednoduchou funkci `secti`, která vrací součet předaných argumentů, a tuto funkci jsme zaregistrovali jako server (webovou službu). Konstruktor objektu `SoapServer` přejímá jako parametr

právě WSDL dokument, který popisuje nejen metodu `secti`, ale také webovou službu jako celek – její název, umístění apod. (Výpis příslušného WSDL dokumentu by byl příliš dlouhý, proto je vynechán.)

Klient by potom vypadal nějak následovně:

```
<?php
$client = new SoapClient("URI-k-dokumentu-kalkulacka.wsdl");
$vysledek = $client->secti(2,3);
?>
```

V PHP lze tedy můžeme poměrně snadno tvořit webové služby a také je z kódu PHP konzumovat, bohužel však obtížnou částí zůstává příprava WSDL dokumentu. Navíc, informace o webové službě jsou duplikovány – jednak zde máme kód jako takový, jednak WSDL dokument. Po některých změnách kódu je tedy nutno ručně upravit i WSDL dokument. Přesto jsou webové služby v PHP docela dobře použitelné.[18]

#### 10.3.4.4 Webové služby v ASP.NET

Jestliže jsem PHP označil za poměrně dobře připravené pro webové služby, o ASP.NET se to takto říci nedá. Tam jsou totiž webové služby integrální součástí celé platformy a .NET Framework pro práci s nimi nabízí téměř dokonalý komfort.

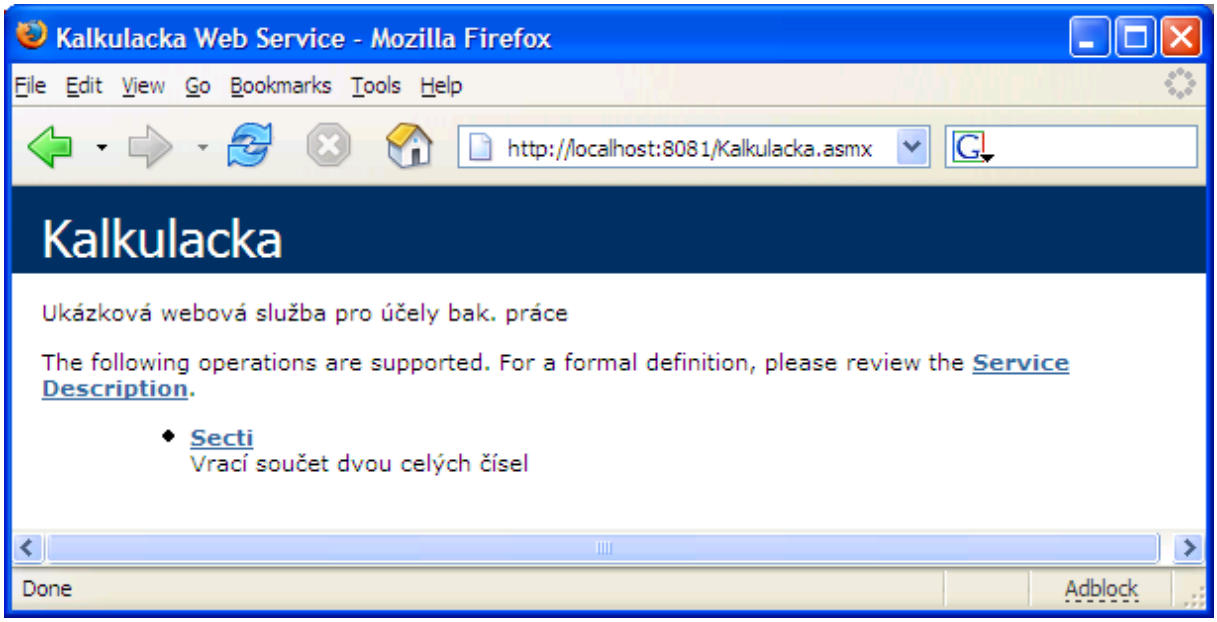
Server vypadá následovně:

```
<%@ WebService language="C#" class="Kalkulacka" %>

using System;
using System.Web.Services;
using System.Xml.Serialization;

[WebService(Namespace="http://example.bak.prace/websluzby",
            Name="Sčítací webová služba",
            Description="Ukázková webová služba pro účely bak. práce")]
public class Kalkulacka {

    [WebMethod(Description="Vrací součet dvou celých čísel")]
    public int Secti(int a, int b) {
        return a + b;
    }
}
```

Jedná se o soubor s příponou asmx – tato přípona je pro webové služby vyhrazena. Soubor vypadá celkem nepřehledně, celý vykonávaný kód je však velmi krátký (je zvýrazněn tučně). Zajímavé jsou oblasti uzavřené v hranatých závorkách. To jsou tzv. *atributy*, další syntaktická vlastnost jazyka C# (respektive všech .NET jazyků). Zjednodušeně řečeno to jsou jakési dodatečné informace o popisované entitě – lze odtud řídit chování vláken (threadů), verze, ukládat uživatelské informace jako třeba jméno autora či seznam změn apod. V případě webových služeb jsou atributy elegantně využívány pro jejich deklarativní popis, což je v případě PHP nutno provádět ručně ve WSDL dokumentu.

Příjemnou vlastností ASP.NET je, že pokud URL výše uvedené webové služby zadáme do adresního řádku prohlížeče, bude nám automaticky vytvořen jakýsi webový klient:

[18] Pěkný článek o podpoře SOAP v PHP 5 lze nalézt v dokumentu [PHP5SOAP], ze kterého byly přejaty zde uvedené informace.




**Obrázek 3 – Automaticky vytvořený webový klient**

Po kliknutí na `Secti` lze snadno otestovat, zda služba funguje tak, jak má. Možná je trochu zarážející, že služba vůbec funguje, když jsme nikde nedefinovali WSDL dokument. Není to však proto, že by Microsoft webové služby řešil nějak „po svém“, ale proto, že ASP.NET vše vygenerovalo zcela automaticky. WSDL dokument můžeme zobrazit, pokud do adresního řádku za stávající URL dopíšeme ?wsdl, tedy např. http://localhost:8081/Kalkulacka.asmx?wsdl. (WSDL soubor je samozřejmě cachován, aby nemusel být pořád dokola vytvářen.)

Velmi elegantní a se zbytkem ASP.NET konzistentní je potom vytváření klienta. Nejdříve je však potřeba seznámit se s pojmem *proxy*. Ten nemá nic společného s proxy serverem, název je odvozen od zástupných objektů v technologii COM. Tam se jedná o objekty umístěné na straně klienta, které vypadají přesně stejně jako objekty serveru, akorát že nevykonávají žádnou funkčnost, pouze zprostředkovávají komunikaci. Stejně tak jsou proxy objekty webových služeb uměle vygenerované objekty, které se tváří přesně tak, jako cílové objekty. .NET Framework SDK nabízí nástroj wsdl.exe, který umí z WSDL dokumentu vygenerovat zdrojový kód proxy třídy, Visual Studio tento nástroj integruje.

Je-li vše připraveno, volání služby vypadá následovně:

```
Kalkulacka kalk = new Kalkulacka();
int vysledek = kalk.Secti(2,3);
```

K webovým službám lze tedy přistupovat naprosto stejně jako k jakékoliv jiné metodě běžného objektu.[19]

#### 10.3.4.5 Shrnutí

Podle mého názoru je podpora webových služeb v ASP.NET daleko lepší než v PHP. Tam se sice pomocí nového rozšíření lze se „zbytkem světa“ také relativně dobře dorozumět, ale v ASP.NET je integrace dotažena daleko dál. Model zástupných objektů (stubů, proxy, jakkoli se to nazývá) je rozšířen a akceptován v rozličných technologiích (COM, EJB, CORBA, .NET) a kromě pohodlí poskytovaného platformou .NET umožňuje také zcela čitelný a konzistentní zápis (se vším všudy – není např. problém ve webové službě vyhazovat výjimku a v klientovi ji známým způsobem ošetřovat). Problém není ani s relativně složitými datovými typy, jako např. s DataSetem (ona „in-memory“ databáze zmiňovaná výše) – i ty lze velmi jednoduše předávat jako parametry metod nebo je přijímat jako návratové hodnoty. Webové služby jsou navíc plně podporovány ve Visual Studiu, které dělá jejich vývoj ještě snadnější.

[19] Komplexnější příklad je třeba v [29].

## 10.4 Další vlastnosti technologií

### 10.4.1 Konfigurace

Důležitou součástí každé aplikace je možnost její konfigurace, ať už se jedná o nastavování systémových nebo uživatelských záležitostí. Hlavním cílem je snadná udržovatelnost – při dobrém návrhu aplikace lze i poměrně velké zásahy aplikovat bez změny kousku kódu pouze tím, že deklarativně změníme konfigurační soubor (touto velkou změnou může být třeba změna databáze, v ideálním případě nám ale bude stačit pouze změnit connection string a přihlašovací údaje).

Začneme s ASP.NET, protože to má podle mého názoru konfigurovatelnost vyřešenu velmi dobře. Vše je ukládáno v XML souborech, přičemž ASP.NET zná dva druhy souborů: Machine.config, nastavení pro celý web server, a Web.config, což je nastavení pro určitou část aplikace.

Soubor Web.config je platný pro skripty v aktuálním adresáři a pro všechny adresáře podřízené. Pokud se v nějakém adresáři vyskytne další soubor Web.config, zdědí všechna nastavení od svého rodiče a má možnost další vlastnosti přidat nebo stávající modifikovat, překrýt. Protože by bylo poměrně složité při každém požadavku rekonstruovat celý hierarchický strom souborů Web.config, aby se zjistilo, jaké vlastně nastavení platí pro aktuální stránku, je po prvním vyřešení vše cachováno a při příštím požadavku ihned k dispozici. Dojde-li však ke změně konfigurace, ASP.NET znovu celou hierarchii zkontroluje a aplikuje změny (na závěr je samozřejmě vše opět uloženo do cache). Konfigurační soubory jsou pochopitelně rozšiřitelné (XML = eXtensible ML) a přes URL nedosažitelné (požadavek na http://muj.web/aplikace/Web.config vyvolá HTTP chybu 403 – forbidden, [30]).

Přes soubor Web.config toho lze nastavit už ve výchozím stavu opravdu hodně (možná proto se na internetu objevilo několik specializovaných editorů). Namátkou je možno nastavit autentizaci/autorizaci, řízení sessions, nastavení bezpečnostní politiky, webových služeb, řízení chyb atd. atd.

PHP pro konfigurovatelnost svých aplikací mnoho nenabízí. Vše v podstatě končí souborem php.ini, ve kterém se nacházejí všechna dostupná nastavení. Jejich seznam však není rozšiřitelný a kaskádový princip souborů Web.config také nefunguje. V této oblasti mi tedy připadá ASP.NET jasně lepší.

### 10.4.2 Výkonnost

S rostoucím počtem uživatelů webové aplikace se stává otázka výkonnosti čím dál důležitější, protože uživatelé velmi neradi čekají na zobrazení stránky. V extrémním případě se tak výkonnost při výběru technologie může stát kriteriem číslo jedna.

Původně jsem myslel, že zde uvedu jakýsi „benchmark“ obou technologií, aby bylo možno jednoduše posoudit, která z nich je rychlejší. Bylo by však nutno vyřešit několik zásadních otázek. První by byla, v jakém prostředí má skript běžet, jestli ve stejném (tedy asi Windows + IIS) nebo v nativním (pro PHP tedy v prostředí Linux + Apache, pro ASP.NET v prostředí Windows + IIS), případně nakolik by byly výsledky ovlivněny rozdílností operačních systémů namísto odlišností ve webových technologiích samotných. Velkým problémem je také, co vlastně měřit. „Skript“ je obecný pojem a může se jednat jak o téměř prázdný zdrojový dokument, tak o jednoduché „Ahoj světe“, o matematický výpočet, o složité dotazování na databázi nebo o generování náhledů obrázků. Není potom vyloučeno, že některou oblast bude lépe zvládat jedna technologie, zatímco jinou zase ta druhá. Jak potom dojít k objektivnímu závěru?

Kvalitní benchmark by tedy jistě byl zajímavým námětem pro jinou práci, bohužel se mi žádnou takovou nepodařilo najít. Faktem zůstává, že ASP.NET je velmi výkonnou technologií (ve svých počátcích .NET velmi drtivě porážel J2EE, viz např. [31]; Java se samozřejmě od té doby výrazně zlepšila). Namísto konkrétních čísel se však podíváme na to, jaké prostředky jednotlivé technologie pro podporu výkonnosti nabízejí.

První otázkou je, zda je kód kompilovaný, protože ten se zákonitě vykonává rychleji než kód interpretovaný[20]. ASP.NET, jako celý .NET, skripty kompiluje do MSIL (Microsoft Intermediate Language), což je obdoba bytecode v Javě. Tento pseudokód je potom do strojového kódu kompilován metodou Just-In-

[20] Interpretovaný kód, jak název napovídá, je potřeba nejdříve interpretovat, přeložit do strojového kódu a teprve potom vykonat. Kompilovaný kód naproti tomu už interpretovaný je.

Time, tedy při prvním požadavku. Vzniká tak kompilovaný skript, který následně ASP.NET cachuje, aby překlad z MSIL nemuselo provádět při každém požadavku. Na ASP.NET je pěkně vidět rozdíl mezi interpretovaným a kompilovaným kódem – při prvním požadavku na složitou stránku trvá její vygenerování na běžném desktopovém počítači několik sekund (MSIL je potřeba interpretovat), zatímco druhý požadavek je vyřízen prakticky okamžitě. To je dáno také tím, že JIT kompilace se snaží o co nejoptimalizovanější překlad MSIL do strojového kódu, což má pozitivní výkonnostní důsledky. Dostatečným důkazem právě uvedeného tvrzení by mělo být to, že nevím o žádném produktu, který by generovaný kód nějakým způsobem dále optimalizoval (ASPAccelerator.NET využívá jiné techniky než optimalizaci generovaného kódu).

PHP je v drtivé většině zdrojů označováno za interpretované, není to však tak úplně pravda. Také PHP skripty jsou nejdříve kompilovány do něčeho, co se jmenuje *opcode* a v podstatě se jedná o ekvivalent MSIL. Tento opcode je potom **interpretován** pomocí Zend Engine, což je v podstatě srdce i mozek PHP. Jedná se tedy o jakýsi hybrid mezi kompilovaným a interpretovaným přístupem.

Problémem PHP je, že nedochází k žádné optimalizaci kódu ani ke cachování strojových kódů. Naštěstí lze na internetu sehnat několik nástrojů zdarma (alternativní komerční řešení nabízí firma Zend).

Dalším účinným prostředkem řešení výkonností otázky je cachování výstupu (output caching). Také v této oblasti nabízí ASP.NET velmi účinné řešení – poměrně jednoduše lze řídit cachování celé stránky nebo jejích fragmentů, navíc lze řídit dobu vypršení cachovaného výstupu nebo držet více verzí stránky podle toho, jaký bude uživatelský vstup. Vše je jednoduše deklarovatelné v záhlaví aspx souboru přidáním direktivy `@OutputCache`, např. `<%@OutputCache Duration="3600" VaryByParam="mesto" %>` určuje, že daná stránka bude cachována po dobu jedné hodiny a že v cachi budou ukládány různé verze stránek pro jednotlivá města (příkladem je stránka doručující mapy na základě zadaného města).

PHP bohužel opět nic podobného nenabízí. Aby nedošlo k mýlce: PHP od verze 4 obsahuje funkce pro tzv. output buffering, lze tedy napsat vlastní framework pro podporu cachování výstupu. Tento framework však není součástí PHP jako takového, vývojář má pouze možnost si ho vytvořit[21], navíc ne s takovými možnostmi, jaké nabízí ASP.NET. Systémovým řešením je komerční produkt Zend Performance Suite, který slibuje zvýšení výkonnosti až 25krát. Co však konkurence nabízí zcela zdarma, je v případě PHP nutno pořídit za tučnou částku 1875 dolarů.

Popravdě řečeno mi není příliš jasné, proč jsou v otázce výkonnosti obě technologie tolik rozdílné, když je to tolik důležitá oblast. ASP.NET je velmi výkonným řešením, což se o PHP automaticky říct nedá, naopak mu chybí základní vlastnosti implikující výkonnost – vestavěná optimalizace kódu, cachování kompilovaných skriptů a cachování výstupu. Je otázkou, zda je pouze náhoda, že firma Zend vyvíjející jádro PHP současně nabízí poměrně drahá řešení zde uvedených nedostatků.

### 10.4.3 Podpora platforem

Internet je heterogenním prostředím s velkým množstvím různých platforem, proto je dost zajímavé, nakolik jsou ASP.NET a PHP platformově nezávislé. Slovo „platforma" přitom v tomto kontextu vyjadřuje jakési běhové prostředí, je to tedy třeba operační systém, ale i webový server nebo kombinace obojího (.NET Framework i PHP jsou samy o sobě platformami pro běh webových aplikací, v tomto kontextu je však za platformy považovat nebudeme).

Velkou předností PHP je jeho platformová nezávislost, rozběhneme ho prakticky všude. Má širokou podporu jak pro operační systémy (Unix, Linux, Windows, MacOS) tak pro různé webové servery (Apache, IIS, Xitami a další). Jakýmsi domovským prostředím je Linux v kombinace s Apachem, žádný problém však nedělá ani běh na strojích s Windows (firma Zend navíc nabízí produkt WinEnabler, který slibuje zvýšenou stabilitu na tomto operačním systému). Často si lze vybrat, zda PHP poběží jako CGI (nedoporučeníhodné) nebo jako modul web serveru (výkonnější řešení).

ASP.NET jako technologie Microsoftu je samozřejmě „doma" na platformě Windows + IIS. Podpora ostatních platforem je problematická a bývá častým dotazem v různých diskuzích nebo na závěr prezentací. Filištínskou odpovědí bývá to, že Microsoft sice přímo jiné platformy nepodporuje, nicméně jim dává k dispozici dokumentaci a velká část jeho technologií je standardizovaná skrze nezávislou autoritu ECMA.

[21] Docela pěkný článek je na http://www.ilovejackdaniels.com/php/caching-output-in-php/.

Kompletně jsou navíc přístupné zdrojové kódy pro kompilátor jazyka C# a pro Common Language Infrastructure (CLI).

Objevují se tedy porty na ostatní platformy. Nejznámějším projektem je Mono, jehož domovská stránka je http://www.go-mono.org/. Jedná se o Open Source projekt sponzorovaný firmou Novell, který má velký počet příznivců a jako svou část obsahuje i podporu ASP.NET. Veřejně dostupné aplikace jako IBuySpy (ukázková implementace dostupná už od alfa verzí ASP.NET) nebo nGallery (obrázková galerie) pod Monem fungují. Dostupný je také mód pro Apache, takže ASP.NET není nutně vázané na IIS. Požadavky ASP.NET na webový server jsou navíc dobře zdokumentovány a např. vývojový nástroj Web Matrix (freeware vývojové prostředí od Microsoftu) obsahuje plně funkční webový server komunikující s ASP.NET (celá velikost vývojového prostředí spolu s webovým serverem je navíc jen 1,3 MB).

Existují i jiné projekty ala Mono ([32]), přesto např. pan Michael Juřek na přednášce pro studenty VŠE vyjádřil určitou skepsi. Podle jeho názoru jsou tyto projekty nutně technologicky pozadu (nejdříve musejí počkat, co Microsoft do příští verze přidá, a teprve potom mohou začít s vlastní implementací). První verze Mona byla např. vydána 30. června 2004, přičemž kompatibilita stále není dokonalá. Jenže za pár měsíců zde budeme mít druhou verzi .NET Frameworku a Mono může znovu začít dohánět.

V podpoře nejrůznějších platforem tedy zcela jednoznačně boduje PHP, které je skutečně „platform-independent". Microsoft naproti tomu nijak nezakrývá, že cílovou platformou pro ASP.NET je Windows + IIS, nicméně se portům jinam nebrání. Celý .NET Framework je dobře dokumentovaný, standardizovaný a k některým věcem existují přímo zdrojové kódy.

Poznámka na závěr: podle mého osobního názoru otázka platformy sama o sobě není nejdůležitější, ačkoliv je dnes velmi moderní být zarytým příznivcem či odpůrcem platformy xxx nebo platformy yyy. Určité porty .NETu existují, rozhodne-li se tedy Microsoft zítra ASP.NET zpoplatnit jedním dolarem za jeden obsloužený požadavek, alternativa existuje. Navíc lze vždy přejít ke konkurenci. Myslím, že důležitější než jméno platformy jsou její další parametry, např. cena, bezpečnost apod. Pokud by se ukázalo, že nejlevnější, nejbezpečnější a kdo ví co ještě je BestOS v kombinaci s BestWebServerem, nebude vadit, že nějaká technologie poběží pouze tam.

### 10.4.4 Bezpečnost

Bezpečnost je jednoznačným trendem dnešní doby. Dříve platilo „nejdřív funkčnost, potom bezpečnost", dnes se razí heslo „funkčnost s bezpečností ruku v ruce".

Bezpečnost technologie má podle mého názoru dvě složky. Tou první je bezpečnost technologie jako takové, tou druhou je potom bezpečnost řešení jako celku (tj. celé platformy včetně operačního systému a webového serveru).

Co se první části týče, obě technologie samozřejmě dělají, co mohou. Obě podporují autentizaci / autorizaci (ASP.NET opět poměrně elegantně pomocí souborů Machine.config nebo Web.config, tedy deklarativně), nabízejí funkce pro bezpečné zapisování řetězců do databází (PHP např. obsahuje tzv. magic quotes, což je sada funkcí a konfiguračních direktiv, které umožňují automaticky řešit časté problémy s podvrženými SQL vstupy) atd. atd. Zákonitě však zůstanou problémy, které nelze na aplikační úrovni řešit – např. problém bezpečnosti cookies, sessions apod. Zde musejí ke slovu přijít specializované technologie typu SSL.

Druhou bezpečnostní otázkou je bezpečnost platformy jako celku. Co se týče operačního systému, výrazně lepší reputaci mají systémy Unix a GNU/Linux, zatímco Windows svou bezpečností rozhodně neprosluly. Microsoft sice v rámci své politiky „trustworthy computing" dělá co může, nálepku bezpečného systému ale asi jen tak nezíská.

Podobně je to s webovými servery. Apache je uznáván jako bezpečný a kvalitní server, zatímco IIS za bezpečný považován není. Zde bych už pochyboval, zda je to skutečnost nebo spíše mýtus. Uznávaným bezpečnostním webem je securityfocus.com, který pro IIS 6.0 registruje jedinou nalezenou chybu (v modulu pro webovou administraci, který není defaultně instalován), zatímco pro Apache 2.0 27 chyb

(včetně přetečení zásobníku nebo DOS).[22] Jiný zdroj, [33], dospívá ke třem chybám IIS – rovněž tedy velmi slušné číslo.

Důležitý je však rozdíl v opravách chyb. Zatímco v Open Source řešeních může být chyba ihned po jejím zveřejnění opravena, a to kýmkoliv, Microsoft vždy vydává opravy s určitým zpožděním. Další výhodou otevřených řešení je, že jsou slabá místa odhalována daleko dříve, protože je dostupný zdrojový kód. Bezpečnostní předpoklady mají tedy výrazně lepší Linux nebo Apache, ačkoliv se třeba aktuálně IIS 6 ukazuje jako bezpečný server.

Na webu securityfocus.com jsem rovnou zkontroloval také PHP a ASP.NET. PHP verze 5 nemělo ani jeden záznam, zatímco ASP.NET verze 1.1 tři[23]. Jelikož bylo PHP vydáno teprve před několika dny, není tento údaj příliš informativní, tři chyby ASP.NET je podle mého názoru velmi slušný výsledek.

V současnosti se tedy obě technologie zdají být dostatečně bezpečné. O PHP v kombinaci s Linuxem a Apachem nemůže být pochyb (tato kombinace je prověřena miliony instalací a bezpečnost je navíc podtržena dostupností zdrojových kódů k celé trojici produktů), Microsoft se poslední dobou také velmi snaží a ASP.NET v kombinaci s Windows 2003 Serverem a IIS 6 se také zdá být celkem bezpečným řešením. Stále však platí, že Open Source řešení jsou obecně bezpečnější.

### 10.4.5 Rozšiřitelnost

PHP i ASP.NET nabízejí velké množství vestavěných funkcí pro různé oblasti, samozřejmě se však nelze vyhnout situacím, kdy něco bude chybět. Pro PHP i ASP.NET je dostupné velké množství skriptů, respektive komponent. Přístup obou technologií k rozšiřitelnosti je však jiný.

PHP lze rozšiřovat v zásadě dvěma způsoby. Tím prvním je napsání skriptu přímo v PHP, zatímco druhou možností je vytvoření nového modulu (nejčastěji v jazyku C). Oba přístupy však mají vážné nevýhody. Skript v PHP bude mít bohužel všechny výkonnostní problémy, kterými PHP trpí, modul v C je zase obtížné vytvořit.

Zcela v kontrastu je rozšiřitelnost ASP.NET, která je jednoduše postavena na principech OOP. Velmi jednoduše lze vytvářet nové knihovny, které budou přístupné naprosto stejně jako knihovny standardní, navíc lze vytvářet vlastní serverové ovládací prvky. Ty lze potom například zakomponovat do palety nástrojů Visual Studia, přístup k nim je tedy naprosto totožný jako k prvkům vestavěným.

### 10.4.6 Cena

Nyní přichází to, co je skoro vždy nejdůležitější: cena. První dobrou zprávou je, že nominálně jsou obě technologie zdarma, tj. že je lze zadarmo stáhnout z internetu, nainstalovat a začít používat. Existuje tu drobný rozdíl: zatímco PHP nerozlišuje něco jako runtime a vývojové prostředí (prostě lze PHP nainstalovat a nakonfigurovat, otevřít Notepad, napsat skript a za chvilku se radovat, jak to pěkně funguje), ASP.NET má dvojakou podobu. Je sice součástí .NET Frameworku, aby však bylo možno aplikace plnohodnotně vyvíjet, je potřeba například kompilátor pro jazyk C# (nebo pro jiný použitý jazyk). Tento kompilátor je však součástí tzv. *.NET Framework SDK*. Abychom tedy mohli říct, že je ASP.NET zadarmo, musí být zadarmo jak framework samotný, tak i jeho SDK. To tak naštěstí je.

V ceně se však bude lišit něco, bez čeho se neobejdeme – operační systém. Za Windows budeme muset zaplatit, za GNU/Linux nikoliv. Problematika však není černobílá, protože nominální cena je jen málokdy směrodatná, nehledě na to, že mnoho distribucí Linuxu je rovněž placených. Cena vs. TCO, Windows vs. Linux, komerční produkt vs. Open Source, to jsou dnes ožehavá témata, rád bych se proto vyhnul nějakým kontroverzním závěrům. Myslím si, že v principu je důležité asi tolik: necháme-li stranou projekty typu Mono, za ASP.NET je v současnosti nevyhnutelně nutné zaplatit cenu operačního systému, což v principu není u GNU/Linuxu nutné, i když pokročilejší distribuce placené jsou. Také si myslím, že jednorázový výdaj za licenční poplatek není něco nepřekonatelného – zaplatit několik tisíc jednou za pár let by neměl být problém pro běžného smrtelníka, natož pak pro podnik.

---

[22] Bohužel na výsledky nelze uvádět přímé odkazy. Je tedy nutno přejít na stránku http://www.securityfocus.com/bid/vendor/ a zde zvolit výrobce, produkt a verzi.

[23] Jedná se samozřejmě o bezpečností chyby. Chyb ve funkčnosti („bugs“) PHP 5 za jeden měsíc své existence (od 13. července do 12. srpna 2004) nasbíralo 37 (http://www.php.net/ChangeLog-5.php), ASP.NET jich za rok a 4 měsíce od svého vydání dne 9. dubna 2003 nasbíralo 28 (podle webu http://www.devasp.net/net/search/display/category867.html, který chyby převzal z knowledge base webu MSDN).

Nebezpečí licenčních poplatků je podle mého názoru v tom, že firma (Microsoft) může vydělávat na vynucování upgradu na vyšší verze. Pokud se např. IIS 6 za nějaký čas ukáže jako málo bezpečný nebo jinak nevyhovující a Microsoft novou verzi „přiváže“ na nový operační systém, bude to velký tlak na upgrade systému, což bude znovu stát licenční poplatky. GNU/Linux je zdarma „navždy“ (opět platí jen pro základní distribuce).

Problematika licenčních poplatků je velmi složitá, někdy svým charakterem spíše připomíná politiku než IT, takže to radši nechám být. Osobně sympatizuji s přístupem světa Free software, který nikoho k ničemu nenutí, i když si myslím, že ceny za licence produktů Microsoftu nejsou něco, co by se nedalo přežít.

Co se týče webových serverů Apache a IIS, tak za ty se explicitně žádné licenční poplatky neplatí.

Posledním číslem, které musíme do ceny technologie připočítat, jsou náklady na vývoj. Dokonce si myslím, že právě tato složka je pro stanovení ceny klíčová. Zaměstnavatele stojí jedna hodina vývojáře několik set korun, za pár dní je tedy jeden člověk dražší než celý Windows 2003 Server. Srovnat tedy musíme vývojářskou produktivitu, závěry o ceně je možno činit až pak.

### 10.4.7 Produktivita

Microsoft často jako jednu z hlavních předností ASP.NET uvádí právě „developer productivity“. Slavným a na všech přednáškách všudypřítomným příkladem je vytvoření webové stránky s kalendářem během několik sekund. Potom následuje klik klik klik a kalendář je pěkně zformátovaný a reaguje na událost výběru dne. Na první pohled je to velmi působivé, ale „nejhorší“ na tom je, že tento pocit zůstává, i když to člověk vidí nebo sám dělá podruhé a potřetí. Myslím, že koncept znovupoužitelných serverových ovládacích prvků, které nejsou zapisovány imperativně, ale deklarativně, má skutečně co činit s vývojářskou produktivitou.

ASP.NET rovněž obsahuje některé další zajímavé vlastnosti související s produktivitou – jmenujme např. konzistentní framework, možnost psát kód v libovolném .NET jazyce (vývojář se nemusí učit nový jazyk), oddělení skriptového kódu od HTML kódu, plné nasazení OOP implikující znovupoužitelnost, viewstate umožňující automaticky uchovávat stav ovládacích prvků, snadný a konzistentní způsob konfigurace aplikace pomocí souborů Web.config a Machine.config, integrace webových služeb apod. Ne že bych si myslel, že se v PHP nedá vyvinout aplikace rychle, nicméně ASP.NET na mě působí skutečně tak, že produktivita byla jedním z důležitých cílů návrhu.

Nakonec ještě nemůžu zatajit chloubu Microsoftu – vývojové prostředí Visual Studio .NET. Je otázka, zda je relevantní uvádět jakési IDE v souvislosti s produktivitou technologie jako takové, nicméně dva důvody mám: zaprvé Microsoft rozhodně nevyvíjí ASP.NET a Visual Studio odděleně. Naopak, tyto produkty jsou provázány natolik, že i kdyby byl .NET Framework 2.0 hotov třeba před týdnem, Microsoft bude s jeho uvolněním čekat, dokud nebude odladěno Visual Studio 2005. Souvislost mezi produkty tedy není zanedbatelná. Zadruhé, Visual Studio ovlivňuje produktivitu zcela zásadním způsobem, a to nejen proto, že by bylo tak geniální (což zdaleka není). Je to především proto, že při návrhu ASP.NET byl kladen důraz na to, aby šlo vývoj ASP.NET aplikací snadno v nějakém IDE podporovat (viz koncept serverových ovládacích prvků). Že se tímto IDE stalo zrovna Visual Studio a ne třeba nějaký nástroj od Borlandu, je shoda okolností (nebo dobrým vyznamenáním Microsoftu).

PHP takovéto IDE nemá. Existuje sice několik velmi dobrých, ale podle mých zkušeností nedosahují kvalit Visual Studia.

Vraťme se tedy ještě na chvilku k „ceně“ technologií. Obě jsou nominálně zdarma, běží na webových serverech, které jsou nominálně zdarma, liší se však licenční politikou podporovaných operačních systémů a produktivitou. Zatímco PHP může běžet na téměř libovolné platformě a lze si tedy vybrat takovou, která bude úplně zdarma, u ASP.NET takovou svobodu nemáme, protože je vázáno na operační systém Windows, kde se musí platit (projektům typu Mono sice držím palce, nicméně se stále jedná o neplnohodnotnou alternativu). V mých očích však ASP.NET vše dohání v úspoře prostředků zvýšením produktivity vývojáře, naopak si myslím, že je dlouhodobě levnější. Je to však jen můj názor.

## 10.5 Jiné související záležitosti

### 10.5.1 Vývojová prostředí

Ačkoliv lze obě technologie vyvíjet pouze s kombinací SDK + Notepad (v případě PHP se runtime rovná SDK), je několik dobrých důvodů, proč tak nečinit. Vývojová prostředí (IDE, Integrated Development Environment) totiž výrazně zvyšují efektivitu práce, proto by běžné textové editory měly dostat přednost pouze v případě, že slouží k výukovým účelům (IDE totiž mohou některé důležité koncepty skrývat za roušku automatizace).

Jak již bylo řečeno v úvahách o produktivitě technologií, IDE je velmi účinným prostředkem, jak vývoj urychlit. Zde jsou některé vlastnosti, které by podle mého názoru mělo kvalitní IDE splňovat:

- co nejlepší podpora při psaní kódu
- integrace všech potřebných nástrojů tak, aby se nebylo nutno přepínat jinam
- integrace nápovědy a jiných informačních zdrojů
- ladění
- podpora projektů
- podpora práce v týmu
- WYSIWYG návrh

Podporu při psaní kódu považuji za obecně velmi důležitou. Zvýrazňování syntaxe a automatické odsazování považuji za samozřejmost, součástí každého IDE by měla být i vlastnost automatického dokončování názvů (ať už systémových knihoven nebo uživatelsky definovaných proměnných). Velmi užitečná je rovněž okamžitá kontrola syntaxe tak, aby nebylo nutno chyby odhalovat až během kompilace (v ASP.NET) nebo v horším případě až při spuštění skriptu po HTTP dotazu (v PHP a někdy také v ASP.NET – podle toho, zda je používáno code behind a zda je kód kompilován předem nebo až při požadavku na stránku). Mezi ne tak časté vlastnosti editoru kódu patří šablony, které umožňují snadno vkládat běžné jazykové konstrukty, jako např. for cyklus nebo iteraci přes všechny prvky pole. Ještě o stupínek výše stojí možnost následné manipulace s kódem, tzv. refaktoring. Vrcholem možností editace jsou potom inteligentní značky podobné těm, které se vyskytují v novějších Wordech nebo ve špičkovém IDE pro Javu jménem IntelliJ IDEA (tyto inteligentní značky automaticky nabízejí určité změny kódu, např. pokud stojíme na příkazu, který může generovat výjimku, automatická značka nabízí vygenerování příslušného try/catch bloku).

Integrace potřebných nástrojů je to, co z IDE dělá IDE. Měly by být integrovány všechny nástroje, které jsou například v SDK přístupné pomocí příkazového řádku tak, aby je bylo možno pohodlně používat z vývojového prostředí. Často je k vidění také integrace FTP klienta pro pohodlné editování vzdálených souborů, součástí by měl být také XML editor a nějaký nástroj pro jednoduchou práci s datovými zdroji (v ideálním případě by mělo jít jednoduše do projektu přetáhnout příslušnou tabulku s tím, že se automaticky vygeneruje příslušný přístupový kód).

Důležitou součástí vývoje je také kvalitní dokumentace (které se tato práce věnuje o něco níže). Její integrace do prostředí je pro vývojáře velkým plusem – nejlepší je, pokud se lze z editoru kódu nějak jednoduše dostat k odpovídajícímu tématu dokumentace. Zpříjemněním práce je také integrace nápovědy do automatického dokončování názvů, které nejen že nabízí adekvátní možnosti, ale také v plovoucí nápovědě podává stručnou informaci o tom, co daná funkce/vlastnost dělá.

Podpora ladění (debugging) je důležitou vlastností především u větších projektů, kde nemusí být snadné chybu najít.

Podpora práce v týmu je pak nejčastěji poskytována skrze integrovaného klienta nějakého systému pro správu verzí (např. CVS, Visual SourceSafe atd.).

WYSIWYG návrh je trochu ošemetnou vlastností, kterou ocení jen určitá část uživatelů. Je sice pravda, že vizuální návrh může některé projekty dramaticky urychlit, naproti tomu je někdy lépe přesně vědět, jaký kód se skrývá v pozadí. Určitě je ale dobré, když je WYSIWYG návrh k dispozici.

To tedy byly podle mého názoru podstatné požadavky na IDE, jaké vývojové nástroje jsou ale pro jednotlivé technologie k dispozici?

Pro PHP existuje nepřeberné množství editorů a vzhledem k podstatě PHP, u kterého se běhové prostředí v podstatě rovná SDK, mnohé editory lze téměř označit za IDE. Ať už vezmeme výborný český PSPad, světově uznávaný shareware UltraEdit nebo Open Source jEdit, všechny vynikají především dobrou podporou při psaní kódu. Integrace ostatních nástrojů není u PHP tak důležitá, protože PHP prakticky žádné podpůrné nástroje při vývoji nepotřebuje. Častou a užitečnou pomůckou jsou tak především průzkumníci kódu, kteří umožňují udržet si dobrý přehled nad aktuálním skriptem. Nápovědu rovněž většinou není problém integrovat a nějakým způsobem má management projektů zvládnutý každý editor, většinou však chybí podpora ladění a podpora vizuálního návrhu.

Mezi nejlepší „čistokrevná" IDE patří Zend Studio, což je vývojový nástroj zhruba za 250 dolarů. Podle mého osobního názoru je toto IDE přeci jen pro vývoj vhodnější než univerzální editory. Podpora při psaní kódu je lepší, Zend Studio při psaní hlídá syntaxi, podporuje šablony, umožňuje správu verzí pomocí integrovaného CVS, umí ladit aplikace a hledat potenciální chyby v kódu (vzpomeňme na problém s neinicializovanými proměnnými) a navíc umí pracovat jak v režimu pro PHP 4, tak pro PHP 5. Opět zde však chybí podpora WYSIWYG návrhu, což je ale asi u PHP spíš plusem. Koncepce tohoto jazyka totiž podporu vizuálního návrhu velmi komplikuje, protože např. obrázek může být vložen jak pomocí `<img src="obrazek.jpg" alt="Obrázek" />`, tak pomocí `<?php echo "<" . /* nyní to přijde, vložíme obrázek */ "img " . "s" . "r" . "c" . "=\"obrazek.jpg\" alt=\"Obrázek\"" . " />"`. Pro IDE je tedy velmi obtížné poznat, že má do WYSIWYG editoru vložit nějaký obrázek.

ASP.NET lze rovněž vyvíjet v libovolném editoru, nebude to však stejně komfortní jako vývoj PHP. Pokud tedy budeme trvat na tom, že chceme vyvíjet v prostředí, které je zadarmo, je jasnou volbou Web Matrix. Jedná se o jakési odlehčené IDE od Microsoftu, které má své plusy i mínusy. Největším mínusem je prakticky nulová podpora při ručním psaní kódu. Editor zvýrazňuje syntaxi, ale tím to asi tak končí. Nic jako doplňování názvů nebo něco podobného, což je velká škoda. Rovněž zde chybí podpora projektů. Toto vývojové prostředí má však i plusy. Vizuálně se podobá např. Delphám nebo Visual Studiu, můžeme tedy editovat vlastnosti objektů na záložce Properties, máme k dispozici paletu ovládacích prvků, integrován je kompilátor i nástroj pro jednoduché přidávání datových zdrojů, nechybí ani podpora webových služeb (i když se v podstatě jedná pouze o zprostředkování command-line utility wsdl).

Stěžejní lodí je však samozřejmě Visual Studio .NET, jehož kvality mu nelze upřít. Začněme cenou za Visual C#, což je v podstatě Visual Studio podporující pouze jazyk C# (plné Visual Studio podporuje i další jazyky). Tato cena je dostupných 109 dolarů, což je podstatně méně, než stojí Zend Studio. Podstatnější jsou však vlastnosti. Zjednodušeně se dá říct, že naplňuje všechny dříve uvedené body. Podpora psaní kódu je velmi dobrá – ihned po instalaci se snadno může měřit se Zend Studiem, existují však addiny[24], které dokáží editor kódu ještě více zdokonalit např. o refaktoring nebo o inteligentní značky. Příkladem je například ReSharper od stejné společnosti, která vyvíjí IDE IntelliJ IDEA. Také ve všech dalších bodech je Visual Studio silné. Má však ještě jednu další velmi důležitou přednost – má veřejně přístupné API, takže je snadno rozšiřitelné. Příkladem byl právě uvedený ReSharper, ještě lepším příkladem je však to, že z prostředí Visual Studia lze vyvíjet pro BizTalk server – sám Microsoft tedy rozšiřitelnosti Visual Studia využívá a vývojář se tak nemusí učit ovládat další IDE.

Osobně mi tedy připadá, že pro ASP.NET existuje daleko lepší vývojové prostředí. Zend Studio je sice dobré IDE, ale s Visual Studiem se příliš srovnávat nedá (nehledě na to, že před časem zveřejněná beta verze Visual Studia 2005 už nyní velký rozdíl ještě zvětší). Možná je to proto, že má Microsoft tisíckrát víc vývojářů, možná proto, že Visual Studio se skutečně povedlo, ale nepochybně také proto, že ASP.NET bylo vyvinuto s ohledem na to, aby bylo jeho vývoj možné snadno podporovat ve vývojových prostředích. PHP je poněkud živelně vzniklou technologií, nějaké ohledy na IDE tedy při jeho návrhu patrně brány nebyly.

### 10.5.2 Dokumentace, komunity

Technologie se liší i v přístupu k dokumentaci. PHP nabízí poměrně útlý, ale zato velmi přehledný manuál. Na webu jsou navíc pod každým tématem uvedeny poznámky uživatelů, které informační hodnotu tématu někdy velmi výrazně zvyšují (nebo převyšují). Dokumentaci lze rovněž stáhnout v několika různých formátech. Nyní, na konci srpna 2004, byla konečně dopsána část dokumentující nové funkce páté

[24] Add-in je Microsoftem používaný ekvivalent termínu plugin.

verze (zhruba měsíc k novinkám neexistovala žádná oficiální dokumentace), takže je PHP manuál dobře použitelným zdrojem informací.

ASP.NET jako takové žádnou speciální dokumentaci nemá, na oficiálním webu http://asp.net jsou ke zhlédnutí pouze tutoriály (které nejsou stáhnutelné – Microsoft asi věří, že všichni vývojáři na světě mají trvalé paušální připojení). Velmi rozsáhlá dokumentace celého .NET Frameworku, jehož je ASP.NET součástí, je tak k dispozici na webu MSDN (http://msdn.microsoft.com) nebo v offline verzi po zakoupení Visual Studia („MSDN Library for Visual Studio .NET 2003" na třech CD). Tato dokumentace obsahuje nesmírně mnoho informací (s manuálem PHP se to vůbec nedá srovnat), to je však zároveň jejím problémem. Najít adekvátní téma je podle mé zkušenosti dost obtížné, člověk přitom často nepotřebuje znát detailní podrobnosti, ale spíše potřebuje rychle najít odpovídající informaci. Paradoxně tak často rychleji dá odpověď Google, což je věc, nad kterou by se měli autoři MSDN Library zamyslet.

Zdrojem informací však nejsou pouze oficiální manuály. Velmi mnoho informací se dá získat od komunit vytvořených kolem technologií – typickým příkladem jsou otázky „jak něco udělat", na které v manuálech často nejsou odpovědi, pokud neobsahují speciální how-to kapitoly.

Kolem PHP je vytvořena nesmírně silná komunita. Je to dáno jednak tím, že je PHP Open Source (otevřená řešení jsou na komunitách přímo postavena), jednak jeho rozšířeností. Na internetu lze sehnat ohromné množství hotových skriptů i celých rozsáhlých aplikací (např. několik redakčních systémů), existují Usenet skupiny i specializovaná diskusní fóra (v českém jazyce patří mezi nejlepší diskusní fórum na serveru builder.cz).

Také ASP.NET si během své relativně krátké existence získalo mnoho příznivců a vzniklo mnoho desítek dedikovaných webů. Také ve světě ASP.NET lze sehnat mnoho hotových skriptů, komponent nebo aplikací, přímo z webu Microsoftu jsou dokonce ke stažení tzv. starter kits, což jsou hotové odladěné aplikace, které lze buďto rovnou nasadit (některé jsou užitečné, např. Community Starter Kit nebo Commerce Starter Kit, viz http://asp.net/Default.aspx?tabindex=8&tabid=47), nebo čerpat informace přímo z jejich zdrojových kódů.

Myslím tedy, že co se týče komunit, nemá ASP.NET horší podporu než PHP. Navíc je jakýmsi módním trendem Microsoftu přibližovat členy vývojových týmů běžným vývojářům – přímo na jejich webu lze nalézt hromadu odborných blogů, vývojáři Microsoftu se účastní některých diskusních fór apod.

S otázkou dokumentace souvisejí také komentáře v kódu. PHP jako takové nezná žádnou normu, jak komentáře zapisovat (kromě syntaktické definice), zatímco v ASP.NET lze používat XML dokumentaci, ze které lze snadno vygenerovat HTML stránky, které se na chlup podobají těm z MSDN Library (standardní dokumentace). PHP tento nedostatek řeší pomocí externího nástroje PHPDoc, který svou syntaxí vychází z JavaDoc, zatím je však pouze ve fázi beta.

PHP a ASP.NET si tedy svou odlišnost drží až do samého závěru. Co se týče dokumentace samotné, PHP volí cestu minima přehledně uspořádaných informací se zakomponovanými připomínkami uživatelů, ASP.NET zase dokumentaci poskytuje v podobě několikagigabajtového molocha, ve kterém je snad úplně vše, ale zato se to nedá příliš snadno nalézt. Osobně stále preferuji přístup PHP, musím však přiznat, že po několikaměsíční práci s MSDN Library, kdy jsem trochu pochopil logiku jejího uspořádání, je to také poměrně dobře použitelný nástroj.

V otázce komentování kódu asi vede ASP.NET, které používá svůj formát založený na XML, takže lze poznámky snadno přeměnit ve standardní dokumentaci ala MSDN Library. PHP komentáře v kódu nijak neřeší.

# Oddíl třetí

# ZÁVĚR

## 11 Sumarizace výsledků

Jsme u konce, zbývá však ještě sumarizovat závěry. V předchozím textu lze najít poměrně mnoho oblastí, o kterých bylo alespoň okrajově pojednáno. Nemělo by příliš smysl se zde k dílčím závěrům znovu vracet – na konci každé kapitolky jsem se snažil učinit nějaké krátké shrnutí. Zde bych rád uvedl závěry z obecnějšího pohledu, respektive bych rád vyzdvihl to, co je podle mého názoru nejdůležitější.

Vývojář se nejčastěji bude setkávat s kódem samotným, proto začneme zde. ASP.NET přináší velmi zajímavý programovací model založený na serverových ovládacích prvkách. Ty mají několik zásadních vlastností:

- jsou zapisovány deklarativně a samy určují, jak bude generován jejich výstup (pro jiné klientské zařízení může být generován jiný výstup)
- reagují na události, umožňují tedy událostmi řízené programování
- umožňují oddělení skriptu od HTML kódu; „skript“ se tak stává spíše plnohodnotným objektově orientovaným programem než skriptem v tradičním slova smyslu
- jsou znovupoužitelné
- pokud není stanoveno jinak, automaticky si pamatují svůj stav mezi požadavky

To má dalekosáhlé důsledky např. pro udržovatelnost kódu, pro podporu ve vývojových prostředích, pro způsob přemýšlení nad řešenou situací apod.

PHP naproti tomu používá model „spaghetti code“, tedy míchání HTML kódu se serverovými vsuvkami. Hlavní výhodou tohoto přístupu je dokonalá kontrola nad generovaným výstupem, což může být v některých scénářích užitečné. Obecně je ale tento model považován spíše za problém, především kvůli obtížné čitelnosti a udržovatelnosti.

Co se týče syntaxe samotné, také zde najdeme propastný rozdíl. PHP je skriptováním v pravém slova smyslu, najdeme zde tedy přímočaré funkce pro řešení problémů, výstup je možno generovat pomocí jednoduchého příkazu `echo` atd. atd. Syntaxi PHP a základní úkony se lze tedy naučit doslova za odpoledne. Do verze 5 byla navíc přidána solidní podpora OOP, vývojář má tedy na výběr. Tato jazyková flexibilita je silnou devizou.

ASP.NET je naopak „opravdovým“ objektově orientovaným programováním (nejedná se žádný pseudo-objektový hybrid typu Visual Basic 6). Dobré zvládnutí ASP.NET je tak otázkou výrazně delšího času, už jen proto, že je potřeba se nejdříve seznámit s .NET Frameworkem jako takovým.

Který přístup je správný? Jediná odpověď patrně neexistuje a obě technologie se od sebe musí učit. Hlavní architekt jazyku C# Anders Hejlsberg uvažuje o zjednodušení syntaxe tím, že by nebylo nutné explicitně deklarovat typ proměnné (celý .NET ale samozřejmě typový zůstane, pouze by se typ získával např. z přiřazovaného výrazu), vývojáři PHP zase na plné obrátky pracují na kvalitnější podpoře OOP. Mým osobním dojmem je, že se PHP více snaží přiblížit jazyku C# než naopak (ve skutečnosti se PHP snaží přiblížit Javě, Java a C# však mají téměř shodnou syntaxi).

Co se týče typických problémů řešených při vývoji webových aplikací, jsou ASP.NET i PHP podobně vybavené (jinak by stěží mohly získat svou dnešní oblíbenost). Technologie se však většinou liší

v přístupu. PHP prokazuje, že je naprosto účelovou technologií, jakýmsi „quick & dirty“ nástrojem. Je potřeba vyřešit nějaký problém? Tady je příslušná funkce.

ASP.NET na mě působí daleko konzistentnějším dojmem. Je potřeba vyřešit nějaký problém? Tady je odpovídající technologie. Pro datové zdroje zde nenajdeme pouze funkce pro připojení k Oracle 8i a nějaké další funkce pro vykonání dotazu, ale máme zde pokročilé API jménem ADO.NET. Pro práci s XML zde nemáme jednoduchou knihovnu, která umí XML elegantně načíst a zpracovat, ale celý jmenný prostor System.Xml, který je plně integrovaný do .NET Frameworku jako celku. Webové služby? V PHP rozšíření pro práci se SOAP, v ASP.NET hluboká integrace do .NET Frameworku. Technologie se zkrátka liší svou filosofií.

ASP.NET také bodovalo v několika dalších důležitých aspektech. Především to byla výkonnost, kde má PHP ještě hodně co dohánět. Rovněž konfigurace není v PHP nijak systémově řešena (nepočítám-li několik málo konfiguračních direktiv v souboru php.ini), zatímco v ASP.NET je tato otázka velmi dobře vyřešena pomocí hierarchicky uspořádaných XML souborů. ASP.NET je také oproti PHP velmi snadno rozšiřitelné a s Visual Studiem nelze srovnávat žádné IDE pro PHP.

PHP má ale i své silné stránky. Především se jedná o otevřenou technologii (PHP je vyvíjeno jako Open Source), což má automaticky několik příjemných důsledků (je možno pozměnit funkčnost, snadno lze opravit nalezenou bezpečnostní chybu, PHP může být vyvíjeno, i kdyby projekt všichni současní vývojáři opustili atd. atd.). Především má však PHP skvělou podporu různých platforem – prakticky lze říct, že ho lze rozběhnout úplně všude.

Naopak vázanost ASP.NET na platformu Windows + IIS je jeho hlavní nevýhodou. Jednak je to platforma podléhající licenčním poplatkům, jednak není úplně jisté, že je to platforma pro prostředí Internetu dostatečně bezpečná. Můžeme argumentovat nízkým počtem nalezených chyb v IIS 6 nebo v ASP.NET, vázanost k jedné platformě je však jednoznačně nevýhodou.

Takže, jaký je závěr? Myslím, že ASP.NET prokázalo, že se jedná o kvalitní technologii, která PHP v lecčems předčí. Měla by dostat přednost s výjimkou několika případů, kdy bude PHP vhodnější:

- aplikace musí běžet na více platformách, respektive jinde než na platformě Windows + IIS
- aplikaci bude chtít vytvořit někdo, kdo není dostatečně seznámen s OOP nebo dává přednost jednoduchému jazyku PHP
- rozhodujícím faktorem při výběru bude cena hostingu (pro PHP jsou v současnosti hostingy o dost levnější)
- aplikace je natolik jednoduchá, že ji vývojář „napíše dříve, než by mu vůbec naběhlo Visual Studio“

Důležité je, že tyto scénáře jsou velmi časté. Ačkoliv tedy PHP v celé řadě věcí za ASP.NET zaostává, daří se mu nabízet dvě naprosto klíčové vlastnosti: platformovou nezávislost a jednoduchost. Proto nesouhlasím s radikálním tvrzením zarytých stoupenců ASP.NET, že v dnešním světě už není pro PHP místo. Naopak myslím, že PHP zůstane ještě dlouho oblíbené, ačkoliv celkově považuji ASP.NET za technologii vyspělejší.

# 12 Náměty na další práci

V předchozím textu se vyskytlo několik problémových pasáží, které nemohly být v rámci této práci řešeny, ať už kvůli rozsahu nebo kvůli značné složitosti. Tyto problémy mohou sloužit jako náměty na další práci.

## 12.1 Měření výkonnosti

V kapitole 10.4.2 bylo řečeno, že provést výkonnostní srovnání PHP a ASP.NET by sice bylo pěkné, ale zároveň by bylo potřeba vyřešit několik problémů (podrobnosti viz zmíněná kapitola). Osobně bych považoval nějaký „benchmark“ za informačně velmi přínosný, protože výkonnost považuji za jednu z důležitých vlastností technologie.

## 12.2 Produktivita

Produktivita vývojářů má velký vliv na celkové náklady technologie, problematika jejího měření je tak podle mého názoru zajímavým námětem. Třeba by šlo srovnávat, jak lze typické úlohy (např. získání dat z databáze a jejich zobrazení na stránce) řešit rychle (tj. s kolika řádky kódu) apod.

## 12.3 SEO

SEO (Search Engine Optimization, optimalizace pro vyhledávače) v poslední době narůstá na významu. PHP, které používá serverové vsuvky a má dokonalou kontrolu nad každým znakem odesílaného HTML, je samozřejmě pro SEO dobře připraveno, horší je to ale s ASP.NET. Tam je HTML kód generován automaticky, což může být problém. Schopnost s ASP.NET aplikovat SEO tedy také může být zajímavým tématem.

# Zdroje

Zdroje, ze kterých čerpala tato práce, často nejsou na kvalitativně stejné úrovni, proto jsem je rozdělil do několika skupin.

V té první budou uvedeny hlavní informační zdroje, jako např. knižní publikace nebo oficiální weby jednotlivých technologií. Zdroje z hlavní skupiny se poznají tak, že jsou v textu odkazovány pomocí textu v hranatých závorkách, např. [Prog.NET].

Ve druhé skupině budou uváděny zdroje menšího informačního rozsahu a také nižší garantované kvality, např. různé články nebo diskuse.

Třetí skupinu budou tvořit přímé odkazy na webové stránky, které byly použity jako zdroj nějaké konkrétní informace nebo které mají nějakou informaci potvrdit. Např. v textu po uvedení definice z Wikipedie bude následovat odkaz na zdroj z této třetí skupiny, kde bude uvedeno konkrétní URL, které dokládá pravost definice. Tyto zdroje budou pro odlišení odkazovány pomocí čísla v hranatých závorkách, např. [1], [2] atd.

Za hypertextovými odkazy následuje v hranatých závorkách datum, ke kterému byl odkaz platný a relevantní.

## I. Hlavní zdroje

### Publikace


**[Prog.NET]** PROSISE, J. *Programming Microsoft .NET*. Microsoft Press, 2002. 816 s. ISBN 0735613761

**[.NET+VS]** KAČMÁŘ, D. *Programujeme .NET aplikace ve Visual Studiu .NET*. Computer Press, 2001. 344 s. ISBN 8072265695

**[ASPza21dni]** PAYNE, Ch. *Naučte se ASP.NET za 21 dní*. Computer Press, 2002. 786 s. ISBN 8072266055

**[C#vkostce]** Drayton, P., Albahari, B., Neward, T. *C# v kostce - pohotová referenční příručka*. Grada, 2003. 788 s. ISBN 8024704439

**[PHPprof]** Castagnetto, J., Rawar, H., Schumann, S. aj. *PHP programujeme profesionálně*. 1. vyd. Computer Press, 2002. 656 s. ISBN 8072263102

**[PHPKosek]** Kosek, J. *PHP – tvorba interaktivních internetových aplikací*. Grada, 1999. 492 s. ISBN 8071693731


### Webové zdroje


**[php.net]** *PHP: Hypertext Preprocessor*, http://www.php.net/

**[zend.com]** *Zend Technologies*, http://zend.com/

**[msdn]** *MSDN Homepage*, http://msdn.microsoft.com/

**[asp.net]** *ASP.NET Web*, http://asp.net/

**[gotdotnet]** *GotDotNet: The Microsoft .NET Framework Community*, http://gotdotnet.com


## II. Články a jiné „menší“ zdroje


**[C#vsJava]** RADECK, K. *C# and Java: Comparing Programming Languages*. http://download.microsoft.com/download/7/4/7/7472c2f5-8c1a-4263-a1a3-772bc4d2142d/csharp_java_final.exe [17. 8. 2004]

**[SessFix]** Kolšek, M. *Session Fixation Vulnerability in Web-based Applications*. http://www.acros.si/papers/session_fixation.pdf [17. 8. 2004]

**[PHP5SOAP]** Stogov, D. *PHP SOAP Extension*. http://zend.com/php5/articles/php5-SOAP.php [17. 8. 2004]

**[PHP5XML]** Stocker, Ch. *XML in PHP 5 - What's New?*. http://zend.com/php5/articles/php5-xmlphp.php [17. 8. 2004]

**[ASPnetWS]** Tong, C. *Create and Consume Web Services at Server and Client*. http://www.asp101.com/articles/colin/webservices/default.asp [17. 8. 2004]

**[C#Anders]** *The C# Design Process: A Conversation with Anders Hejlsberg*. http://msdn.microsoft.com/vcsharp/homepageheadlines/hejlsberg/default.aspx [17. 8. 2004]

# III. Odkazy

**[1]** http://www.microsoft.com/windows2000/en/server/iis/htm/core/iigloss4.htm [17. 8. 2004]
**[2]** http://en.wikipedia.org/wiki/Web_application [17. 8. 2004]
**[3]** http://en.wikipedia.org/wiki/Application [17. 8. 2004]
**[4]** http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/glossary.html [17. 8. 2004]
**[5]** http://en.wikipedia.org/wiki/Common_Gateway_Interface [17. 8. 2004]
**[6]** http://zend.com/php5/intro_letter.php [17. 8. 2004]
**[7]** http://www.microsoft.com/cze/net/basics/whatis.asp [17. 8. 2004]
**[8]** http://www.php.net/manual/en/ref.java.php [17. 8. 2004]
**[9]** http://www.csh.rit.edu/~jon/projects/pip/ [17. 8. 2004]
**[10]** http://www.gotdotnet.com/team/lang/, http://www.jasonbock.net/dotnetlanguages.html [17. 8. 2004]
**[11]** http://www.w3.org/TR/REC-xml/ [17. 8. 2004]
**[12]** http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconwebusercontrolsvscustomwebcontrols.asp [17. 8. 2004]
**[13]** http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/webdevinwhidbey.asp [17. 8. 2004]
**[14]** http://www.devcity.net/net/article.aspx?alias=msil_1_intro [17. 8. 2004]
**[15]** http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconwhatiscommonlanguagespecification.asp [17. 8. 2004]
**[16]** http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconthecommontypesystem.asp [17. 8. 2004]
**[17]** http://cz.php.net/manual/en/function.strpos.php [17. 8. 2004]
**[18]** http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconAssemblies.asp [17. 8. 2004]
**[19]** http://en.wikipedia.org/wiki/Polymorphism_%28computer_science%29 [17. 8. 2004]
**[20]** http://www.artima.com/intv/nonvirtual.html [17. 8. 2004]
**[21]** http://zend.com/expert_qa/qas.php?id=43&single=1 [17. 8. 2004]
**[22]** http://www.php.net/manual/en/ref.strings.php [17. 8. 2004]
**[23]** http://pear.php.net/ [17. 8. 2004]
**[24]** http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconwhyadonet.asp [17. 8. 2004], http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconoverviewofadonet.asp [17. 8. 2004]
**[25]** http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpcontheadonetdataset.asp [17. 8. 2004]
**[26]** http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemxml.asp [17. 8. 2004]
**[27]** http://zend.com/php5/articles/php5-xmlphp.php [17. 8. 2004]
**[28]** http://msdn.microsoft.com/library/en-us/vbcon/html/vbconchoosingserverstateoption.asp [17. 8. 2004]

**[29]** http://www.asp101.com/articles/colin/webservices/default.asp [17. 8. 2004]
**[30]** http://msdn.microsoft.com/library/en-us/cpguide/html/cpconaspnetconfiguration.asp [17. 8. 2004]
**[31]** http://gotdotnet.com/team/compare/Nile%20Benchmark%20Results.doc [17. 8. 2004]
**[32]** http://www.dotmugs.ch/links.aspx?list=opendotnet [17. 8. 2004]
**[33]** http://msmvps.com/bernard/archive/2004/06/10/7882.aspx [17. 8. 2004]

# Terminologický slovník

Zde jsou uvedeny nejdůležitější pojmy, které se v práci objevily. Jejich význam je pospán podle potřeb této práce, v žádném případě se nejedná o definice.

| | |
|---|---|
| ***.NET [dot net]*** | „Sada softwarových technologií společnosti Microsoft pro propojování světa informací, lidí, systémů a zařízení.“ |
| ***.NET aplikace*** | Aplikace určená pro platformu *.NET*. |
| ***.NET Framework*** | Jádro platformy *.NET*, běhové prostředí a sada podpůrných prostředků pro *.NET aplikace*. |
| ***.NET Framework SDK*** | Balík nástrojů pro vývoj na platformě *.NET*. |
| ***.NET jazyk*** | Jazyk, který vyhovuje pravidlům *CLS (Common Language Specification)* a může být používán pro vývoj *.NET aplikací*. |
| ***abstraktní třída*** | Třída, která nemůže být instanciována (slouží jako třída předka). |
| ***add-in*** | Výraz Microsoftu pro plugin. |
| ***ADO.NET*** | *API* pro přístup k *datovým zdrojům* na platformě *.NET*. |
| ***Apache*** | Rozšířený *Open Source* webový server. |
| ***API, Application Programming Interface*** | Rozhraní (sada funkcí) pro programování aplikací. |
| ***ASP, Active Server Pages*** | Skriptovací technologie pro webové aplikace od Microsoftu. |
| ***ASP.NET*** | Nová platforma pro vývoj webových aplikací, která je součástí *.NET Frameworku*. |
| ***atribut (v .NET jazycích)*** | Syntaktický prvek, který umožňuje danou entitu (metodu, členskou proměnnou, …) popisovat pomocí meta dat. |
| ***brzká vazba*** | To, která metoda bude použita, je určeno v době kompilace (nikoliv až za běhu jako u *pozdní vazby*). |
| ***C# [sí-šárp]*** | Jeden z hlavních *.NET jazyků*. |
| ***cachování [kešování]*** | Uchovávání dynamicky generovaných věcí na nějakém místě, aby příště nemusely být znovu generovány. |
| ***case sensitive*** | Citlivost na velikost písmen. |
| ***CGI, Common Gateway Interface*** | Rozhraní definující komunikaci mezi webovým serverem a externí aplikací označovanou za CGI skript. |
| ***CLI, Common Language Infrastructure*** | Specifikace pro podporu *jazykové neutrality .NET Frameworku*. Základními částmi jsou společný typový systém, metadata, *CLS* a virtuální běhový systém. |
| ***CLS, Common Language Specification*** | Specifikace určující, jaké vlastnosti musí každý *.NET jazyk* mít. |
| ***code- behind*** | Technika oddělení HTML kódu od skriptu používaná v ASP.NET. |
| ***cookies*** | Textové informace posílané v hlavičce *HTTP*, které mohou být uloženy na straně klienta a později odeslány zpět serveru. |
| ***CSS, Cascading Style Sheets, kaskádové styly*** | Jazyk a technologie pro určení stylu (většinou vzhledu) stránky. |
| ***datový zdroj*** | Abstraktní pojmenování určitého zdroje dat, kterým může být relační databáze, soubor na disku, Excel apod. |

| | |
|---|---|
| ***dědičnost*** | Princip *OOP*, který umožňuje potomkovské třídě zdědit veškeré atributy a metody od svého rodiče. |
| ***ECMA, European Computer Manufacturers Association*** | Mezinárodní asociace založená v roce 1961, jejímž hlavním cílem je standardizace ICT systémů. V roce 1994 byla přejmenována na Ecma International, známa je především díky ECMAScriptu. |
| ***getters, setters*** | Veřejné metody, které slouží k přístupu k soukromým členským proměnných třídy (obvykle se vytvářejí podle syntaxe `getPromenna()` a `setPromenna(…)`) |
| ***HTML, HyperText Markup Language*** | Jazyk používaný pro WWW stránky se schopností odkazovat na další dokumenty Internetu. |
| ***HTTP, HyperText Transfer Protocol*** | Protokol určený pro komunikaci mezi klientem a serverem v prostředí *WWW*. |
| ***IDE, Integrated Development Environment*** | Integrované vývojové prostředí. |
| ***IIS, Internet Information Services*** | Bývá ztotožňováno s webovým serverem, balík navíc obsahuje FTP a SMTP servery. Autorem je Microsoft. |
| ***J#*** | *.NET jazyk* se syntaxí *Javy*. |
| ***Java*** | Platforma a jazyk firmy Sun. |
| ***jazyková neutralita*** | Taková vlastnost technologie, která umožňuje pro vývoj používat více než jeden jazyk. |
| ***jmenné prostory, namespaces*** | Obdoba balíčků (packages) v *Javě* umožňující sdružování funkčně podobných tříd. |
| ***JSP, JavaServer Pages*** | Objektová webová technologie firmy Sun. |
| ***Mobile Internet Toolkit*** | Sada nástrojů pro vývoj pro mobilní zařízení (PDA apod.). |
| ***modifikátory přístupu*** | Klíčová slova jazyka, která určují dostupnost členů třídy v různých kontextech. |
| ***Mono*** | Nejnadějnější *Open Source* implementace *.NET Frameworku* mimo platformu Windows. |
| ***MSDN, Microsoft Developers Network*** | Komunita vývojářů kolem produktů Microsoftu. |
| ***MSIL, Microsoft Intermediate Language*** | Obdoba bytecode *Javy*, mezijazyk, do kterého se kompilují *.NET aplikace*. |
| ***MySQL*** | *Open Source* relační databáze. |
| ***neinicializovaná proměnná*** | Proměnná, která je použita, aniž by jí dříve byla přiřazena nějaká hodnota. |
| ***netypový, dynamicky typovaný jazyk*** | Jazyk, u kterého v době překladu typy nemusejí být známy, dosazují se až za běhu. |
| ***objektové technologie*** | Pracovně používaný termín! Technologie, jako např. *JSP* nebo *ASP.NET*, které podporují vývoj aplikací objektovým způsobem. |
| ***OOP (objektově orientované programování)*** | Programování, které využívá objektových principů (objekty, abstrakce, zapouzdření, dědičnost, polymorfismus apod.). |
| ***opcode*** | Mezijazyk, do kterého se kompilují PHP skripty. Tento kód je při požadavku dále interpretován. |
| ***Open Source*** | V užším významu aplikace s přístupným zdrojovým kódem, v širším významu hnutí prosazující vývoj aplikací otevřeným způsobem. |
| ***output caching, cachování výstupu*** | Cachování výstupu tak, aby při příštím požadavku mohla být rovnou doručе- |

| | |
|---|---|
| | na statická kopie namísto nového dynamického generování téhož výstupu. |
| ***ovládací prvek, control*** | Softwarová komponenta, ať už vizuální (tlačítko, kalendář, …) nebo nevizuální (připojení k databázi, časovač, …). |
| ***PHP, PHP: Hypertext Preprocessor*** | Jedna z nejpoužívanějších skriptovacích technologií pro tvorbu webových aplikací. |
| ***polymorfismus*** | Princip *OOP* založený na pozdní vazbě. Často bývá ztotožňován s možností vytvářet pro různé objekty metody se stejnými názvy. |
| ***pozdní vazba*** | To, která metoda bude ve skutečnosti použita, je určeno až za běhu (nikoliv při kompilaci jako u *brzké vazby*). Souvisí s *virtuálními metodami*. |
| ***proxy*** | Zástupný objekt na straně klienta, který se tváří úplně stejně jako objekt na straně serveru. |
| ***přetěžování, overloading*** | Vytvoření více metod stejného názvu v rámci jedné třídy. Metody se musí lišit signaturou (počtem, pořadím nebo typem parametrů). |
| ***query string*** | Část URL za otazníkem (http://priklad/soubor?**toto=je&query=string**) |
| ***refaktoring kódu*** | Změny kódu bez vlivu na jeho funkčnost (např. přejmenování určité proměnné na všech místech jejího výskytu). |
| ***renderování*** | Generování výstupu. |
| ***SDK, Standard Development Kit*** | Balík vývojových nástrojů. |
| ***serverové ovládací prvky, server controls*** | *Ovládací prvky*, ze kterých se skládají ASP.NET stránky. |
| ***serverové vsuvky*** | Určité části HTML dokumentu, které jsou před odesláním klientovi zpracovány příslušným programem. |
| ***session id*** | Identifikátor relace, textový řetězec přenášený s každým HTTP požadavkem mezi klientem a serverem. |
| ***sessions, relace*** | Způsob řešení nestavovosti *HTTP* založený na serveru. Stavová data jsou ukládána na serveru a klient je identifikován pomocí *session id*. |
| ***skládání*** | Technika *OOP*, kdy jsou do určité třídy vkládány objekty jiných tříd. |
| ***skriptovací technologie*** | Pracovně používaný termín! Technologie, jako např. *PHP* nebo *ASP*, které používají vkládání skriptového kódu do *HTML* a nemají příliš dobrou podporu pro vývoj aplikací objektovým způsobem. |
| ***skriptový kód, serverový kód*** | Pracovně používané pojmy pro kód zpracovávaný na straně serveru. |
| ***SOAP (Simple Object Access Protocol)*** | Protokol pro přenos XML zapouzdřených dat. Používá se ke vzdálenému vyvolávání metod a přijímání výsledků mezi systémy. |
| ***spaghetti code*** | Promíchání HTML kódu se skriptovým kódem. Důsledek použití serverových vsuvek. |
| ***SSI (Server Side Includes)*** | Sada direktiv, které mohou být vloženy do HTML a zpracovány webovým serverem. |
| ***starter kits*** | Ukázkové odladěné ASP.NET aplikace s dostupným zdrojovým kódem. |
| ***typový, staticky typovaný jazyk*** | Jazyk, u kterého jsou známy typy proměnných už v době překladu. |
| ***událostmi řízené programování*** | Tok programu je řízen událostmi. (V tradičním programu je tok řízen předem danými pravidly, většinou je program prováděn sekvenčně od začátku do konce s výjimkou různých „specialit“ typu větvení nebo cyklů.) |
| ***user agent*** | Řetězec identifikující prohlížeč. Je odesílán v HTTP požadavku. |

***validita HTML*** Dodržení standardu deklarovaného v DOCTYPE.

***vázání dat, data binding*** Přivázání dat k nějakému ovládacímu prvku (např. přivázání dat z databáze k datové mřížce způsobí jejich automatické vykreslení, není potřeba v cyklech iterovat přes všechny řádky a sloupce).

***viewstate*** Skryté formulářové pole, které *ASP.NET* využívá k automatickému zapamatování stavů *serverových ovládacích prvků*.

***virtuální metoda*** Metoda, jejíž případné volání je určeno až za běhu. Souvisí s *pozdní vazbou*.

***Visual Basic .NET*** Jeden z hlavních *.NET jazyků*.

***Visual Studio .NET*** *IDE* pro platformu *.NET* (tedy také pro *ASP.NET*).

***vlastnosti, properties*** Syntaktická záležitost, která místo *getters a setters* používá přirozenější přístup k členským proměnným třídy (místo `setCislo(5)` se píše `Cislo = 5`).

***výjimky, exceptions*** Mechanismus pro konzistentní zpracování problémových („výjimečných“) stavů.

***WAP (Wireless Application Protocol)*** Otevřený mezinárodní standard pro aplikace, které používají bezdrátovou komunikaci. Především má zpřístupnit internet mobilním telefonům.

***Web Forms*** Jádro ASP.NET, koncept, pomocí kterého lze webové stránky skládat z komponent podobně jako při vývoji okenních aplikací.

***webová aplikace*** Aplikace postavená na modelu klient/server, kde koncovým klientem je webový prohlížeč, serverem je webový server a pro komunikaci mezi nimi se používá protokol *HTTP*.

***webová služba*** Aplikace stavící na modelu webových služeb.

***webové služby*** V konkrétním významu množné číslo od webové služby, v obecném významu sada protokolů a standardů pro komunikaci mezi systémy.

***WSDL (Web Services Description Language)*** XML formát pro popis webové služby.

***WWW (World Wide Web)*** Hypertextový systém operující přes Internet.

***WYSIWYG (What You See Is What You Get)*** Vizuální způsob návrhu.

***XHTML*** Nástupce standardu *HTML*, který používá syntaxi *XML*.

***XML (eXtensible Markup Language)*** Rozšiřitelný značkovací jazyk umožňující sémantický popis dat a komunikaci mezi systémy.

***zapouzdření*** Princip *OOP*, skrývání implementace a zpřístupnění funkčnosti pomocí rozhraní.

***Zend*** Komerční tvář *PHP*, firma vyvíjející *Zend Engine*, vývojové prostředí *Zend Studio* a několik dalších produktů navázaných na PHP.

***Zend Engine*** Mozek a srdce *PHP*, jádro vykonávající PHP skripty.

***Zend Studio*** *IDE* pro *PHP*.

# Příloha: Jiná srovnání PHP s ASP.NET

Název: **PHP and ASP.NET Go Head-to-Head**
Autor: Sean Hull
URL: http://otn.oracle.com/pub/articles/hull_asp.html

Poněkud tendenčně napsaný článek, ze kterého jako jasný vítěz vychází PHP. Poměrně drtivou kritiku lze nalézt na adrese http://weblogs.asp.net/bleroy/articles/193608.aspx, další kritika je např. zde: http://ryangregg.com/archive/2004/07/19/982.aspx.

Název: **Head-to-Head: PHP vs. ASP.NET**
Autoři: Robert Lair, Jason Lefebvre
URL: http://www.msdnaa.net/content/?ID=2315

Článek napsaný v květnu 2003, který byl nejdříve umístěn na webu MSDN a posléze přesunut na MSDNAA (MSDN Academic Aliance). Je psán pokud možno neutrálním tónem, často vyzdvihuje různé vlastnosti PHP, přesto má však budit dojem, že má ASP.NET navrch (aniž by byl závěr takto explicitně vyjádřen). Článku chybí komplexnost, zaměřuje se pouze na pár záležitostí a je zaměřen na starší verzi PHP 4.

Název: **PHP, ASP or ASP.NET?**
Autor: Daniel Winter
URL: http://uk.builder.com/webdevelopment/scripting/0,39026636,20283074,00.htm

Spíše popisný článek, který stručně jednotlivé technologie představuje. V závěru se dozvíme pouze to, že každá technologie se může v určitých situacích hodit.

Název: **Top 6 Reasons to Use .NET**
Autor: Mattias Johansson
URL: http://www.sitepoint.com/article/871

Podle mého názoru trochu příliš nadšený článek o dokonalosti ASP.NET. Argumenty jsou uváděny jednostranně a zjednodušeně – autor např. pokládá vývoj v ASP.NET za levnější, protože „nedostižný“ Web Matrix je zadarmo, zatímco UltraEdit stojí 35 dolarů a Bullet Proof FTP 30 dolarů. Rovněž je asi za hranicí únosnosti jako jednu z šesti největších výhod ASP.NET uvést její nezávislost na platformě.

Název: **Top 10 .NET Myths Exposed**
Autor: Harry Fuecks
URL: http://www.sitepoint.com/article/870?ct=1

Článek pochází ze stejného serveru jako ten předchozí a má být pohledem z druhé strany. Nevyjmenovává však x nejúžasnějších věcí na PHP, ale spíš ho obhajuje proti výrokům typu „ASP.NET je na tvoření webových aplikací nejlepší!“. Podle nadpisu má článek vyvracet 10 největších mýtů o .NETu, nečiní to však příliš přesvědčivě, občas autor sklouzává až k záměrnému překrucování skutečností. Příkladem je „mýt č. 4“, který tvrdí, že ASP.NET stránky jsou kompilované, zatímco PHP skripty nejsou. Autor velmi razantně tento „mýt“ vyvrací s tím, že v .NETu slovo „kompilovaný“ naznamená totéž co v jazyku C++. To má sice pravdu, určitě ale sám dobře ví, že nakonec je MSIL stejně kompilováno do strojového kódu a teprve ten je potom opakovaně prováděn. Tuto „drobnost“ však již pro jistotu nezmiňuje. I dále se argumentace nese v poněkud podezřelém duchu, i tak kontroverzní prohlášení, že PHP má lepší podporu OOP než .NET, si vystačilo s jedním odkazem na jakési FAQ (v cílovém dokumentu je navíc pojednáváno pouze o nedeterministické destrukci objektů, což není argument, ale spíš námět na debatu). Celkově tento článek hodnotím jako silně tendenční – zatímco v předchozím článku jsou věcné chyby spíše kvůli nezvládnutému opojení .NETem, sem byly podle mého názoru zakomponovány zcela záměrně.