Interreg
EVROPSKÁ UNIE
Rakousko-Česká republika
Evropský fond pro regionální rozvoj

INFORMATIKA

Základy webových aplikací

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

VŠTE

EVROPSKÁ UNIE

Obsah

1. Komunikace, sítě, protokoly 4
1.1. Komunikační protokoly 4
1.2. TCP/IP 4
1.2.1. Aplikační vrstva 6
1.2.2. Fyzická vrstva 6
1.2.3. Linková vrstva 6
1.2.4. Síťová vrstva 6
1.2.5. Transportní vrstva 7
1.3. ICMP 7
1.3.1. Velikost paketů 7
1.4. IPv4 9
1.5. IPv6 9
1.6. ICMP 9
1.7. TCP 10
1.8. UDP 10
1.9. SCTP 10
2. Úvod do jazyka HTML 11
2.1. Úvod do HTML 11
2.2. Historie HTML 11
2.3. Co je potřeba pro tvorbu html stránek 11
2.4. Používané termíny v HTML. 11
2.5. HTML 13
2.6. Jazyky pro prezentaci webového obsahu - CSS 14
2.6.1. Historie CSS 14
2.6.2. Další možnosti použití 14
2.6.3. Syntaxe 16
2.6.4. CSS styly 18
2.6.5. DHTML 18
2.6.6. CSS styly a třídy, identifikátory a style 18
3. Logika na straně klienta - JavaScript 19
3.1. Co je JavaScript 19

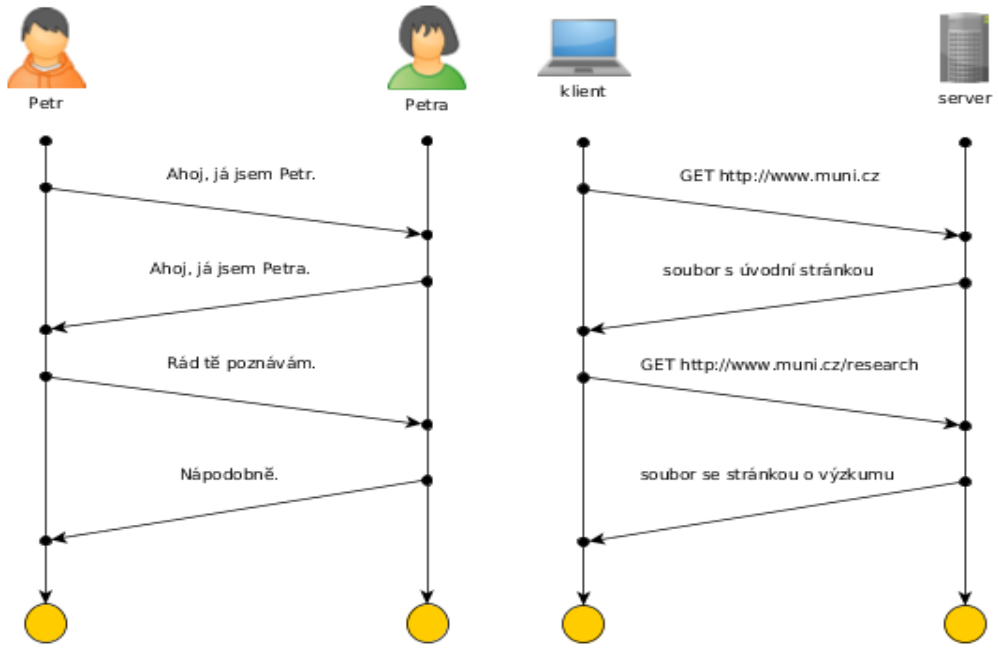
3.1.1. Charakteristiky jazyka 20
3.1.2. Vysvětlení skriptu 20
3.1.3. Zápis skriptu 22
4. Architektura webu 23
4.1. Návrh architektury webu 23
4.2. Programování webových aplikací 24
4.3. Technologie našich webových aplikací 27
4.4. Co to je architektura webu? 27
4.4.1. Jak na dokonalou architekturu 28
5. PHP /basics/ 32
5.1. PHP – základní informace 33
K čemu je PHP dobré? 33
Co je k tvorbě v PHP potřeba? 33
6. Zpracování http dotazu 36
6.1. Metody dotazu 36
GET 36
POST 36
HEAD 36
PUT/DELETE 36
TRACE 36
6.1.1. Dotazové hlavičky 37
Accept* 37
Host 37
6.1.2. Hlavičky odpovědi 37
Content* 37
Expires 37
6.2. Základní rysy protokolu HTTP 38
6.2.1. Formát požadavku protokolu HTTP 38
Příklad: 39
7. Metody požadavku protokolu HTTP 40
7.1. Metoda GET 40
7.2. Metoda POST 41
7.3. Použití datových zdrojů 42

7.4. Co je ODBC ... 42
8. Datové zdroje ... 44
9. Databázový přístup ... 47
10. Data strukturovaná a nestrukturovaná ... 50
10.1. Strukturovaná data ... 50
10.2. Data nestrukturovaná ... 50
10.3. Objemy dat - strukturovaná i nestrukturovaná data ... 51
10.4. Technologické charakteristiky datového skladu ... 52
10.5. Logická struktura datového skladu ... 52
11. Technologie AJAX ... 54
12. Frameworky ... 59
12.1. Co jsou frameworky? ... 61
13. 10 nejlepších PH frameworků pro vývojáře ... 62

1. KOMUNIKACE, SÍTĚ, PROTOKOLY

1.1. Komunikační protokoly

V kontextu počítačových sítí se často setkáváme s komunikačními protokoly. Ty přesně definují způsob, jakým probíhá komunikace realizující konkrétní funkci a to na všech úrovních. Máme tak protokoly pro zasílání dat, navazování zabezpečených kanálů, vyhledání síťové adresy odpovídající doménovému jménu, doručení emailu atd. Protokol je známý oběma komunikujícím stranám a popisuje přesně jaký obsah, v jakém pořadí a s jakým časováním je předáván. Odklon od takto strukturované komunikace je možné interpretovat jako chybu.



Obrázek 1: Ilustrace komunikačního protokolu

1.2. TCP/IP

Základním principem prostupujícím architekturu počítačových sítí je rozdělení komunikace do vrstev podle abstrakce. Každá vrstva je zodpovědná za popis přenosu od úrovně aplikace až po komunikaci po fyzických spojích. Síťový model TCP/IP je základním kamenem všech dnešních sítí a i celého internetu a je pojmenován podle dvou hlavních protokolů zajišťujících směrování a transport dat mezi uzly. Protokol IP popisuje adresaci uzlů, rozklad dat na pakety a jejich směrování uvnitř sítě.

Komunikace mezi stejnými vrstvami dvou různých systémů je řízena komunikačním protokolem za použití spojení vytvořeného sousední nižší vrstvou. Architektura umožňuje výměnu protokolů jedné vrstvy bez dopadu na ostatní. Architektura TCP/IP je členěna do čtyř vrstev (na rozdíl od referenčního modelu OSI se sedmi vrstvami):

- aplikační vrstva (*application layer*)
- transportní vrstva (*transport layer*)
- síťová vrstva (*internet layer*)
- vrstva síťového rozhraní (*network interface*)

Obrázek 2: Vrstvy síťového modelu TCP/IP

TCP je transportní protokol stavící „spojovanou“ službu nad IP přenosem: zajišťuje kontrolu úspěšnosti přenosu a případné přeposílání chybějících či poškozených částí a volí optimální rychlost přenosu.

Rodina protokolů TCP/IP, kterou dnes používáme k realizaci naprosté většiny síťové komunikace, byla navržena na přelomu 70. a 80. let s cílem vytvořit robustnější model síťové komunikace, který by byl schopen se do určité míry vypořádat i s výpadky částí sítě. Základní (a v té době celkem revoluční) myšlenkou je *packet switching* (přepínání paketů): data nejsou posílána jako souvislý proud (stream), ale po samostatných blocích (paketech), a jednotlivé uzly sítě samy rozhodují, kudy budou pakety dále posílat.

V praxi je tato myšlenka realizována sadou protokolů, implementujících potřebné funkce. Protokoly obvykle rozdělujeme do několika úrovní (vrstev). Místo abstraktního ISO/OSI modelu, který pracuje se sedmi vrstvami, při výkladu TCP/IP většinou používáme zjednodušený pětivrstvý model (některé protokoly v TCP/IP modelu zastávají funkci více vrstev ISO/OSI modelu).

1.2.1. Aplikační vrstva

Nejvýše je *aplikační vrstva*, tak označujeme data aplikačních protokolů jednotlivých síťových služeb. zahrnuje protokoly síťových aplikací: elektronické pošty, HTTP (webové stránky), DNS (doménová služba) a další .Takových protokolů existuje obrovské množství, z nejznámějších uveďme např. HTTP, SMTP, FTP, NTP. Z pohledu TCP/IP se jedná o data, která je třeba přenést k cílovému příjemci. O to se starají nižší vrstvy.

1.2.2. Fyzická vrstva

Nejníže je v modelu vrstva *fyzická*. Na rozdíl od vyšších vrstev se nejedná o softwarovou vrstvu (protokol), tímto označením rozumíme konkrétní fyzické médium, které používáme k přenosu dat. Příkladem může být např. twisted pair (kroucená dvoulinka) kabeláž ve většině lokálních ethernetových sítí, koaxiální kabel, optické vlákno nebo telefonní linka. Médium ale nemusí být hmotné - např. v případě bezdrátových sítí v mikrovlnném pásmu (wi-fi, breezenet) nebo optických pojítek.

1.2.3. Linková vrstva

Nejníže ze softwarových vrstev je *linková vrstva*. Jedná se o nejnižší komunikační protokol, sloužící k přenášení dat po fyzickém médiu. Tento protokol je většinou úzce svázán s konkrétní volbou média, ale tato korespondence nemusí být 1:1, např. ethernet bývá v praxi implementován nejen na twisted-pair kabeláži, ale můžeme se setkat s jeho implementacemi pomocí koaxiálního kabelu nebo naopak optických vláken. Jiným příkladem protokolu linkové vrstvy je PPP, protokol používaný k realizaci vytáčeného připojení (dial-up) nebo propojení počítačů přes sériovou linku. Podstatnou vlastností protokolů linkové vrstvy je skutečnost, že řeší pouze komunikaci mezi uzly, které jsou *přímo* spojeny (odtud i název).

1.2.4. Síťová vrstva

Globální adresaci a směrování má na starosti vrstva *síťová*, v praxi realizovaná téměř výhradně protokolem IP (vyskytujícím se ve dvou verzích, IPv4 a IPv6). Zatímco i u protokolů linkové vrstvy existují adresy a např. v případě ethernetových MAC adres jsou (nebo by aspoň měly být) dokonce globálně jednoznačné, nelze je použít ke směrování paketů, protože z takových adres nelze poznat, kde cíl hledat. Adresy protokolu IP (IP adresy) jsou ale přidělovány hierarchicky tak, že delegace jednotlivých rozsahů odpovídá topologii sítě. Z cílové IP adresy lze proto určit, kudy máme paket dále poslat, tedy alespoň následujícího prostředníka (hop) po cestě. Kromě této své základní funkce

řeší protokol IP ještě některé další, např. fragmentaci (rozdělení příliš dlouhých paketů na několik kratších) nebo označení paketů podle typu provozu (ToS - type of service).

1.2.5. Transportní vrstva

Nejpoužívanějšími protokoly transportní vrstvy jsou dnes UDP a TCP. UDP (User Datagram Protocol) lze chápat jako minimalistický transportní protokol, zavádějící pouze pojem portu, který lze chápat jako adresu konkrétního procesu (přesněji socketu) v rámci cílového uzlu. UDP je ale stále bezstavový protokol (nelze tady mluvit v pravém slova smyslu o spojení), neřeší otázku ztracených paketů ani jejich pořadí. Přesto se často používá pro svou jednoduchost a nižší režii, a to zejména tam, kde tyto otázky řešit nepotřebujeme.

Opačný přístup je reprezentován protokolem TCP (Transmission Control Protocol). Ten naopak zavádí *spojení* mezi dvěma porty na koncových uzlech. Z pohledu klientské aplikace se takové spojení chová podobně jako roura pro komunikaci mezi dvěma procesy (ale na rozdíl od roury je TCP spojení obousměrné), je zaručeno, že posloupnost bytů (stream), kterou jedna strana odešle, dostane ve stejné podobě druhá strana. Protokol TCP se stará o detekci a opakované odeslání ztracených dat, stejně jako o přerovnání dat z paketů, které dojdou ve špatném pořadí. TCP tak poskytuje aplikační vrstvě poměrně vysokou míru komfortu, většina komunikace proto dnes používá jako transportní protokol TCP. Nevýhodou ale může být vyšší režie a relativně pomalá reakce na výpadky, proto se pro některé účely (např. většina DNS dotazů nebo VoIP) dává přednost UDP.

1.3. ICMP

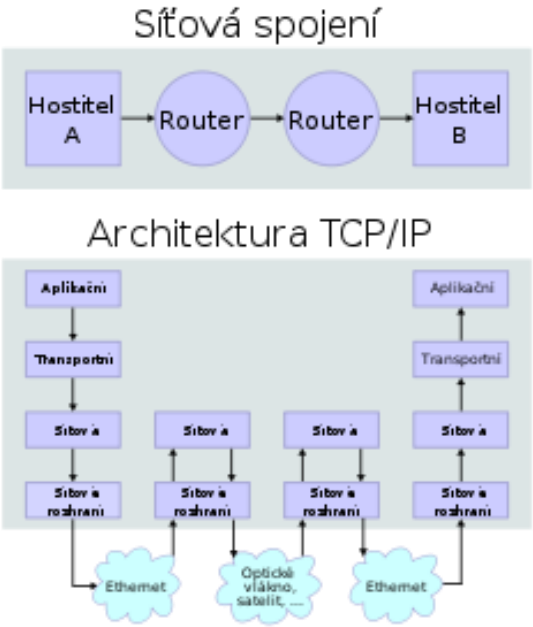
Ze struktury vrstev se trochu vymyká protokol ICMP (Internet Control Message Protocol). Pakety (message) tohoto protokolu jsou přenášeny přímo prostřednictvím IP protokolu, ICMP ale nelze považovat za transportní protokol, protože neslouží k přenášení aplikačních dat. Tento protokol slouží k diagnostickým a servisním účelům. Příkladem aplikací ICMP jsou zprávy o nedoručitelnosti paketu (destination unreachable), pakety generované příkazem echo (ICMP echo a echo reply) nebo některé servisní typy zpráv (redirect).

1.3.1. Velikost paketů

Maximální (teoretická) velikost IP paketu je 65535 B, ale limitujícím faktorem je většinou linková vrstva. Protože většina paketů aspoň jednou projde přes ethernet (nebo jeho ekvivalent), bývá většinou velikost paketů volena podle jeho limitu (1536 B), odtud nejobvyklejší hodnota 1500 B. To je ale samozřejmě pouze maximální hodnota, pakety

často

bývají i výrazně kratší, zejména u interaktivních aplikací. Hlavičky IP a TCP mají velikost 20-60 B (obvyklejší jsou hodnoty u dolní hranice), UDP a ICMP hlavičky mají 8 B, ethernetová hlavička 14 B (navíc 2 B na konci paketu kontrolní součet).



Vrstvy TCP/IP zajišťující přenos mezi dvěma hostiteli prostřednictvím dvou routerů.

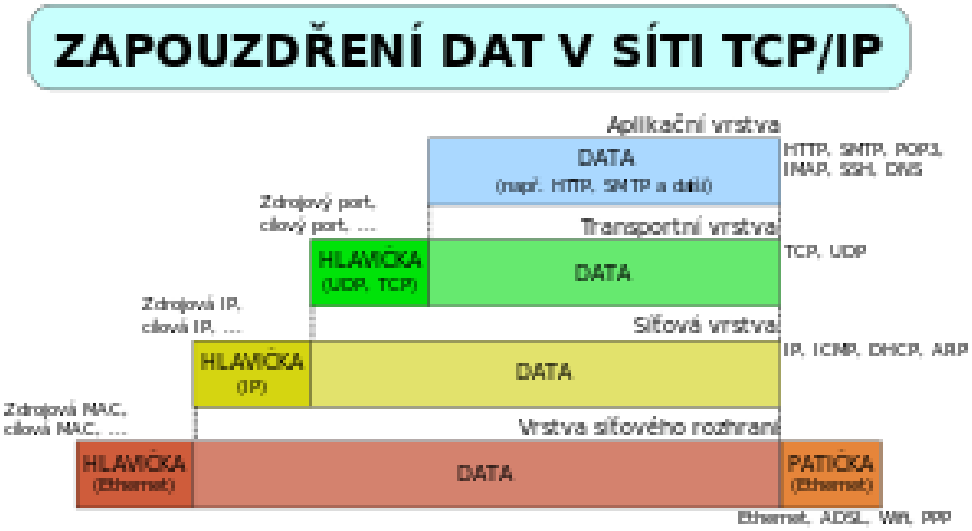


Schéma zapouzdření aplikačních dat na vrstvách TCP/IP.

Vzhledem ke složitosti problémů je síťová komunikace rozdělena do tzv. vrstev, které znázorňují hierarchii činností. Výměna informací mezi vrstvami je přesně definována. Každá vrstva využívá služeb vrstvy nižší a poskytuje své služby vrstvě vyšší.

Internet Protocol je základní protokol síťové vrstvy a celého Internetu. Provádí vysílání datagramů na základě síťových IP adres obsažených v jejich záhlaví. Poskytuje vyšším vrstvám síťovou službu bez spojení.

V současné době je převážně používán protokol IP verze 4. Nová verze 6, která řeší nedostatek adres v IPv4, bezpečnostní problémy a vylepšuje další vlastnosti protokolu IP, je celosvětově používána jen několika procenty zařízení připojených k internetu, ale jejich počet rychle roste.

1.4. IPv4

- Internet protokol verze 4
- 32 bitové adresy
- cca 4 miliardy různých IP adres, dnes nedostačující
- formát: xxx.xxx.xxx.xxx kde xxx je libovolné číslo od 0 do 255 (8 bitů)

1.5. IPv6

- Internet protokol verze 6
- 128 bitové adresy
- podpora bezpečnosti
- podpora pro mobilní zařízení
- funkce pro zajištění úrovně služeb (QoS - Quality of Service)
- fragmentace paketů - rozdělování
- není zpětně kompatibilní s IPv4

Address Resolution Protocol se používá k nalezení fyzické adresy MAC podle známé IP adresy. Protokol v případě potřeby vyšle datagram s informací o hledané IP adrese a adresuje ho všem stanicím v síti. Uzel s hledanou adresou reaguje odpovědí s vyplněnou svou MAC adresou. Pokud hledaný uzel není ve stejném segmentu, odpoví svou adresou příslušný směrovač.

1.6. ICMP

Internet Control Message Protocol slouží k přenosu **řídících hlášení**, které se týkají chybových stavů a zvláštních okolností při přenosu. Používá se např. v programu *ping* pro testování dostupnosti počítače, nebo programem *traceroute* pro sledování cesty paketů k jinému uzlu.

1.7. TCP

Transmission Control Protocol vytváří virtuální okruh mezi koncovými aplikacemi, tedy **spolehlivý přenos dat**. Vlastnosti protokolu:

- Spolehlivá transportní služba, doručí adresátovi všechna data bez ztráty a ve správném pořadí.
- Služba se spojením, má fáze navázání spojení, přenos dat a ukončení spojení.
- Transparentní přenos libovolných dat.
- Plně duplexní spojení, současný obousměrný přenos dat.
- Rozlišování aplikací pomocí portů.

1.8. UDP

User Datagram Protocol poskytuje nespolehlivou transportní službu pro takové aplikace, které nepotřebují spolehlivost, jakou má protokol TCP. Nemá fázi navazování a ukončení spojení a už první segment UDP obsahuje aplikační data.

1.9. SCTP

Spolehlivý protokol pro přenos datagramů ve více proudech. Je využívaný zejména v telekomunikacích. Doplňuje některé vlastnosti, které TCP postrádá:

- Multihoming - komunikující uzel může mít několik IP adres.
- Členění datového toku na datagramy.
- Používání více proudů dat - omezuje blokování komunikace způsobené chybějícím blokem dat, ke kterému může dojít v TCP.
- Výběr a sledování cesty - Pokud má primární adresa problémy s dostupností lze používat alternativní.

2. ÚVOD DO JAZYKA HTML

2.1. Úvod do HTML

HTML je jednoduchý značkovací jazyk, kterým se tvoří webové stránky, tyto stránky jsou jen obyčejné textové soubory, které většinou obsahují nějaký text a pár html značek, které určují význam a vzhled jednotlivých částí stránky.

Při tvorbě www stránek se nepoužívá jen html, ale i další jazyky: css, php, javascript a další.

HTML je základ, až budete znát alespoň základy můžete se přiučit něco o css stylech ty určují vzhled stránek a později něco z php, ale to už je programování.

2.2. Historie HTML

Web vznikl v roce 1989 od této doby se neustále vyvíjí, aktuálně se pracuje na HTML 5.

2.3. Co je potřeba pro tvorbu html stránek

Jednoduchý textový editor, může se použít poznámkový blok (notepad), nebo speciální editory s podporou html kódu, tyto programy zvýrazňují syntaxy a umožňují rychle přepínat mezi kódem a náhledem, práce s takovým editorem je mnohem efektivnější než s obyčejným poznámkovým blokem.

2.4. Používané termíny v HTML.

Tag Základní značka html, zapis tagu: <tag>.

Atribut Zapisuje se přímo do tagu a nastavuje nějakou jeho vlastnost, zápis atributu: <tag atribut="hodnota">.

Element Zápis nadpisu: <h1>Nadpis stránky</h1>.

První stránka.

HTML stránky jsou obyčejné textové soubory obohacené o tagy.

Tagy

Tagy se zapisují mezi znaky < >, některé jsou párové a některé nejsou.

Zápis nepárového tagu:

```
<tag>
```

Zápis párového tagu:

```
<tag>Nějaký text</tag>
```

U párového tagu je důležíte v ukončovací značce napsat lomítko / jinak by to prohlížeč nepochopil.

Atributy

Atributy se zapisují přímo do tagu.

```
<tag atribut="hodnota">
<tag atribut="hodnota">Párový tag s atributem.</tag>
```

Zákaz křížení tagů.

Tagy se mohou do sebe vkládat, nesmějí se však křížit.

```
<b><i>Tučná kurzíva</b></i>
```

Správný zápis:

```
<b><i>Tučná kurzíva</i></b>
```

Velikosti písmen

V html na velikosti písmen nezáleží můžete tedy psát **<TAG>**, nebo **<tag>** je to jedno, ale v xhtml což je vlastně novější verze html se už musí psát tagy i atributy jen malými písmeny.

Pozor v **url** je nutné zachovat velikost písmen, např. SOUBOR.html nerovná se soubor.html.

2.5. HTML

HTML znamená Hypertext Markup Language, tedy hypertextový značkovací jazyk. Hypertext markup language se vyvinul ze SGML a stal se používaným jazykem pro tvorbu webových stránek. V historii se nejvíce používaly verze HTML 2.0, HTML 3.2, HTML 4.01 a HTML 5. Z HTML se vyvinulo také XHTML (extended hypertext markup language) jako aplikace XML, které osobně považuji za slepou vývojovou větev (vývoj mi dal za pravdu). V roce 2010 se začalo mluvit o používání HTML 5 a větší část jeho novinek už se v roce 2017 používá.

Struktura html souboru

Nejčastější "šablona" stránky:

```
<!DOCTYPE HTML>
  <html>
        <head>
              <meta charset=”windows-1250”>
              <title>Jméno</title>
        </head>
        <body>

samotný text stránky

        </body>
  </html>
```

2.6. Jazyky pro prezentaci webového obsahu - CSS

2.6.1. Historie CSS

CSS vzniklo někdy kolem roku 1997. Je to kolekce metod pro grafickou úpravu webových stránek. Ta zkratka znamená Cascading Style Sheets, česky "kaskádové styly". Kaskádové, protože se na sebe mohou vrstvit definice stylu, ale platí jenom ta poslední. To teď není důležité.

Existuje návrh CSS 2, vylepšené a složitější formy stylů, které ale v nejrozšířenějším prohlížeči Internet Exploreru moc nefungují.

Kdy používat CSS

V roce 2015 se dá říct, že už se celý web formátuje pomocí CSS. Ze starého HTML formátování zůstalo tak maximálně ztučnění a kurziva. Proto je dobré se v CSS orientovat, jestliže chcete dělat webové stránky. V první řadě je ale potřeba vědět, jak funguje HTML. Pokud HTML ani trochu neznáte, není dobré začínat s CSS. Kdy se vyplatí CSS studovat:

- chcete mít stránky hezky a moderně zformátované. Barvy, zarovnání, rozvržené sloupce atd.,
- často píšete texty určené pro Internet a nechcete ztrácet čas složitým formátováním,
- zabýváte se skriptováním, zejména Javascriptem,
- spravujete (či zatím jen plánujete) větší web s mnoha stránkami, které by měly vypadat podobně.

2.6.2. Další možnosti použití

Trojí použití CSS. Styl se může nadeklarovat třemi způsoby, níže uvádím příklady. Stačí, když se pro začátek naučíte jeden ze tří způsobů:

Přímo v textu zdroje u formátovaného elementu pomocí atributu style="...". Tomu říkám **přímý styl**. Je to nešikovné, ale občas se to používá.

Pomocí "**stylopisu**" (angl. "stylesheet") v hlavičce stránky. Stylopis je jakýsi seznam stylů. Je v něm obecně napsáno, co má být jak zformátováno, například že nadpisy mají být zelené. Do stránky se stylopis píše mezi tagy <style> a </style>.

Použitím externího stylopisu -- to je **soubor *.css**, na který se stránka odkazuje tagem <link>. V souboru je umístěný stylopis. Hlavní výhoda je v tom, že na jeden takový soubor se dá nalinkovat mnoho stránek, takže pak všechny vypadají podobně.

Samozřejmě stačí ovládnout jenom jeden způsob.

Příklady

Chci udělat odstavec červeným písmem pomocí CSS. Jak už jsem popsal, jde to třemi způsoby:

Přímý zápis

Do zdroje se napíše tato deklarace odstavce:

```
<p style="color: red">Tento odstavec bude červený.</p>
```

Vysvětlení: <p> je značka vymezující odstavec; z anglického paragraph. Atribut "style" je obecný atribut použitelný u každého prvku. Color znamená barva a red je červená.

Stylopisem

Do hlavičky dokumentu se napíše stylopis uzavřený mezi tagy <style></style>:

```
<style>
p    {color: red}
</style>
```

a do těla stránky se mohou psát odstavce:

```
<p>Tento odstavec bude červený. </p>
<p>Tento mimochodem také, protože červené budou všechny.</p>
```

To, jak zařídit, aby nebyly červené všechny, ale jenom některé odstavce, se dá pomocí "tříd" a "identifikátorů".

Externím CSS souborem

Vytvoří se soubor, který se pojmenuje třeba *styly.css*. V něm bude pouze tento text:

```
p {color: red}
```

Do hlavičky html dokumentu, který chci stylem ovlivnit, musím napsat odkaz na tento soubor:

```
<link rel="stylesheet" type="text/css" href="styly.css">
```

V těle dokumentu pak budou opět všechny odstavce červené.

2.6.3. Syntaxe

CSS nejsou součástí HTML, a tak se zapisují zcela jiným způsobem, jak už jste si možná všimli. Pokud vám tabulka přijde příliš teoretická, všimněte si pouze příkladů ve spodní části.

Přímý styl:	<tag style="zápis vlastností">stylovaný element</tag>
Ve stylopisu:	<style> tag {zápis vlastností} 2.tag {zápis vlastností} </style>
Zápis vlastností zjednodušeně:	vlastnost: hodnota; 2.vlastnost: 2.hodnota
Zápis vlastností obecně:	vlastnost: hodnota [, hodnota2] [; další zápis vlastností]

Příklady:

Příklad přímého stylu	text červeného odstavce
Příklad stylopisu	<style> p {color: red} body {background-color: yellow;} </style>
Příklad jednoduchého zápisu vlastností	color: red
a složitějšího zápisu vlastností	font-family: Arial, Arial CE, sans-serif; color: red;

Je nutné všimnout si, kde se používají uvozovky, dvojtečky, složené závorky, středníky a čárky. Příklad správného zápisu:

```
h2 {color: green; background-color: yellow}
```

Mezery a konce řádků příliš velkou roli nehrají, mohou se přidávat a vypouštět. Velikost písmen nehraje roli. K dispozici je seznam vlastností a jejich hodnot.

Hodnoty, které prohlížeč nezná, ignoruje.

Komentáře ve stylopisech se dělají podobně jako v Javě mezi znaky /* a */. Dvě lomítka nefungují.

Příklad s nadpisem

Ve stylopisu nebo v externím css souboru se to dá udělat docela snadno.

```
<style>
  h1{color: green;}
  h2 {color: blue;}
</style>
```

Potom v celém dokumentu budou nadpisy první úrovně zelené a nadpisy druhé úrovně modré. To ovšem pouze za předpokladu, že při psaní textu byly použity tagy <h1> a <h2>. Jinými slovy, stylopisy se dají efektivně použít pouze u dobře strukturovaných textů.

2.6.4. CSS styly

CSS styly jsou kaskádovací styly, používají se k vytvoření stylu webové stránky (barva, písmo, velikost písma). S CSS styly můžete pomocí jednoho souboru ovlivňovat design celého webu.

Zastaralé metody

Před CSS styly se k stylu stránky používal prvek <font face="písmo" color="barva" size="velikost">, který už byl překonán a zavrhnut. Oproti CSS stylu má nevýhody:

- Pokud jste často měnili styl textu, objevoval se tento tag ve zdrojovém kódu velmi často a to značně zpomalovalo běh stránky.
- Umožňuje měnit pouze písmo, barvu a velikost: <font face="písmo" color="barva" size="velikost">

2.6.5. DHTML

Možná už jste se někdy setkali se zkratkou DHTML, nebo Dynamické HTML - věřte, že tento jazyk je tvořen takřka jen z JavaScriptu, VBScriptu (jazyk s podobnými vlastnostmi jako JavaScript) a CSS styly. Tento jazyk využívá síly HTML, JavaScriptu a CSS a vytváří tak dokonalý design a stránky na které se dá dívat.

2.6.6. CSS styly a třídy, identifikátory a style

CSS - Cascading Style Sheets - kaskádovací styly, poprvé implementovala společnost Microsoft v roce 1996 do Internet Exploreru 3.0. CSS styl zcela nahrazuje prvek <font> a zavádí prvek <style>. Pomocí CSS stylů můžete definovat kromě barvy, písma a velikosti spoustu dalších věcí (rámeček, podtržení, tučnost, vlnitost, zobrazení, odrážky, okraje..)

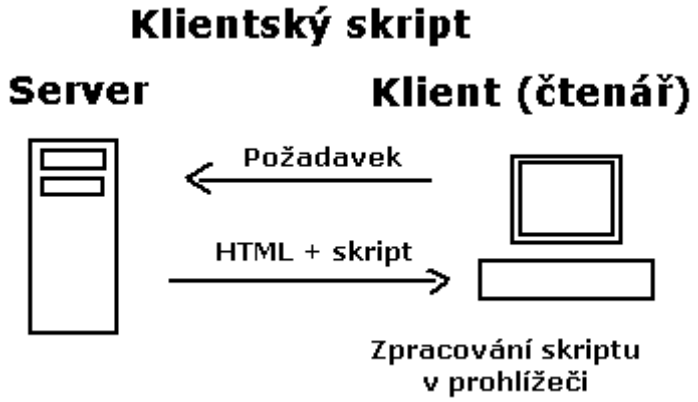
CSS styly se aplikují hlavně pomocí tříd a identifikátorů. Ty umožňují tvorbu CSS stylu jediným atributem a vy tedy nemusíte opakovat stejný kód desetkrát. Kromě toho můžete také definovat styl prvkům (h1, p, table at.) pomocí selektorů. Např. každý prvek <input> bude mít vždy červený text - to je možné udělat jediným řádkem CSS.

3. LOGIKA NA STRANĚ KLIENTA - JAVASCRIPT

3.1. Co je JavaScript

JavaScript je programovací jazyk, který se používá v internetových stránkách. Zapisuje se přímo do HTML kódu, což je velká výhoda, protože je to jednoduché.

JavaScript je klientský skript. To znamená, že se program odesílá se stránkou na klienta (do prohlížeče) a teprve tam je vykonáván. (Protikladem klientských skriptů jsou skripty serverové, které jsou vykonávány na serveru a na klienta jdou už jen výsledky.)



Existují i jiné jazyky klientských skriptů, například VBScript. Jsou ale tak málo používané, že když se dnes mluví o "skriptech", myslí se tím JavaScripty.

JavaScript není Java

JavaScript je často zaměňován s Javou. Java je samostatný programovací jazyk. Má s JavaScriptem pouze podobnou syntaxi.

Co je potřeba umět

- HTML, alespoň základy
- základy programování, alespoň trochu

3.2. Charakteristiky jazyka

JavaScript je jazyk:

- interpretovaný -- nemusí se kompilovat
- objektový -- využívá objektů prohlížeče a zabudovaných objektů
- závislý na prohlížeči -- funguje ale ve většině prohlížečů
- case sensitivní -- záleží na velikosti písem v zápisu
- syntaxí podobný jazykům C, Java a podobným
- Omezení jazyka
- JavaScript funguje pouze v prohlížeči.
- Uživatel může JavaScript zakázat
- Existují různé odlišné verze jazyka i prohlížečů, což vede k častým chybám.
- Neumí přistupovat k souborům (kromě cookies) ani k žádným systémovým objektům.
- Neumí žádná data uložit (kromě cookies).
- To vše z něj dělá pouze jazyk druhořadý, účelově použitelný pouze v HTML stránkách.

Jak se učit JavaScript

Po zvládnutí základů je nejlepší všímat si cizích skriptů na cizích stránkách. Většina skriptů je zapsána přímo ve zdrojovém kódu stránky, takže se dá zkopírovat (některé kódy jsou v externích souborech, ale i ty jsou stáhnutelné).

3.2.1. Vysvětlení skriptu

Skript se zapisuje do HTML mezi tagy <script> a </script>. Všechno, co je mezi těmi tagy, je program psaný v jazyce Javascript.

V příkladě je použit příkaz **document.write()**. Ten způsobuje normální zápis do proudu dokumentu. Zapsaný text se ihned zobrazí v prohlížeči.

Pokud se zapisuje normální text, musí se obalit uvozovkami (na rozdíl od proměnné). Mezi uvozovkami se nesmí zalomit řádek.

Každý příkaz JavaScriptu se ukončuje středníkem nebo stačí zalomit řádek.

Jak vytvořit první skript

Vše, co vytvoříte v JavaScriptu se nazývá skript. Ten můžete volně umístit do stránky, nebo na něj vytvořit odkaz - stránka pak sama natáhne do stránky JavaScript. Samostatné soubory psané v JavaScriptu mají přípony .js nebo .jse. Přípona .js je obvyklejší. K vytváření skriptů stačí editor zdrojového kódu (PSPad, textový editor, nebo libovolný editor HTML).

K prohlížení potřebujete prohlížeč prohlížeč (nejlépe alespoň Internet Explorer a Mozilla Firefox, abyste skripty mohli kontrolovat v těchto nejčastěji používaných prohlížečích).

Vkládání skriptu do stránky

A teď prakticky. Skript píšeme mezi značky <script> a </script>. Ty můžete vložit do sekce body nebo do sekce head (záleží na účelu skriptu).

```
<html>
  <head>
        ...
        <script type="text/javascript">
             javascript script body
        </script>
        ...
  </head>
  <body>

document body
  <script type="text/javascript">
        javascript script body
  </script>

document body
  </body>
</html>

Tag <script>
```

Syntax of the tag <script> is the following:

```
<script type="text/javascript" src="url of external file">
<!--
        javascript script content
//-->
<script>
```

Atribut type označuje typ skriptu (v případě JavaScriptu "text/javascript"). Protože existují prohlížeče, které nemusí rozumět JavaScriptu, je vhodné zapsat na začátek skriptu <!-- a na konec //-->, jinak by došlo k tomu, že by prohlížeč vypsal skript jako normální text (nyní ho bude považovat za komentář a neukáže ho).

3.2.2. Zápis skriptu

JavaScript netoleruje záměnu velkého písmena za malé (je case-sensitive), proto document.write není to samé jako DOCUMENT.write. Toto pravidlo je nezbytné dodržovat, jinak skript nebude fungovat.

Tím jsme si prošli úvod do jazyka a můžeme se pustit do praktických příkladů. Když si jich pár vyzkoušíte, bude vše jasnější.

JavaScript je multiplatformní, objektově orientovaný skriptovací jazyk, jehož autorem je Brendan Eich z tehdejší společnosti Netscape.

Nyní se zpravidla používá jako interpretovaný programovací jazyk pro WWW stránky, často vkládaný přímo do HTML kódu stránky. Jsou jím obvykle ovládány různé interaktivní prvky GUI (tlačítka, textová políčka) nebo tvořeny animace a efekty obrázků.

JavaScript byl původně obchodní název implementace společnosti Netscape, kde byl vyvíjen nejprve pod názvem Mocha, později LiveScript, ohlášen byl společně se společností Sun Microsystems v prosinci 1995 jako doplněk k jazykům HTML a Java. Pro verzi firmy Microsoft je použit název JScript. Ten je podporován platformou .NET.

Program v JavaScriptu se obvykle spouští až po stažení WWW stránky z Internetu (tzv. na straně klienta), na rozdíl od ostatních jiných interpretovaných programovacích jazyků (např. PHP a ASP), které se spouštějí na straně serveru ještě před stažením z Internetu. Z toho plynou jistá bezpečností omezení, JavaScript např. nemůže pracovat se soubory, aby tím neohrozil soukromí uživatele.

JavaScript je možné použít i na straně serveru. První implementací JavaScriptu na straně serveru byl LiveWire firmy Netscape vypuštěný roku 1996, dnes existuje několik možností včetně opensource implementace Rhinola založené na Rhino, gcj, Node.js a Apache.

Kromě DHTML se JavaScript používá k psaní rozšíření pro mnohé aplikace, například Adobe Acrobat.

JavaScript je také možno spouštět v operačních systémech Windows pomocí programu Windows Script Host a nahradit tak dávkové soubory MS-DOS.

4. ARCHITEKTURA WEBU

4.1. Návrh architektury webu

Web nikdy nezačínáme tvořit grafickým návrhem. Na začátku nejprve definujeme cíle jednotlivých stránek a pak to, kde se jaké informace budou na webu nacházet.

Proč je to důležité?

Díky návrhu architektury webu si můžeme lépe představit, jak bude výsledný web vypadat a jaké budou jeho funkce.

Díky tomu můžeme lépe odhalit a odstranit případné nedostatky. Tím ušetříme spoustu peněz, které by mohla stát pozdější oprava naprogramované aplikace.

Návrh struktury webu

V návrhu struktury vydefinujeme všechny stránky webu, jejich vzájemné vazby a obvyklé uživatelské scénáře. Dbáme na to, aby byly informace vhodně strukturované, dobře použitelné a vedly k naplnění cílů.

Návrh navigace webu

Správně zvolená navigace je důležitou součástí webu, která pomáhá návštěvníkovy stránek rychle nalézt to, co hledá. Tomuto bodu věnujeme při návrhu webu velikou péči.

Rozkreslení modelů webu (wireframe)

Jakmile známe obsah webu, tak se můžeme pustit do kreslení drátěných modelů webu (wireframe), které jsou základem pro grafický návrh webu.

Jazyk PHP

PHP je jedním z nejvíce rozšířených programovacích jazyků používaných k vytváření webových aplikací. PHP se používá na straně serveru a slouží tedy ke generování HTML/XHTML kódu stránky, jenž pak server odesílá do prohlížeče (na rozdíl od klientského JavaScriptu, který funguje až při zobrazení stránky v prohlížeči).

Hlavním kladem PHP je jeho nezávislost na platformě (Windows, Linux, Unix…), mezi výhody PHP patří i široké možnosti použití. PHP například umí pracovat se soubory a s mnoha různými databázemi, s PHP lze generovat a upravovat grafiku, umí odesílat a přijímat emaily, vytvářet PDF, podporuje všechny důležité internetové protokoly…

Protože má PHP poměrně volnou syntaxi (způsob zápisu), snadno se učí, zejména pokud již máte zkušenosti s jinými programovacími jazyky. Společně s webovým serverem Apache a databází MySQL tvoří PHP tzv. triádu, trojici programů nejčastěji používaných pro generování stránek. Z toho plyne i další výhoda PHP – na internetu existuje obrovské množství fragmentů, uživatelsky definovaných funkcí a hotových řešení obvyklých problémů.

4.2. Programování webových aplikací

Jako **webová aplikace** se obvykle označuje skript (kód zajištující funkci programu) běžící na straně serveru. Často bývá propojen s některou z databází, systému uchovávajícího data webové aplikace (zjednodušeně si lze databázi představit jako soubor MS Excel). Výstupem skriptu je potom samotná webová stránka, která je předána k zobrazení prohlížeči.

Úkolem webových aplikací je většinou zvýšit interakci internetové prezentace s jejími návštěvníky, případně též usnadnit správu webu, tj. ušetřit opakující se práci při tvorbě www stránek.

Podle požadavků na funkčnost může být webovou aplikací pouhých několik řádků kódu (např. při odesílání konkaktního formuláře), výjimkou však nejsou ani webové aplikace o mnoha tisících řádcích.

Složitější webové aplikace bývají často propojeny i na další software uvnitř firmy, např. na objednávkové systémy, účetní programy atd. To umožňuje ještě efektivněji šetřit drahou lidskou práci. Problémem nemusí být ani napojení aplikace na online platební systémy.

Příklady jednodušších webových aplikací

- Kontaktní formulář
- Kniha návštěv
- Diskuze nebo chat
- Katalogy a ceníky
- Různé slovníky
- Bannerový systém

U větších prezentací se často vyplatí řešit rovnou celou prezentaci dynamickým způsobem, ať už pomocí šablon či rovnou nasazením na redakční systém. Specifickou formou takové webové aplikace je potom internetový obchod. Pro své mnohé výhody se také stále více rozmáhají weblogy, taktéž různé intranety a extranety patří mezi speciální druhy webových aplikací.

Při programování všech webových aplikací dbáme především na tyto aspekty:

Bezpečnost – má prvořadý význam u jakékoliv webové aplikace, vždy hrozí riziko ztráty či zničení dat, nebezpečí číhá i v podobě krádeže informací nepovolanou osobou nebo v nabourání webového serveru skrze aplikaci (hrozba ztráty image společnosti).

Využívání dostupných zdrojů – při programování každé webové aplikace se snažíme používat již hotové úseky kódu z jiných zdrojů, k tomuto účelu vlastníme i rozsáhlý archív skriptů. Šetříme si tak práci potřebnou na vývoj aplikace a tím i vaše finance a celkový čas na realizaci zakázky.

Rozšiřitelnost – jakmile se webová aplikace osvědčí v praxi, začíná se obvykle pracovat na jejích dalších úpravách, vylepšeních a nadstavbách. Pokud bylo s těmito modifikacemi počítáno už při návrhu aplikace, jejich zapracování je mnohem jednodušší a tedy i levnější. I z tohoto důvodu se snažíme řešit většinu složitějších webových aplikací modulárním způsobem.

Rychlost – pomalá webová aplikace je jen velmi málo použitelná, problémy s ní mají i vyhledávače. Proto se snažíme optimalizovat všechny skripty na rychlost a proto také doporučujeme nasadit hotovou aplikaci na naše servery. Protože zde zároveň máme instalované nejmodernější vývojové nástroje, jaké na mnoha serverech pro komerční webhosting chybí, i samotný vývoj webové aplikace se tím urychlí a zlevní.

Maximální zátěž webové aplikace je pojem spojený s vysokou návštěvností. Pokud váš web navštíví současně větší množství návštěvníků (např. při uvedení očekávaného produktu), server je často nezvládne všechny obsloužit (říkáme, že server „spadl‘). Schopnost webové aplikace odolávat vysoké zátěži vyžaduje především volbu vhodných nástrojů, optimalizaci databáze i výpočtů a další speciální techniky, jako je například *předkešování*. Podle našich zkušeností právě malá odolnost vůči zátěži mnohdy vypovídá o nízké úrovni programátorů, kteří webovou aplikaci vytvořili.

Důkladné testování – před spuštěním webové aplikace vždy důkladně testujeme všechny její funkce na vývojovém serveru. Ten mívá stejnou konfiguraci jako server ostrý, což umožňuje další redukci případných problémů při samotném spuštění.

4.3. Technologie našich webových aplikací

Při programování webových aplikací bývá dnes po celém světě nejčastěji využíván skriptový jazyk PHP a databáze MySQL. Tato kombinace, společně s webovým serverem (programem) jménem Apache nazývaná **triáda**, se i nám osvědčuje pro svou flexibilitu. Mezi další výhody této sestavy patří široká dostupnost hotových funkcí a fragmentů kódu a také neustálý vývoj těchto programů.

Jsou-li k tomu důvody z hlediska požadované funkčnosti webové aplikace nebo zjednodušení jejího vývoje, používáme i další programovací jazyky, např. Perl, Python či databázi PostgreSQL. Každý z nich se hodí na určité specifické potřeby konkrétní webové aplikace.

Největším kladem všech zmiňovaných technologií je jejich zařazení k open source. Znamená to, že jsou **šířeny zdarma** (což je také jedním z významných faktorů jejich dosavadní úspěšnosti a rozšířenosti) a vy tedy za samotné využívání těchto produktů nic neplatíte.

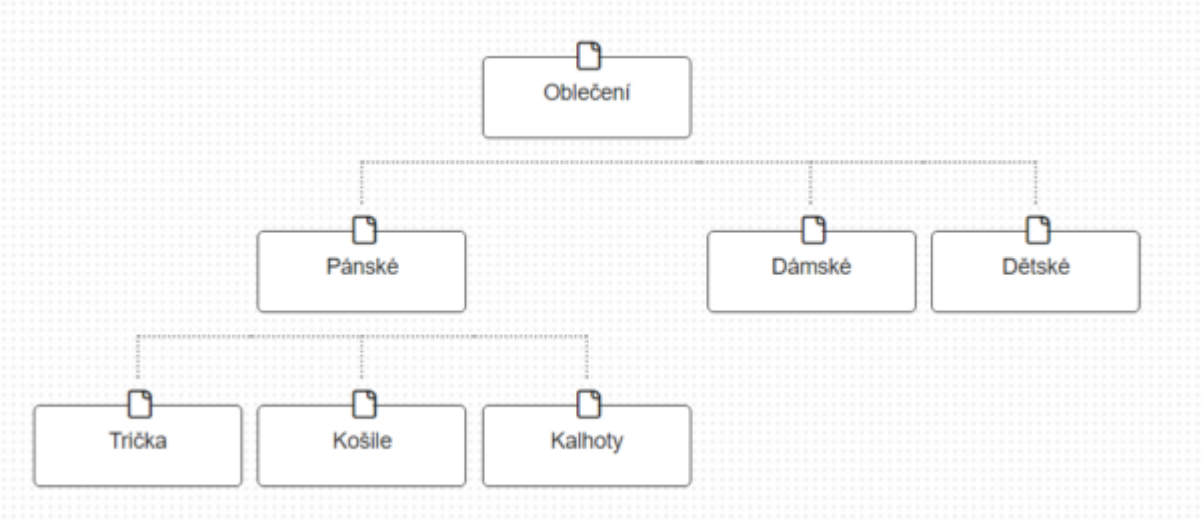
4.4. Co to je architektura webu?

Informační (obsahová) architektura webu je způsob, jakým poskládáte informace na webu do logického celku.

Je jasné, že všechny informace nebudete mít na jediné stránce, ale že web bude obsahovat více stránek. A tyto stránky jsou uspořádané do různých úrovní. Od toho obecného po podrobné. A vše je tematicky provázané.

Pokud například přijdete na web prodávající oblečení, tak se budete do webu zanořovat nějak takto: pánské oblečení > košile > s krátkým rukávem. Způsob, jakým jsou informace na webu uspořádané, se nazývá informační architektura webu.

Více naznačuje tento obrázek:



Proč řešit architekturu webu

Dobrá architektura webu je důležitá z několika důvodů:

- Vede návštěvníka webu rychle a intuitivně k cíli – tedy k obsahu, pro který si přišel.
- Tvoří logiku celého webu, kterou vnímají návštěvníci i vyhledávače.
- Umožňuje uspořádat obsah webu tak, aby byl snadno dohledatelný ve vyhledávačích.
- Jestliže připravujete nový web, nebo e-shop, obsahovou (informační) architekturu webu byste měli řešit v rámci základního návrhu webu (před grafickým návrhem a realizací webu – viz 4 kroky profesionálního webdesignu).
- Kvalitní architektura webu zajistí, že na něm budete mít správně uspořádaná všechna důležitá témata. To se odrazí nejenom v lepší orientaci, ale také v lepší dohledatelnosti webu na konkrétní klíčová slova ve vyhledávačích. A jak je známo – návštěvnost z vyhledávačů je zadarmo a je vysoce relevantní.

4.4.1. Jak na dokonalou architekturu

V rámci MD webdesign tvoříme architekturu webu na základě analýzy klíčových slov. To znamená, že zjistíme, jaká klíčová slova lidé používají ve vyhledávačích Google a Seznam. Analyzujeme jejich hledanost, tedy důležitost. A klíčová slova pak rozřadíme do kategorií, abychom získali větší nadhled.

Platí, že pokud chcete být na nějaké klíčové slovo dohledatelní ve vyhledávači, je vhodné mít na webu stránku věnovanou tématu klíčového slova a klíčové slovo samotné.

Díky analýze klíčových slov získáte přehled o tom, co uživatele vyhledávačů zajímá. A zařazením těchto témat (klíčových slov) do architektury webu vám výrazně zvýší šance na získání dalších pozic ve vyhledávání.

Když už vytvoříme informační architekturu, popíšeme i obsah jednotlivých stránek. Samotný obsah webu navrhujeme na základě analýzy zákazníka/klienta formou tzv. person (více o personách na Wikipedii). Díky tomu můžeme navrhnout takový obsah, který odpovídá očekávání konkrétních zákazníků. A jsme schopni mu dodat takové informace, o které stojí. A co nejdříve odbourat jeho/její obavy před provedením poptávky/objednávky.

Tvorba webových stránek: začínáme informační architekturou

Informační architektura je plán pro třídění informací na webu. Pro správnou funkci budoucích webových stránek je zapotřebí věnovat nemalé úsilí vytvoření srozumitelné informační architektury.

Stejně jako architekt při návrhu domu pracuje s prostorem, světlem a tvary, informační architekt pracuje s informacemi, strukturou a prioritou.

Obecně platí, že správně navržená informační architektura by měla být pochopitelná i "nahá", tedy tak, jak jsme ji vytvořili v textové podobě. Barvičky ani šipky návštěvníkovi při průchodu webem nepomohou, pokud informační architektuře webu nerozumí.

Mezi hlavní důvody "ztracení se návštěvníka na webu" patří: - použitá terminologie - uspořádání vyžadující profesní znalost - nelogické uspořádání informací.

Abychom se výše uvedeným problémům vyhnuli, měl by návrh informační architektury zahrnovat tyto kroky:

- Určení cílové skupiny
- Shromáždění informací
- Seskupení informací
- Určení priorit
- Vytvoření informační architektury

Kroky postupně:

1. <u>Určení cílové skupiny</u>

Je důležité si na otázku, kdo je cílová skupina webu, odpovědět dříve, než začneme navrhovat informační architekturu, protože dramaticky změní náš pohled na zveřejňované informace.

U webů veřejné správy je cílovým uživatelem prakticky každý, stěží proto budeme schopni definovat znalosti nebo zvyky, které bude většina uživatelů sdílet. Můžeme ale předpokládat, které znalosti uživatelé mít nebudou.

Praxe ukazuje, že uživatelé, kteří přistupují na webové stránky měst a obcí, zpravidla neznají rozdíl mezi městem a úřadem. Neznají úřední terminologii a stěží se vyznají v organizační struktuře úřadu. Situaci nepomáhá skutečnost, že totožné agendy jsou v

různých městech spravovány různě nazvanými odbory.

Plyne z toho, že interní struktura úřadu není vhodnou inspirací pro tvorbu informační architektury, kterou chcete prezentovat laické veřejnosti.

2. Shromáždění informací

Druhým krokem při návrhu informační architektury by mělo být ujasnit si, jaké informace bude návštěvník na webu hledat. Veřejná správa musí řadu informací zveřejňovat ze zákona, ale zdaleka to nejsou všechny informace, které by se na webu měly vyskytovat.

Weby veřejné správy zpravidla obsahují obrovské množství informací, od zápisů jednání zastupitelstev, přes vyhlášky, rozpočty, nařízení, rozhodnutí až po aktuální dění v obci a nabídku kulturních či sportovních akcí.

Součástí tohoto kroku je i rozhodnutí, které informace na webu zveřejňovat nechceme. Je skutečně nutné na webu úřadu zveřejňovat kalendář akcí nebo seznam místních restaurací? Vždy je třeba zvážit, které informace bude provozovatel webu schopen pravidelně aktualizovat. Problémům s neaktuálními informacemi je dobré předejít již při návrhu informační architektury.

3. Seskupení informací

Když víme, které informace chceme na webu prezentovat, je na čase vytvořit tematické celky. Tyto celky by měly pojmout veškeré informace, které jsme nashromáždili v předchozím kroku, a to pokud možno bez přesahů.

V tuto chvíli je vhodné do procesu zapojit alespoň jednoho zástupce laické veřejnosti. Jako pracovník úřadu máte jasno ve struktuře úřadu a v procesech, kterými se řídí. Pro laika je však tento svět cizí a je tedy nepraktické po návštěvníkovi pro úspěšný průchod webem požadovat vnitřní znalost úřadu.

Návštěvník postupuje webem “shora” – začne na nejobecnější úrovni a postupně pomocí dalších klikání svůj dotaz upřesňuje, až dosáhne konkrétního výsledku. Při návrhu informační architektury však postupujeme “zdola” – třídíme a kategorizujeme jednotlivé informace do celků a skupin. Tento rozpor může velmi snadno návštěvníka zavést do slepých uliček. Návštěvník nezná obsah webu a neví, zda hledaná informace na webu vůbec existuje. Je proto klíčové otestovat srozumitelnost informační architektury při průchodu “shora”.

4. Určení priorit

Doposud jsme si vystačili s trochou selského rozumu a testování s uživateli. Při určení priorit v návrhu informační architektury se však pouštíme do analytické části a je dobré mít v ruce konkrétní čísla, například analýzu nejčastěji vyhledávaných výrazů na webu.

Priority se navíc mohou měnit v čase, například v létě bývá častý dotaz na koupaliště, zatímco v zimě uživatele zajímá především úklid sněhu.

Způsobů upřednostnění informací je celá řada: informaci můžeme umístit na přední místa ve struktuře, případně umístit ji přímo na titulní stránku nebo hlavní stránku tematického celku. Dále lze použít vizuální zvýraznění za pomoci kontrastu, barev, velikosti atd.

Například na stránce komunálního odpadu by mohla být informace o aktuální výši a datu splatnosti poplatku, aniž by návštěvník byl nucen procházet PDF s vyhláškou. A protože se jedná o často hledanou informaci, můžeme na ni navíc upozornit větším fontem.

5. Vytvoření informační architektury

Při vytváření informační architektury může pomoci vhodný nástroj, ale žádný nástroj není samospásný. Informační architektura ve své nejjednodušší podobě připomíná strom.

5.PHP /BASICS/

PHP is a programming language working on the side of server. PH/ enables to store and change website data. Původní význam zkratky PHP byl Personal Home Page. Vzniklo v roce 1996, od té doby prošlo velkými změnami a nyní tato zkratka znamená PHP: Hypertext Preprocessor.

Možnosti PHP

PHP není nijak těžké pochopit a už se základy si lze vystačit. Umí ukládat, měnit a mazat data. Vše se odehrává na webovém serveru (kde jsou uloženy zdrojové kódy webových stránek). PHP skript se nejprve provede na serveru a potom odešle prohlížeči pouze výsledek (znamená to, že nejprve spočítá kolik je 300/30 a pak prohlížeči odešle jen číslo 10). Proto ve zdrojovém kódu najdete jen "10" (to je rozdíl oproti JavaScriptu, který počítá přímo v prohlížeči). Zdrojový kód PHP narozdíl od JavaScriptu a HTML v prohlížeči nezobrazíte.

Pomocí PHP je možné vytvořit diskuzní fórum, knihu návštěv, počítadlo, anketu, graf a dokonce si pomocí jednoduchého kódu můžete zlikvidovat celý obsah webu. Navíc máte možnost propojit vaše stránky s databázemi, např. MySQL.

K čemu PHP?

Je rozhodně alespoň jedna funkce PHP, která se hodí snad do každého webu. Na webových stránkách se obvykle opakují některé části, hlavička s odkazy, menu, patička. S PHP si můžete snadno vytvořit šablonu pro web, do které se budou vkládat soubory s menu, patičkou atd. Můžete tedy mít menu jen jednou zapsané a do dalších stránek ho pouze kopírovat. Až budete chtít menu změnit, bude to nesmírně jednoduché. Více v článku PHP menu.

Soubory PHP

Webová stránka s prvky PHP má nejčastěji koncovku .php. Lze se však setkat i s dalšími koncovkami, např. .phtml, php3, php4, php5. Některé hostingy dle koncovky určovaly, pod jakou verzí PHP skript spustit (aktuální je 7). To je však velice výjimečné a v naprosté většině případů byste si měli vystačit s koncovkou .php.

Instalace

PHP je jazyk, který si nevystačí jen s prohlížečem určité verze (třeba jako HTML nebo JavaScript), ale je nutné ho na počítač nainstalovat. Základ tvoří webový server a knihovny. K podpoře PHP je třeba instalovat a konfigurovat server, obvykle Apache. Nejlepší je využít k instalaci PHP program PHP Triad, který vše sám nainstaluje.

Webhosting s PHP

Ne každý webhosting zahrnuje podporu PHP. Potřebná podpora je u webhostingu nadstandardní službou za příplatek. Nicméně lze sehnat webhosting zdarma s podporou PHP (např. Webzdarma.cz, PHP 5). Při výběru webhostingu pro PHP stránky si pečlivě přečtěte, co nabídka zahrnuje.

5.1. PHP – základní informace

Dynamické stránky (tj. stránky nejprve vygenerované serverem a poté odeslané klientovi) jsou v současné době nezbytnou součástí každé složitější internetové prezentace. K hlavním skriptovacím jazykům, které se používají k tvorbě těchto stránek, patří ASP (Active Server Pages) a PHP. Právě úvodu do jazyka PHP je věnován tento seriál.

Před časem již na Intervalu úvodní článek o PHP vyšel, takže zde je jen přehled základních informací o tomto jazyce:

Co to je PHP?
PHP je skriptovací jazyk vykonávaný na straně serveru vkládaný do běžného HTML kódu. Co to znamená? Každou stránku, která obsahuje PHP skripty, server nejprve vezme a vykoná všechny příkazy v PHP, které jsou ve stránce uvedené, poté pošle klientovi již čistý HTML kód, který je výsledkem běhu skriptu. Server může PHP skripty teoreticky hledat ve všech odesílaných souborech, ale zpravidla je nakonfigurován tak, aby je hledal v souborech s příponami .php, .php3 nebo .phtml. Příkazy PHP jsou vkládány přímo do HTML kódu a jsou od něj odděleny tagy <? a ?> (nebo <?php a ?>).

K čemu je PHP dobré?

PHP je velmi všestranný jazyk, ve kterém lze relativně snadno naprogramovat třeba zpravodajský server nebo virtuální obchod. Data si lze ukládat buď do obyčejných textových souborů, nebo do databáze (PHP si dobře rozumí s většinou běžně používaných databází, jmenujme alespoň populární mySQL). Hračkou je zpracovávání dat z formulářů, snadno vytvoříte různé on-line testy včetně statistik úspěšnosti dosavadních návštěvníků nebo naprogramujete kvalitní reklamní systém. O síle PHP svědčí jeho používání na serverech Email.cz, Centrum.cz či BillBoard.cz

Co je k tvorbě v PHP potřeba?

PHP se vkládá do HTML, takže k tvorbě PHP skriptů obvykle postačuje jakýkoliv běžný HTML editor. Já osobně při tvorbě jak HTML, tak PHP skriptů zcela vystačím s Poznámkovým blokem (Notepadem). Nejdůležitější je ale mít vytvořené skripty kde umístit a vyzkoušet. K tomu musí být na serveru nainstalována podpora PHP (PHP – nyní

ve verzi 4 – je zdarma ke stažení na www.php.net, zde je i podrobný postup instalace). Nejvýkonnější je PHP jako modul serveru Apache pod operačním systémem Linux, ale lze ho používat i pod Windows – tato informace vás samozřejmě zajímá jen tehdy, pokud se staráte o svůj server sami. Pokud používáte některou z mnoha nabídek webhostingu zdarma, není podpora PHP moc pravděpodobná, nicméně např. server www.kgb.cz webhosting zdarma včetně podpory PHP (byť s jistými omezeními – nemožnost používání databází atd.) poskytuje. Máte-li placený webhosting, zeptejte se na PHP svého providera – je slušná šance, že podporu PHP buď zdarma, nebo za určitý příplatek nabízí. Provider by vám také měl sdělit, jakou příponu máte požívat pro soubory obsahující PHP skripty (jak již bylo řečeno, většinou je to .php, .php3 nebo .phtml).

První skript v PHP

Zde je náš první PHP skript, který (jak originální :-) vypíše aktuální čas:

```
<html>
  <head>
        <meta http-equiv="Content-Type" content="text/html; char-
        set=windows-1250">
        <title>PHP – ukázka 1</title>
  </head>
  <body bgcolor="#FFFFFF" text="#000000">
        <center><font face="Arial CE, Arial" size="5">
              Aktuální čas: <?php echo Date („H:i:s"); ?>
        </font></center>
  </body>
</html>
```

Uložte ho na server třeba jako soubor ukazka1.php a poté se na něj podívejte přes nějaký prohlížeč. K tomu, aby skript pracoval, musí být nejprve interpretován serverem, tzn. nelze si ho prohlížet off-line ze svého harddisku.

Jak skript přesně pracuje?

Důležité je si všimnout, že jde v podstatě o klasickou HTML stránku, která navíc obsahuje jeden PHP příkaz, a to echo Date („H:i:s"), který je od okolního HTML kódu oddělen značkami <?php a ?>. Server nejprve vezme požadovaný soubor ukazka1.php, vidí, že má příponu .php, a proto ho nejprve prožene interpretem PHP a vykoná všechny příkazy – ty hledá právě mezi oddělovacími značkami <?php a ?>. Narazí na příkaz echo, který slouží k zápisu do výsledného souboru. Funkce Date vrací datum a čas, v závorce jsou její parametry, které v tomto případě určují formát, ve kterém má být čas zobrazen (hodiny,

sekundy oddělené dvojtečkami – více o funkci Date v dalších dílech). Následující středník slouží k oddělení více příkazů od sebe – v tomto případě je nadbytečný, protože příkaz je jen jeden, ale je dobré zvyknout si ho psát. Interpret PHP tedy namísto příkazu PHP zapíše do stránky aktuální čas a server poté stránku odešle návštěvníkovi. Pokud si dáte zobrazit zdroj výsledku skriptu, opravdu uvidíte pouze čisté HTML, žádné PHP. Z toho vyplývá další hezká vlastnost PHP – na rozdíl od client-side jazyků, jako je třeba JavaScript, se k vašemu pracně vytvořenému kódu nikdo nedostane a nemůže ho tak lacino okopírovat.

Shrnutí anebo co si je třeba pamatovat

- Příkazy PHP se vkládají do běžného HTML kódu, oddělují se značkami <?php a ?> (nebo pouze <? a ?>).
- Aby server věděl, že má v souboru hledat příkazy PHP, musí mít soubor správnou příponu, obvykle .php, .php3 nebo .phtml.
- Návštěvník od serveru dostane pouze čistý HTML kód (tedy kód po vykonání všech PHP příkazů).

6.ZPRACOVÁNÍ HTTP DOTAZU

6.1. Metody dotazu

GET

je nejpoužívanější metoda. Slouží k vyzvednutí objektu (html soubor, obrázek, cokoliv...) ze serveru. Odpověď je „kešovatelná“. Proto GET dotaz doprovází spoustu hlaviček ve kterých se specifikuje, jak je dokument starý, zda byl modifikován atd. GET dotaz obvykle nemá tělo.

POST

Pomocí této metody se dají v těle dopravit na server informace od uživatele (velmi často se POST používá pro odeslání rozsáhlejších dat z webových formulářů, pro upload souboru a podobně).

HEAD

se chová naprosto stejně jako GET, ale v odpovědi se nepřenáší tělo. Tento dotaz se hodí například ke zjištění, zda objekt existuje (při kontrole odkazů na stránce).

PUT/DELETE

vytvoří/smaže daný objekt ze serveru. Tyto metody se v praxi příliš nevyužívají.

OPTIONS

slouží ke zjištění informací o daném kontextu (nebo „*“ pro celý server). Klient může zjistit, které dotazy může na daný kontext zaslat.

OPTIONS * HTTP/1.1

Host: www.root.cz

Ukázka implicitního nastavení serveru

TRACE

se používá ke sledování cesty celého dotazu. V těle odpovědi klient dostane pěkně seřazené všechny dotazy jednotlivých systémů, kterými požadavek procházel. Tato metoda je používaná administrátory a webovými programátory, kteří chtějí zjistit, proč jim server vrací například prošlý (expirovaný) dokument apod.

Hlavičky

Protokol HTTP verze 1.1 definuje velké množství hlaviček pro dotazy i odpovědi. Zde jsou některé z nich:

6.1.1. Dotazové hlavičky

Accept*

Hlavičky tohoto typu indikují, co všechno je schopen klient zpracovat. Server pak vybere nejvhodnější alternativu. Patří sem hlavičky Accept (MIME typy dokumentů), Accept-Charset (znaková sada, v českém prostředí velmi důležité), Accept-Encoding (kódování přenášených dat, většinou slouží k výběru komprese) a Accept-Language (jazyk dokumentu).

Connection

V protokolu HTTP 1.1 je definován parametr „close“, který požaduje okamžité uzavření spojení po přenosu prvního vyžádaného dokumentu.

Referer

Klient touto hlavičkou sděluje URI stránky, ze které byl odkaz vygenerován.

Host

HTTP 1.1 zavádí podporu tzv. name-based virtuálních serverů. Tato metoda umožňuje provozovat více virtuálních serverů na jediné IP adrese, klient ovšem musí pomocí této hlavičky specifikovat jméno serveru, s nímž chce komunikovat.

User-Agent

Touto hlavičkou by se měl klientský program identifikovat, ať už pro účely statistické či pro poskytování odlišného obsahu různým prohlížečům a podobně.

6.1.2. Hlavičky odpovědi

Content*

Hlavičky popisující obsah (tělo) odpovědi. Mohou obsahovat například délku obsahu (Content-Length), jeho MD5 digest (Content-MD5), jazyk (Content-Language), typ dokumentu (Content-Type) a další atributy. Nutno podotknout, že tyto hlavičky se nepoužívají pouze v odpovědích – pakliže obsahuje požadavek i tělo (např. při metodě POST), je obvykle nutné je rovněž použít.

Server

Tato hlavička slouží serveru k vlastní identifikaci (obvykle zde najdeme jeho jméno, verzi a někdy i další informace).

Expires

Server může prostřednictvím tohoto údaje sdělit, kdy vyprší platnost dokumentu. Po uplynutí této doby by si měl klient stáhnout novou verzi.

Existuje i řada dalších hlaviček, kterými může například klient řídit stažení dokumentu („stahuj pouze pokud byl dokument modifikován od …“) nebo třeba předat serveru uživatelské jméno a heslo pro přístup k neveřejným částem serveru. Podobně i server může jemněji popsat svou odpověď a například sdělit klientovi, kdy byl naposledy dokument modifikován nebo zda je povoleno jeho kešování ve veřejných či privátních keších.

6.2. Základní rysy protokolu HTTP

K úplnému pochopení článku budeme muset znát už trochu více základní rysy HTTP, a jak tento protokol vlastně pracuje (znalost předchozího článku také pomůže). Protokol HTTP je protokolem aplikační úrovně pro distribuované hypermediální informační systémy. V praxi to znamená, že tento protokol se obecně na internetu používá nejen pro přenos dat mezi klientem a serverem, ale i pro mnoho dalších úloh. Protokol HTTP je bezstavový, tj. že nerozpoznává klienty, od nichž chodí požadavky. Pokud jeden klient odešle požadavek a vzápětí ten stejný klient odešle další požadavek, server nepozná, že jde o stejného klienta.

HTTP existuje ve 3 verzích a to 0.9, 1.0 a 1.1 . První z nich, označována za HTTP/0.9 existovala jako jednoduchý protokol, který uměl v omezené podobě přenášet data na internetu. Verzí HTTP/1.0 nabyl protokol možnosti přenášení informací ve formátu MIME, takže mohl obsahovat i metainformace o přenášených datech. Nejpodstatnějším vylepšením protokolu verzí HTTP/1.1, což je zároveň poslední, aktuální verze, bylo to, že všechna spojení se stala trvalými. To znamená, že se spojení uzavře, až když jeden z dvojice klient-server odešle hlavičku pro uzavření. Dříve HTTP uzavíralo spojení po každé odpovědi serveru. Tímto vylepšením se také nesrovnatelně zvýšila rychlost přenosu, protože server už pro každý obrázek, rám a applet nemusí otevírat nové spojení.

6.2.1. Formát požadavku protokolu HTTP

Požadavek protokolu HTTP má následující formát:

METODA URL-DOKUMENTU VERZE-HTTP

HLAVIČKY

prázdný řádek

DALŠÍ-DATA Pouze u metody POST

Metoda požadavku značí způsob, jakým má server požadavek zpracovat. O metodách si více povíme později. Hlavičky se odesílají v následujícím formátu:

JMÉNO-HLAVIČKY: HODNOTA-HLAVIČKY

Každá hlavička musí být na samostatném řádku. Všechny řádky musí být ukončeny sekvencí znaků CRLF (\r\n). Na konci všech hlaviček musí následovat prázdný řádek, i kdyby za ním už neměla být žádná data.

Požadavky se v PHP odesílají přes tzv. *sockety*. Socket je v podstatě jakési spojení mezi serverem a klientem. Abychom s nimi mohli pracovat, musíme takový socket nejdříve otevřít. K tomu slouží funkce fsockopen:

```
fsockopen(server, port);
```

Příklad:

```
$sock = fsockopen(”www.interval.cz”, 80);
```

Socket máme otevřený, můžeme odeslat svůj požadavek pomocí funkce fputs:

```
fputs(socket, request);
```

Příklad:

```
fputs($sock, ”GET /index.html HTTP/1.1\r\nHost: www.inter-
val.cz\r\n\r\n”)
```

Více v PHP již není třeba, pojďme se podívat na metody požadavku http.

7. METODY POŽADAVKU PROTOKOLU HTTP

V HTTP/1.1 existuje sedm základních metod požadavků HTTP:

- GET
- POST
- HEAD
- OPTIONS
- PUT
- DELETE
- TRACE

Za každým požadavkem ještě mohou následovat jednotlivé hlavičky. Ve verzi HTTP 1.1 je v každém požadavku povinná hlavička Host, která specifikuje hostitele. Po hlavičkách (jak už jsem psal) musí následovat prázdný řádek.

7.1. Metoda GET

Metoda GET je ta nejjednodušší a patří mezi základní. Prakticky pokaždé, když načítáte stránku ze serveru a neodeslali jste předtím formulář s metodou POST, používá se k obdržení stránky ze serveru právě tato metoda. Výsledkem je tudíž stránka a její hlavičky, na kterou se pomocí metody ptáme. Formát této metody je následující:

GET URL-STRÁNKY VERZE-PROTOKOLU

HLAVIČKY

prázdný řádek

Příklad:

GET /index.asp HTTP/1.1

Host: www.interval.cz

prázdný řádek

7.2. Metoda POST

Metoda POST funguje v podstatě sejně jako metoda GET, ale u POST máte možnost za hlavičkami (a prázdným řádkem) poslat skriptu data. Touto metodou se např. odesílají data z formuláře s metodou POST. Formát tohoto požadavku je následující:

POST URL-STRÁNKY VERZE-PROTOKOLU
HLAVIČKY
prázdný řádek
DATA Z FORMULÁŘE

Příklad:

POST /zpracujdata.php HTTP/1.1
Host: www.formulare.cz
Content-Length: 29
Content-Type: application/x-www-form-urlencoded
prázdný řádek
pole1=hodnota1&pole2=hodnota2

U této metody nám přibyly některé hlavičky, o kterých jsem se předtím nezmínil. Content-Length značí, jak dlouhá jsou data z formuláře (v bytech) a Content-Type: application/x-www-form-urlencoded značí MIME typ dat z formuláře.

Metody HEAD, OPTIONS, PUT, DELETE, TRACE

Pokud zrovna neprogramujete internetový prohlížeč (a v PHP asi ne), pravděpodobně se s těmito metodami nikdy nesetkáte. Takže pouze stručně uvedu jejich význam v tabulce:

Metoda	Popis
HEAD	Metoda HEAD funguje prakticky stejně jako GET, ale HEAD nevrací tělo stránky, pouze hlavičky. Toho se např. využívá při zjišťování, jestli se stránka od posledního požadavku změnila.
OPTIONS	Používá se pro dotaz na možnosti serveru
PUT	Funguje jako GET, ale uchovává tělo požadavku na místě daném požadovaným URL. Podobně jako odesílání souborů přes FTP.
DELETE	Odstraňuje dokument ze serveru. Dokument, který má být odstraněn, je dán URL požadavku.
TRACE	Používá se pro sledování požadavku přes všechny proxy servery a firewally, přes které požadavek jde. Podobá se nástroji TraceRoute.

7.3. Použití datových zdrojů

Moderní a výkonné aplikace pro zpracování dat neukládají informace přímo ve svých vlastních souborech, ale používají služeb některého z externích datových zdrojů. Datových zdrojů může být celá řada. Od jednoduchého textového souboru, přes dříve oblíbené souborové databáze třeba ve formátu DBF až po databáze uložené na SQL serverech.

Každý takový zdroj zpravidla používá svůj vlastní formát pro uložení dat a metody pro přístup k těmto datům. Aby programátoři aplikací nemuseli čelit problému s různorodostí datových zdrojů a vyvíjet aplikace pro každý zdroj zvlášť, byl vytvořen standardní nástroj, který umožňuje přistupovat k různým zdrojům dat pomocí jednotné standardizované platformy – ODBC.

Takto standardizovaná platforma přináší uživatelům možnost komplexně a přitom jednoduchým způsobem využívat data z různých zdrojů na různých místech. Můžete například přímo ve Wordu nebo Excelu použít údaje pořízené v Accessu, aniž by bylo nutné je nejprve exportovat a pak složitě načítat do požadované aplikace.

7.4. Co je ODBC

ODBC je zkratka pro Open Database Connectivity. Datové zdroje ODBC jsou aplikacím přístupné přes příslušný ovladač, který si můžeme představit jako prostředníka pro komunikaci mezi uživatelskou aplikací a externím zdrojem dat. Aplikace svůj dotaz na data předá do ODBC. Příslušný ovladač přeloží tento dotaz tak, aby mu rozuměl externí zdroj dat, a zašle mu jej. Odpověď od zdroje dat opět putuje přes ODBC, které přeloží výsledek do standardní podoby a vrátí jej volající aplikaci.

Princip práce ODBC je dán standardem, proto jakákoliv aplikace, která umí použít ODBC ovladače, může přistupovat k jakémukoliv externímu zdroji dat od libovolného výrobce, který pro to poskytne příslušný ovladač. ODBC tak umožnil standardizovat na straně aplikací přístup k datům a nestarat se o to, jakým konkrétním způsobem pracuje zdroj dat.

Na jedné straně tedy aplikaci stačí, když umí pracovat s ODBC. Na straně druhé výrobci různých zdrojů dat k nim poskytují příslušné ODBC ovladače, aby jejich zdroje byly lehce přístupné aplikacím. To je výhodné pro programátory aplikací, kteří mají k dispozici nezávislý standardizovaný přístup k datům, a potažmo i pro uživatele, kteří tím pádem získají aplikace přistupující k různým datům levněji a v kratším čase.

Práce s ODBC ve Windows

Ovládací prvky pro nastavování ODBC jsou již integrální součástí operačního systému Microsoft Windows. **Správce zdrojů dat ODBC** naleznete v ovládacích panelech. Na

tomto místě je nutno upozornit, že uvedené postupy i obrázky platí beze zbytku v operačním systému Windows 10, ale obdobně platí i pro další verze Windows. Na některých systémech ale může být správce ODBC přístupný odlišným způsobem.

8. DATOVÉ ZDROJE

K naplnění seznamů příjemců a šablon těmi správnými daty ve správný okamžik využívá Mailkit podporu pro **XML & RSS datové zdroje**. Jejich použití je omezeno v závislosti na typu účtu uživatele. Verze Mailkit Base je omezena na použití XML zdrojů dat, určených pouze pro import příjemců do seznamů, zatímco Mailkit Syndicate a Agency podporuje jak XML, tak RSS datové zdroje, a to i za účelem načítání dat do šablon.

Použití XML datových zdrojů pro seznam příjemců

Pro vytvoření nového seznamu příjemců z datového zdroje je nejprve potřeba nastavit nový XML datový zdroj.

- Jméno - jméno datového zdroje. Pokud bude datový zdroj sloužit pro seznam příjemců, dostane seznam příjemců shodné jméno.
- Popis - popis datového zdroje.
- Zdroj - URL adresa, na které se nalézá XML, či RSS, jež má sloužit jako datový zdroj.
- Autorizace - zaškrtne se v případě, že je přístup na umístění datového zdroje zaheslován.
- Typ - z roletkové lišty se vybírá, zda se jedná o RSS či XML datový zdroj
- Cíl - z roletkové lišty se vybírá cíl datového zdroje. Zda bude použit pro seznam příjemců nebo pro šablonu.
- Automaticky aktualizovat - pokud se tato volba zaškrtne, bude se zdroj automaticky aktualizovat těsně před rozesláním kampaně.
- Doba expirace - po jak dlouhé době se má datový zdroj v případě automatické aktualizace znovu aktualizovat.
- Poslední aktualizace - zde se zobrazuje datum poslední aktualizace datového zdroje.
- Prázdné záznamy: Vynulovat - údaje u příjemců budou vynulovány dle prázdných záznamů v importu. Ponechat současnou hodnotu - údaje zůstanou u příjemce ve stejné podobě jako před importem.

Jak připravit datový zdroj

Datový zdroj nemá pevně definovanou strukturu a může být ve formátech XML nebo JSON, který musí být plně validní. Soubor datového zdroje musí být vystaven na URL dostupné ze serverů Mailkitu a zabezpečen proti přístupu třetích stran neboť se jedná o citlivé údaje.

Protože každý klient využívá jiný informační systém s jinými možnostmi, systém datových zdrojů je postaven maximálně univerzálně a nepředepisuje specificky strukturu požadovaných dat. Na datové zdroje se však vztahují jistá technická omezení:

- Plně validní XML nebo JSON
- U XML formátu nejsou podporovány attributy (např. first_name="Jana" gender="f" country="cz")
- Doporučené kódování znaků UTF8
- Nepoužívat znak "," jako oddělovač pro více hodnot, ale použít znak "|"
- Unikátním identifikátorem záznamu a zároveň jediným povinným polem je e-mailová adresa. Pokud bude více záznamů s totožnou e-mailovou adresou, dojde k jejich vzájemnému přepsání.

Jak již bylo napsáno, jediným povinným údajem je **e-mail** a všechny další údaje jsou nepovinné, nicméně důležité. Obecně platí pravidlo "čím více, tím lépe", ale také "nic se nesmí přehánět". Do datového zdroje by tak mělo přijít maximum dostupných údajů o příjemcích, které je možné pro vaše současné, ale i budoucí e-mailové kampaně využít.

Import dat z datového zdroje

K získání obsahu se musí přiřadit jednotlivé větvě zdroj k polím kontaktu. Pro přiřazení se klikne na název zdroje dat a dále na tlačítko **Zobrazit strukturu**. Po přiřazení všech polí, se klikne na tlačítko **Uložit**. Následně bude možné pokračovat tlačítkem **Import** pro aktuální import dat. V tomto momentě bude vytvořen nový **Seznam příjemců** (jež bude pojmenován totožně jako zdroj, jejž ho vytvořil) a data z datového zdroje budou importována dle Vašeho předchozího přiřazení.

Uživatelé účtu typu Syndicate a Agency mají zároveň možnost automatické aktualizace. Pokud označí automatickou aktualizaci, datový zdroj bude automaticky importován a seznam příjemců bude aktualizován před počátkem doručování každé kampaně. Tento parametr nedoporučujeme používat u datových zdrojů obsahující více než nižší tisíce záznamů, neboť samotná aktualizace může trvat i několik minut a o to bude zpožděna rozesílka kampaně. Pro větší datové zdroje doporučujeme využít možnosti pravidelné plánované aktualizace ve stanovenou denní hodinu, kterou pro vás nastaví naše zákaznická podpora.

Použití XML & RSS datových zdrojů v šablonách

Nastavení XML & RSS datových zdrojů pro použití v šablonách je velmi podobné jako pro použití v seznamech příjemců, ale bez nutnosti přiřazování významů jednotlivých polí. Hodnoty jsou určeny řetězcem jmen v šabloně, proto je snadné nastavit jakýkoliv XML či RSS zdroj.

Výše je uvedený příklad kódu šablony, pro kterou je použit RSS datový zdroj pojmenovaný *EXAMPLE*. Příkaz *FOREACH* vytváří smyčku pro parsování a nalezení všech záznamů. Každý

standardních RSS tagů je snadno řešen a vložen do HTML kódu, což umožňuje výstup dat do šablony. Více informací viz. Šablony emailů.

Produktové datové zdroje

Datové zdroje je možné využít i pro přenos produktové nabídky do Mailkitu a následné použití produktových informací v kampaních. Právě zde se pak projevuje síla datových zdrojů a programovatelných šablon, která umožňuje kombinovat data z vícero zdrojů a zcela automatizovaně tak personalizovat obsah na míru jednotlivých příjemců.

Pro produktové informace je možné použít kterýkoliv z běžných formátů produktových feedu pro srovnávače Heureka, Zbozi, Google a další, nebo generovat vlastní feed s potřebnými informacemi. Protože produktové feedy jsou velice rozsáhlé a je důležitá rychlost práce s daty v nich obsaženými, tyto datové zdroje se přenášejí přímo do SQL databáze a i tak je s nimi možné pracovat. Pro nastavení produktového datového zdroje se obraťte na zákaznickou podporu, která vám pomůže s jeho implementací a následným využitím.

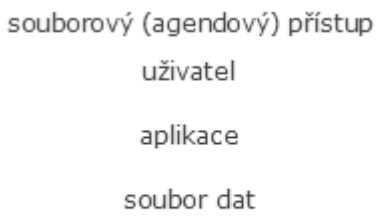
Delivery feedy

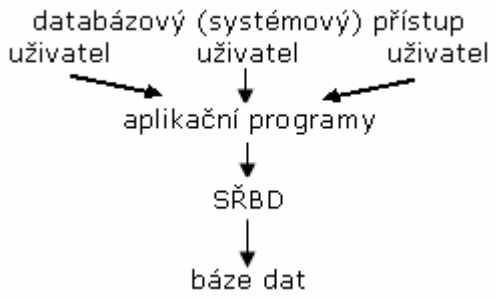
Delivery feedy jsou speciální datové zdroje, jež slouží k předání strukturovaných informací pro realizaci rozesílky kampaně. Zatím co obvykle kampaň využívá stanovený seznam příjemců, na který probíhá její rozesílka dle nastavených pravidel, v případě delivery feedu je kampaň rozesílána pouze na adresy uvedené ve feedu. Jedná se o alternativu k API volání mailkit.sendmail_mass, tzn. způsob jak do Mailkitu dostávat vysoce strukturovaná data, např. z personalizačních systémů či CRM, jež se mají při odesílání kampaně zpracovat. Tyto feedy pak musí mít striktně definovanou strukturu ve formátu XML.

9. DATABÁZOVÝ PŘÍSTUP

Požadavky na databázový systém:

- kontrola konzistence dat - databáze má být schopna zajistit dodržování určitých pravidel tzv. integritních omezení a zabezpečit data před případnými nehodami, které mohou vzniknout v průběhu transakcí,
- transakce - posloupnost manipulací s daty, která musí proběhnout, aby data byla uložena správně, např. převod z 1 účtu na jiný účet v bance (musí proběhnout korektně na obou účtech),
- velké objemy dat - relativně k možnostem paměťových médií musí být databáze schopna uchovávat odpovídající objem dat,
- správa dat - etapy vývoje.





Databázový přístup

- velké databázové systémy - firmy Informix, Sybase – nákladné menší (cenově dostupnější) databázové systémy - MS Access, Paradox, FoxPro, dále pak malé databázové systémy dostupné zcela zdarma - např. My SQL
- jazyk SQL - standard umožňující využívání datových zdrojů spravovaných různými databázovými systémy

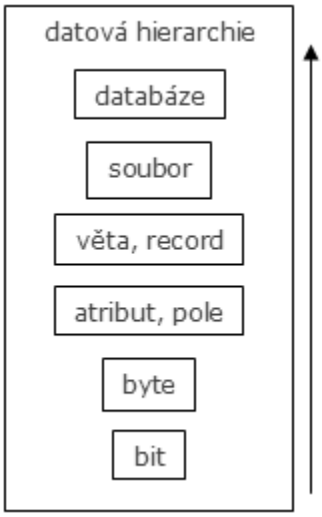
Tvorba datové základny pro IS organizace - složitá záležitost vyžadující péči lidí s různým odborným zaměřením. Při návrhu konceptuálního schématu datové základny se rozhoduje o tom, co v datové základně bude. Během provozu IS je pro uživatele důležité, zda umí datovou základnu využívat jako informační zdroj.

Požadavky na počítačový IS:

- horizontální a vertikální integrace informací,
- rychlé agregování informací od nižších stupňů zařízení,
- racionální prezentace informací v čase, formě a prostoru,
- časová frekvence,
- rozsah uchovávaných informací.

Organizace dat - soubory a databáze:

- zpracování transakcí - dávkové zpracování nebo zpracování on-line,
- současný trend - objektově orientované a hypermediální databáze.



Návrh strukturované datová základny

- realita, jejímž odrazem má být navrhovaná datová základna, sestává z různých objektů neboli entit,
- mezi sledovanými entitami mohou existovat různé vztahy (např. mezi entitami stejného typu = rekurzivní vztah).

Kardinalita vztahu: symbolické označení 1:1 (profesor XY má za manželku ZN); 1:N (které studenty učí profesor XY); M:N (které studenty učí kteří profesoři).

Integritní omezení datové základny - veškerá pravidla vymezující přípustné hodnoty (a kombinace hodnot) atributů, formát zobrazení.

Relační model dat

- Předpokládá existenci jedno hodnotových atributů:
- představa zobrazení formou relační tabulky, ve které odpovídá pojmu n-tice řádek a pojmu atribut sloupec

Relační databáze

- všechna data mají tvar 1 nebo více tabulek s pojmenovanými sloupci
- každý sloupec obsahuje data z 1 domény (tj. 1 datového typu)
- prvky jednotlivých sloupců (jímž je dáno jméno a typy) se nazývají obvykle položky nebo pole a pojem **řádek** splývá s pojmem **záznam (věta)**
- relacemi ve smyslu relačního modelu dat se obecně popisují jak entity, tak vztahy mezi nimi

- podle tohoto dvojího využití pak můžeme terminologicky rozlišovat mezi tzv. **entitními relacemi** neboli množinami uspořádaných n-tic atributů popisujících samotné entity a „vztahovými relacemi“ (tj. množiny uspořádaných n-tic)

Datové sklady

Princip tzv. Warehouse - 2 hlavní záměry:

- Sjednocení pohledu na data v jednotlivých tzv. produkčních systémech poskytne přehledný přístup k datům
- rozličně nazývané jedny a tytéž věci budou viděny jakožto věc jedna a různě měřené veličiny

10. DATA STRUKTUROVANÁ A NESTRUKTUROVANÁ

10.1. Strukturovaná data

Základní typy (dělení z důvodu rozlišení povolených a nepovolených manipulací a hodnot):

- textová (řetězce znaků, vyjádření informací pomocí text. kódu, pouze určitá množina prvků, které můžeme zaznamenávat, mohou definovat syntaxi,
- číselná - čísla reálná racionální,
- datum, čas - omezeno jakých hodnot bude nabývat (30. únor, 27. hodin),
- logická - splnění podmínek existence či neexistence vlastností objektů - 2 hodnoty (0 a 1, A a N),
- kategorie - hodnota vlastností vybraná ze škály (často číselníky, umožňuje zaznamenávání hodnoty pouze kódem).
- Strukturováním je vytvořena taková organizace dat, která umožňuje efektivně uložit, zpracovat a vyhledat údaje podle potřeby → strukturovaná data vytvářejí vyhledávací klíče (někdy též identifikační klíče)- klíče, jež jednoznačně identifikují datový záznam, jsou nazývány privátní klíče (identifikační klíče) - základní podmínka a datového a databázového systému.
- Údaje o něčem (odraz reality) - jméno příjmení, adresa, věk, tel, číslo, váha, cena, ..., počet bodů, kategorie, prům. známka, ..., počet kusů, počet stran.
- Operace aneb co s nimi mohu dělat - sčítání, zaokrouhlení, násobení, připojení (jméno+příjmení), zkrácení, řazení, ..., den v týdnu, negace, ...
- Datový typ (musí být definován) - číslo, textový údaj, datum a čas, logický údaj (ano/ne).
- Zakódovaná data - různá kódování - text písmena - různé kódové tabulky (ascii, ebdic, ...) národní abecedy; datum a čas (jak píšeme datum).

10.2. Data nestrukturovaná

Data typu: volný text, audio, video, grafika, multimédia

- Poskytují více dat než pouhé strukturované údaje
- Problém: podle nestrukturovaných dat lze velmi těžko vyhledávat - používáme řešení - nestrukturovaná data bývají doplněna daty strukturovanými (název mp3)

10.3. Objemy dat - strukturovaná i nestrukturovaná data

- stránka textu ascii (notepad) 1,8 kB
- stránka textu word 50 kB
- vektorová grafika A4 30 kB
- bitmapový obraz A4(jpg, rgb) 5 MB
- záznam 1 minuty zvuku (wav) 10 MB
- záznam 90 minut obrazu 3 GB

Datový sklad (anglicky Data Warehouse, případně DWH) je zvláštní typ relační databáze, která umožňuje řešit úlohy zaměřené převážně na analytické dotazování nad rozsáhlými soubory dat.

Orientace na subjekt

U běžné relační databáze je obvyklá snaha o co nejmenší redundanci uložení dat, které je dosahováno jejich normalizací do třetí normální formy a vnitřním provázáním jednotlivých logických funkčních celků. V datovém skladu je naproti tomu řešení vždy vedeno snahou o jasnou vnitřní separaci jednotlivých funkčních celků – výsledkem je struktura, která je čitelnější pro uživatele (manažera, business analytika) za cenu zvýšených nároků na paměťový prostor.

Integrovanost

Běžná provozní aplikace (program) nad relační databází řeší určitý specifický okruh úloh nad „svými“ specifickými daty. V datovém skladu je třeba naproti tomu shromáždit informace z mnoha různých zdrojů a seskupit je nikoliv podle původu, ale podle logického významu (úzce souvisí s orientací na subjekt – všechna data týkající se určité funkční oblasti potřebuji mít „na jedné hromadě“ bez ohledu na to, odkud pocházejí).

Nízká proměnlivost

Data jsou do datového skladu obvykle nahrávána ve větších dávkach (například v denních nebo týdenních intervalech) a pak již nejsou nijak modifikována.

Historizace

Data jsou v datovém skladu obvykle udržována v historické podobě, nikoliv pouze v aktuálním stavu. To je dáno nutností provádění analýz zaměřených na vývoj v čase. V běžné relační databázi je z pohledu uživatelů obvykle zajímavý pouze aktuální stav datových objektů.

10.4. Technologické charakteristiky datového skladu

Z požadavků na datový sklad vyplývají jeho technologické charakteristiky:

- Datový sklad musí obsahovat nástroj pro nahrávání dat z různých datových zdrojů, tyto zdroje mohou mít různé datové formáty a různé fyzické umístění, nemusí se zdaleka jednat pouze o relační databáze.
- Datový sklad ukládá data nikoliv s ohledem na co nejlepší podmínky pro editaci, ale s ohledem na co nejlepší a nejrychlejší provádění složitých dotazů – proto je pro uložení dat používána často technologie OLAP.
- Nelze předem vědět, jaké dotazy a jaké úlohy budou chtít uživatelé nad datovým skladem v budoucnosti řešit. (V době budování datového skladu je obvykle známý pouze typ úloh, nikoliv všechny jednotlivé dotazy a úlohy.) Z toho vyplývá potřeba dostatečně flexibilních a přitom uživatelsky přívětivých analytických nástrojů.

10.5. Logická struktura datového skladu

Data v datovém skladu jsou z logického (uživatelského) pohledu členěna do schémat – každé schéma odpovídá jedné analyzované funkční oblasti.

Jádro každého schématu tvoří jedna nebo několik faktových tabulek. V nich jsou uložena vlastní analyzovaná data – číselné a finanční hodnoty, které jsou použity k analytickým výpočtům – agregacím, třídění apod. Většinu paměťového místa v datovém skladu zabírají faktové tabulky, které obsahují detailní údaje ze všech zdrojů – tedy řádově více údajů než ostatní tabulky.

Faktové tabulky jsou pomocí cizích klíčů spojeny s dimenzemi. Dimenze jsou tabulky, které obsahují seznamy hodnot sloužících ke kategorizaci a třídění dat ve faktových tabulkách.

Příklad

V datovém skladu je třeba uložit informace o všech prodejích z pokladen hypermarketů, data budou dále analyzována na základě doby prodeje, prodejny, typu zboží, dodavatele, probíhajících marketingových akcí a způsobu platby (kartou, hotově).

Schéma Prodej bude obsahovat faktovou tabulku Položky prodeje, kde bude pro každou prodanou položku uložen údaj o typu prodaného zboží, ceně a počtu kusů (případně prodané hmotnosti).

Kromě této faktové tabulky bude schéma obsahovat také dimenze pro třídění položek prodeje: časové dimenze Datum a Hodina (v rámci dne), dimenzi Prodejna, dimenzi Typ zboží, kde bude jeden řádek pro každou jednotlivou položku (například „Choceňský jogurt

borůvkový 250ml“), dimenzi Kategorie zboží (obsahující řádky jako například „Jogurt“), dimenzi Oddělení (obsahující řádky jako například „Mléčné výrobky“), dimenzi Dodavatel (obsahující řádky jako například „Choceňská mlékárna a.s.“) a tak dále.

Faktová tabulka musí být pomocí cizího klíče přímo nebo nepřímo propojena s každou z těchto dimenzí.

V uložení hierarchických dimenzí mám v zásadě dvě možnosti:

- Z celé hierarchie vytvořím jednu dimenzní tabulku, ve které budou údaje pro vyšší stupně hierarchie uloženy redundantně. Vznikne schema, kde je každá dimenzní tabulka vázána přímo na faktovou tabulku – podle tvaru svého diagramu se takové schéma nazývá hvězda (Star schema).
- Na hierarchickou dimenzi budu aplikovat normalizační doporučení 3NF, takže pouze dimenze na nejnižším stupni hierarchie bude vázána přímo na faktovou tabulku, ostatní pak na některou z nižších dimenzí v hierarchické struktuře – podle tvaru svého diagramu se takové schéma nazývá vločka (Snowflake schema).

11. TECHNOLOGIE AJAX

AJAX (Asynchronous JavaScript and XML) je v informatice obecné označení pro technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich kompletního znovunačítání za pomoci asynchronního zpracování webových stránek pomocí knihovny napsané v JavaScriptu. Na rozdíl od klasických webových aplikací poskytují uživatelsky příjemnější prostředí, ale vyžadují použití moderních webových prohlížečů.

Tyto aplikace jsou vyvíjeny s využitím technologií:

- HTML (nebo XHTML) a CSS pro prezentaci informací;
- DOM a JavaScript pro zobrazování a dynamické změny prezentovaných informací;
- XMLHttpRequest pro asynchronní výměnu dat s webovým serverem (typicky je užíván formát XML, ale je možné použít libovolný jiný formát včetně HTML, prostého textu, JSONči EBML).
- Podobně jako DHTML, LAMP nebo SPA, Ajax ve skutečnosti není konkrétní jednotlivá technologie, ale pojem označující použití několika technologií dohromady s určitým cílem.

Výhody

Mezi výhody patří odstranění nutnosti znovunačtení a překreslení celé stránky při každé operaci, které jsou nutné u klasického modelu WWW stránek. Pokud například uživatel klikne na tlačítko pro udělení hlasu v nějaké anketě, celá stránka se musí znovu načíst ze serveru, třebaže se na ní jen například aktualizují výsledky hlasování a veškerý zbytek obsahu zůstává stejný. Prostřednictvím AJAXu proběhne odeslání hlasu uživatele na pozadí, server zašle jen ty části stránky, které se změnily, a jen tyto části se uživateli na stránce aktualizují a překreslí. Taktéž nedochází k nepříjemnému efektu, kdy se po dané akci v průběžně načítané stránce postupně přizpůsobují a „za běhu“ formátují a zarovnávají její blokové elementy, obrázky atd. – obtěžující může být i to, že po dané akci uprostřed delší stránky (odscrollované dolů) se nově načtená stránka zobrazí vyscrollovaná nahoru. S AJAXem má uživatel pocit mnohem větší plynulosti práce, která se (zejména u rychlejšího internetového připojení) blíží běžným desktopovým aplikacím.

Z toho vyplývá také potenciál snížit zátěž na webové servery a síť obecně. Jelikož není potřeba při každém požadavku sestavit celý HTML dokument, ale pouze provedené změny, je množství vyměňovaných dat výrazně nižší a teoreticky to může mít příznivý vliv i na zátěž databázových serverů či dalších backendových systémů.

Nevýhody

AJAX však naopak může zvýšit *počet* vyměňovaných HTTP požadavků, a třebaže přenášejí nižší množství dat, tak při nevhodné implementaci zátěž neklesne.

Mezi nevýhody patří hlavně změny v paradigmatu používání webu: webové stránky se chovají jako plnohodnotné aplikace se složitou vnitřní logikou, nikoli jako posloupnost stránek, mezi kterými se lze navigovat i pomocí tlačítek Zpět a Další. Obdobným způsobem není možno předat URL stránky, ve které již bylo pomocí technologie AJAX něco „naklikáno". Moderní AJAXové aplikace jsou schopny procházení v historii (přinejmenším částečně) obnovit za použití různých technik (např. využití části adresy za znakem # či pomocí neviditelných IFRAMEs). To ale ve výsledku ztěžuje návrh stránek a vyžaduje více času a práce na implementaci pomocí AJAXu.

Problémem AJAXových aplikací také může být síťová latence: potřeba komunikace přes Internet má negativní dopady na rychlost odezvy a interaktivitu uživatelského rozhraní. Pokud uživateli není jasně signalizováno, že aplikace zpracovává jeho požadavek (a na pozadí komunikuje se serverem), jediné, co zaregistruje, je zpožděná reakce (mezitím se dokonce může snažit operaci spustit znovu, neboť se domnívá, že systém jeho příkaz ignoroval, a tím vygenerovat větší zátěž serveru a/nebo způsobit něco, co neměl v úmyslu, např. objednat deset vstupenek místo jedné). Jako vhodné řešení se doporučuje po dobu mezi odbavením požadavku na server a jeho odpovědí nějakým způsobem zobrazit, že uživatelův požadavek se zpracovává, například textem nebo animovanou ikonou.

Další nevýhodou AJAXu je nutnost používat moderní grafické prohlížeče, které podporují potřebné technologie. Všechny dnešní běžné prohlížeče však tyto technologie alespoň v základu podporují, problém tak zůstává jen u minoritních prohlížečů typu Lynx nebo na hardwarově slabších zařízeních pro prohlížení, například na (některých) mobilních telefonech a PDA. V rámci webové přístupnosti se vyžaduje, aby stránky byly přístupné i z prohlížečů bez podpory AJAXu, což pro vývojáře znamená více času a práce a objednavatelům výsledných stránek větší náklady.

AJAX

Zkratka AJAX pochází z anglického Asynchronous JavaScript and XML. AJAX je moderní technologie často využívaná v současných webových aplikacích. Hodně se o ní mluví, neboť je součástí RIA, nového směru programování vedoucímu k vyššímu uživatelskému komfortu a funkčnosti aplikací.

AJAX však ve skutečnosti není žádnou novou technologií, pouze novou kombinací technologií již dávno známých, tj. HTML (nebo XHTML), JavaScriptu, XML a XMLHttpRequest.

A proč je AJAX tak výhodný? Aplikace využívající AJAX umí odeslat a získat zpět data ze serveru bez nutnosti znovu nahrávat celou stránku (na rozdíl od klasických odkazů). AJAX tak může

mít mnohá využití, příkladem mohou být různé našeptávače (formuláře, které se automaticky předvyplňují podle stisknuté klávesy), AJAX ankety a další složitější aplikace, které dokáží uživateli usnadnit práci.

AJAX však má i určité nevýhody, především při nevhodném užití snižuje významně použitelnost stránek. Proto je třeba, stejně jako u jiných technologií, AJAX aplikaci dobře promyslet a před nasazením i důkladně otestovat na uživatelích.

Jak jistě většina z Vás ví, Javascript je tzv. client-side jazyk. Provádí se tedy na straně webového prohlížeče na klientském počítači a je interpretován javascriptovým interpreterem. Disponuje řadou vlastností, mimo jiné i tou, která nás teď bude zajímat nejvíce – schopností provádět asynchronní operace/úlohy.

V čem vlastně spočívá asynchronnost v AJAXu? Jde o schopnost Javascriptu zavolat na serveru nějaký skript, nebo prvek API a nečekat bezprostředně na odpověď. Místo toho pokračuje provádění kódu (uživateli se například zobrazí stránka a může s ní běžně pracovat). Když potom odpověď přijde, provádění hlavního kontextu kódu se pozastaví (dříve, či později v závislosti na prioritě prováděné úlohy) a dojde k zavolání tzv. callbacku – funkce, která se provede v případě, že ze serveru dostaneme odpověď. Takovémuto kódu, či takovéto funkci se pak říká neblokující, protože neblokuje průběh provádění skriptu čekáním na odpověď, či zpracování.

Co tedy AJAX je a co nám přináší? Jde o zkratku **Asynchronous Javascript And XML**, to nám zatím neřekne víc, než že to souvisí s XML a Javascriptem, nicméně půjdeme na to postupně a všechno si vysvětlíme.

AJAX je způsob, jak programovat v Javascriptu – není to tedy žádný nový jazyk ani framework nebo knihovna třetí strany. Jde o způsob, kterým můžeme vyměňovat data v naší aplikaci s databází, serverovými skripty (PHP, Java, ASP, atd.) aniž bychom aktualizovali/reloadovali celou stránku, technicky vzato neaktualizujeme (ve smyslu refreshe) vůbec žádnou část naší aplikace, či stránky a v tom vlastně tkví ta "magická síla" a podstata AJAXu.

Proč se v názvu vyskytuje zkratka XML? Jednoduše je to jeden z formátů dat, ve kterém AJAX umí obdržet informaci ze serveru, nejznámější a často používané jsou JSON, XML, text, binární data. Každý z nich má něco do sebe, každý se hodí na něco jiného *a používá se trochu jinak.

AJAX vám například dává možnost zkontrolovat data zadávaná uživatelem, ještě než je uživatel odešle potvrzením formuláře. Do jisté míry je toho schopný i doteď známý Javascript, ale co když jde o registrační formulář s podmíněným heslem, uživatelským jménem nebo e-mailem, které musí být unikátní v rámci databáze/tabulky? Přesně pro tyto a mnohé další účely je tato technologie to pravé. Dovolí vám totiž provést ty samé

operace, které byste prováděli voláním například PHP skriptů při přesměrování po odeslání formuláře, ale na rozdíl od nich vám umožní provádět tato volání na pozadí.

Ajax v základech

Myšlenka Ajaxu spočívá v tom, že jsou části webové stránky natahovány asynchronně a tímto způsobem se mění obsah stránky. Znamená to, že při změně obsahu nebude znovu natažena celá stránka, nýbrž pouze modifikovaný obsah. Za tímto účelem je z JavaScriptu volán objekt XMLHttpRequest. Tento požadavek (Request) se dotazuje serveru na obsah, který je typicky vracen zpět pomocí XML a JavaScriptu. Skript pak toto XML analyzuje. Data z XML skript potom překládá a odpovídajícím způsobem mění pomocí DOM (Document Object Model) webovou stránku.

Základní principy práce s Ajaxem si popíšeme na příkladu, který má blízko k jeho praktickému využití – konkrétně se jedná o diskusní fórum, v němž jsou diskuse v HTML zobrazeny jako strom. Aby si uživatel mohl přečíst daný příspěvek, musí kliknout na jeho nadpis. V případě klasického modelu by byl zaslán požadavek na server a následně je vytvořena celá stránka, která kromě původního stromu obsahuje navíc i text příspěvku. U modelu založeného na Ajaxu bude jako reakce na dotaz na server poslán pouze samotný text příspěvku: Připojení tohoto textu k jinak nezměněné stránce zajistí příslušný program v JavaScriptu.

Pro provedení této úlohy byl u starších verzí Internet Exploreru (počínaje verzí 5.0) k dispozici ActiveX objekt. IE 7 už stejně jako jiné prohlížeče dává odpovídající objekt k dispozici nativně. Aktuální API XMLHttpRequest Internet Exploreru je k dispozici na webu MSDN a dostupné je rovněž API projektu Mozilla.

Vlastním trikem Ajaxu je právě XMLHttpRequest – ten může být totiž zpracováván asynchronně, což v této souvislosti znamená asynchronně k uspořádání zbytku stránky a rovněž asynchronně vůči na stránce běžícím skriptům. To znamená, že stažení dat může být provedeno prostřednictvím HTTP a že data mohou být stažena bez toho, aby tím trpěla interakce uživatele se stránkou. Sestavení požadované stránky běží také na pozadí.

Co to vlastně je AJAX? Stručně řečeno, AJAX je technologie, která pomocí skriptů zprostředkovává stránce komunikaci s webovým serverem. Lze tak měnit obsah stránky bez potřeby ji znovu načítat.

Teoreticky lze tedy vytvořit dynamické stránky bez serverových skriptů, tedy mít jednu stránku pro celý web, která pouze mění svůj obsah. Toto řešení však nedoporučuji, protože, jak již bylo řečeno, se jedná o scriptovou technologii, tudíž je příliš závislá na klientu a není zaručeno, zda bude fungovat. Využití však najde v doplňcích webové stránky, jako je anketa, nebo pomocník pro vyhledávání, jako má např. Seznam.cz. Zde je poměrně jednoduché udělat alternativní řešení - u ankety se hlas pošle klasickým formulářem a u vyhledávání zas chybějící pomocník nikoho příliš trápit nebude.

Pokud máte jistotu, že u návštěvníků vašich stránek AJAX poběží, je možné jej použít i pro složitější aplikace. Osobně například používám chat, který si stahuje pouze nové příspěvky a pokud je na něm návštěvník sám, kontrola aktualizací se zpomalí, což šetří mnoho přenesených dat.

12. FRAMEWORKY

Framework (aplikační rámec) je softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovny API, podporu pro návrhové vzory nebo doporučené postupy při vývoji.

- Účel
- Architektura
- Příklady
- Související články
- Externí odkazy

Účel

Cílem frameworku je převzetí typických problémů dané oblasti, čímž se usnadní vývoj tak, aby se návrháři a vývojáři mohli soustředit pouze na své zadání. Například tým, který používá Apache Struts k vývoji webových stránek pro banku, se může zaměřit na to, jak se budou provádět bankovní operace, a ne jak zajistit bezchybnou navigaci mezi jednotlivými stránkami.

Vyskytují se námitky, že použitím frameworku bude kód pomalý či jinak neefektivní a že čas, který se ušetří použitím cizího kódu, se musí věnovat nastudování frameworku. Nicméně při jeho opakovaném nasazení nebo ve velkém projektu dojde k výrazné úspoře času. Při odinstalování frameworku již nebude možné některé aplikace spustit.

Architektura

Framework se skládá z tzv. frozen spots a hot spots. Frozen spots definují celkovou architekturu softwarové struktury, její základní komponenty a vztahy mezi nimi. Tyto části se nemění při žádném použití frameworku. Naproti tomu hot spots jsou komponenty, které spolu s kódem programátora vytvářejí zcela specifickou funkcionalitu, a proto jsou skoro pokaždé jiné.

V objektově orientovaném prostředí je framework tvořen abstraktními a klasickými (neabstraktními) třídami. Frozen spots pak mohou být reprezentovány abstraktními třídami a vlastní kód (hot spots) se přidá implementací abstraktních metod.

Framework je softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovnu API, návrhové vzory nebo doporučené postupy při vývoji.

Další velmi důležitým faktorem při výběru je to, k čemu framework potřebuji a k čemu je určen. Obecně lze frameworky rozdělit na dvě skupiny:

- Sady scriptů – knihoven – pokrývající všemožné potřeby
- Scripty vytvářející jednu konkrétní webovou aplikaci

Výhody

- Rychlejší vývoj
- Méně kódu
- Univerzální kód – změny či nové funkce přidáte 3× jednodušeji
- Pěkná URL

Nepochybně se vaše rychlost vývoje zrychlí. Framework vám usnadní práci. Už nebudete programovat rutiny jako je připojení k databázi, zacheckování správného option u selectu, nebo zvalidování formuláře. Cool-Url budou pro vás automatická věc. Už žádné vrásky s XSS. Oddělená aplikační a prezentační logika, super! Změna DB serveru z MySQL na PostgreSQL? Ne, to vás nemůže rozhodit.

Díky frameworku se můžete opravdu soustředit jen na ten opravdový vývoj. Neřešíte blbůstky. To vše je ale vykoupeno časem, kdy se učíte s frameworkem pracovat. Znáte to, něco konečně uděláte, ale za týden, až proniknete ještě hlouběji do dané problematiky, zjistíte, že to šlo třikrát jednodušeji. Nelze říct, že učením frameworku strávíte měsíc. Ne, někdy, u těch složitějších a komplexnějších, to může být i rok. Nakonec si začnete framework rozšiřovat o své knihovny. A za rok se ohlédnete a zjistíte, že to, co byste předtím programovali týden, máte už teď za jeden den. Framework je úspora času a čas, to jsou peníze.

Výběr není jednoduchý. Jsem s to, že je třeba mít aspoň dva ve své nabídce. Jeden na věci složitější, druhý na ty jednoduší. A pak před každým projektem učinit rozhodnutí, co je pro něj vhodnější. Samozřejmě, nejlépe se naučit ty nejznámější, ty, které mají dobrou velkou komunitu, ty, u kterých máte jistotu, že vývoj za týden neskončí.

- **CakePHP**: celkem jednoduchý na naučení; celkem mocný; kvalitní komunita; dlouhý vývoj nové verze
- **Zend** Framework: velmi mocný; pokrývající veškeré potřeby při tvorbě jakýkoliv webových aplikací; veliká komunita; složitý; dlouhé názvy;
- **CodeIgniter**: maličký, jednoduchý; vyvíjen jednou firmou, ne komunitou; v něčem lepší než Cake, v něčem velmi „hloupý“ – např. nemá layouty

Monolitické frameworky se postupně rozpadají do samostatných (decoupled) komponent. A to přináší řadu výhod. Zatímco dříve bylo použití jen jedné části frameworku obtížné až nemožné, dnes si prostě nainstalujete jeho komponentu. Vývojový cyklus jednotlivých komponent může mít různé tempo. Mají vlastní repozitáře, issue trackery, mohou mít vlastní vývojářské týmy.

Komponenty můžete aktualizovat na nové verze průběžně, bez čekání, než vyjde další verze celého frameworku. Nebo naopak se můžete rozhodnout určitou komponentu neaktualizovat, třeba kvůli BC breaku.

Význam slova framework se tak posouvá, o verzích už takřka nelze hovořit. Místo frameworku XYZ ve verzi 2.3.1 používáte sadu komponent v různých verzích, které spolu fungují.

12.1. Co jsou frameworky?

Při tvorbě moderních webových aplikací se poměrně často využívají tzv. frameworky, neboli vývojové rámce. Jedná se o soubor knihoven a zdrojového kódu, který lze opakovaně použít k usnadnění práce a pokrýt jeho funkcionalitou část tvořené aplikace. Frameworky dokáží řešit problémy, které lze nějakým způsobem zobecnit, jelikož by jinak nemělo smysl tvořit framework. Tvůrce aplikace se tak může více koncentrovat na funkce pro aplikaci výjimečné a jedinečné a neřešit problémy týkající se rutinních úkolů.

Využití javascriptových frameworků

Framework lze vytvořit ke většině programovacích jazyků. Proto javascriptový framework je framework, který slouží k ulehčení práce a programování v javascriptu. Jejich využití je značně široké a každý dnem se rozšiřuje portfolio problémů, které javasriptové frameworky dokáží řešit. Obecně lze říci, že je lze využít ke psaní efektivního zdrojového kódu, z velké části řeší také vzájemnou kompatibilitu (resp. nekompatibilitu) mezi webovými prohlížeči a napsaného kódu, šetří programátorův čas a umožňují využít prvků, které by bylo jinak velmi obtížné programovat svépomocí. Jejich síla a využití je spojena především s technologií AJAX, kde vylepšují vzájemnou interakci uživatele a aplikace pomocí asynchronní komunikace mezi klientem a serverem. Dále pak umožňují dynamicky a hlavně jednoduše přistupovat a měnit jednotlivé elementy stránky (DOM model), GUI prvky, přiřazovat jim události a efektně je animovat. V neposlední řadě některé frameworky obsahují již předpřipravené GUI komponenty, které buď není možno pomocí jiné technologie realizovat, nebo je to příliš náročné a neefektivní. S těmito prvky je možno jednoduše v rámci frameworku pracovat a začlenit je do aplikace jen pomocí několika řádků kódu.

13. 10 NEJLEPŠÍCH PH FRAMEWORKŮ PRO VÝVOJÁŘE

PHP, známý po celém světě jako nejoblíbenější skriptovací jazyk na straně serveru, se hodně rozvinul od dob, kdy se ve statických HTML souborech začaly objevovat první inline fragmenty kódu.

V těch časech museli vývojáři budovat složité weby a webové aplikace, a nad jistou úrovní složitosti **zabíralo příliš mnoho času a úsilí vždycky začínat úplně od začátku**. Odtud vznikla potřeba strukturovanějšího přirozeného způsobu vývoje. Adekvátní řešení této potřeby poskytují vývojářům PHP frameworky.

Proč vůbec používat nějaký PHP framework

Podívejme se nejprve na nejpádnější důvody, proč mnozí vývojáři rádi používají PHP frameworky, a jak mohou tyto frameworky pozvednout úroveň vývojového procesu. Co PHP frameworky poskytují:

- Umožňují rychlý vývoj.
- Poskytují dobře organizovaný, opětovně využitelný a udržovatelný kód
- Umožňují růst v průběhu času, protože webové aplikace běžící na frameworcích jsou škálovatelné.
- Zbavíte se starostí souvisejících s nízkoúrovňovou bezpečností webu.
- Dodržují vzor MVC (Model-View-Controller), který zajišťuje separaci prezentace a logiky.
- Prosazují moderní webové vývojové postupy, mezi něž patří nástroje objektově orientovaného programování.

1. Laravel

Přestože je Laravel poměrně nový PHP framework (byl vydán v roce 2011), je podle nejnovějšího online průzkumu na webu Sitepoint nejoblíbenějším frameworkem mezi vývojáři. Laravel má obrovitý ekosystém s platformou připravenou k okamžitému hostování a rozmisťování a jeho oficiální web nabízí mnoho návodů ve formě screencastů, jimž zde říkají Laracasty.

2. Symfony

Komponenty frameworku Symfony 2 používají mnohé impozantní projekty, jako jsou systém pro řízení obsahu Drupalnebo software phpBB pro spouštění fór, ale **spoléhá se na něj i** Laravel – framework uvedený výše. Symfony má **rozsáhlou vývojářskou komunitu** a mnoho vášnivých příznivců.

3. CodeIgniter

CodeIgniter je odlehčený PHP framework, který už je téměř deset let starý (původně byl vydaný v roce 2006). CodeIgniter má velmi přímočarý instalační proces, který požaduje jen minimum konfigurace, takže může ušetřit soustu trápení. Je také ideální volbou, pokud se chcete **vyvarovat konfliktů s verzemi PHP**, protože **funguje hladce téměř na všech sdílených a dedikovaných hostovacích platformách** (v současné době požaduje jen PHP 5.2.4).

4. Yii 2

Pokud zvolíte framework Yii, dodáte svému webu vzpruhu pro výkon, protože je **rychlejší než ostatní PHP frameworky**, používá totiž **extenzívně techniku „pohodového" načítání (lazy loading).** Yii 2 je čistě **objektově orientovaný** a je založený na kódovacím pojetí **DRY** (Don't Repeat Yourself, „neopakujte se"), takže poskytuje **dost jasnou a logickou kódovou bázi**.

5. Phalcon

Framework Phalcon byl vydán v roce 2012 a rychle si mezi vývojáři PHP vydobyl popularitu. Říká se o něm, že je rychlý jako sokol (anglicky falcon), protože **byl napsán v jazycích C a C++, aby se dosáhlo nejvyšší možné úrovně optimalizace výkonu**. Dobrou zprávou je, že se nemusíte učit jazyk C, protože funkcionalita je **vystavena jako PHP třídy připravené k okamžitému použití v jakékoli aplikaci**.

6. CakePHP

CakePHP už má za sebou celou dekádu existence (první verze byla vydaná v roce 2005), pořád však patří mezi nejoblíbenější PHP frameworky, protože vždy dbal na to, aby držel krok s dobou. Nejnovější verze CakePHP 3.0 rozšířila správu relací, **vylepšila modularitu** tím, že oddělila několik komponent, a zvýšila způsobilost **vytvářet soběstačnější knihovny**.

7. Zend Framework

Zend je robustní a stabilní PHP framework zabalený spolu se spoustou konfiguračních voleb, proto se obvykle **nedoporučuje pro menší projekty,** je však **vynikající pro ty složitější**. Mezi partnery Zend patří takové společnosti, jako IBM, Microsoft, Google a Adobe. Nadcházející hlavní vydání, Zend Framework 3, bude optimalizované pro PHP 7, bude však i nadále podporovat PHP 5.5.

8. Slim

Slim je PHP mikro framework, který poskytuje vše, co potřebujete, a nic, co nepotřebujete. Mikro frameworky jsou v designu minimalistické, jsou **vynikající pro menší aplikace**, pro které by byl framework, který má plnohodnotnou výbavu pro úplně všechno, něco jako

kanón na vrabce. Tvůrce Slimu se inspiroval mikro frameworkem Ruby, který se jmenuje Sinatra.

9. FuelPHP

FuelPHP je flexibilní plnohodnotný PHP framework, který podporuje nejen obyčejný vzor MVC, ale na úrovni architektury též jeho rozvinutou verzi, HMVC (Hierarchical Model-View-Controller). FuelPHP přidává mezi vrstvy Controller a View **nepovinnou třídu** s názvem Presenter (dříve se jmenovala ViewModel), aby **obsahovala logiku potřebnou ke generování pohledů**.

10. PHPixie

PHPixie je zcela nový framework, odstartoval v roce 2012 s cílem vytvořit vysoce výkonný framework pro weby určené jen ke čtení. PHPixie stejně jako FuelPHP **implementuje návrhový vzor HMVC** a je vybudovaný pomocí nezávislých komponent, které **lze používat také bez samotného frameworku**. Komponenty PHPixie jsou stoprocentně prověřené přes jednotkové testy a požadují jen minimum závislostí.