

Jakub Vrána

1001 TIPŮ A TRIKŮ PRO

PHP



CD obsahuje

veškeré zdrojové kódy z knihy,
instalaci Apache, MySQL
a další užitečné nástroje

Nejlepší postupy a řešení pro vaše webové aplikace

- ▲ Využití AJAXu, návrhových vzorů, export a import dat
- ▲ Zabezpečení dat, ladění výkonu aplikace
- ▲ Práce se soubory, XML dokumenty, knihovnamy a frameworky
- ▲ Regulární výrazy, zpracování formulářů a e-mailů

Jakub Vrána

1001 tipů a triků pro PHP

Computer Press
Brno
2012

1001 tipů a triků pro PHP

Jakub Vrána

Obálka: Martin Sodomka

Odpovědný redaktor: Martin Herodek

Technický redaktor: Jiří Matoušek

Objednávky knih:

<http://knihy.cpress.cz>

www.albatrosmedia.cz

eshop@albatrosmedia.cz

bezplatná linka 800 555 513

ISBN 978-80-251-2940-1

Vydalo nakladatelství Computer Press v Brně roku 2012 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 16097.

© Albatros Media a. s. Všechna práva vyhrazena. Žádná část této publikace nesmí být kopírována a rozmnožována za účelem rozšiřování v jakékoli formě či jakýmkoli způsobem bez písemného souhlasu vydavatele.

Dotisk 1. vydání.

 **ALBATROS** MEDIA a.s.

Stručný obsah

Úvod	33
Přehledný kód	35
Běžové prostředí	47
Instalace a konfigurace	53
Syntaxe jazyka	75
Jazykové konstrukce	89
Proměnné	103
Pole	121
Objektové programování	131
Návrhové vzory	153
Regulární výrazy	159
Webové aplikace	167
Zpracování formulářů	189
Práce se soubory	209
Databáze	227
Ukládání dat	271
Vícejazyčné aplikace	289
E-mail	303
AJAX	317
Práce s obrázky	321
XML dokumenty	325
Export a import dat	335
Knihovny a frameworky	347
Bezpečnost aplikace	355
Zabezpečení dat	377
Výkonnost	395
Hotové aplikace	417
Vývojové prostředí	423
Vývoj PHP	437
Poslední tip	443

Obsah

Úvod	33
Přehledný kód	35
1 Jak je to s velikostí písmen	35
2 Jak je to s velikostí písmen u zkratek	35
3 Jak je to s velikostí písmen v URL	35
4 Diakritika v identifikátorech	36
5 Diakritika v komentářích	36
6 Rozmístění tříd do souborů	36
7 Pojmenování rozhraní	36
8 Co to je značka Byte-Order-Mark	36
9 Čeština nebo angličtina?	36
10 Jazyk textů a komentářů	37
11 Pojmenování funkcí	37
12 Název aplikace v identifikátorech	37
13 Velikost písmen v SQL dotazech	38
14 Jak vypadá Spaghetti code	38
15 Jak na odsazování kódu	38
16 Znak pro odsazování kódu	39
17 Odsazování konstrukce switch	39
18 Odsazování kódu v kombinaci s HTML	39
19 Zarovnávání operátorů	40
20 Odřádkování před operátorem	40
21 Jak na konce řádků	41
22 Konec řádku na konci souboru	41
23 Ukončení posledního prvku pole	41
24 Pořadí parametrů funkcí	41
25 Pojmenování parametrů funkcí	42
26 Závorky kolem ternárního operátoru	42
27 Použití apostrofy nebo uvozovky?	42
28 Řídící konstrukce s bloky kódu	43
29 Volání funkce před její definicí	43
30 V jakém pořadí definovat metody	44
31 Funkce vypisující data	44
32 Postradatelné proměnné a funkce	44

	Běhové prostředí	47
33	Použití PHP pro webové aplikace	47
34	Použití PHP z příkazové řádky	47
35	Syntaktická kontrola skriptu	47
36	Jiná konfigurace při spuštění z příkazové řádky	47
37	Spuštění PHP kódu bez vytváření souboru	48
38	Interaktivní vyhodnocování kódu	48
39	Vstup a výstup z příkazového řádku	48
40	Chyba skriptu spuštěného z příkazového řádku	48
41	Použití chybového výstupu	49
42	Načtení vstupu z webové služby	49
43	Zápis na výstup z webové aplikace	49
44	Načtení konfigurace PHP	50
45	Detekce běhového prostředí	50
46	Definice konfiguračního souboru	50
47	Pravidelné spuštění úloh	51
48	Pravidelné spuštění úloh z webu	51
49	Aktuální adresář při spuštění skriptu	51
50	Co je to PHP-GTK	51
51	Jak na PHP pro Javu	52
52	Jak na PHP pro .NET	52
	Instalace a konfigurace	53
53	Jak zjistit verzi PHP	53
54	Aktuální verze PHP	53
55	Nastavení konfigurace PHP	53
56	Konfigurace pro adresář	53
57	Centrální konfigurace pro adresář	54
58	Vynucení konfigurace	54
59	Použití konstant v konfiguraci PHP	54
60	Použití jednotek v konfiguraci PHP	54
61	Pravdivostní hodnoty konfiguračních direktiv	55
62	Otevírání vzdálených souborů	55
63	Vkládání vzdálených souborů	55
64	HTML nebo XHTML?	56
65	Výběr kódování	56
66	Nastavení kódování	56
67	Předefinování řetězcových funkcí	56

68	Jak na hlášení chyb	57
69	Práce s neinicializovanými proměnnými	57
70	Proměnná s chybovou hláškou	58
71	Využití obsluhy chyby pro získání chybové hlášky	58
72	HTML kód v chybách	58
73	Chyby v SQL dotazech	59
74	Instalace PHP na vlastním počítači	59
75	Instalace PHP na IIS	60
76	Instalace PHP a aplikací na IIS	60
77	Jaká je licence PHP	60
78	Jaká je licence MySQL	60
79	Nahrávání extenzí	61
80	Název souboru s extenzí	61
81	Vývojové a produkční prostředí	61
82	Odkazy z chybových hlášek	62
83	Konfigurace aplikace specifická pro server	62
84	Povolení ladícího režimu	62
85	Nevhodné povolení ladícího režimu	63
86	Registrace proměnných zvenku	63
87	Pořadí proměnných zvenku	64
88	Automatické ošetřování vstupu od uživatele	64
89	Vypnutí automatického ošetřování vstupu od uživatele	64
90	Automatické ošetření načítaných dat	65
91	Automatické zahájení session	65
92	Platnost session proměnných	65
93	Mazání starých session dat	66
94	Přidání session identifikátoru do URL	66
95	Odmítnutí session identifikátoru z URL	66
96	Konstanta se session identifikátorem	66
97	Vytváření session identifikátoru	67
98	Oddělovač parametrů URL	67
99	Nastavení limitů PHP	67
100	Nastavení omezení paměti	68
101	Co je to bezpečný režim	68
102	Omezení adresáře skriptů	69
103	Umístění dočasných souborů	69
104	Jak skrýt přítomnost PHP	69
105	Vložení souboru do všech skriptů	69

106	Konfigurace e-mailu	70
107	Jak nastavit časové pásmo	70
108	Konfigurace MySQL	70
109	Co je to MySQLnd	70
110	Zastaralé direktivy	71
111	Jaký použít oddělovač adresářů	71
112	Oddělovač cest	71
113	Vypsání XML značky	71
114	Použití konstant pro vlastní konfiguraci	72
115	Konstanty bez rozlišení velikosti písmen	72
116	Definice absolutní cesty	72
117	Definice absolutního URL	73
118	Kde je uložena cesta k sobě	73
119	Instalace vlastní aplikace	73
	Syntaxe jazyka	75
120	Možnosti uzavírání PHP kódu	75
121	Vynechání koncové značky	75
122	Konec řádku za koncovou značkou	75
123	Ukončení příkazu	75
124	Ignorování zbytku souboru	75
125	Ukončení příkazu pro ignorování zbytku souboru	76
126	Druhy komentářů	76
127	Ukončení jednořádkového komentáře	76
128	Dokumentační komentáře	77
129	Značky dokumentačních komentářů	77
130	Jak na „Nutno dodělat“	77
131	Zpracování dokumentačních komentářů	78
132	Jak na anotace	78
133	Co je operátor identity	78
134	Jaký použít operátor nerovnosti	78
135	Pořadí porovnávání	79
136	Co je to ternární operátor	79
137	Zkrácený ternární operátor	79
138	Jak používat operátor plus	79
139	Pořadí násobení a dělení	80
140	Bitový posun	80
141	Obvyklý bitový posun	80
142	Bitová negace	80

143	Definice konstant	81
144	Použití konstant	81
145	Nedefinované konstanty	81
146	Konstanty obsahující pole	81
147	Jak uložit návratovou hodnotu funkce	82
148	Interní funkce s proměnným počtem parametrů	82
149	Předefinování funkce	83
150	Definice výjimky	83
151	Rozšíření výjimky	83
152	Probublání výjimky	84
153	Eskalace výjimky	84
154	Příklad chyb na výjimky	85
155	Typy výjimek v PHP	85
156	Zobrazení typu výjimky	85
157	Obsluha nezachycené výjimky	86
158	Skrytí chyb	86
159	Zobrazení všech chyb	87
160	Úklid po chybě	87
	Jazykové konstrukce	89
161	Univerzálnost cyklu for	89
162	Došli jsme na konec cyklu?	89
163	Počet průchodů cyklem	89
164	Jak použít operátor čárka	90
165	Alternativní syntaxe řídicích konstrukcí	90
166	Řetězení podmíněného příkazu	90
167	Vyvolání chyby v podmíněném příkazu	90
168	Pořadí bloků podmíněného příkazu	91
169	Zkrácené vyhodnocování podmínek	91
170	Jaké zvolit pořadí podmínek	92
171	Opakované vyhodnocování podmínky	92
172	Jaké použít logické operátory	92
173	Priorita logických operátorů	92
174	Priorita operátoru zřetězení	93
175	Vyskočení z nadřazeného cyklu	93
176	Co jsou speciální funkce	93
177	Používat include nebo require?	94
178	Kdy používat include_once?	94
179	Kontrola opakovaného vložení zevnitř knihovny	94

180	Ukončení vkládání souboru	95
181	Testování úspěšnosti vložení souboru	95
182	Využití návratové hodnoty vložených souborů	95
183	Globální proměnné ve vkládaných souborech	96
184	Ukončení funkce bez návratové hodnoty	96
185	Inicializace statických proměnných	96
186	Hromadné přiřazení proměnných	97
187	Jakou použít koncovku u vkládaných souborů	97
188	Vkládání souboru z aktuálního adresáře	98
189	Vložení souboru z adresáře zpracovávaného skriptu	98
190	Použití jmenného prostoru	98
191	Co je to aserce	99
192	Funkce s libovolným počtem parametrů	99
193	Co je to anonymní funkce	100
194	Jak používat uzávěry	100
195	Vytvoření funkce z řetězce	101
196	Vyhodnocení kódu v řetězci	101
197	Omezené vyhodnocení kódu v řetězci	101

Proměnné **103**

198	Rozsah viditelnosti proměnných	103
199	Přístup ke globálním proměnným z funkce	103
200	Reference a globální proměnné	104
201	Proměnné proměnné	104
202	Proměnné s nepovolenými znaky	104
203	Proměnné s rozšířenými znaky	105
204	Je proměnná nastavena?	105
205	Jsou proměnné nastavené?	105
206	Zrušení proměnné	106
207	Přiřazení proměnných hodnotou	106
208	Přiřazení proměnných referencí	106
209	Předání parametrů referencí	107
210	Předání parametrů referencí při volání	107
211	Přiřazení objektů referencí	107
212	Volitelný parametr předávaný referencí	108
213	Jak použít přiřazení jako výraz	108
214	Zachování typu proměnné	108
215	Zachování významu proměnné	109
216	Zjištění typu proměnné	109

217	Ověření typu řetězec	109
218	Nastavení typu proměnné	109
219	Převod na pravdivostní hodnotu	110
220	Jaké používá PHP velikosti čísel	110
221	Jak pracovat s přesnými čísly	110
222	Bitový posun	110
223	Zápis hexadecimálních čísel	111
224	Zápis čísel v osmičkové soustavě	111
225	Zápis čísel ve vědecké notaci	111
226	Převod uživatelského vstupu na číslo	111
227	Kontrola číselného řetězce	111
228	Používání čísel v kódu	112
229	Nastavení jazyka	112
230	Řetězec jako posloupnost bajtů	112
231	Přístup k jednomu znaku v řetězci	113
232	Binární bezpečnost řetězcových funkcí	113
233	Speciální znaky v řetězci	113
234	Znaková sada a kódování	113
235	Jak na převod kódování	114
236	Délka řetězce v kódování UTF-8	114
237	Samotné zpětné lomítko v řetězci	114
238	Jak na víceřádkové řetězce	114
239	Co je to heredoc	115
240	Řetězec bez interpretace speciálních znaků	115
241	Přístup k prvkům pole uvnitř řetězce	116
242	Přístup ke složitějším strukturám z řetězce	116
243	Přístup k libovolným strukturám z řetězce	116
244	Je řetězec neprázdný?	116
245	Porovnávání řetězců	117
246	Porovnání řetězců bez rozlišení velikosti písmen	117
247	Porovnání číselných řetězců	118
248	Porovnání prvních několika znaků řetězce	118
249	Končí text tímto řetězcem?	118
250	Záměna více řetězců najednou	118
251	Záměna více bajtů najednou	119
252	Ošetřování parametrů funkcí	119
253	Ošetření chyby při deserializaci proměnných	119

	Pole	121
254	Sjednocení polí	121
255	Spojení polí	121
256	Přidání prvku na konec pole	121
257	Přidání prvku na začátek pole	121
258	Vložení prvků doprostřed pole	122
259	Jaký je typ klíčů pole	122
260	Načítání unikátních identifikátorů	123
261	Procházení polí	123
262	Procházení pole referencí	123
263	Získání náhodného prvku pole	124
264	Jak realizovat víceúrovňová pole	124
265	Výchozí parametry funkce v poli	124
266	Třídění polí	125
267	Stabilita třídění	125
268	Třídění českých polí	125
269	Zřetězení prvků pole	126
270	Obalení prvků pole řetězcem	126
271	Výchozí hodnota nenastaveného prvku pole	126
272	Vyhledání hodnoty v poli	127
273	Získání prvního prvku pole	127
274	Získání libovolného prvku pole	127
275	Průchod více poli současně	128
276	Pole pevně dané velikosti	128
277	Jak implementovat frontu a zásobník	128
278	Rychlejší fronta a zásobník	129
279	Jak detekovat pole	129
	Objektové programování	131
280	Vytvoření objektu bez parametrů konstruktoru	131
281	Jak na konstruktor v PHP 4	131
282	Jak na konstruktor od PHP 5	131
283	Zakázání vytvoření objektu	131
284	Co je to vícenásobný konstruktor	132
285	Volání destrukturu při chybě	133
286	Přístup k objektu vrácenému funkcí	133
287	Co je to standardní třída	133
288	Přetypování na objekt	133
289	Jak se odkázat na aktuální objekt	134

290	Zjištění názvu třídy	134
291	Inicializace vlastností	134
292	Veřejné vlastnosti	135
293	Neveřejné vlastnosti v PHP 4	135
294	Deklarace více vlastností	135
295	Jak definovat interní veřejné metody	135
296	Převod na řetězec	136
297	Explicitní převod na řetězec	136
298	Práce s neinicializovanými vlastnostmi	137
299	Vyvolání chyby při zápisu do neinicializované vlastnosti	137
300	Vlastnosti pouze pro čtení	137
301	Jak na serializaci objektů	138
302	Jak na deserializaci objektů	139
303	Jak exportovat objekty	139
304	Jak klonovat objekty	139
305	Co je mělká kopie objektu	140
306	Co je hluboká kopie objektu	140
307	Automatická hluboká kopie objektu	141
308	Předávání metod k zavolání	141
309	Předávání statických metod k zavolání	141
310	Automatické nahrávání tříd	142
311	Jmenné prostory při automatickém nahrávání tříd	142
312	Více funkcí pro nahrávání tříd	142
313	Deserializace objektů bez definovaných tříd	142
314	Viditelnost protected	143
315	Viditelnost členů jiných objektů stejné třídy	143
316	Viditelnost pro přátele	143
317	Využití abstraktních tříd	144
318	Určení typu parametru	145
319	Co je to rozhraní	145
320	Implementace více rozhraní	146
321	Změna parametrů zděděných metod	146
322	Procházení objektů	147
323	Přístup k indexu objektu	148
324	Jak zjistit počet prvků objektu	148
325	Objekt jako pole	149
326	Jak zavést silnou kontrolu typů	149
327	Kontrola skalárních typů v parametrech	149

328	Co je to reflexe	150
329	Využití reflexe pro testování	150
330	Objekty jako klíče pole	150
	Návrhové vzory	153
331	Singleton	153
332	Důkladný Singleton	153
333	Registry	154
334	Registry s odloženou inicializací	154
335	Factory	155
336	Observer	156
337	Model-View-Controller	156
338	Active record	157
	Regulární výrazy	159
339	Knihovny regulárních výrazů	159
340	Oddělovač regulárních výrazů	159
341	Zpětné lomítko v regulárním výrazu	159
342	Nadbytečná zpětná lomítka	160
343	Speciální znaky v regulárním výrazu	160
344	Libovolný počet opakování	160
345	Konec řetězce v regulárním výrazu	161
346	Co jsou to třídy znaků	161
347	Nalezení výčtu znaků v řetězci	161
348	Kontrola alfanumerických znaků	161
349	Jak vyhledat všechna písmena	162
350	Ošetření textu v regulárním výrazu	162
351	Nalezené číslo řádku	162
352	Nalezení více regulárních výrazů	162
353	Neuložení nalezeného výskytu	163
354	Pojmenování nalezených výskytů	163
355	Zvýraznění odkazů v textu	163
356	Zkrácení odkazů v textu	164
357	Jak zjistit počet provedených záměn	165
358	Co jsou hladové kvantifikátory	165
359	Aserce v regulárních výrazech	165

	Webové aplikace	167
360	Struktura adresy URL	167
361	Předání parametrů v URL	167
362	Speciální znaky v názvech parametrů	168
363	Odstranění parametru z URL	168
364	Sestavení parametrů URL	168
365	Odkaz na první stránku	169
366	Odkazy na části stránky	169
367	Absolutní a relativní odkazy	169
368	Použití relativních odkazů	170
369	Jak používat stavové hlavičky	170
370	Alternativní způsob nastavení stavových hlaviček	171
371	Okamžik odeslání hlaviček	171
372	Poslání stavového kódu v případě chyby	171
373	Způsoby přesměrování	172
374	Jak adresovat výchozí stránku aplikace	172
375	Přesměrování na domácí stránku aplikace	172
376	Jaká je struktura souboru pro roboty	173
377	Skrytí stránek před vyhledávači	173
378	Co znamená bezstavovost webových aplikací	173
379	Uložení informací o uživateli do cookies	174
380	Nastavení a přečtení cookie	174
381	Práce s odeslanou cookie	174
382	Odstranění cookie	174
383	Použití cookies pro předvyplnění formuláře	175
384	Nastavení doby platnosti cookies	175
385	Prodloužení doby platnosti cookie	175
386	Velikost cookies	176
387	Cookies pro více adresářů	176
388	Cookies pro více domén	177
389	Cookies třetích stran	177
390	Co jsou session proměnné	177
391	Nastavení a přečtení session proměnné	177
392	Trvalé nastavení filtru uživatele	178
393	Automatické vložení konce stránky	178
394	Umístění vkládaných souborů do jiného adresáře	179
395	Umístění vkládaných souborů do podadresáře	179
396	Jak vytvořit pěknou URL	179

397	Přepis URL	180
398	Přepis pěkných URL	180
399	Přesměrování starých URL	180
400	Automatické vytvoření pěkného URL	181
401	Pěkná URL s identifikátorem	182
402	Přepis všech URL	182
403	Změna systému URL	183
404	Jak na přesměrování domén	183
405	Doména s www nebo bez?	183
406	Pěkná URL pro chudé	184
407	Pěkná URL přes chybovou stránku	184
408	Jak nastavit chybovou stránku	184
409	Velikost písmen v URL	185
410	Návrat na předchozí stránku	185
411	Detekce zastaralého odkazu	186
412	Automatické zastavení skriptu	186
413	Externí vyhledávání na stránkách	186
414	Vyhledávání v placených stránkách	187
415	Jak správně realizovat demoverzi	187
416	Zjištění aktuální verze	188
	Zpracování formulářů	189
417	Rozdíl mezi metodami GET a POST	189
418	Přístup k datům od uživatele	189
419	Přístup k datům od uživatele z funkcí	190
420	Přesměrování po odeslání formuláře metodou POST	190
421	Zobrazení informace o provedené operaci	191
422	Uložení informace o provedené operaci	191
423	Předání informace o provedené operaci	192
424	Kód přesměrování po odeslání formuláře	192
425	Formát hlavičky Location	193
426	Hlavička Location se zabezpečenými stránkami	193
427	Hlavička Location s číslem portu	193
428	Přesměrování na část dokumentu	193
429	Typ dat z formuláře	194
430	Prázdná hodnota z formuláře	194
431	Odeslání pole	195
432	Zpracování odeslaného pole	195
433	Indexace prvků v odeslaném poli	195

434	Složitější index v odeslaném poli	196
435	Odeslání formuláře na sebe sama	196
436	Metoda GET přepíše parametry v URL	196
437	Detekce odeslání formuláře	197
438	Jak nastavit název odesílacího tlačítka	197
439	Jak realizovat více odesílacích tlačítek	198
440	Odesílací tlačítko pro každý řádek	198
441	Jak realizovat košík s produkty	199
442	Ukládání košíku do session proměnné	199
443	Ukládání košíku do cookie	199
444	Ukládání košíku do databáze	200
445	Přidání položky do košíku JavaScriptem	200
446	Jak zjistit celkovou cenu košíku	201
447	Převod košíku do databáze	202
448	Opakované odeslání formuláře	202
449	Náhled komentáře	203
450	Smazání odeslaného komentáře	204
451	Logování IP adresy	205
452	Logování identifikátoru prohlížeče	205
453	Zobrazení IP adresy	205
454	Jak vytvořit výběrový seznam	206
455	Jak vytvořit výběrový seznam se skupinami	207
456	Zpracování formuláře po automatickém odhlášení	207
457	Jaká je maximální délka řetězců	208
458	Vypnutí doplňování hodnot	208
	Práce se soubory	209
459	Získání celého obsahu souboru	209
460	Jak určit délku načtených dat	209
461	Vypsání obsahu souboru	209
462	Jak detekovat konce řádků	209
463	Jak zjistit čitelnost souboru	210
464	Cache informací o souboru	210
465	Zapsání proměnné do souboru	210
466	Co je atomicita operací	210
467	Jak na zamykání souborů	211
468	Atomická práce se soubory	211
469	Datum poslední modifikace aktuálního souboru	211
470	Vytvoření dočasného souboru	211

471	Práce s daty v paměti	212
472	Procházení adresářů	212
473	Rekurzivní procházení adresářů	212
474	Vestavené rekurzivní procházení adresářů	213
475	Průchod souborů ve více adresářích	213
476	Poslání dat metodou POST	213
477	Získání souboru přes proxy server	214
478	Přihlašovací údaje ke vzdálenému souboru	214
479	Získání vrácených hlaviček	214
480	Stažení souboru extenzí CURL	214
481	Přenos cookies při stahování souboru	215
482	Ruční zpracování cookies	215
483	Stažení souboru na pozadí	216
484	Ruční získání HTTP odpovědi	216
485	Ruční stažení souboru	217
486	Zpracování HTML stránky	217
487	Zjištění kódování HTML stránky	218
488	Zjištění jazyka HTML dokumentu	219
489	Zjištění kódování českého textu	219
490	Zapsání binárních dat	220
491	Načtení binárních dat	220
492	Práce s komprimovanými soubory	220
493	Nahrávání souborů	221
494	Práva k nahraným souborům	221
495	Nahrání více souborů	221
496	Přesunutí souboru od uživatele	222
497	Kontrola nahraného souboru	222
498	Původní název nahraného souboru	222
499	Načtení souboru od uživatele	222
500	Vlastník nahraného souboru	223
501	Výchozí práva vytvářených souborů	223
502	Velikost nahrávaného souboru	223
503	Zjištění koncovky souboru	223
504	Ukládání souborů do databáze	224
505	Uložení souboru od uživatele do databáze	224
506	Načítání souborů z databáze	224
507	Ukládání souborů do PostgreSQL	225
508	Stažení části souboru	225

	Databáze	227
509	Nezávislost na databázovém systému	227
510	Výběr databázového systému	227
511	Jednotná práce s různými databázovými systémy	227
512	Co umožňuje extenze PDO	228
513	Ošetření chyb v PDO	228
514	Co umožňuje knihovna Dibi	229
515	Co umožňuje knihovna NotORM	229
516	Přenášení nevyužitých dat z databáze	230
517	Co je to ORM	230
518	Jak používat operátor OR v MySQL	231
519	Hledání v seznamu hodnot	231
520	Zapisování identifikátorů	231
521	Ošetření identifikátorů	232
522	Co umožňuje databáze information_schema	232
523	Vestavěná databáze	232
524	Vlastní funkce v SQLite	233
525	Pevná struktura databáze	233
526	Pojmenování databázových tabulek	233
527	Využití typu int unsigned	233
528	Uložení desetinných čísel	234
529	Využití typu char	234
530	Maximální délka řetězce	234
531	Zadání neplatných dat	234
532	Zobrazení varování	234
533	Využití typu enum	235
534	Jaký je typ dat pocházejících z databáze	235
535	Formát kalendářního data	235
536	Nastavení časového pásma	236
537	Přehledný zápis SQL dotazů	236
538	Názvy sloupců při spojení více tabulek	237
539	Seznam sloupců v dotazu INSERT	237
540	Pevný název tabulky	237
541	Co dělá hvězdička v dotazu SELECT	238
542	Způsoby spojení tabulek	238
543	Význam hodnoty NULL	238
544	K čemu slouží indexy	238
545	Co jsou unikátní indexy	239

546	Využití unikátních klíčů	239
547	Co je to primární klíč	240
548	Co jsou automatické identifikátory	240
549	Jak na víceloupcové indexy	240
550	Existuje záznam?	241
551	Zjištění počtu řádků	241
552	Zjištění celkového počtu řádků	242
553	Jak realizovat stránkování	242
554	Omezení počtu stránek	243
555	Trvalé odkazy na stránkování	243
556	Počet zobrazovaných záznamů	244
557	Existuje další stránka?	244
558	Získání počtu měněných záznamů	245
559	Mezery v SQL příkazu	245
560	Postupné přenášení dat z databáze	246
561	Komunikace s databází během postupného přenášení dat	246
562	Hromadné vložení více záznamů	246
563	Položení více dotazů najednou	247
564	Zpracování vícenásobného výsledku	247
565	Asynchronní spuštění dotazu	248
566	Přístup k datům vráceným z databáze	248
567	Uložení vráceného řádku do proměnných	249
568	Automatické připojení k databázovému serveru	249
569	Určení kódování přenášených dat	249
570	Výchozí kódování přenášených dat	250
571	Ošetření řetězců	250
572	Ohraničování řetězců	250
573	Ošetření hodnot	250
574	Ošetření proměnných v PDO	251
575	Vázání proměnných	251
576	Vázání proměnných v MySQLi	252
577	Vázání proměnných v PDO	252
578	Zástupný znak ve vázání proměnných	253
579	Emulace vázání proměnných	253
580	Kdy nelze použít vázání proměnných	254
581	Uvolnění výsledku dotazu	254
582	Zavření připojení k databázovému serveru	254
583	Práce s více databázemi	255

584	Více připojení k databázovému serveru	255
585	Více připojení ke stejnému serveru	255
586	Co je to replikace MySQL	256
587	Jak vyřešit zpoždění replikace	256
588	Funkce pro vytvoření seznamu hodnot	257
589	Pořadí záznamu	258
590	Konstantní počet dotazů	258
591	Fronta požadavků	259
592	Co je perzistentní připojení	259
593	Omezení počtu perzistentních připojení	259
594	Úklid perzistentního připojení	260
595	Jak na perzistentní připojení v MySQLi	260
596	Jak na obnovení připojení	260
597	Jaká nabízí MySQL úložišť	260
598	Jak realizovat transakce	261
599	Získání dat pro úpravu	262
600	Chyby v transakcích	262
601	Zamykání tabulek	264
602	Využití cizích klíčů	264
603	Rekurzivní mazání odkazovaných záznamů	264
604	Vypnutí referenční integrity	265
605	Prohledávání databáze	265
606	Jak vytvořit fulltextový index	266
607	Operátory fulltextové vyhledávání	266
608	Fulltextový index s tabulkami InnoDB	266
609	Vyhledávání ve více tabulkách	267
610	Fulltextové vyhledávání Sphinx Search	267
611	Sphinx Search pro češtinu	268
612	Nalezení textu pomocí Sphinx	268
613	Zvýraznění nalezeného textu Sphinx	269
	Ukládání dat	271
614	Určení pořadí záznamů	271
615	Vlastní pořadí záznamů	271
616	Ukládání odpovědí ankety	271
617	Ukládání hlasů v anketě	272
618	Změna hlasu v anketě	272
619	Vložení složitějšího objektu do textu	273
620	Záměna značky v textu na objekt	273

621	Složení článku z elementárních objektů	274
622	Zjištění počtu přístupů po dnech	274
623	Import dat do databáze	275
624	Aktualizace importovaných dat	275
625	Přepsání importovaných dat	276
626	Smazání starých dat při importu	276
627	Uložení PSČ	277
628	Zobrazení PSČ	277
629	Kontrola rodného čísla	277
630	Kontrola IČ	278
631	Odkaz na počáteční písmena	278
632	Výpis záznamů od daného písmene	279
633	Zvýraznění aktivních počátečních písmen	280
634	Zobrazení otevírací doby	280
635	Je otevřeno?	281
636	Zobrazení otevíracích hodin	281
637	Zvýraznění dnešní otevírací doby	282
638	Jak ukládat parametry produktů	282
639	Zobrazení parametrů	282
640	Uložení pravdivostních hodnot	283
641	Práce s datem	283
642	Jak realizovat sezónní ceníky	284
643	Zjištění aktuální sezónní ceny	284
644	Zjištění ceny za více období	284
645	Zobrazení kalendáře	284
646	Získání části data	285
647	Posun v čase	285
648	Uložení nastavení do databáze	286
649	Co je to tag cloud	286
650	Jak ukládat hierarchická data	287
651	Logaritmičké odsazování diskusí	287
	Vícejazyčné aplikace	289
652	Internacionalizace a lokalizace	289
653	Formátování výstupů	289
654	Formátování čísel	289
655	Formátování čísla s pevnou mezerou	290
656	Formátování data	290
657	Převod data z mezinárodního formátu	290

658	Zpracování formátovaného data	291
659	Jak zobrazit české názvy měsíců	291
660	Jak zobrazit české názvy dní v týdnu	292
661	Zobrazení českého formátu data	292
662	Zobrazení relativního českého data	293
663	Jak zvolit identifikátor překladů	294
664	Překlady obsahující proměnnou	294
665	Vyhledání textů k překladu	294
666	Označení textů k překladu	295
667	Překlad textů ve zdrojovém kódu	296
668	Jednotné a množné číslo	296
669	K čemu slouží knihovna Gettext	297
670	Načítání překladů z databáze	298
671	Doplňování překladů do databáze	298
672	Jak překládat texty s vazbou na data	299
673	Odlišná data pro každý jazyk	299
674	Stejná data pro všechny jazyky	299
675	Práce s nepřeloženými texty	300
676	Jak předat adresu jazykových verzí	300
677	Jak předat kód jazyka	301
678	Detekce jazyka uživatele	301
679	Zobrazení detekované jazykové verze	302
680	Přepnutí jazykové verze	302
	E-mailý	303
681	Skrytí e-mailové adresy	303
682	Kontrola e-mailové adresy	303
683	Odesílání e-mailů	304
684	Oddělovač hlaviček	304
685	Čeština v těle zprávy	305
686	Čeština v hlavičkách zprávy	305
687	E-mail v kódování UTF-8	305
688	Jakou strukturu má patička zprávy	306
689	E-mailý ve formátu HTML	306
690	Převod HTML na text	307
691	Připojení souboru k e-mailu	307
692	Odeslání sestavené zprávy	308
693	Obrázky v e-mailu	308
694	Vložení obrázku do e-mailu	308

695	Odpovědní formulář	309
696	Kontakt na uživatele bez vyzrazení jeho e-mailu	310
697	Potvrzení e-mailové adresy	310
698	Upozornění na nové komentáře	311
699	Jedno upozornění na nové komentáře	311
700	Co zadat do adresy odesílatele	312
701	Notifikace o platbě	312
702	Stahování notifikací o platbě	312
703	Stažení seznamu plateb	313
704	Podepisování e-mailů	313
705	Podpis zprávy pomocí PGP	314
706	Podpis zprávy pomocí PKCS #7	314
707	Šifrování e-mailů	315
	AJAX	317
708	Využití AJAXu	317
709	Jak zjistit stav aplikace	317
710	Informování uživatele o provádění požadavku	317
711	Pořadí zpracování požadavků	318
712	Minimalizace počtu požadavků	318
713	Typ požadavků na server	318
714	Alternativní formát odesílaných dat	319
715	Formát odpovědi ze serveru	319
716	Identifikace požadavků AJAX	319
	Práce s obrázky	321
717	Přidání rozměrů obrázků	321
718	Doplnění rozměrů obrázků do HTML kódu	321
719	Co jsou to gravatary	322
720	Jaké volit formáty obrázků	322
721	Načtení obrázku v libovolném formátu	322
722	Jak vytvářet grafy	323
723	Co jsou to QR kódy	323
724	Vytvoření QR kódu	324
	XML dokumenty	325
725	Co obsahuje hlavička XML dokumentů	325
726	Načtení RSS exportu	325
727	Pravidelné načítání RSS exportu	325

728	Informování o RSS exportu	325
729	RSS export vyhledávání	326
730	Zrušení RSS exportu	326
731	Počet položek v RSS exportu	326
732	Co je to PubSubHubbub	327
733	Data načtená pomocí SimpleXML	327
734	Vyhledávání v SimpleXML	328
735	Test HTML atributu class	328
736	Získání textového obsahu značky	328
737	Jmenné prostory XML	329
738	Zjištění pořadí ve výsledcích vyhledávání	329
739	Entity při načtení XML dokumentů	330
740	Kontrola XML podle DTD	330
741	Kódování při zpracování HTML dokumentů	331
742	Vnucení kódování při zpracování HTML dokumentů	331
743	Zpracování velkých XML dokumentů	331
744	Převod XML dokumentu	332
745	Vytváření XML dokumentů	333
746	Úprava XML dokumentů	333
	Export a import dat	335
747	Co je to JSON	335
748	Načtení dat ve formátu JSON	335
749	Ošetření proměnné pro JavaScript	335
750	Využití protokolu data	336
751	Export do CSV	336
752	Jak nastavit oddělovač v CSV	336
753	Co nabízí formát Excelu	337
754	Export do Excelu	337
755	Export do šablony Excelu	337
756	Co nabízí formát PDF	337
757	Export do PDF	338
758	Převod HTML do PDF	338
759	Import kurzů měn	338
760	Jak vytvořit mapu serveru	339
761	Priorita v mapě serveru	340
762	Vytvoření mapy serveru	340
763	Mapa serveru z databáze	340
764	Odkaz na mapu serveru	341

765	Velikost mapy serveru	341
766	Export zboží	342
767	Formát exportu zboží	342
768	Vygenerování exportu zboží	342
769	Export geografických dat	343
770	Co umožňují webové služby	343
771	Položení XML-RPC požadavku	344
772	Co je WSDL dokument	344
773	Připojení k SOAP serveru	344
774	Vytvoření SOAP serveru	345
775	Ladění SOAP požadavků	345
	Knihovny a frameworky	347
776	Inteligentní objekty	347
777	Jak na ladění aplikací	347
778	Jak vytvořit inteligentní formuláře	348
779	Zadávání formátovaného textu	349
780	Kopírování formátovaného textu do Texy	349
781	Vizuální editory formátovaného textu	349
782	Vyčištění HTML kódu	350
783	Opravení HTML kódu	350
784	Jak použít šablony Smarty	350
785	Dědičnost šablon	351
786	Co je Nette Latte	351
787	Použití Nette Latte s atributy	352
788	Zvýraznění zdrojového kódu	352
789	Zvýraznění sebe sama	352
790	Geolokace z IP adresy	353
791	Zkrácení adresy	353
792	Zjištění cíle zkráceného odkazu	353
	Bezpečnost aplikace	355
793	Validace dat	355
794	Sanitizace dat	355
795	Cross-site scripting	355
796	Vytvoření HTML entit	356
797	Množina ošetřovaných dat	356
798	Nedůvěryhodné proměnné serveru	356
799	Odstranění značek	357

800	Využití šablon	357
801	Kontextově citlivé ošetřování dat	357
802	Oddělení HTML a PHP kódu	357
803	Určení kódování stránky	358
804	Kontrola kódování dat	358
805	Okamžik ošetřování dat	358
806	Jak na filtrování dat	359
807	Nastavení výchozího filtru	359
808	Vypnutí výchozího filtru	359
809	K čemu slouží blacklist a whitelist	360
810	Vkládání souborů	360
811	Inicializace proměnných	360
812	Nepodmíněná inicializace proměnných	361
813	Cookies neviditelné z prohlížeče	362
814	Jak vytvořit bezpečné cookies	362
815	Co je to Cross-Site Request Forgery	362
816	Obrana proti CSRF	363
817	Obrana proti CSRF pomocí hlavičky Referer	363
818	Obrana proti CSRF využitím parametru URL	364
819	Rozsah obrany proti CSRF	364
820	Obrana proti CSRF při e-mailových operacích	364
821	Co je to ClickJacking	365
822	Co je to Session Hijacking	365
823	Skrytí adresy stránky při navštívení odkazu	366
824	Jak zjistit totožnost uživatele	366
825	Co je to Session Fixation	367
826	Podvržení hlaviček	367
827	Pseudonáhodná a náhodná čísla	367
828	Inicializace náhodných čísel	368
829	Používání pseudonáhodných čísel	368
830	Skutečně náhodné číslo	368
831	Ukládání souborů od uživatele	369
832	Ověření názvu souboru od uživatele	369
833	Kontrola souborů od uživatele	369
834	Posílání souborů od uživatele	370
835	Omezení počtu operací podle IP adresy	370
836	Uložení informace o proxy serveru	371
837	Co jsou to reverzní proxy servery	371

838	Pozor na veřejné proxy servery	371
839	Jak odhalit Tor	371
840	Rozlišení robota a člověka	372
841	Zmatení robota	372
842	Detekce člověka pomocí JavaScriptu	373
843	Nespoléhat se na CAPTCHA	373
844	Zavření prohlížeče	373
845	Uživatel pro práci s databází	374
846	Více uživatelů na jednom serveru	374
847	Oddělení více uživatelů mimo PHP	374
848	Zakázání spuštění externích programů	374
849	Co je to penetrační testování	375

Zabezpečení dat

377

850	Co je to autentizace a autorizace	377
851	Způsoby autentizace	377
852	Způsoby získání hesla od uživatele	377
853	Základní HTTP autentizace	377
854	Pokročilá HTTP autentizace	378
855	Zobrazení přihlašovacího formuláře	378
856	Volitelné přihlášení	379
857	Umístění přihlašovacího formuláře	379
858	E-mail jako přihlašovací jméno	379
859	Co je to Security by Obscurity	380
860	K čemu slouží hašovací funkce	380
861	Ukládání hesel	380
862	Jak na omezení hesla	381
863	Jak kontrolovat složitost hesla	381
864	Bezpečnost hašovacích funkcí	382
865	Bezpečnější ukládání hesel	382
866	Hláška při zadání neplatného hesla	383
867	Opakované zadání neplatného hesla	383
868	Poslání zapomenutého hesla	384
869	Změna hesla	385
870	Předvyplnění hesla	385
871	Vypnutí doplnění přihlašovacího jména a hesla	386
872	Zadání části hesla	386
873	Přihlášení na více doménách	386
874	Přihlášení na více doménách s využitím JavaScriptu	387

875	Ukládání čísel kreditních karet	387
876	Vytvoření klíče pro asymetrickou kryptografii	388
877	Kdy použít protokol HTTPS	389
878	Jak na důvěryhodné certifikáty	389
879	Protokol výzva – odpověď	389
880	Proč použít role	391
881	Okamžik kontroly oprávnění	391
882	Reakce na chybějící oprávnění	392
883	Uložení session dat	392
884	Skrytí zdrojových kódů	394
885	Uložení přihlašovacích údajů k databázi	394
	Výkonnost	395
886	Jak použít HTTP hlavičky	395
887	K čemu slouží hlavička Expires	395
888	K čemu slouží hlavička Last-Modified	395
889	Jak zajistit vždy aktuální data	396
890	K čemu slouží hlavička If-Modified-Since	396
891	Čas poslední modifikace v databázi	396
892	Poslední modifikace souvisejících objektů	397
893	K čemu slouží hlavička ETag	397
894	K čemu slouží hlavička If-None-Match	397
895	Privátní cache	398
896	Cache při používání session proměnných	398
897	Pořadí předávaných parametrů	398
898	Reverzní proxy servery	398
899	HTTP hlavičky v požadavku	398
900	Bufferování výstupu	400
901	Bufferování celé stránky	400
902	Vysypání výstupu	401
903	Komprese výstupu	401
904	Komprese statických souborů	401
905	Jak zjistit aktuální datum a čas	401
906	K čemu slouží akcelerátory	402
907	Vyhodnocení podmínky cyklu	402
908	Automatické nahrávání tříd	402
909	Minimalizace kódu	403
910	Vlastní minimalizace kódu	403
911	Jak vytvořit archiv PHP skriptů	404

912	Načtení dat z PHP archivu	404
913	Vytvoření PHP archivu	404
914	Vytvoření PHP archivu s webovou aplikací	404
915	Převod PHP kódu do C++	405
916	Měření rychlosti	405
917	Měření rychlosti webové aplikace	405
918	Co je to profilování	406
919	Jak zjistit množství zabrané paměti	406
920	Chytřejší uvolňování paměti	406
921	Informování uživatele o průběhu operace	406
922	Pořadí zpracování šablon	406
923	Kopírování při zápisu	407
924	Kopírování s ternárním operátorem	407
925	Kopírování a reference	408
926	Uvozovky nebo apostrofy	408
927	Mikrooptimalizace	408
928	Prohledávání pole	409
929	Vyhledání více hodnot v poli	409
930	Ukládání do souborové cache	409
931	Jak funguje sdílená paměť	410
932	Vzdálená sdílená paměť	410
933	Ukládání session dat do Memcache	411
934	Zamykání session souborů	411
935	Ukládání session souborů do adresářů	411
936	Ruční mazání zastaralých session souborů	411
937	Někdy se dá obejít bez session proměnných	412
938	Jak realizovat frontu požadavků	412
939	Kešování SQL dotazů	412
940	Využití transakcí	413
941	Rychlý commit, pomalý rollback	413
942	Synchronizace disku při dokončení transakce	413
943	Zobrazení vysokého čísla stránky	414
944	Paralelní zpracování	414
945	Spouštění programů na pozadí	415
	Hotové aplikace	417
946	MediaWiki	417
947	WordPress	417
948	phpBB	418

949	phpMyAdmin	419
950	Adminer	419
951	Adminer Editor	420
952	Rozšíření pro Adminer	421
953	Synchronizace databáze	421
	Vývojové prostředí	423
954	Proč používat systém pro správu verzí	423
955	Co je to Subversion	423
956	Co je to Git	424
957	Správa požadavků	424
958	Propojení správy požadavků s verzováním	424
959	Propojení Subversion se správou požadavků	424
960	Použití verzovacího systému pro nasazení aplikace	425
961	Co je to Phing	425
962	Balíčky operačního systému	426
963	Prostředí pro vývoj svobodného softwaru	426
964	Testování aplikací	426
965	Testování pomocí PHPUnit	427
966	Testování pomocí PHPT	427
967	Vlastní spuštění testů	427
968	Co je to Selenium IDE	428
969	Co je to Selenium RC	428
970	Pokrytí kódu	429
971	Ladicí výpis proměnných	430
972	Formátovaný ladicí výpis proměnných	430
973	Ladicí informace v hlavičkách	431
974	Ladění pomocí FirePHP	431
975	Ladění aplikací	431
976	Připojení k FTP	432
977	K čemu slouží editor a vývojové prostředí	432
978	Co nabízí editor SciTE	432
979	Jak vypadá ikona PHP skriptů	433
980	Jak nastavit vlastní písmeno disku pro skripty	433
981	Asociace PHP skriptů	434
982	Asociace PHP skriptů pro Firefox	434
983	Jak na zálohování	434

	Vývoj PHP	437
984	Přímé odkazy do PHP manuálu	437
985	Poznámky dokumentace	437
986	Zrcadla stránek PHP	437
987	Vývojová verze dokumentace	438
988	Jak na hlášení chyb	438
989	Proč používat diskusní fóra	438
990	Kde hledat zdrojové kódy PHP	439
991	Jaká je architektura PHP	439
992	Jak realizovat více vláken	440
993	Tvorba extenzí	440
994	Křížová reference zdrojových kódů	440
995	K čemu slouží testy PHP	440
996	Zdrojové kódy dokumentace	441
997	Sestavení dokumentace	441
998	Pseudotypy používané v dokumentaci	441
999	Překlad oficiální dokumentace	442
1000	České weblogy o PHP	442
	Poslední tip	443
1001	Používejte vámi vytvářené aplikace	443
	Rejstřík	445

Úvod

Kromě toho, že se živím programováním, tak se také podílím na tvorbě oficiální dokumentace PHP a učím základy vytváření webových stránek na vysoké škole. Vedle toho píšu *blog.php.vrana.cz* a vedu několik školení o tvorbě webových aplikací se zaměřením na PHP, MySQL a JavaScript. Ze všech těchto vstupů čerpám inspiraci, a proto doufám, že všechny tipy a triky v této knize budou pro čtenáře nějakým způsobem zajímavé. S vatovými triky následujícího typu se tedy v této knize rozhodně nesetkáte:

1. Věděli jste, že oblouk nakreslíte funkcí `imageArc`?
2. Věděli jste, že obdélník nakreslíte funkcí `imageRectangle`?
3. Věděli jste, že čáru nakreslíte funkcí `imageLine`?

Místo toho najdete třeba i tipy týkající se výkonnosti webových aplikací, verzovacích systémů nebo dokonce vývoje PHP. I to je totiž pro programátora v PHP důležité.

Knihy u čtenáře předpokládá znalost syntaxe PHP a schopnost dohledávat si informace v dokumentaci. Pokud tedy nějaký tip např. pojednává o zajímavém využití nějaké funkce, tak se nevysvětluje význam jednotlivých parametrů, protože ten je popsán v dokumentaci.

Knihy byla napsána v době PHP 5.3, v době zrušení vývojové verze PHP 6, která měla zavést podporu pro Unicode. Zabývá se tedy verzemi PHP 5.3 a staršími.

Kompletní text knihy lze prohledávat pomocí hledání na webu *php.vrana.cz*.

Komu je kniha určena

Tipy jsou rozděleny do tří kategorií pro začátečníky, pokročilé a znalce. Rozdělení ale není ani tak podle náročnosti, jako spíš podle jejich potřebnosti.



začátečník

Tipy pro začátečníky by měl znát i začínající programátor (i když někdy mohou být složitější).



pokročilý

Tipy pro pokročilé by měla znát většina programátorů PHP.



znalec

Bez znalosti tipů pro znalce se dá často obejít, i když samozřejmě mohou poskytnout nějakou zkratku nebo usnadnění práce.

Odlišení písma

V knize se používá tučné písmo pro názvy produktů, kurziva pro odborné termíny, neproporcionální písmo pro ukázky kódu. V ukázkách kódu jsou tučně klíčová slova a vestavěné funkce, kurzivou proměnné.

Poděkování

Chtěl bych poděkovat Davidu Grudlovi, který je autorem několika knihoven použitých v této knize a také spoluautorem kódu u několika tipů. Také mu děkuji za řadu cenných připomínek.

Doprovodné CD

Doprovodný disk obsahuje kromě zdrojových kódů také řadu odkazů na užitečné stránky a také několik užitečných nástrojů, jež vám programování v jazyce PHP výrazně usnadní nebo alespoň zpříjemní.

CD stačí vložit do počítače a rozhraní se spustí automaticky. Pokud nemáte automatické spouštění disků povoleno, vyhledejte na CD kořenový adresář a otevřete soubor `spustit_CD.html`.

Jestliže rozhraní CD otevřete v prohlížeči Internet Explorer, Opera nebo Google Chrome, budete z CD moci instalovat doprovodný software okamžitě. V případě jiných prohlížečů se zobrazí výzva k uložení instalačního souboru na pevný disk. V tomto případě doporučujeme spustit instalaci přímo z CD. Obsah CD najdete ve složce obsah.

Přehledný kód

1 Jak je to s velikostí písmen



V důsledku toho, že PHP převzalo řadu funkcí a obrátů z jiných programovacích jazyků, nemá jednotné pravidlo pro používání velkých a malých písmen. Alespoň některá pravidla jsou ale společná:

- Proměnné a funkce je zvykem začínat malým písmenem.
- Názvy tříd obvykle začínají velkým písmenem.
- Konstanty se píšou celé velkými písmeny a slova se oddělují podtržítkem.

Nejednotnost tkví hlavně v oddělování slov. Většina vestavěných funkcí PHP používá pro oddělení slov podtržítko (např. `array_keys`), pokud to ovšem nejsou zkratky (jako např. `strpos`). Některé funkce se toho ale nedrží a slova nijak neoddělují (např. `imageCreate`).

Objektový kód obvykle slova odděluje zvětšením písmene (např. `PDO::errorCode`), i když ani to neplatí vždy (`mysqli::select_db`).

U funkcí a tříd PHP velikost písmen nerozlišuje, u proměnných ale ano. V této knize budeme pro oddělování slov používat zvětšení písmene, z důvodu lepší čitelnosti i u vestavěných funkcí PHP.

2 Jak je to s velikostí písmen u zkratk



Pokud je součástí identifikátoru (např. proměnné nebo funkce) nějaká zkratka, je vhodné ji zapisovat malými písmeny stejně jako ostatní slova. Jinak bude identifikátor hůře čitelný, obzvlášť v situaci, kdy by zkratek bylo více za sebou.

```
<?php
// dobře čitelné
createXmlHttpRequest($xmlHttp);

// špatně čitelné, proměnné by navíc měly začínat malým písmenem
createXMLHttpRequest($XMLHTTP);
?>
```

3 Jak je to s velikostí písmen v URL



Adresu URL je zvykem zapisovat malými písmeny a slova oddělovat spojovníkem, stejný způsob pojmenování tedy volíme u skriptů přímo zobrazovaných na webu.

U souborů vkládaných do jiných skriptů odvíjíme název souboru od toho, co v souboru definujeme. Pokud to je např. třída, tak soubor můžeme pojmenovat stejně jako třídu, včetně velikosti písmen.

4 Diakritika v identifikátorech



začátečník

PHP dovoluje v názvech proměnných a v dalších identifikátorech používat i znaky z horní poloviny ASCII tabulky. Pokud se tedy rozhodneme zadávat identifikátory česky, můžeme je psát i s diakritikou. V praxi se toho ale téměř nevyužívá, protože tento způsob zápisu nemusí podporovat všechny editory.

5 Diakritika v komentářích



začátečník

Pokud se rozhodneme komentáře k programu psát česky, měli bychom zásadně používat diakritiku (háčky a čárky). Souvislejší texty bez diakritiky se totiž velmi špatně čtou. Zdrojové kódy se v poslední době obvykle píšou v kódování UTF-8.

6 Rozmístění tříd do souborů



pokročilý

Každá třída se obvykle umísťuje do samostatného souboru pojmenovaného podle jejího názvu. Toto pravidlo se občas porušuje u definic výjimek, které se buď definují hromadně v jednom souboru, nebo spolu s třídou, která je vyvolává.

7 Pojmenování rozhraní



znalec

Název rozhraní (interface) v některých projektech začíná znakem I, zdaleka to ale není pravidlem. Stejně jako třídy se rozhraní obvykle umísťují do souborů pojmenovaných podle jejich názvu.

8 Co to je značka Byte-Order-Mark



pokročilý

Soubory ve znakové sadě Unicode mohou začínat značkou *Byte-Order-Mark* (BOM). To je posloupnost bajtů na začátku souboru, která obsahuje informaci o použitém kódování. Platí to i pro kódování UTF-8, které lze používat pro vytváření PHP skriptů. Tuto značku ale PHP nijak neinterpretuje, takže ji není vhodné používat. Jinak může dojít k tomu, že např. funkce header skončí chybou, protože už skript odeslal nějaký výstup (neviditelnou značku BOM) nebo že skript odešle těchto značek více (pokud do sebe vkládáme více skriptů, které by měly značku uvedenou).

9 Čeština nebo angličtina?



začátečník

Při programování bychom si měli vybrat nejen programovací jazyk (což je nevyhnutelné), ale i ten lidský, který budeme používat pro identifikátory a texty v programu. U identifikátorů, jako je název proměnných nebo funkcí, je obvykle přirozené používat angličtinu, protože i návazný kód používá angličtinu.

```
<?php
$xmlReader = new XmlReader; // působí přirozeně
$cteckaXml = new XmlReader; // působí krkolmně
?>
```

V některých případech se navíc používají ustálené obraty jako např. název metody `getConnection`. České varianty, ať už v podobě `getPripojeni` nebo třeba `dejPripojeni`, působí nepřirozeně.

V knize tedy budeme používat anglické identifikátory.



Poznámka: Stejně pravidlo lze použít i pro pojmenování souborů.

10 Jazyk textů a komentářů



začátečník

U vypisovaných textů je situace daná tím, zda vytváříme jednojazyčnou nebo vícejazyčnou aplikaci. U jednojazyčné je přirozené psát texty rovnou ve výsledném jazyce, u vícejazyčné potom v angličtině (pokud je jedním z výsledných jazyků).

U komentářů záleží na tom, kdo s nimi bude pracovat. Pokud všichni autoři i čtenáři komentářů ovládají dostatečně dobře angličtinu, tak je vhodné psát komentáře právě v ní, aby byl jazyk v souladu s identifikátory. Pokud taková situace nenastane, tak je vhodné psát komentáře jazykem, který všichni zúčastnění ovládají nejlépe.

Vypisované texty a komentáře v knize budou tedy česky.

11 Pojmenování funkcí



začátečník

Název funkcí a metod je vhodné začínat slovesem určujícím, co přesně vlastně funkce nebo metoda dělá. Např. `getConnection` je lepší název než samotné `connection`, protože je z názvu jasné, že jde o získání připojení, a ne např. o jeho nastavení. Slovesem je vhodné název funkce vždy začínat, `getConnection` je lepší než `connectionGet`. U neobjektového kódu je možné názvu funkce předradit modul, kterého se týká (např. `imageCreate`), u metod objektů to není potřeba.

12 Název aplikace v identifikátorech



pokročilý

Budoucí verze PHP mohou používat nová klíčová slova nebo jiné identifikátory, které by mohly kolidovat s identifikátory použitými naší aplikací. V zájmu dopředné kompatibility bychom tedy v hlavním jmenném prostoru aplikace neměli používat příliš obecné identifikátory. Např. PHP 5.1 zavedlo třídu `Date` (ta byla po sporech nakonec přejmenována na `DateTime`). Stejně tak mohou nové identifikátory zavádět používané knihovny.

Předřazovat všem identifikátorům název aplikace ale také není zrovna nejšťastnější, protože to identifikátory prodlužuje a zpřehledňuje.

```
<?php
// v budoucnu může kolidovat
class Database {
}

// dlouhý a nepřehledný identifikátor
class MyAppDatabase {
}
```

?>

Čisté řešení tohoto problému nabízí jmenné prostory, které jsou k dispozici od PHP 5.3.

```
<?php
namespace MyApp;
class Database {
}
new Database; // použití zevnitř jmenného prostoru
new \MyApp\Database; // použití kdekoliv
?>
```

13 Velikost písmen v SQL dotazech



začátečník

V SQL dotazech je zvykem klíčová slova uvádět velkými písmeny a vlastní identifikátory (především názvy tabulek a sloupců) malými písmeny. Slova se obvykle oddělují podtržítkem. Někdo se kloní k oddělování slov zvětšením písmene stejně jako v PHP kódu, s tím ale mohou být za určitých okolností problémy (např. MySQL v závislosti na konfiguraci může názvy tabulek převádět na malá písmena).

Pokud se u identifikátorů rozhodneme používat češtinu, je zvykem ji psát bez diakritiky, i když by si většina databázových serverů poradila i s ní. V komentářích tabulek a sloupců je naopak diakritiku vhodné používat.

14 Jak vypadá Spaghetti code



pokročilý

O špagetovém kódu mluvíme tehdy, když jsou jednotlivé části aplikace promíchané tak, že se v nich dá jen těžko vyznat. U webových aplikací je obzvláště jednoduché takový kód vytvořit, protože se v nich používá řada jazyků zapisovaných v prostém textu:

- **JavaScript** je vhodné vyčlenit do samostatného souboru, který z HTML odkážeme značkou `<script src="">`. Přímou v HTML kódu je vhodné uvádět pouze inicializaci proměnných JavaScriptu, které jsou specifické pro daného uživatele.
- **CSS** je také vhodné dát do samostatného souboru a odkázat ho značkou `<link rel="stylesheet" href="">`. Provázání s HTML kódem zajišťují názvy značek, tříd a identifikátorů.
- **HTML** je vhodné nemíchat s řídicím PHP kódem, ale vyčlenit ho do samostatné šablony. Ta může využívat nějaký šablonovací systém nebo může být zapsaná také v PHP, vždy ale platí, že už slouží jen k vypsání dat.
- **SQL** je užitečné centralizovat do tzv. *modelu* (což je část aplikace zajišťující především získávání a ukládání dat), případně se bez něj díky ORM můžeme dokonce obejít.

Výhodou rozdělení aplikace do jednotlivých částí je jejich menší složitost a snazší udržitelnost. Umožňuje to také větší specializaci vývojářů, kdy nehrozí, že HTML kodér poškodí práci PHP programátora.



Poznámka: Některé ukázky kódu v knize se této zásady nebudou kvůli jednoduchosti držet.

15 Jak na odsazování kódu



Bílé znaky jako mezery a konce řádků se sice v PHP až na výjimky ignorují, i tak je ale vhodné kvůli čitelnosti dodržovat pevná pravidla jejich zápisu. Každý příkaz se obvykle píše na jeden řádek – dlouhé příkazy lze rozdělit, ale nikdy bychom neměli na jeden řádek dávat příkazů více. Bloky kódu je zvykem odsazovat, otevírací závorka se umísťuje obvykle na konec předchozího řádku, u funkcí a tříd ji lze dát i na samostatný řádek.



Poznámka: V této knize se budeme zásad správného formátování kódu držet, otevírací složená závorka bude vždy na konci předchozího řádku.

16 Znak pro odsazování kódu



O znacích používaných pro odsazování kódu se vedlo mnoho bezvýsledných debat. Někdo používá dvě mezery, někdo čtyři, někdo trvá na osmi mezerách. Osobně používám tabulátor z těchto důvodů:

- Každý programátorský editor dovoluje nastavit jeho zobrazovanou šířku. Ostatní editory ho obvykle zobrazují jako osm mezer.
- Odsazení je otázkou jednoho stisknutí klávesy v libovolných podmínkách.
- V uloženém souboru zabírá tabulátor jen jeden bajt.

17 Odsazování konstrukce switch



Formátování konstrukce `switch` je ošemetné, protože vytváří dvě úrovně. Někdo doporučuje zarovnávat `case` na stejnou úroveň jako `switch`, někdo o úroveň dál. Pokud je ve větších `case` jen jeden příkaz, tak se mi zdá nejpřehlednější ho uvést rovnou na řádek `case`:

```
<?php
function numbers($number) {
    switch ($number) {
        case 0: return "nula";
        case 1: return "jedna";
        case 2: return "dva";
        default: return null;
    }
}
?>
```

18 Odsazování kódu v kombinaci s HTML



Pokud kombinujeme PHP kód s HTML, tak může být správné odsazování oříškem. Můžeme odsazovat buď pouze PHP kód, nebo pouze HTML, nejpřehlednější je ale asi odsazování obojího.

```
<!-- Odsazování pouze PHP -->
<?php foreach ($data as $row) { ?>
    <tr>
        <td><?php echo htmlspecialchars($row["name"]); ?></td>
```



```

        </tr>
<?php } ?>

<!-- Odsazování pouze HTML -->
<?php foreach ($data as $row) { ?>
<tr>
    <td><?php echo htmlspecialchars($row["name"]); ?></td>
</tr>
<?php } ?>

<!-- Kombinace odsazování -->
<?php foreach ($data as $row) { ?>
    <tr>
        <td><?php echo htmlspecialchars($row["name"]); ?></td>
    </tr>
<?php } ?>

```

Situace se ovšem zkomplikuje, když chceme v bloku PHP kódu vypsat neuzavřenou značku, kterou uzavřeme později. Tam kombinace odsazování příliš dobře nefunguje.

19 Zarovnávání operátorů



začátečník

S formátováním kódu není vhodné to přehánět. Někteří programátoři se vyžívají třeba v tom, že při více přiřazeních pod sebou zarovnávají operátor přiřazení:

```

<?php
$int  = 0;
$s    = "";
$array = array();
?>

```

S takovýmto odsazováním je spousta práce, která čitelnost žádným zásadním způsobem nezlepší. Problém nastane hlavně v případě, kdy chceme přiřadit další proměnnou s delším názvem – pak musíme zarovnání všude opravit.

Zcela tragicky takovéto zarovnání dopadne v případě, kdy se na zdrojový kód podíváme s proporcionálním písmem, které někteří programátoři používají (protože je čitelnější a vejde se ho na obrazovku víc).

```

<?php
$int  = 0;
$s    = "";
$array = array();
?>

```

Obrázek 1. Zobrazení kódu proporcionálním písmem v editoru SciTE

20 Odřádkování před operátorem



začátečník

Pokud by jeden příkaz vytvořil příliš dlouhý řádek, lze ho rozdělit na více řádků. Odřádkování je vhodné udělat před operátorem a pokračování příkazu odsadit:

```

<?php
$ma i l

```

```
->setFrom("alice@example.com")
->addTo($email)
->setSubject("Pokusná zpráva")
;
?>
```

21 Jak na konce řádků



Na Windows se řádky ukončují znaky CR+LF (v PHP řetězci se zapisuje jako `\r\n`), na Linuxu pouze znakem LF (`\n`). PHP koncům řádek nepřikládá zvláštní význam, takže je mu jedno, co použijeme. Programátorské editory si poradí s oběma druhy konců řádků, primitivní textové editory (např. **Notepad** na Windows) vyžadují nativní konce řádků.

Kvůli menší velikosti se obvykle používá znak LF, každopádně je vhodné v celém souboru používat stejný znak.

22 Konec řádku na konci souboru



Poslední řádek souboru je vhodné ukončit koncem řádku (klávesa `Enter`). Usnadňuje to přidávání dalších příkazů a zpřehledňuje to rozdíly mezi verzemi, které něco doplňují na konec souboru.

23 Ukončení posledního prvku pole



Za posledním prvkem v definici pole je možné v PHP uvést čárku stejně jako za ostatními prvky. Velmi užitečné je to hlavně u definice složitějších polí, kde jednotlivé prvky napíšeme na samostatné řádky. Zjednodušuje se tím budoucí přidávání nebo prohazování prvků.

```
<?php
$array = array(
    "první",
    "druhý",
    "třetí",
);
?>
```

24 Pořadí parametrů funkcí



Při vytváření funkce s více než jedním parametrem bychom se měli zamyslet nad tím, v jakém pořadí budeme parametry předávat. U funkcí, které s něčím manipulují, je zvykem v prvním parametru předávat právě tuto hodnotu. Pokud funkce manipulují s objektem, tak je obvykle lepší funkci změnit na metodu tohoto objektu. Další parametry se obvykle uvádí od těch přesnějších po ty vágnější (nejdříve např. předáme ID a až potom název). Volitelné parametry uvádíme pochopitelně až na konci, jinak to v PHP ani nejde.

Funkce by každopádně neměly mít parametrů příliš mnoho, ukázkou z tohoto pohledu nevhodně navržené funkce je `setCookie`, která má 7 parametrů.

25 Pojmenování parametrů funkcí



PHP na rozdíl od některých jiných programovacích jazyků nedovoluje při volání funkce pojmenovat předávané parametry. Všechny parametry je potřeba předat v pořadí, které funkce definuje. Pokud má funkce více podobných parametrů za sebou, je vhodné si hodnotu parametru zapsat do proměnné s popisným názvem.

```
<?php
// nepřehledné
$sqlite->singleQuery($query, true, false);

// přehlednější
$firstRowOnly = true;
$decodeBinary = false;
$sqlite->singleQuery($query, $firstRowOnly, $decodeBinary);

// třetí možnost
$sqlite->singleQuery($query,
    $firstRowOnly = true,
    $decodeBinary = false
);

// využití komentáře
$sqlite->singleQuery($query,
    true, // firstRowOnly
    false // decodeBinary
);

// dá se využít i toho, že se řetězec převede na pravdivostní hodnotu
$sqlite->singleQuery($query, "firstRowOnly", false);
?>
```

Dobrý editor zdrojového kódu nám samozřejmě význam parametru odhalí, ale při prohlížení kódu nemusí být editor k dispozici (ukázky kódu v článcích, prohlížení kódu z repozitáře, atd.).

26 Závorky kolem ternárního operátoru



Kolem ternárního operátoru (neboli podmíněného výrazu) je vhodné dělat závorky, což zlepší jeho čitelnost:

```
<?php
$a = ($m ? f() : ($n ? g() : $o));
?>
```

Ještě přehlednější je v takovýchto případech použít řídicí konstrukci `if`.

27 Použití apostrofy nebo uvozovky?



PHP dovoluje pro zápis řetězců používat uvozovky nebo apostrofy. Uvnitř uvozovek lze používat proměnné a speciální znaky, uvnitř apostrofů ne. Při výběru zápisu bychom

se měli řídit hlavně přehledností. Pokud vypisovaný kód obsahuje uvozovky, tak bývá přehlednější ho uzavřít do apostrofů, jinak budeme v této knize používat uvozovky.

```
<?php
echo '<option value="">(vyberte)</option>';
echo "<div>$html</div>\n";
?>
```

28 Řídicí konstrukce s bloky kódu



Pokud za řídicími konstrukcemi jako `if`, `while` nebo `foreach` uvedeme jenom jeden příkaz, nemusíme ho uzavírat do bloku kódu (složených závorek). I tak je vhodné to ale udělat, protože se tím zlepší přehlednost kódu a usnadní se jeho další modifikace (přidání příkazu).

```
<?php
// přehledný a snadno rozšiřitelný kód
if ($valid) {
    f();
}

// funkce se zavolá vždy, protože se vykoná pouze prázdný příkaz (;)
if ($valid); // chyba
    f();
?>
```

29 Volání funkce před její definicí



PHP dovoluje použít funkci v kódu dříve, než je definovaná. Definice funkce totiž vzniká ve fázi kompilace skriptu, ale zavolá se až ve fázi spuštění. Lepší je ale obvykle funkci nejprve definovat a pak teprve zavolat. Pokud bychom se totiž v budoucnu rozhodli funkci přesunout do vloženého souboru, tak přestane fungovat (vkládané soubory se totiž kompilují až při spuštění hlavního skriptu).

```
<?php
// bude fungovat
function defineFirst() {
}
defineFirst();

// bude fungovat
runFirst();
function runFirst() {
}

// bude fungovat
include "defineFirst.inc.php";
defineFirst();

// nebude fungovat
runFirst();
include "runFirst.inc.php";
?>
```

Do souboru s definicí funkcí je lepší nedávat žádný globální kód. Když toto pravidlo, např. u jednoduchého skriptu spouštěného z příkazové řádky, porušíme a v jednom souboru nadefinujeme funkce i globální kód, tak je zvykem nejprve uvádět definice konstant a funkcí a pak teprve spouštěný kód.

30 V jakém pořadí definovat metody



Na pořadí definic metod stejně jako u funkcí nezáleží, kvůli čitelnosti je ale vhodné dodržovat určitá pravidla. Pořadí definic ve třídě může být takovéto:

- Nejprve vlastnosti v pořadí `public`, `protected`, `private`.
- Konstruktor, destruktorka a další magické metody.
- Další metody seřazené opět podle viditelnosti a podle logických celků.

31 Funkce vypisující data



Ve většině programovacích jazyků je lepší se vyhnout vytváření funkcí, které přímo vypisují nějaká data. Co kdybychom pak výstup chtěli použít někde jinde, třeba ho umístit do e-mailu? Místo toho proto funkce obvykle vrací řetězec, který vypíše až volající.

V PHP se velmi pohodlně kombinuje textový výstup s programovým kódem a jakýkoliv výstup lze snadno zachytit. Proto lze ospravedlnit i vytváření funkcí, které něco přímo vypisují. Pokud bychom náhodou s výstupem chtěli udělat něco jiného, můžeme použít *output buffering*:

```
<?php
ob_start();
phpInfo();
$phpInfo = ob_get_clean();
?>
```

32 Postradatelné proměnné a funkce



Pokud výsledek nějakého výpočtu používáme na více místech, je vhodné si ho přiřadit do proměnné. Pokud stejnou posloupnost příkazů používáme víckrát, je vhodné je uzavřít do funkce a volat tuto funkci. Když více funkcí pracuje se stejným objektem, je vhodné je uzavřít do třídy. Tyto zásady vedou k vytváření přehledného, výkonného a snadno udržitelného kódu.

Co když ale výsledek výpočtu používáme jen na jednom místě, co když funkci voláme jen jednou? Vyplatí se proměnnou nebo funkci vytvářet v takovém případě? Z pohledu běhu programu to je zbytečné, výkonnost to zcela nepatrně zhorší, ale program to obvykle zpřehlední – kus kódu tak vlastně dostane své jméno. Porovnejme následující tři ukázky kódu:

```
<?php
// nejefektivnější, ale nepřehledný kód
echo simplexml_load_file(mysql_result(mysql_query("
```

```
SELECT filename
FROM xml
WHERE id = $id
"), 0))->title;

// pojmenování jednotlivých výpočtů
$result = mysql_query("SELECT filename FROM xml WHERE id = $id");
$filename = mysql_result($result, 0);
$xml = simplexml_load_file($filename);
echo $xml->title;

// pojmenování celé operace
function getTitleFromXml($id) {
    $result = mysql_query("SELECT filename FROM xml WHERE id = $id");
    $filename = mysql_result($result, 0);
    $xml = simplexml_load_file($filename);
    return $xml->title;
}
echo getTitleFromXml($id);
?>
```

Postradatelné proměnné a funkce jsou vlastně způsob dokumentování kódu. Pokud si tedy nějaký blok kódu zaslouží komentář, může ho zpřehlednit také vyčlenění do funkce nebo pojmenování vypočtených hodnot.

Běhové prostředí

33 Použití PHP pro webové aplikace



pokročilý

Nejběžnější použití PHP je pro tvorbu webových aplikací. K tomu je potřeba provozovat webový server (např. **Apache** nebo **IIS**), se kterým lze PHP propojit třemi základními způsoby:

- *CGI* je nezákladnější způsob, kdy se pro zpracování každého skriptu spouští vlastní interpret PHP. V praxi se tento způsob v podstatě nepoužívá, protože je velmi pomalý.
- *Modul webového serveru* je nejběžnější způsob propojení PHP s webovým serverem Apache. Díky těsné vazbě na webový server jde o velmi výkonné řešení.
- *FastCGI* nechává interpret PHP spuštěný na pozadí a předává mu skripty, které je potřeba zpracovat. Díky tomu je toto řešení srovnatelně rychlé jako modul webového serveru, díky volnější vazbě s webovým serverem ale navíc umožňuje skripty spouštět pod jejich vlastníkem (v modulu všechny skripty běží pod společným uživatelem). To je užitečné hlavně na sdíleném webhostingu.

34 Použití PHP z příkazové řádky



začátečník

PHP lze spouštět i z příkazové řádky. Pro toto použití se používá termín *CLI (Command Line Interface)*. Hodí se hlavně pro spouštění testů nebo třeba pravidelně spouštěných úloh.

35 Syntaktická kontrola skriptu



začátečník

Syntaktickou správnost skriptu lze ověřit bez jeho spuštění příkazem `php -l`. Tuto kontrolu je vhodné provádět před jakoukoliv publikací skriptu (ať už třeba uložením do verzovacího systému nebo nahráním na server) nejlépe automaticky. Pokud totiž syntaktická kontrola neprojde, tak se skript vůbec nespustí.



Poznámka: V PHP 5 existovala také funkce `php_check_syntax`, v PHP 5.0.5 ale byla odstraněna, protože zpřístupnila funkce a třídy definované v kontrolovaném souboru.

36 Jiná konfigurace při spuštění z příkazové řádky



pokročilý

Při spuštění z příkazové řádky lze nastavit odlišný konfigurační soubor nebo samotnou konfigurační direktivu. Zajišťuje to parametr `-c` resp. `-d`.

```
# celý konfigurační soubor
php -c php.ini file.php
```

```
# jedna nebo více konfiguračních direktiv
php -d safe_mode=0 file.php
```


37 Spuštění PHP kódu bez vytváření souboru



Pro spuštění PHP kódu přímo z příkazové řádky bez vytváření souboru lze použít parametr `-r`. Tomu se předává samotný PHP kód, který se vyhodnocuje už v kontextu PHP značek, ty se tedy neuvádí.

```
php -r 'phpcredits(CREDITS_DOCS);'
```

Velký pozor je potřeba dát na ošetřování znaků, které mají speciální význam jak na příkazové řádce, tak v PHP kódu.

38 Interaktivní vyhodnocování kódu



Parametrem `-a` příkazového řádku lze spustit tzv. interaktivní režim, kdy jsou příkazy vyhodnocovány bezprostředně po jejich zapsání. Tento režim se dá použít především pro různé pokusy, osobně ale preferuji vytvoření souboru s testovaným kódem a jeho běžné spuštění, protože se v něm snáze dělají změny.

39 Vstup a výstup z příkazového řádku



Při spuštění z příkazového řádku můžeme pracovat se standardním vstupem, standardním výstupem a chybovým výstupem pomocí konstant `STDIN`, `STDOUT` a `STDERR`. Tyto prostředky ani není vhodné zavírat.

```
<?php
// načtení celého standardního vstupu
$input = stream_get_contents(STDIN);
?>
```

Pokud bychom s těmito vstupy a výstupy chtěli pracovat i mimo příkazový řádek, může se si otevřít virtuální soubor `php://stdin`, `php://stdout` a `php://stderr`, obvykle to ale není potřeba.

40 Chyba skriptu spuštěného z příkazového řádku



Pokud program z příkazového řádku nemůže pokračovat (třeba proto, že chybí povinný parametr), tak je vhodné skript ukončit nenulovým stavovým kódem. To zajistí příkaz `exit` s číselným parametrem. Ostatní programy příkazového řádku pak podle tohoto stavového kódu mohou poznat, jestli skript uspěl nebo ne.

```
<?php
if ($argc != 2) {
    echo "Použití: php example.php parametr\n";
    exit(1);
}
?>
```

Při spuštění externího programu z PHP skriptu můžeme stavový kód získat např. v druhém parametru funkce `system`.



Poznámka: Příkaz `die` je *aliasem* příkazu `exit`.

41 Použití chybového výstupu



pokročilý

Pokud program příkazového řádku kromě běžného výstupu vypisuje také chybová hlášení, je vhodné je vypisovat na samostatný chybový výstup. Při běžném spuštění se sice oba výstupy (běžný i chybový) zobrazí společně na obrazovce, při přesměrování výstupu je možné je ale rozdělit.

```
<?php
if (!is_readable($filename)) {
    // nevhodné řešení
    echo "Soubor $filename nelze přečíst.\n";

    // lepší řešení
    fwrite(STDERR, "Soubor $filename nelze přečíst.\n");
}
?>
```

42 Načtení vstupu z webové služby



znalec

Pokud skriptu pošleme data z formuláře metodou POST, tak je PHP rozebere do proměnné `$_POST`. Když ale skript dostane data v jiném formátu, než který používají formuláře (např. XML z webové služby nebo požadavku AJAX), tak potřebujeme načíst jejich původní podobu.

K načtení dat odeslaných metodou POST můžeme použít virtuální soubor `php://input`. Za tímto účelem lze použít i proměnnou `$HTTP_RAW_POST_DATA`, její naplňování se ale dá zakázat konfigurační direktivou `always_populate_raw_post_data`.

```
<?php
$xml = simplexml_load_file("php://input");
?>
```



Poznámka: Ani jeden zdroj dat není k dispozici, pokud jsme z formuláře dovolili odeslání souboru (typ `multipart/form-data`).

43 Zápis na výstup z webové aplikace



pokročilý

Pokud chceme nějaká data zapsat na výstup, můžeme k tomu použít kromě prostého vypsání příkazem `echo` i virtuální soubor `php://output`. Hodí se to v situaci, kdy s výstupem potřebujeme pracovat jako se souborem.

```
<?php
// vypsání binárního objektu databáze Oracle
$ociLob->export("php://output");
?>
```

44 Načtení konfigurace PHP



PHP hledá konfigurační soubor na několika místech:

1. V adresáři určeném konfigurační direktivou `PHPIniDir` webového serveru Apache.
2. Podle proměnné prostředí `PHPRC`.
3. Na Windows se cesta hledá také v registrech v klíči `HKLM\SOFTWARE\PHP\IniFilePath` (za větvi PHP může následovat ještě číslo verze ve formátu `x.y.z`, `x.y` nebo `x`).
4. Aktuální pracovní adresář.
5. Adresář webového serveru.
6. Nakonec se soubor hledá v adresáři `C:\Windows` na Windows, jinde podle nastavení kompilační volby `--with-config-file-path`.

V těchto adresářích se hledá nejprve soubor `php-SAPI.ini`, kde SAPI je kód běhového prostředí (např. `apache` nebo `cli`), a potom soubor `php.ini`. Použitý soubor nám sdělí funkce `phpInfo`.

45 Detekce běhového prostředí



Skript můžeme napsat tak, aby se dal spustit z různých běhových prostředí, typicky přes webový server a z příkazové řádky. Pro zjištění běhového prostředí můžeme použít konstantu `PHP_SAPI` (nebo funkci `php_sapi_name`), při spuštění z příkazové řádky má hodnotu `"cli"`.

Podle běhového prostředí můžeme upravit výstup.

```
<?php
if (PHP_SAPI == "cli") {
    echo "$error\n";
} else {
    echo "<p>" . htmlspecialchars($error) . "</p>\n";
}
?>
```



Poznámka: Funkce `phpInfo` je ukázkou vestavěné funkce, která se chová různě v různých prostředích. Při spuštění přes webový server vypíše úhlednou tabulku, při spuštění z příkazové řádky pouze jednoduchý textový výstup.

46 Definice konfiguračního souboru



S PHP se distribuují konfigurační soubory `php.ini-development` a `php.ini-production`, které se dají použít jako základ konfigurace pro vývojové a produkční prostředí. Tyto soubory jsou velmi obsáhlé, obsahují bohaté komentáře a nastavují většinu konfiguračních direktiv včetně těch, které aplikace nastavit nepotřebuje, případně odpovídají výchozí hodnotě. Konfigurační soubor se značně zpřehlední, pokud se v něm nastaví jen direktivy nutné pro provoz aplikací na daném serveru. Volitelně je možné nenastavovat ani direktivy na jejich výchozí hodnotu, ta se ale v budoucích verzích PHP může změnit, takže pokud na nich aplikace závisí, tak je lepší je explicitně uvést.

47 Pravidelné spouštění úloh



Pro pravidelné spouštění úloh se obvykle používá aplikace **cron**. Ta dovoluje v daný čas a ve stanoveném intervalu spouštět určené programy. Používá se např. pro pravidelný import dat nebo naopak pro mazání zastaralých dat.

Spouštěný program by v případě úspěchu neměl nic vypisovat, protože jakýkoliv výstup se považuje za chybu a odešle se e-mailem uživateli.



Poznámka: Na Windows lze k pravidelnému spouštění využít ovládací panel Naplánované úlohy.

48 Pravidelné spouštění úloh z webu



Někdy může být výhodné spouštět pravidelné úlohy nikoliv z příkazového řádku, ale přímo v kontextu webové aplikace. Může to být potřeba kvůli oprávněním nebo kvůli zajištění totožné konfigurace. Tomuto způsobu spouštění se obvykle říká **WebCron** a některé webhostingy ani jiný způsob pravidelného spouštění nenabízejí.

Periodické spouštění			
Doména	Cesta ke skriptu	Interval	Povoleno
example.com	<input type="text"/>	05	<input checked="" type="radio"/> Ano <input type="radio"/> Ne
			<input type="button" value="Vložit"/>

Obrázek 2. Formulář pro pravidelné spouštění úloh aplikace WebControl

49 Aktuální adresář při spuštění skriptu



Při spuštění PHP skriptu z příkazové řádky (obzvlášť ze služby **cron**) je běžné, že aktuální pracovní adresář není nastaven na adresář, ve kterém je umístěn skript. Relativní cesty se vyhodnocují vzhledem k aktuálnímu pracovnímu adresáři, takže když takovéto cesty ve skriptu používáme, nemusí skript při spuštění z příkazového řádku pracovat.

Spolehlivější je tedy definovat všechny cesty jako absolutní. Můžeme k tomu využít cestu k momentálně zpracovávanému skriptu, která je vždy uložena v „konstantě“ `__FILE__`. Adresář skriptu můžeme z této konstanty získat funkcí `dirname`, od PHP 5.3 se dá použít také přímo konstanta `__DIR__`.



Poznámka: Není vhodné pracovní adresář měnit funkcí `chDir`, protože navazující programy mohou počítat s tím, že jsou spuštěny z daného adresáře.

50 Co je to PHP-GTK



Projekt **PHP-GTK** se dá použít pro vývoj desktopových aplikací v PHP s využitím grafického prostředí GTK. To se používá hlavně na Linuxu, je ale k dispozici i pro Windows nebo OS X. Desktopové aplikace mají proti webovým tu výhodu, že nemusí

ze serveru přenášet uživatelské rozhraní, a to má také bohatší výrazové prostředky. Webové aplikace mají zase tu výhodu, že se dají spustit odkudkoliv a díky AJAXu a dalším technologiím nabízí solidně rychlé a bohaté prostředí také. PHP se proto pro vývoj desktopových aplikací v praxi příliš nepoužívá.

```
<?php
$window = new GtkWindow;
$window->set_title("Ahoj světe!");
$window->connect_simple('destroy', array('Gtk', 'main_quit'));

$label = new GtkLabel("Jen jsme chtěli říct\r\n'Ahoj světe!'");
$window->add($label);

$window->show_all();
Gtk::main();
?>
```

51 Jak na PHP pro Javu



Kromě standardní implementace PHP vytvořené v jazyce C existuje i verze pro Javu. Aplikace, která dovoluje PHP skripty spouštět v prostředí Javy, se jmenuje **Quercus** a využívá se především pro spouštění PHP skriptů v prostředí **Google App Engine**, které dovoluje provozovat vlastní aplikace na serverech firmy Google a snadno je škálovat.

52 Jak na PHP pro .NET



PHP skripty lze spouštět také v rámci frameworku **.NET**. Využít se k tomu dá kompilátor **Phalanger**, jehož hlavní vývojáři pochází z České republiky. Toto běhové prostředí se dá použít především pro vytváření desktopových aplikací, které mohou využívat plnou sílu frameworku **.NET**, např. včetně komunikace se zvukovou kartou.

Toto prostředí se dá použít také pro vytváření **Silverlight** aplikací v PHP.

Instalace a konfigurace

53 Jak zjistit verzi PHP



Kód je vhodné psát tak, aby fungoval ve všech podporovaných verzích PHP. Použitou verzi můžeme zjistit z konstanty `PHP_VERSION` (případně z funkce `phpVersion`), od PHP 5.2.7 můžeme použít také konstanty `PHP_MAJOR_VERSION`, `PHP_MINOR_VERSION`, `PHP_RELEASE_VERSION` a `PHP_EXTRA_VERSION`, kde jsou jednotlivé části čísla verze k dispozici samostatně. Od PHP 5.2.7 je k dispozici také konstanta `PHP_VERSION_ID` vyjadřující verzi jako celé číslo (např. 50207).

Verze v PHP mají tvar 5.3.2RC1 a je potřeba je porovnávat funkcí `version_compare`.

54 Aktuální verze PHP



Téměř každá nová verze PHP opravuje řadu bezpečnostních chyb, je proto vhodné používat vždy aktuální verze. Aktuální verze ale není obvykle jen jedna, protože vývojáři udržují více větví PHP po delší dobu. V jeden okamžik tak mohou být aktuální např. verze 5.3.2 i 5.2.13. Linuxové distribuce mohou navíc udržovat bezpečnost i starších verzí, takže ve stejný okamžik může být z pohledu bezpečnosti aktuální i verze 5.2.6-1+lenny8.

55 Nastavení konfigurace PHP



PHP dovoluje konfiguraci nastavit na několika místech. Vedle vlastního konfiguračního souboru `php.ini` ji lze nastavit v konfiguraci webového serveru – tyto dva způsoby vyžadují pro načtení konfigurace restart webového serveru. Webový server může dovést nastavit konfiguraci i pro adresář (typicky v souboru `.htaccess`), tento způsob může být ale zakázaný. A konečně některé konfigurační direktivy lze nastavit i funkcí `ini_set`, ta ale na některých hostinzích může být také zakázaná. Proto existují i funkce, které vybrané konfigurační direktivy nastavují přímo (např. `session_set_cookie_params`).

Možnost nastavení jednotlivých konfiguračních direktiv je popsána v manuálu PHP.

56 Konfigurace pro adresář



Pokud PHP skript běží jako modul webového serveru Apache s povolenou konfigurační direktivou `AllowOverride`, tak je možné v souboru `.htaccess` definovat konfiguraci specifickou pro aktuální adresář a jeho podadresáře. Vedle konfigurace webového serveru lze pomocí direktiv `php_value` a `php_flag` nastavovat i konfiguraci PHP. Nejčastěji se v tomto souboru nastavují konfigurační direktivy, které se projevují ještě před spuštěním skriptu jako `magic_quotes_gpc` nebo `register_globals`, ale lze nastavit i většinu ostatních konfiguračních direktiv (ty, které mají měnitelnost `PHP_INI_PERDIR`).

```
php_flag magic_quotes_gpc Off
php_value error_reporting 6135
```



Poznámka: Ve webovém serveru IIS slouží ke stejnému účelu soubor `Web.config`.

57 Centrální konfigurace pro adresář



pokročilý

Povolení souboru `.htaccess` je velmi praktické, ale vede k jistému zpomalení, protože webový server při každém požadavku prochází všechny adresáře od aktuálního až po kořenový a hledá v něm tento soubor. Odlišná konfigurace pro jednotlivé adresáře serveru se proto dá nastavit i v konfiguračním souboru Apache pomocí direktivy `<Directory>`.

V konfiguračním souboru `php.ini` se dá nastavit odlišná konfigurace pro jednotlivé adresáře v sekci `[PATH]`. Tuto možnost lze použít při spuštění PHP přes `FastCGI` a je k dispozici od PHP 5.3.



Poznámka: Nevýhodou centrální konfigurace může být to, že k jejímu načtení musíme restartovat server.

58 Vynucení konfigurace



znalec

Pokud nechceme, aby uživatelé přepsali hodnotu nějaké konfigurační direktivy, můžeme ji nastavit v konfiguraci webového serveru Apache pomocí direktiv `php_admin_value` a `php_admin_flag`. Takto nastavené hodnoty nelze přepsat v souboru `.htaccess` ani funkcí `ini_set`.

59 Použití konstant v konfiguraci PHP



začátečník

Některé konfigurační direktivy PHP přijímají číselnou hodnotu, která se může vyjádřit jako bitová maska konstant. Typickým zástupcem takové direktivy je `error_reporting`. Konstanty můžeme samozřejmě používat při volání funkce `ini_set`, dají se použít také v souboru `php.ini`, nelze je ale použít v konfiguračních souborech Apache (typicky hlavně `.htaccess`), kde je nutné použít číslo.

60 Použití jednotek v konfiguraci PHP



začátečník

Direktivy ovlivňující množství zabrané paměti nebo místa na disku dovolují za číslem uvést jednotku. Ta může být `K` (1024 bajtů), `M` (1024 KB) nebo `G` (1024 MB). Velikost písmen se nerozlišuje a jednotky lze použít i při nastavení z konfiguračních souborů webového serveru.

Funkce `ini_get` vrací hodnotu tak, jak byla zadaná.

61 Pravdivostní hodnoty konfiguračních direktiv



Pravdivostní konfigurační direktivy mohou být v konfiguračních souborech webového souboru **Apache** nastaveny pomocí `php_flag`. Nastavení pomocí `php_value` bude také fungovat, ale funkce `ini_get` bude vracet řetězec. Pokud tedy ve skriptu chceme spolehlivě zjistit hodnotu pravdivostní konfigurační direktivy, musíme zohlednit všechny hodnoty, které se vyhodnotí jako `true`.

```
<?php
/** Zjištění pravdivostní hodnoty konfigurační direktivy
 * @param string
 * @return bool
 */
function getIniBool($ini) {
    $value = strtolower(ini_get($ini));
    return (intval($value)
        || $value == "on"
        || $value == "true"
        || $value == "yes"
    );
}
?>
```

62 Otevírání vzdálených souborů



Konfigurační direktivou `allow_url_fopen` můžeme v PHP povolit otevírání vzdálených souborů přístupných protokolem HTTP a několika dalšími protokoly.

```
<?php
file_get_contents("http://www.example.com");
?>
```

Pokud je tato direktiva vypnutá, můžeme ke stažení vzdálených souborů použít extenzi **cURL**.



Poznámka: Některé ukázky v této knize předpokládají zapnutí této direktivy.

63 Vkládání vzdálených souborů



Možnost vkládání vzdálených souborů konstrukcemi `include` a `spol.` ovlivňuje od PHP 5.2 direktiva `allow_url_include`, která je ve výchozím nastavení vypnutá (ve starších verzích se k tomu používala direktiva `allow_url_fopen`). Vzhledem k tomu, že prakticky nedává smysl spouštět vzdálené soubory jako PHP kód, není důvod tuto direktivu zapínat.

I tak je ale nezbytné vkládané soubory kontrolovat, protože útočník může pro útok využít soubor na disku serveru.

64 HTML nebo XHTML?



Webové stránky se obvykle posílají v jazyce HTML nebo v jeho alternativním zápisu XHTML. Ten je o něco přísnější – vyžaduje například uzavírání všech značek a obalování všech hodnot atributů do uvozovek nebo apostrofů. Stránky v jazyce XHTML by se ideálně měly posílat s typem `application/xhtml+xml`, k jehož odeslání lze použít HTTP hlavičku `Content-Type`. Tento typ ale dělá potíže starším verzím **Internet Exploreru**, proto se obvykle neposílá. Prohlížeče pak pracují s XHTML dokumentem jako s HTML. Výběr varianty jazyka musíme sdělit i některým funkcím, např. `n12br`.



Poznámka: Tato kniha uvádí ukázky kódu v jazyce XHTML.

65 Výběr kódování



Pro kódování českých znaků se tradičně používají kódování `windows-1250` (používané hlavně na Windows), `iso-8859-2` (používané hlavně v Linuxu) a `utf-8` (způsob kódování znakové sady Unicode). První dvě zmíněná kódování ukládají každý znak do jednoho bajtu a dají se použít pro střeoevropské jazyky. Třetí uvedené kódování je univerzální a dovoluje uložit prakticky všechny znaky, které se na světě používají. Využívá k tomu proměnlivý počet bajtů na znak – písmena bez diakritiky se ukládají do jednoho bajtu, písmena s diakritikou do dvou bajtů a například čínské znaky do tří bajtů. České texty uložené v kódování UTF-8 jsou proto mírně delší než v jednobajtových kódováních, v HTML dokumentu představuje nárůst asi 4 %.

Vzhledem k univerzálnosti je kódování UTF-8 nejlepší současná volba.

66 Nastavení kódování



V PHP se výstupní kódování nastavuje konfigurační direktivou `default_charset` a mělo by být vždy nastaveno, protože jinak kódování určí prohlížeč automatickou detekcí, která nemusí být vždy spolehlivá.

Přímo ze skriptu můžeme kódování určit funkcí `header("Content-Type: text/html; charset=utf-8")`. Obdobným způsobem bychom měli kódování určit i u exportů, např. RSS.

Kromě toho bychom měli kódování specifikovat i v HTML značce `<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />`. To je důležité pro případ, kdy si uživatel uloží kopii stránky do souboru – prohlížeče v tom případě uloží pouze samotný HTML kód bez hlaviček.

67 Předefinování řetězcových funkcí



Extenze **MBstring** dovoluje pomocí konfigurační direktivy `mbstring.func_overload` předefinovat řetězcové funkce tak, aby respektovaly kódování nastavené touto extenzí. Tomuto předefinování je ale obvykle lepší se vyhnout, protože knihovny používané aplikací se mohou spoléhat na standardní chování řetězcových funkcí.

68 Jak na hlášení chyb



Pomocí konfigurační direktivy `error_reporting` nebo stejnojmenné funkce můžeme ovlivnit, o jakých chybách chceme být informováni. Chyby se dají rozdělit do tří základních skupin:

- Fatální chyby – `E_ERROR`, `E_USER_ERROR`
- Varování – `E_WARNING`, `E_USER_WARNING`
- Poznámky – `E_NOTICE`, `E_USER_NOTICE`

Fatální chyby způsobí ukončení běhu programu, varování nás upozorňují na jasné chyby, které ale neukončí skript, a poznámky nás upozorňují na obraty, které mohou být chybou, ale nemusí. Poznámky upozorňují především na práci s neinicializovanými proměnnými. Takové proměnné mají v PHP hodnotu `null`, ale zapomenutí inicializace proměnné může vést k těžko odhalitelné chybě (např. překlep).

Důsledná inicializace proměnných nicméně může kód zpřehlednit. Představme si, že chceme do košíku (reprezentovaného `session` proměnnou) přidat jeden kus nějakého produktu. V PHP k tomu můžeme použít následující kód:

```
<?php
$_SESSION["basket"][$id_produk]++;
?>
```

Kód je přehledný a jednoznačný, pokud ale produkt v košíku ještě nebude, tak PHP vygeneruje poznámku. Aby se tato poznámka nevygenerovala, je potřeba kód přepsat:

```
<?php
if (isset($_SESSION["basket"][$id_produk])) {
    $_SESSION["basket"][$id_produk]++;
} else {
    $_SESSION["basket"][$id_produk] = 1;
}
?>
```

Takto zapsaný kód je mnohem delší a méně přehledný. Příklady v této knize jsou nicméně napsány tak, aby žádné chyby včetně poznámek negenerovaly, mohou se ale spoléhat na existenci proměnných, které používají.



Poznámka: Existují i další chybové úrovně (např. `E_STRICT` nebo `E_RECOVERABLE_ERROR`) a skoro každá větší verze PHP nějakou novou zavádí.

69 Práce s neinicializovanými proměnnými



Upozornění na práci s neinicializovanými proměnnými je hlavním úkolem chybové úrovně `E_NOTICE` (poznámky), PHP to ale bohužel dělá špatně ze dvou důvodů:

1. Na práci s neinicializovanou proměnnou jsme upozorněni až v době spuštění kódu, což vzhledem k bezpečnostním rizikům tohoto přístupu může být pozdě.

2. O některých chybných obrazech se nedozvíme (např. přidání prvku do neinicializovaného pole).

Pro kontrolu práce s proměnnými bez obou neduhů lze použít program **php-initialized**, který skript zkontroluje bez jeho spuštění. Autorem tohoto programu je autor této knihy.

70 Proměnná s chybovou hláškou



Konfigurační direktivou `track_errors` můžeme dosáhnout toho, že se poslední vyvolaná chyba uloží do proměnné `$php_errormsg`. Tuto proměnnou můžeme využít např. pro zaznamenání chyby.

Od PHP 5.2 lze použít také funkci `error_get_last`, která o chybě vrací i další informace (typ chyby a soubor a řádek, kde k chybě došlo).

71 Využití obsluhy chyby pro získání chybové hlášky



Pokud se nemůžeme spolehnout na zapnutí konfigurační direktivy `track_errors` a nechceme ovlivňovat okolní prostředí jejím povolením (třeba v kódu knihovny), tak můžeme pro zachycení chybové hlášky použít vlastní obsluhu chyb:

```
<?php
class Error {
    private $error = "";

    function _errorHandler($level, $error) {
        $this->error = $error;
    }

    function trigger() {
        set_error_handler(array($this, '_errorHandler'));
        // zde bude zavolání kódu, který může obsahovat chybu
        pg_connect("");
        restore_error_handler();
        echo "$this->error\n";
    }
}

$error = new Error;
$error->trigger();
?>
```

72 HTML kód v chybách



Konfigurační direktiva `html_errors` způsobuje, že se chybové zprávy vytváří ošetřené pro HTML. To je velmi praktické při jejich vypisování, ale nepraktické při jakékoliv jiné manipulaci s nimi. Pokud nechceme ovlivňovat okolní prostředí vypnutím této direktivy, můžeme HTML ošetřování zneutralizovat:

```
<?php
$error = $php_errormsg;
if (ini_get("html_errors")) {
    $error = html_entity_decode(strip_tags($error));
}
// pokud nás navíc nezajímá ani funkce, ve které došlo k chybě
$error = preg_replace('~^[^:]*: ~', '', $error);
?>
```

73 Chyby v SQL dotazech



PHP ve výchozím nastavení nezobrazuje chyby v SQL dotazech. Funkce jako `mysql_query` a podobné v případě chyby pouze vrátí `false`. Konfigurační direktiva `mysql.trace_mode` se dá použít k automatickému vytváření PHP varování z SQL chyb.

Tato konfigurační direktiva ale zároveň vyvolá varování při kompletním procházení tabulek (což může být důležité kvůli výkonnosti) a pro výsledky neuvolněné funkce `mysql_free_result` (to je důležité kvůli šetření paměti). První uvedená vlastnost je navíc realizována voláním příkazu `EXPLAIN` pro každý `SELECT`, což znemožňuje používat obraty jako `SELECT FOUND_ROWS()`.

V praxi se tato konfigurační direktiva téměř nepoužívá, a pokud chceme chyby v SQL příkazech hlásit, je vhodné si za tím účelem napsat vlastní funkci:

```
<?php
/** Položení MySQL dotazu s nahlášením chyby
 * @param string
 * @param resource mysql link
 * @return resource mysql result
 * @throws Exception v případě chyby v dotazu
 */
function mysqlQueryError($sql, $connection = null) {
    $args = func_get_args();
    $return = call_user_func_array("mysql_query", $args);
    if (!$return) {
        array_shift($args); // funkcím předáme pouze $connection
        $message = call_user_func_array("mysql_error", $args);
        $code = call_user_func_array("mysql_errno", $args);
        throw new Exception($message, $code);
    }
    return $return;
}
?>
```



Poznámka: Čistší řešení nabízí extenze MySQLi, kde lze ze základní třídy vytvořit potomka, který může chyby hlásit požadovaným způsobem.

74 Instalace PHP na vlastním počítači



Pokud chceme na vlastním počítači provozovat PHP aplikaci, můžeme si samostatně nainstalovat webový server, PHP a databázový server. Tato volba je obvyklá na Linuxu,

kde jsou aplikace distribuovány formou balíků nebo zdrojových kódů, můžeme ji ale využít i na Windows.

Druhou možností je využít nějakou aplikaci, která všechny produkty nainstaluje společně a navíc přidá služby pro jejich snadnou správu a konfiguraci. Mezi nejlepší takové balíky patří **XAMPP**, který je k dispozici pro Windows, Linux i další platformy. Pro Windows lze použít také **WampServer** nebo **VertrigoServ**.

75 Instalace PHP na IIS



PHP se nejčastěji používá spolu s webovým serverem **Apache**, srovnatelně výkonné je ale zhruba od verze 5.2 také pod serverem **IIS** na Windows. K jeho instalaci pod tímto serverem lze použít binární soubory PHP – od verze 5.3 se používá verze zkompileovaná jako *VC9 Non Thread Safe* (pro Apache se používá verze *VC6 Thread Safe*).

76 Instalace PHP a aplikací na IIS



K instalaci PHP pod webovým serverem **IIS** můžeme využít také **Microsoft Web Platform**. Ta umožňuje instalovat na server **IIS** webové aplikace napsané v různých programovacích jazycích včetně PHP. Kromě instalace aplikací se stará také o instalaci jejich závislostí, např. databázový server, frameworky nebo právě PHP. K instalaci PHP můžeme tedy použít i tuto platformu.

77 Jaká je licence PHP



PHP je k dispozici pod licencí *PHP License*. Ta se vztahuje na redistribuci zdrojových a binárních kódů PHP a na vytváření jeho odvozenin. Pokud bychom tedy chtěli vytvořit např. verzi PHP, která nebude obsahovat datový typ `resource`, tak bychom tuto licenci museli dodržet. Licence je poměrně velkorysá, pouze nedovoluje použít „PHP“ v názvu odvozeného produktu.

Na skripty vytvořené pro PHP se tato licence nevztahuje, a můžeme je tedy šířit s libovolnou licencí. Neplatilo by to pouze v situaci, kdy bychom je šířili jako jeden celek spolu s PHP.

78 Jaká je licence MySQL



MySQL je možné získat buď s licencí GPL nebo s komerční licencí. Licence GPL vyžaduje, aby program distribuovaný spolu s MySQL používal opět tuto licenci. To především znamená, že musíme dát k dispozici jeho zdrojový kód. Komerční licence to nevyžaduje.

Pokud svoji aplikaci pouze provozujeme na webovém serveru, tak k distribuci nedochází, a zdrojový kód tedy k dispozici dávat nemusíme. V tom případě můžeme použít MySQL s GPL licencí nezávisle na tom, zda jde o komerční nebo nekomerční aplikaci.

79 Nahrávání extenzí



Funkčnost PHP lze rozšiřovat pomocí řady existujících extenzí. Ty dovolí z PHP pracovat třeba s obrázky nebo s FTP. Ostatně i běžně používaná práce s databázemi je zajištěna extenzemi. Pro nahrání extenze můžeme použít konfigurační direktivu `extension`, kterou lze zapsat pouze do souboru `php.ini` (a ne např. do konfigurace webového serveru). Pro nahrání extenze se dá použít také funkce `d1`, ta je ale k dispozici jen při spuštění z příkazového řádku (navíc se dá zakázat pomocí `enable_dl`).



Poznámka: Pro nahrání extenzí rozšiřujících jádro PHP (jako jsou např. akcelerátory nebo debuggery) se používá konfigurační direktiva `zend_extension` (ve starších verzích PHP rozdělená podle způsobu kompilace).

80 Název souboru s extenzí



Skript spuštěný z příkazového řádku si může nahrát potřebné extenze funkcí `d1`. Problém je ten, že se soubory s extenzemi na různých operačních systémech jmenují různě. Na Windows začínají `php_` a mají koncovku `.dll`, na Linuxu mají koncovku `.so`. Správná koncovka je uložena v konstantě `PHP_SHLIB_SUFFIX`. Můžeme si proto napsat funkci, která tento rozdíl skryje:

```
<?php
/** Nahrání extenze podle názvu
 * @param string
 * @return int
 */
function loadExtension($name) {
    $filename = "$name." . PHP_SHLIB_SUFFIX;
    if (PHP_SHLIB_SUFFIX == "dll") {
        $filename = "php_$filename";
    }
    return dl($filename);
}
?>
```



Poznámka: Nahrané extenze můžeme otestovat funkcí `extension_loaded`.

81 Vývojové a produkční prostředí



Aplikaci bychom měli vždy vyvíjet na veřejně nedostupném místě (obvykle na vlastním počítači) a teprve po jejím úplném odladění bychom ji měli umístit na cílové místo. Těmto dvěma prostředími se říká vývojové a produkční.

Ve vývojovém prostředí je vhodné být ihned informován o všech chybách, ke kterým v programu dojde. Jejich zobrazení zajistí konfigurační direktiva `display_errors`. V produkčním prostředí je naopak vhodné všechny chyby touto direktivou skrýt. Jejich zobrazení totiž jednak působí velice amatérsky, jednak z nich mohou uniknout citlivé informace (např. přihlašovací údaje k databázovému serveru).

V produkčním prostředí je naopak vhodné zapnout logování chyb direktivou `log_errors`. Umístění logu můžeme ovlivnit direktivou `error_log`. V produkčním prostředí by měl log zůstat pokud možno prázdný, což můžeme ověřit třeba tím, že si ho v případě neprázdnosti necháme zasílat.

Ve vývojovém prostředí je možné logování chyb zapnout taky, protože pokud se chyba zobrazí např. uvnitř hodnoty HTML atributu, můžeme ji snadno přehlédnout.

82 Odkazy z chybových hlášek



Chybové hlášky generované PHP obsahují odkazy do dokumentace. URL dokumentace můžeme nastavit direktivou `docref_root`, koncovku pomocí `docref_ext`:

```
docref_root = http://www.php.net/manual/en/
docref_ext = .php
```



Poznámka: Vypnutím volby `html_errors` můžeme vytváření odkazů úplně zrušit.

83 Konfigurace aplikace specifická pro server



Na vývojovém a produkčním serveru můžeme používat odlišnou konfiguraci, což můžeme ve skriptu zohlednit:

```
<?php
if ($_SERVER["SERVER_ADDR"] == "127.0.0.1") {
    // vývojový server
    define("DB_SERVER", "localhost");
    define("DB_USER", "root");
    define("DB_PASSWORD", "");
} else {
    // produkční server
    define("DB_SERVER", "db.example.com");
    define("DB_USER", "web");
    define("DB_PASSWORD", "GwqfUcE0");
}
?>
```

Skripty se velmi zjednoduší, když na vývojovém i produkčním serveru budeme používat stejnou konfiguraci. Nic nám např. nebrání si na vývojovém serveru vytvořit uživatele databáze se stejným jménem a heslem (a se stejnými právy) jako na ostrém serveru. S názvem počítače to je o něco složitější – ten si v souboru `etc/hosts` můžeme namapovat na vývojový databázový server, to má nicméně své nevýhody (např. aplikaci nespustíme bez tohoto nastavení).

84 Povolení ladicího režimu



Podle IP adresy serveru můžeme zjistit, zda aplikaci spouštíme na lokálním počítači nebo na veřejně dostupném serveru. Podle toho můžeme povolit ladicí režim.

```
<?php
define("DEBUG", $_SERVER["SERVER_ADDR"] == "127.0.0.1");
?>
```

Pokud je veřejně dostupný server sdílený, tak by měl poslouchat pouze na veřejné IP adrese, aby si jiný uživatel nemohl vytvořit požadavek na lokální počítač, a tím ladicí režim zapnout. U webového serveru **Apache** k tomu slouží nastavení `Listen`.



Poznámka: Lokálních adres je více, i když 127.0.0.1 je nejběžnější. Knihovna Nette Debug nabízí detekci na základě kompletního seznamu.

85 Nevhodné povolení ladicího režimu



znalec

Při nastavování ladicího režimu by nás mohlo napadnout orientovat se místo IP adresy podle názvu serveru:

```
<?php
// nepoužívat - skrývá nebezpečí
define("DEBUG", $_SERVER["SERVER_NAME"] == "localhost");
?>
```

Tento způsob detekce je ale lepší **nepoužívat**, protože si útočník může snadno určit (obvykle souborem `etc/hosts`), že jeho `localhost` míří na náš server, a tím podstrčit hodnotu proměnné `$_SERVER["SERVER_NAME"]`.

86 Registrace proměnných zvenku



pokročilý

Konfigurační direktiva `variables_order` slouží k určení vnějších zdrojů, pro které PHP vytvoří superglobální proměnné. Jde o řetězec sestavený z těchto znaků:

- E – proměnné prostředí (`$_ENV`).
- G – parametry URL (`$_GET`).
- P – data odeslaná metodou POST (`$_POST`).
- C – cookies (`$_COOKIE`).
- S – proměnné serveru (`$_SERVER`).

Pokud je zapnutá konfigurační direktiva `register_globals`, tak `variables_order` určuje také rozsah a pořadí registrovaných globálních proměnných (pozdější hodnoty přepisují dřívější).



Poznámka: Na registraci globálních proměnných bychom se neměli spoléhat (a `register_globals` je tedy lepší vypnout), takže na pořadí znaků v této direktivě by nám nemělo záležet. Většina webových aplikací se neobejde bez nastavení `variables_order = GPCS`.

87 Pořadí proměnných zvenku



znalec

Do pole `$_REQUEST` se ukládají data odeslaná uživatelem v pořadí určeném direktivou `request_order`, která je k dispozici od PHP 5.3 (ve starších verzích se k tomuto účelu používala direktiva `variables_order`). Hodnota `PG` tedy určí, že pole nebude obsahovat cookies a `GET` proměnné přepíšou `POST` proměnné stejného názvu.



Poznámka: Pole `$_REQUEST` je lepší nepoužívat, protože může obsahovat data z jiného zdroje, než očekáváme. Na obsahu této direktivy by nám tedy nemělo záležet.

88 Automatické ošetřování vstupu od uživatele



začátečník

Pokud data od uživatele posíláme do databáze, musíme je ošetřit tak, aby nemohla ovlivnit samotný dotaz, což by bylo významné bezpečnostní riziko. Protože na to začínající programátoři často zapomínají, byla do PHP zavedena konfigurační direktiva `magic_quotes_gpc`, která se o ošetření postará automaticky. Takto ošetřená data lze ale použít jen v některých případech:

- Data se mohou použít pouze uvnitř SQL řetězce uzavřeného do apostrofů nebo uvozovek.
- Musíme používat jednobajtové kódování nebo UTF-8.
- Ošetření je plně kompatibilní pouze s databázovým serverem MySQL.

Důležité je zmínit, že pokud bychom data chtěli použít v jakémkoliv jiném kontextu (např. je vypsát do stránky), musíme je zase od-ošetřit funkcí `stripSlashes`.

Každá aplikace, která chce s daty od uživatele nějakým způsobem pracovat, musí vědět, jak je tato direktiva nastavena, případně si ji musí sama nastavit. To ale nejde udělat přímo ze skriptu, protože data se ošetřují ještě před jeho spuštěním.

Dá se říci, že tato konfigurační direktiva znehodnotí data od uživatele pro jedno konkrétní použití, a proto byla v PHP 5.3 označena jako zastaralá a z budoucích verzí bude odstraněna. Příklady v této knize počítají s tím, že je vypnuta.

89 Vypnutí automatického ošetřování vstupu od uživatele



pokročilý

Pokud má aplikace pracovat s daty od uživatele a chce být nezávislá na nastavení konfigurační direktivy `magic_quotes_gpc`, může si ošetření sama odstranit. K tomu ale nejde použít běžná funkce `ini_set`, protože při spuštění skriptu jsou data již ošetřena. Je proto nutné to udělat vlastním skriptem:

```
<?php
// odstranění ošetření dat způsobené direktivou magic_quotes_gpc
if (get_magic_quotes_gpc()) {
    $process = array(&$_GET, &$_POST, &$_COOKIE, &$_REQUEST);
    while (list($key, $val) = each($process)) {
        foreach ($val as $k => $v) {
            unset($process[$key][$k]);
            if (is_array($v)) {
```

```

        $process[$key][stripSlashes($k)] = $v;
        $process[] = &$process[$key][stripSlashes($k)];
    } else {
        $process[$key][stripSlashes($k)] = stripSlashes($v);
    }
}
unset($process);
}
?>

```



Poznámka: Automatické ošetřování pole `$_FILES` se v různých verzích PHP chová různě.

90 Automatické ošetření načítaných dat



začátečník

Vedle dat od uživatele je možné automaticky ošetřovat i data načítaná z jiných zdrojů. Slouží k tomu konfigurační direktiva `magic_quotes_runtime`, ale k jejímu použití není rozumné opodstatnění. Pokud aplikace nemůže nastavení této direktivy ovlivnit konfigurační, může ji sama vypnout funkcí `set_magic_quotes_runtime` (ta je ovšem označena jako zastaralá).

91 Automatické zahájení session



pokročilý

Pomocí konfigurační direktivy `session.auto_start` lze zapnout automatické otevření session. Ve skriptech pak není nutné volat funkci `session_start` a session proměnné lze rovnou používat.

Ve většině případů je ale lepší tuto direktivu nezapínat, protože session nemusí být vždy potřeba a někdy je také potřeba ji ve skriptu nakonfigurovat. Otevření session konfigurační direktivou do nich také znemožňuje ukládat vlastní objekty, protože definice tříd se načte až později.

92 Platnost session proměnných



začátečník

Session proměnné mají ve výchozím nastavení velmi omezenou platnost ovlivněnou dvěma faktory.

1. Jednak to je platnost session cookie nastavená direktivou `session.cookie_lifetime`, která má ve výchozím nastavení hodnotu 0, což znamená „do zavření prohlížeče“. To je obvykle vyhovující.
2. Druhá direktiva ovlivňující platnost session proměnných je `session.gc_maxlifetime` určující, jak stará session data se už mohou smazat. Tato direktiva má ve výchozím nastavení hodnotu 1440 sekund, což je jen 24 minut. Důležité je, že tuto direktivu nemůžeme samotnou nastavit jen pro část aplikace (např. administrační rozhraní), protože stará session data může smazat kterýkoliv skript. Tuto direktivu tedy musíme nastavit vždy stejně všem částem aplikace, které ukládají data do stejného adresáře (`session.save_path`).



Poznámka: K automatickému mazání starých session dat dochází jen při výchozím způsobu ukládání session proměnných do souboru. Pokud tento způsob změním, tak si můžeme způsob mazání starých dat zajistit sami.

93 Mazání starých session dat



pokročilý

Při každém zavolání funkce `session_start` se PHP s pravděpodobností `session.gc_probability / session.gc_divisor` pokusí smazat stará session data. Za stará se považují ta, která nebyla použita po dobu `session.gc_maxlifetime` sekund.

K mazání tedy může dojít při spuštění libovolného skriptu, který používá session proměnné. Pokud je session dat hodně, tak ho tato operace může výrazně zdržet. Pro zlepšení odezvy serveru je tedy možné nastavit direktivu `session.gc_probability` na nulu a mazání starých session dat provádět nezávislou úlohou (nejspíš spouštěnou službou `cron`).

94 Přidání session identifikátoru do URL



začátečník

Pomocí direktivy `session.use_trans_sid` lze dosáhnout automatického přidávání session identifikátoru u všech interních odkazů na stránce. Ty se do URL přidají v situaci, kdy PHP neobdrží session identifikátor z cookie. K tomu dojde i při prvním zobrazení stránky uživatelem (protože tehdy se session cookie teprve odešle, ale ještě není přijata).

Vzhledem k tomu, že se do URL přidává z pohledu uživatele nesmyslný identifikátor, tak je lepší tuto direktivu vypnout, pokud můžeme vyžadovat zapnuté cookies. Pokud skriptem odesíláme něco jiného než HTML stránku zobrazovanou v prohlížeči (např. export dat nebo obrázek), je nutné tuto direktivu vypnout, protože může vést k poškození dat.

Předávání session identifikátoru v URL může také znamenat bezpečnostní riziko.



Poznámka: Na adresy posílané funkcí `header` (nejčastěji `Location`) se tato direktiva nevztahuje a session identifikátor je do nich nutné přidat ručně.

95 Odmítnutí session identifikátoru z URL



pokročilý

Pokud po uživateli můžeme požadovat zapnuté cookies, je vhodné zapnout konfigurační direktivu `session.use_only_cookies`, která způsobí ignorování session identifikátoru z URL. Její zapnutí ztíží útok *Session Fixation*, kdy útočník vnucuje uživateli session identifikátor nejčastěji právě pomocí URL.

96 Konstanta se session identifikátorem



znalec

Při zahájení session (obvykle zavoláním funkce `session_start`) vytvoří PHP konstantu `SID`. Pokud od uživatele přišla cookie se session identifikátorem, tak je tato konstanta nastavena na prázdný řetězec, jinak obsahuje název a hodnotu session identifikátoru.

Konstantu tedy můžeme použít např. při přesměrování, kde se session identifikátor automaticky nedoplňuje.

```
<?php
session_start();
$sid = (SID ? "?" . SID : "");
header("Location: http://$_SERVER[SERVER_NAME]/$sid");
?>
```



Poznámka: Bez použití této konstanty se můžeme obejít, pokud po uživatelích vyžadujeme zapnuté cookies.

97 Vytváření session identifikátoru



znalec

PHP vytváří session identifikátory z pseudonáhodné posloupnosti. Aby se vývoj této posloupnosti nedal snadno odhadnout, je vhodné nastavit konfigurační direktivu `session.entropy_file`, která do posloupnosti přidává zdroj náhody z určeného souboru. Na Linuxu lze použít `/dev/random` nebo `/dev/urandom`. Spolu s touto direktivou je potřeba také nastavit `session.entropy_length`.



Poznámka: Vhodné je také nezveřejňovat výstup funkce `uniqid` s druhým parametrem, která používá stejnou pseudonáhodnou posloupnost.

98 Oddělovač parametrů URL



pokročilý

Konfigurační direktivou `arg_separator.input` lze nastavit oddělovače parametrů v adrese. Některé servery používají jako oddělovač středník, protože výchozí znak `&` má zvláštní význam i v HTML, takže se např. odkaz s více parametry musí zapisovat jako ``. Nicméně vzhledem k tomu, že při odesílání formulářů prohlížeče vždy generují `&`, je toto nastavení spíše na škodu, protože vznikají duplicity.

Direktiva `arg_separator.output` se používá při generování odkazů, např. při zapnuté volbě `session.use_trans_sid` nebo ve funkci `http_build_query`. Výchozí hodnota této konfigurační direktivy je `&`, což je problematické kvůli tomu, že se používá při výpisu do HTML kódu. Můžeme ji proto změnit na `&`, pak ale výstup funkce `http_build_query` bez třetího parametru (který dovoluje oddělovač explicitně určit) můžeme použít jen v HTML kódu bez dalšího ošetření.



Poznámka: Jako oddělovač v direktivě `arg_separator.input` lze nastavit i více znaků, to ale není moc vhodné, protože pak mají dvě různé adresy (např. `?a=1&b=5` a `?a=1;b=5`) stejný význam.

99 Nastavení limitů PHP



pokročilý

PHP obsahuje v rámci ochrany před zahlcením serveru několik paměťových a časových limitů. Časové limity jsou dva:

- `max_execution_time` – maximální čistá doba běhu skriptu je ve výchozím nastavení 30 s, u dlouhotrvajících skriptů je vhodné ji zvýšit nebo úplně vypnout. Lze změnit také funkcí `set_time_limit`.
- `max_input_time` – maximální čas pro zpracování vstupních dat, ve výchozím nastavení je vypnut a obvykle ho není potřeba zapínat.

Paměťové limity jsou tři:

- `memory_limit` – celkové množství paměti, kterou může skript zabrat. Od PHP 5.2.1 je tato direktiva nastavena na přijatelných 128 MB.
- `post_max_size` – maximální velikost dat přenášených metodou POST. Výchozí hodnotu 8 MB může být potřeba zvětšit pro nahrání většího množství souborů najednou.
- `upload_max_filesize` – maximální velikost nahrávaného souboru. Výchozí hodnota 2 MB může být pro zpracování větších souborů nedostatečná.

100 Nastavení omezení paměti



znalec

Někdy dopředu nevíme, kolik paměti bude skript potřebovat, můžeme to ale zjistit při běhu skriptu. Tehdy se dá využít toho, že direktiva `memory_limit` se dá nastavit i při běhu skriptu:

```
<?php
// momentální zabraná paměť + požadovaná velikost + rezerva
ini_set("memory_limit", memory_get_usage() + $size + 8000000);
?>
```

101 Co je to bezpečný režim



začátečník

Direktiva `safe_mode` je určena pro zlepšení zabezpečení serveru, kde provozuje skripty více uživatelů. Direktiva má tyto základní funkce:

1. Souborovým funkcím dovolí pracovat jen s těmi soubory, které mají stejného vlastníka jako spuštěný skript nebo leží v adresáři s tímto vlastníkem.
2. Pomocí direktivy `safe_mode_exec_dir` dovoluje omezit programy spouštěné z příkazové řádky.
3. Nedovoluje změnit maximální dobu běhu skriptu a množství maximálně zabrané paměti.
4. Do parametru `realm` HTTP autentizace přidává ID vlastníka skriptu.

První bod je nejvýznamnější a také nejproblematictější. Pokud si totiž soubory vytvářené skriptem ukládáme do adresářů vytvořených skriptem (např. strukturovaná cache), tak k nim kvůli `safe_mode` nebudeme mít přístup. Naopak `safe_mode` dovoluje skriptům vytvořeným pomocí PHP pracovat se soubory vytvořenými skripty jiných uživatelů.



Poznámka: Direktiva `safe_mode` byla v PHP 5.3 označena jako zastaralá a z budoucích verzí bude odstraněna.

102 Omezení adresáře skriptů



Direktivou `open_basedir` od sebe můžeme oddělit uživatele sdíleného webhostingu. Tato direktiva nám dovoluje omezit adresář, do kterého bude mít spuštěný skript přístup. Pokud tuto direktivu nastavíme odlišně např. pro každou doménu, tak skripty běžící na této doméně nebudou moci pracovat s cizími soubory.



Poznámka: Adresář může být nastaven i relativně (např. jako `.`), to je ale nevhodné, protože funkcí `chDir` se dá pracovní adresář snadno změnit.

103 Umístění dočasných souborů



Na sdíleném hostingu bychom měli každému uživateli nastavit vlastní adresář pro nahrávané soubory a ukládání `session` proměnných. Řídí to direktivy `upload_tmp_dir` a `session.save_path` a ideálně by měly mířit do adresáře nedostupného z webu, ale v rámci `open_basedir`.

104 Jak skrýt přítomnost PHP



Vypnutím direktivy `expose_php` můžeme skrýt informaci o tom, že na serveru běží PHP. Tato informace spolu s verzí PHP se jinak předává v hlavičce `Server` nebo `X-Powered-By`. Informaci o verzi PHP mohou využít útočníci pro zjištění bezpečnostních slabín serveru, pravda je ale taková, že útok proti starším verzím PHP můžou vyzkoušet i bez znalosti verze. Lepší je proto používat aktuální verzi PHP a informaci o přítomnosti PHP neskrývat.



Poznámka: Vypnutí této direktivy odstraní také obrázek z výsledku funkce `phpInfo`, protože podle něj by šla přítomnost PHP také zjistit.

105 Vložení souboru do všech skriptů



Pomocí direktiv `auto_prepend_file` a `auto_append_file` můžeme automaticky vložit soubor na začátek resp. konec každého skriptu. Můžeme to použít v situaci, kdy všechny skripty využívají společnou knihovnu, kterou pak nemusíme ručně vkládat do všech skriptů.

Obvykle se nicméně tyto konfigurační direktivy nepoužívají, protože je lepší navrhovat aplikace tak, aby běžely pokud možno nezávisle na konfiguraci. Použít je nicméně můžeme tehdy, když chceme aplikaci třetí strany předřadit vlastní kód (např. ověření oprávnění ke spuštění aplikace). Díky těmto direktivám totiž nemusíme měnit zdrojový kód aplikace, což usnadňuje její aktualizaci.

106 Konfigurace e-mailu



Pro odesílání e-mailů pod Windows používá PHP přímo protokol SMTP. Adresu serveru pro odesílání zpráv je proto potřeba nastavit konfigurační direktivou `SMTP` (na rozdíl od ostatních konfiguračních direktiv se píše velkými písmeny).

V ostatních operačních systémech se používá pro odesílání zpráv program příkazového řádku, který je možno nastavit konfigurační direktivou `sendmail_path`.

107 Jak nastavit časové pásmo



Při použití funkcí `date`, `strtotime` a podobných se zohledňuje časové pásmo. To dokáže PHP detekovat z proměnné prostředí `TZ`, tento způsob je ale nespolehlivý. Od PHP 5.1 lze proto časové pásmo nastavit funkcí `date_default_timezone_set` nebo konfigurační direktivou `date.timezone` a bez jejího nastavení vyvolá PHP při použití datové funkce varování. V České republice lze tuto direktivu nastavit na hodnotu `Europe/Prague`.



Poznámka: Funkce `gmdate` pracuje vždy s centrálním časem, a nastavení časového pásma tedy nepotřebuje.

108 Konfigurace MySQL



Pokud se ze skriptů připojujeme k databázi MySQL, můžeme přihlašovací údaje definovat v konfiguračním souboru. Slouží k tomu direktivy `mysql.default_host`, `mysql.default_user` a `mysql.default_password` při použití extenze **MySQL**, resp. `mysqli.default_host`, `mysqli.default_user` a `mysqli.default_pw` při použití **MySQLi**. Takovéto direktivy nabízí i některé další databázové extenze.

Výhoda spočívá v tom, že ve skriptech potom přihlašovací údaje už nemusíme uvádět:

```
<?php
mysql_connect();
mysql_select_db("test");
?>
```



Poznámka: Takto zadané přihlašovací údaje se zobrazují např. ve výstupu funkce `phpInfo`.

109 Co je to MySQLnd



PHP s databázovým serverem MySQL komunikuje tradičně pomocí knihovny **libmysql**, ale od verze PHP 5.3 dovoluje využít také knihovnu **MySQLnd** (*native driver*). Knihovna **libmysql** je obecná a kromě PHP ji využívá také řada dalších projektů, což je její hlavní nevýhoda. Data si tato knihovna totiž ukládá do vlastních struktur, z kterých je PHP musí převádět, což způsobuje dvojnásobné paměťové nároky a stojí čas vynaložený na konverzi.

Knihovna **MySQLnd** je napsaná na míru PHP a data ukládá přímo v interním formátu, což má pozitivní vliv na výkon. Kromě toho také nabízí funkce pro získání statistik jako např. `mysqli_get_client_stats`.

MySQLnd se zapíná při kompilaci PHP, od PHP 5.3 je ve výchozím nastavení zapnuté.

110 Zastaralé direktivy



pokročilý

Některé konfigurační direktivy PHP byly označené jako zastaralé, nebo už byly z PHP dokonce odstraněny. Tyto konfigurační direktivy je tedy lepší vůbec nepoužívat. Jedná se zejména o následující direktivy:

- `zend.ze1_compatibility_mode`
- `magic_quotes_gpc`
- `magic_quotes_runtime`
- `magic_quotes_sybase`
- `register_globals`
- `register_long_arrays`
- `safe_mode`
- `allow_call_time_pass_reference`
- `enable_dl`

111 Jaký použít oddělovač adresářů



začátečník

PHP dovoluje jako oddělovač adresářů a souborů použít na všech platformách lomítko i zpětné lomítko. Obvykle se používá běžné lomítko. Existuje také konstanta `DIRECTORY_SEPARATOR` obsahující nativní oddělovač, tu bychom mohli použít např. při volání externích programů (např. u příkazu `include` je ale použití této konstanty zbytečné).

112 Oddělovač cest



začátečník

Funkce `set_include_path` dovoluje nastavit více cest, ve kterých se budou hledat vkládané soubory. Cesty se na Linuxu oddělují dvojtečkou, na Windows středníkem. Správný oddělovač lze získat z konstanty `PATH_SEPARATOR`.

```
<?php
$include = dirname(__FILE__) . "/include";
set_include_path($include . PATH_SEPARATOR . get_include_path());
?>
```

113 Vypsání XML značky



začátečník

Konfigurační direktivou `short_open_tag` se dá povolit zahajování PHP kódu značkou `<?`. To ovšem koliduje se značkou `<?xml`, která se používá pro zahájení XML dokumentů. Pro vypsání této značky je tedy lepší použít PHP kód:


```
<?php
echo "<?xml version='1.0' encoding='UTF-8' ?>\n";
?>
```



Poznámka: Značku lze vyspat také trikem – `<<?php ?>?xml version='1.0' encoding='UTF-8' ?>`.

114 Použití konstant pro vlastní konfiguraci



Pokud má mít naše aplikace nějakou globální konfiguraci (určující např. výchozí počet vypisovaných záznamů), je obvykle vhodné ji zapsat do konstant. Ty jsou na rozdíl od proměnných neměnné a automaticky viditelné ve funkcích. Zapisují se obvykle velkými písmeny se slovy oddělenými podtržítky:

```
<?php
// dá se změnit a není přímo vidět ve funkcích
$defaultLimit = 30;

// lepší
define("DEFAULT_LIMIT", 30);
?>
```

115 Konstanty bez rozlišení velikosti písmen



Pomocí třetího parametru funkce `define` lze vytvořit konstanty, u kterých se nebude rozlišovat velikost písmen. Takové konstanty je ale obvykle lepší nevytvářet, protože vedou k psaní nejednotného kódu.

Užitečné to je pouze u výsavních konstant PHP, jako jsou `null`, `true` a `false`.

116 Definice absolutní cesty



Pokud si potřebujeme definovat absolutní cestu k aplikaci v souborovém systému, je vhodné ji odvodit z cesty k aktuálnímu souboru. Jinak totiž aplikace téměř jistě přestane fungovat při přesunu na jiný hosting, protože používané cesty nejsou ustálené.

```
<?php
// nepoužívat
define("APPLICATION_PATH", "/var/www/example.com/htdocs");

// lepší
define("APPLICATION_PATH", dirname(__FILE__));

// možné od PHP 5.3
define("APPLICATION_PATH", __DIR__);
?>
```



Poznámka: Cestu lze většinou zapsat i relativně (např. pomocí odkazu na nadřazený adresář `../`), to může ale způsobit problémy při spuštění skriptu z jiného adresáře (např. na příkazovém řádku nebo v obsluze funkce `register_shutdown_function`).

117 Definice absolutního URL



Pokud naše aplikace potřebuje pracovat s absolutním URL (např. při přesměrování nebo při posílání odkazů do aplikace e-mailem), je možné ho odvodit z aktuálního URL. To umožní aplikaci snadno testovat na vývojovém počítači nebo ji přesunout na jinou adresu:

```
<?php
// neumožňuje provoz jinde
define("ABSOLUTE_URL", "http://www.example.com/");

// lepší
define("ABSOLUTE_URL", "http://$_SERVER[SERVER_NAME]
    . dirname($_SERVER["SCRIPT_NAME"])
);
?>
```



Poznámka: Zcela obecná aplikace by měla nejprve kontrolovat proměnnou HTTP_HOST, protože SERVER_NAME nemusí být vždy nastaveno. Pokud aplikace může běžet na protokolu HTTPS, mělo by to být v absolutním URL také zohledněno.

118 Kde je uložena cesta k sobě



Proměnná `$_SERVER["PHP_SELF"]` obsahuje cestu zobrazené stránky až do případného otazníku. Cestu ke spuštěnému skriptu obsahuje proměnná `$_SERVER["SCRIPT_NAME"]`. Tyto dvě proměnné se liší v případě přepisu URL nebo při použití tzv. *path info*.

Cestu včetně případných parametrů obsahuje proměnná `$_SERVER["REQUEST_URI"]`, ta se může hodit třeba v případě, kdy chceme nastavit cookie platnou jen pro momentálně zobrazenou stránku. Řetězec se samotnými parametry je k dispozici v proměnné `$_SERVER["QUERY_STRING"]`, tato proměnná je ale potřeba zřídka, protože z parametrů vytvoří PHP pole `$_GET`.

119 Instalace vlastní aplikace



Pokud vytváříme aplikaci určenou pro instalaci na více počítačů, můžeme vytvořit jednoduchý instalační skript, který cílovým uživatelům umožní nastavení konfigurace (např. přihlašovací údaje k databázovému serveru). Tento skript může vytvořený konfigurační soubor buď rovnou uložit, nebo nabídnout ke stažení.

Z pohledu bezpečnosti je důležité, aby se skript nedal spustit v běžící aplikaci, protože by mohl začít např. přihlašovací údaje posílat na jiný server. Hlavní aplikace proto může kontrolovat, zda je instalační skript smazán:

```
<?php
if (file_exists("config.php")) {
    echo "Před spuštěním aplikace nejprve smažte instalační skript.";
    exit;
}
?>
```


Syntaxe jazyka

120 Možnosti uzavírání PHP kódu



začátečník

Pro uzavření PHP kódu je možno v PHP použít až čtyři různé zápisy:

1. `<?php ?>`
2. `<? ?>`
3. `<% %>`
4. `<script language="php"> </script>`

Druhý způsob se dá zakázat konfigurační direktivou `short_open_tag`, třetí pomocí `asp_tags`, čtvrtý zase vypadá spíše jako klientský skript. Nejlepší je tedy vždy používat první způsob zápisu.

121 Vynechání koncové značky



začátečník

Pokud obsah souboru končí PHP kódem, není nutné kód ukončovat značkou `?>`. Kromě ušetřeného psaní to má tu výhodu, že pokud by soubor měl na konci nějaké zapomenuté bílé znaky (mezery, tabulátory, konce řádků), tak se při uvedení koncové značky vypíšíou. To může vadit například při odesílání HTTP hlaviček, které je potřeba poslat dříve než jakýkoliv textový výstup.

122 Konec řádku za koncovou značkou



začátečník

Znak konce řádku za koncovou značkou `?>` se nevypisuje. To obvykle vede k lepší přehlednosti výsledného HTML kódu, ale je na to potřeba dát pozor v případě, kdy další řádek začíná textem. V tom případě se tento text připojí k předchozímu řádku. Řešit se to dá vložením dvou konců řádek nebo mezery.

123 Ukončení příkazu



pokročilý

Příkaz se v PHP ukončuje středníkem, před značkou `?>` je ale nepovinný. Vynechání středníku je nicméně vhodné se až na výjimky vyhnout, protože znesnadňuje budoucí přidání příkazu na konec bloku PHP kódu.

124 Ignorování zbytku souboru



znalec

Od PHP 5.1 je možné z pohledu parseru ignorovat zbytek souboru od určitého místa. Toto místo se definuje zavoláním funkce `__halt_compiler`. Tento obrat se dá využít v případě, kdy chceme do souboru zahrnout i data, se kterými skript pracuje, a vytvořit tak např. samorozbalitelný archiv. Bajtovou pozici zakončení zpracování PHP získáme konstantou `__COMPILER_HALT_OFFSET__`.

125 Ukončení příkazu pro ignorování zbytku souboru



U funkce `__halt_compiler` se dá využít toho, že příkaz lze v PHP ukončit značkou `?>`. Za touto značkou navíc může být znak konce řádku, který se ignoruje.

Nepřehledný způsob:

```
<?php
__halt_compiler();textová data
```

Přehledný způsob:

```
<?php
__halt_compiler() // středník chybí záměrně
?>
textová data
```

Textové editory totiž funkci `__halt_compiler` obvykle neznají a při ukončení příkazu středníkem by zbytek souboru zvýrazňovaly také jako PHP kód.

126 Druhy komentářů



PHP dovoluje komentáře zapsat třemi možnými způsoby:

```
<?php
// jednořádkový komentář

# jednořádkový komentář

/*
víceřádkový komentář
*/
?>
```

Jednořádkové komentáře jsou vhodné k popisu částí kódu, víceřádkové komentáře se hodí především pro dokumentační komentáře. Dají se použít také pro dočasné skrytí bloku kódu, nelze je ale vnořovat do sebe. Pro dočasné skrytí kódu můžeme také každý řádek uvodit komentářem `#`, který jinde nebudeme používat. Některé programátorské editory mají klávesovou zkratku, která dovoluje tento komentář přidat nebo odebrat ze všech označených řádků.

127 Ukončení jednořádkového komentáře



Jednořádkový komentář se ukončuje buď koncem řádku, nebo značkou `?>`. To se hodí obzvlášť u jednořádkových bloků PHP kódu: `<?php echo $count; // vypsání počtu ?>`.

128 Dokumentační komentáře



Speciálním druhem komentářů jsou dokumentační komentáře. Ty se uvozují znaky `/**`, a jak už název napovídá, mohou kromě popisu kódu sloužit i k vygenerování dokumentace. K tomu se nejčastěji používá program **phpDocumentor**. Kromě toho dokážou s dokumentačními komentáři pracovat některá pokročilá vývojová prostředí (např. **Eclipse**) a používat je pro nápovědu u funkcí, metod, tříd a dalších prvků kódu.

Dokumentační komentář se píše před dokumentovaný kód a každý řádek bloku musí začínat hvězdičkou. Kromě běžného textu lze v komentáři pomocí znaku `@` uvádět značky, které mají zvláštní význam.

129 Značky dokumentačních komentářů



Řada značek dokumentačních komentářů je již zastaralá. Např. v PHP 5 nemá smysl používat značku `@abstract`, protože abstraktní metody a třídy lze určit přímo pomocí klíčového slova `abstract`. Stejně tak u vlastností a metod není potřeba používat značku `@access` ovlivňující viditelnost, ta ale může mít využití u globálních funkcí. V praxi se tak používají především tyto značky:

```
<?php
/**
 * @author      jméno <email>
 * @copyright   jméno datum
 * @license     URL název (licence)
 * @link        URL
 * @package     název (balíčku)
 * @param       typ popis (parametru funkce nebo metody)
 * @return      typ popis (návrátové hodnoty)
 * @throws      třída (vyvolávané výjimky)
 * @var         typ popis (proměnné nebo vlastnosti)
 */
?>
```

Značky se v dokumentačním komentáři mohou opakovat, platí to třeba i pro informace o vyvolávaných výjimkách.

130 Jak na „Nutno dodělat“



Někdy si v kódu potřebujeme poznačit informaci o tom, že je něco potřeba ještě dodělat. Můžeme k tomu použít buď dokumentační komentář `@todo`, nebo obyčejný komentář začínající textem `TODO`. Některá pokročilá vývojová prostředí z těchto komentářů dokážou vytvořit seznam úkolů.

```
<?php
mysql_query($query); // TODO: předat identifikátor připojení
?>
```

131 Zpracování dokumentačních komentářů



Kromě existujících aplikací pro zpracování dokumentačních komentářů je můžeme zpracovat i sami. Od PHP 5 je vrací funkce `token_get_all`, ale především jsou k dispozici při reflexi. PHP se nestará o vnitřní zpracování dokumentačního komentáře (především získání jednotlivých značek), o to už se pomocí řetězcových operací musíme postarat sami.

132 Jak na anotace



Dokumentační komentáře se dají použít i k ovlivnění chování kódu. Pokud si do dokumentačního komentáře poznamenáme nějakou speciální značku, může ji okolní kód brát v úvahu. Dobrým příkladem využití může být např. označení databázového typu v knihovně ORM. Tento způsob ovlivnění nějaké funkčnosti může být při používání velmi pohodlný, ne každému ale vyhovuje – někomu přijde změna chování programu pomocí anotací až příliš magická.

Pro zpracování anotací můžeme použít reflexi, je to ale poměrně pracné. Raději proto nejspíš sáhneme po hotové knihovně, vhodná je např. **Nette Annotations**.

133 Co je operátor identity



Většina porovnávacích operátorů si nevšímá datového typu a pracuje jen s hodnotou operandů. Pokud se datové typy liší, jsou podle definovaných pravidel sjednoceny. To může být na škodu například u funkce `strpos`, která vrací v případě neúspěchu hodnotu `false` a v případě nalezení řetězce index jeho pozice počínaje nulou.

```
<?php
if (strpos($string, "a") === false) {
    echo "Řetězec nebyl nalezen.\n";
}
?>
```

Operátor identity je vhodné používat ve většině případů místo prostého porovnání, protože to pomůže najít hůře odhalitelné chyby.

134 Jaký použít operátor nerovnosti



V PHP jsou dva operátory pro zjištění nerovnosti dvou výrazů: `!=` a `<>`. Protože PHP vychází především z jazyka C, kde se používá operátor `!=`, je zvykem nerovnost zkoumat právě tímto operátorem i v PHP, druhý operátor má ale zcela totožnou funkci.



Poznámka: Operátor `!==` uspěje při rozdílnosti typu nebo hodnoty.

135 Pořadí porovnávání



Operátor = slouží v PHP k přiřazení hodnoty, a nikoliv k porovnání jako v některých jiných programovacích jazycích. Přiřazení je ale také výraz, který se může vyhodnotit např. v podmínce, takže zápis `if ($a = 5)` je korektní, i když neudělá to, co autor nejspíš očekával (přiřadí do proměnné `$a` hodnotu 5 a tuto hodnotu vyhodnotí jako pravdu).

Pokud se někomu operátory pletou třeba v důsledku přechodu z jiného programovacího jazyka, je možné proměnnou uvádět až na druhém místě porovnání, protože zápis `if (5 = $a)` způsobí syntaktickou chybu.

Obecně vzato je ale přehlednější uvádět nejprve výraz, který porovnáваме, a teprve potom hodnotu, se kterou porovnáваме.

136 Co je to ternární operátor



Operátor `?:` se jmenuje ternární podle toho, že pracuje se třemi operandy. Pokud by takových operátorů bylo více, tak bychom mu mohli říkat podmíněný výraz. Operátor vyhodnotí výraz před otazníkem, a pokud je splněn, vrátí výraz za otazníkem, jinak výraz za dvojtečkou. Je tedy jakousi výrazovou obdobou podmíněného příkazu `if else`.

Operátor je vhodné využívat pouze v případě, kdy jsou všechny tři výrazy dostatečně krátké, jinak je nepřehledný.

137 Zkrácený ternární operátor



Od PHP 5.3 je možno v ternárním operátoru vynechat prostřední část. Místo ní se potom použije testovaný výraz:

```
<?php
echo ($a ?: 0); // totéž jako ($a ? $a : 0)
// pokud proměnná $a není nastavena, dojde k chybě E_NOTICE
?>
```

Hodí se to hlavně v případě, kdy je testovaný výraz složitý a obsahuje např. volání funkce.



Poznámka: Pojmenování ternární operátor je odvozeno z počtu tří operandů, se kterými tento operátor pracuje. Při vynechání prostřední části jde vlastně o binární operátor a přesnější by bylo nazývat ho zkrácený podmíněný výraz.

138 Jak používat operátor plus



Operátor plus se v PHP používá ke sčítání čísel, pro spojení řetězců se používá operátor tečka. Pokud se pokusíme operátor plus použít na dva řetězce, tak se převedou na čísla a sečtou.

Jinak se operátor plus chová s poli, kde provede jejich sjednocení.

139 Pořadí násobení a dělení



Násobení a dělení celých čísel je vhodné dělat v takovém pořadí, aby mezivýsledek zůstal celým číslem v přípustném rozsahu. Jednak jsou celočíselné operace mírně rychlejší, jednak v tom případě nemůže dojít k zaokrouhlovacím chybám.

```
<?php
var_dump(1 / 2 * 2); // float(1)
var_dump(2 * 1 / 2); // int(1)
?>
```

140 Bitový posun



V PHP jsou mírně netradičně zpracovány operátory bitového posunu. Operátor << posouvá bity doleva a zprava doplňuje nuly. V nejvýznamnějším bitu se uchovává informace o znaménku, takže tento operátor může znaménko změnit.

Operátor >> posouvá bity doprava, ale informaci o znaménku zachovává. Zleva se tedy doplňuje u záporných čísel jednička a u kladných nula.

```
<?php
var_dump(PHP_INT_MAX << 1); // vznikne záporné číslo
var_dump(-PHP_INT_MAX >> 1); // číslo zůstane záporné
?>
```

141 Obvyklý bitový posun



Při bitovém posunu doprava je zvykem zleva doplňovat nuly. Takový operátor v PHP bohužel k dispozici není, můžeme si ale napsat vlastní funkci, která se o to postará:

```
<?php
/** Bitový posun doprava s doplňováním nul
 * @param int
 * @param int
 * @return int
 */
function rightShift($number, $shift) {
    return ($number >> $shift) & (PHP_INT_MAX >> ($shift - 1));
}
?>
```

142 Bitová negace



Operátor ~ se používá pro obrácení hodnoty bitů v čísle. V PHP se nejčastěji používá spolu s konstantami.

```
<?php
// nastaví úroveň chyb na všechny kromě poznámek
error_reporting(E_ALL & ~E_NOTICE);
?>
```



Poznámka: Pro zapnutí nebo vypnutí určitého druhu chyb se někdy používá zápis `E_ALL ^ E_NOTICE` (operátor bitový XOR). Tento zápis se ale spoléhá na to, že konstanta `E_ALL` danou chybu obsahuje nebo naopak neobsahuje – její hodnota se ale v průběhu verzí měnila, takže je lepší se tomuto zápisu vyhnout.

143 Definice konstant



Konstanty slouží pro přiřazení symbolického názvu nějaké neměnné hodnotě. Na rozdíl od proměnných nelze jejich hodnotu změnit a nejde je ani zrušit.

```
<?php
define("NOW", time()); // vytvoření konstanty
const ERROR = 'error'; // alternativní vytvoření od PHP 5.3
?>
```

144 Použití konstant



Pokud chceme přistoupit k definované konstantě, stačí uvést její název.

```
<?php
if (defined("NOW")) { // test existence
    echo NOW; // použití
}
?>
```

Kromě toho lze použít také funkci `constant`, což se hodí v případě, kdy název konstanty dynamicky sestavujeme.

```
<?php
$input = '_POST';
constant("FILTER$input"); // dynamicky sestavený název konstanty
?>
```

145 Nedefinované konstanty



Pokud se v PHP pokusíme přistoupit k nedefinované konstantě, tak se vyvolá chyba úrovně `E_NOTICE` a vrátí se místo ní řetězec s identifikátorem.

Toto chování bychom neměli využívat jednak kvůli vyvolané chybě, a jednak kvůli tomu, že při dodatečné definici konstanty přestane kód fungovat.

146 Konstanty obsahující pole



V PHP lze definovat pouze konstanty obsahující skalární hodnoty a `null`, nelze tedy vytvořit konstantu obsahující pole nebo objekt. Tento nedostatek se obchází různými způsoby:

```

<?php
// nelze
define("DATABASE", array(
    "server" => "localhost",
    "login" => "ODBC",
    "password" => "",
));

// musíme volat funkci unserialize při každém použití konstanty
define("DATABASE", serialize(array(
    "server" => "localhost",
    "login" => "ODBC",
    "password" => "",
)));

// definice globální proměnné místo konstanty
$DATABASE = array(
    "server" => "localhost",
    "login" => "ODBC",
    "password" => "",
);
// použití: $GLOBALS["DATABASE"]["login"]

// definice více konstant
define("DATABASE_SERVER", "localhost");
define("DATABASE_LOGIN", "ODBC");
define("DATABASE_PASSWORD", "");

// použití třídnicích konstant
class Database {
    const SERVER = "localhost";
    const LOGIN = "ODBC";
    const PASSWORD = "";
}
// použití: Database::LOGIN
?>

```

Každý přístup má své výhody a nevýhody, obvykle je ale nejjednodušší prostě definovat více konstant.

147 Jak uložit návratovou hodnotu funkce



začátečník

Příkaz `return` použitý uvnitř funkce způsobí její ukončení a předání návratové hodnoty. Někdy si ale potřebujeme návratovou hodnotu jen zapamatovat a funkci nechat pokračovat (např. proto, aby po sobě mohla uklidit). Pro volbu názvu proměnné obsahující návratovou hodnotu lze využít toho, že názvy proměnných nekolidují s jinými identifikátory PHP, a lze ji pojmenovat prostě *\$return*.

148 Interní funkce s proměnným počtem parametrů



znalec

Řada PHP funkcí přijímá proměnný počet parametrů, nepovinné parametry ale nemusí mít definovanou výchozí hodnotu. Funkce se totiž interně rozhodují podle počtu předaných parametrů.

```
<?php
// první funkce vyvolá chybu, druhá použije poslední připojení
mysql_query($query, null);
mysql_query($query);
?>
```

Implementaci těchto funkcí si můžeme představit takto:

```
<?php
function mysqlQuery($query, $connection = null) {
    if (func_num_args() < 2) {
        // získání posledního připojení do proměnné $connection
    }
}
?>
```

Znalost tohoto chování je důležitá při vytváření vyšších vrstev nad těmito funkcemi.

149 Předefinování funkce



znalec

PHP samo o sobě nenabízí možnost předefinování funkce. Jakmile je funkce jednou definovaná, tak pokus o definici funkce se stejným názvem skončí chybou. Tuto schopnost nabízí pouze extenze **APD** a **Runkit** v podobě funkcí `override_function`, resp. `runkit_function_redefine`. Kromě experimentů je ale lepší se této vlastnosti vyhnout a funkce definovat vždy jen jednou.



Poznámka: Přejmenování umožňují funkce `rename_function`, resp. `runkit_function_rename`.

150 Definice výjimky



začátečník

Pro výjimky musíme použít objekt třídy `Exception`, kterému můžeme určit chybovou hlášku a číselný kód chyby. Při zachytávání výjimek v bloku `catch` se ale nerozhoduje podle kódu chyby, ale podle třídy výjimky. Pro vlastní výjimky je tedy zcela běžné definovat vlastní třídu. Ta musí být potomkem třídy `Exception`, ale nemusí ji nijak funkčně rozšiřovat.

```
<?php
class DbException extends Exception {
}
?>
```

151 Rozšíření výjimky



začátečník

Při definici třídy výjimky si často vystačíme s metodami nabízenými třídou `Exception`. Pokud ale třeba výjimce chceme přiřadit nečíselný chybový kód vrácený nižší vrstvou, nic nám v tom nebrání. Praktické je v tom případě přepsat hlavně konstruktor.

```
<?php
class ServiceException extends Exception {
```

```

    /** Konstruktor
     * @param string
     * @param string
     */
    function __construct($message, $code) {
        parent::__construct($message, 0);
        $this->code = $code;
    }
}

try {
    throw new ServiceException("Zpráva", "kód");
} catch (Exception $e) {
    echo $e->getCode() . "\n";
}
?>

```

152 Probublání výjimky



Pokud blokem `catch` zachytíme výjimku a teprve tehdy zjistíme, že ji nedokážeme zpracovat (např. podle chybového kódu), můžeme ji poslat ke zpracování nadřazenému bloku. Výjimku není třeba znovu vytvářet, ale stačí vyvolat tu zachycenou:

```

<?php
try {
    throw new Exception("Probublání", 1);
} catch (Exception $e) {
    if ($e->getCode() == 1) { // takovou výjimku neumíme zpracovat
        throw $e;
    }
    // tady bude běžná obsluha chyby
}
?>

```

153 Eskalace výjimky



Chyby způsobené nějakou námi vyvíjenou knihovnou můžeme chtít reprezentovat jednotnou výjimkou. Někdy se ale může hodit vědět i to, jaká výjimka chybu původně způsobila. Od PHP 5.3 to lze vyřešit předáním třetího parametru konstruktoru výjimky, pomocí kterého lze předat předchozí výjimku:

```

<?php
class LibraryException extends Exception {
}

try {
    throw new Exception("Vyvolání obecné chyby");
} catch (Exception $e) {
    throw new LibraryException("Při komunikaci došlo k chybě", 0, $e);
}
?>

```

154 Překlad chyb na výjimky



Jazyku PHP se někdy vyčítá, že sice obsahuje mechanismus výjimek, sám ho ale až na několik málo případů nepoužívá a místo toho využívá vlastní mechanismus chyb. Nicméně je velmi snadné překládat na výjimky všechny chyby, ke kterým dojde v interních funkcích PHP. Stačí použít funkci `set_error_handler` pro definici vlastní služby chyb a v ní chybu na výjimky přeložit. Od PHP 5.1 pro tento druh výjimky dokonce existuje vlastní třída.

```
<?php
/** Funkce pro set_error_handler překládající chyby na výjimky
 * @param int
 * @param string
 * @param string
 * @param int
 * @throws Exception
 */
function exceptionErrorHandler($level, $message, $file, $line) {
    if (error_reporting() & $level) {
        throw new Exception($message, 0, $level, $file, $line);
    }
}
set_error_handler("exceptionErrorHandler");
strpos("abc", "a", 4); // vyvolá výjimku
?>
```



Poznámka: Před PHP 5.3 obsahuje takto vyvolaná výjimka nesprávný *stack trace*.

155 Typy výjimek v PHP



Kromě třídy `Exception`, která je předkem všech výjimek, a třídy `ErrorException`, která je určena k překladu chyb na výjimky, definuje další standardní výjimky extenze **SPL**. Užitečné mohou být především následující výjimky:

- `LogicException` a její potomci `InvalidArgumentException` a `LengthException`.
- `RuntimeException` a její potomci `OutOfBoundsException` a `UnexpectedValueException`.

156 Zobrazení typu výjimky



Pokud v bloku `catch` chceme vypsát typ výjimky, tak můžeme využít toho, že výjimka je v PHP vždy tvořena objektem, takže můžeme získat její třídu pomocí funkce `get_class`.

```
<?php
try {
    throw new UnexpectedValueException("Nečekaná hodnota");
} catch (Exception $e) {
    echo get_class($e) . ": " . $e->getMessage() . "\n";
}
?>
```



Poznámka: Takovouto obsluhu výjimek je vhodné používat třeba při hlášení chyb v PHP. Pro běžné otestování třídy výjimky lze použít operátor `instanceof`, kromě toho lze objekt výjimky také přímo vypsát.

157 Obsluha nezachycené výjimky



pokročilý

Pokud dojde k výjimce, kterou program nezachytí, skončí fatální chybou. Toto chování lze upravit zavoláním funkce `set_exception_handler` definující *callback*, který se v takovém případě zavolá. Funkci můžeme použít třeba pro odeslání informace o výjimce e-mailem.

```
<?php
function handleException($exception) {
    mail("admin@example.com", "Vyjimka", $exception->getMessage());
}
set_exception_handler('handleException');
throw new Exception("Nezachycená výjimka");
?>
```



Poznámka: Skončením funkce je ukončen i celý program.

158 Skrytí chyb



pokročilý

V PHP je k dispozici operátor `@` (zavináč), který skryje všechny chyby vyvolané následujícím příkazem. Použití tohoto operátoru bychom se ale měli v naprosté většině případů vyhnout. Pokud např. funkce vyvolá chybu v případě, že jí předáme nesprávné parametry, je mnohem vhodnější tyto parametry zkontrolovat před zavoláním samotné funkce. Operátor `@` totiž může skrýt i chyby, se kterými nepočítáme a které se potom špatně odhalují (včetně fatálních). Použití tohoto operátoru také aplikaci zpomaluje.

Operátor bychom neměli používat ani v případě, kdy vznik chyby nedokážeme ovlivnit (např. funkce pro připojení k databázi vyvolá chybu v případě, kdy se k databázovému serveru nedá připojit). O takovéto chybě je vhodné samozřejmě uživatele informovat vlastní zprávou:

```
<?php
if (!mysql_connect()) {
    echo "Nepodařilo se připojit k databázovému serveru.\n";
}
?>
```

Skrytí systémové chyby na produkčním serveru zajistíme vypnutím konfigurační direktivy `display_errors`. Logování těchto chyb naopak povolíme direktivou `log_errors`. V logu se pak dozvíme, kdy připojení k databázi nefungovalo.



Poznámka: Vypnutí všech chyb funkcí `error_reporting(0)` má stejné nevýhody jako použití operátoru `@`.

159 Zobrazení všech chyb



Pokud je nějaká knihovna napsaná s hojným využitím operátoru @ a my v ní potřebujeme dohledat chyby, můžeme zapnout extenzi **scream**, která operátor pro skryté chyby potlačí.

160 Úklid po chybě



Pokud dojde k nějaké chybě, je vhodné po sobě uklidit. Od PHP 5.3 se k tomu dá využít operátor `goto`:

```
<?php
function fillFileGoto($filename) {
    $fp = fopen($filename, "w");
    // první část kódu
    if ($error1) {
        goto clean;
    }
    // druhá část kódu
    if ($error2) {
        goto clean;
    }
    return $fp;
clean:
    fclose($fp);
    unlink($filename);
    return false;
}
?>
```

Ve starších verzích lze použít cyklus `do` s jedním průchodem, kód je ale o něco málo nepřehlednější:

```
<?php
function fillFileBreak($filename) {
    $fp = fopen($filename, "w");
    do {
        // první část kódu
        if ($chyba1) {
            break;
        }
        // druhá část kódu
        if ($chyba2) {
            break;
        }
    }
    return $fp;
} while (false);
fclose($fp);
unlink($filename);
return false;
}
?>
```


Nejčistší je použití výjimek, s těmi je ale spojena jistá časová režie, která může být u časově kritických operací na škodu:

```
<?php
class FillFileException extends Exception {
}

function fillFileThrow($filename) {
    $fp = fopen($filename, "w");
    try {
        // první část kódu
        if ($chyba1) {
            throw new FillFileException;
        }
        // druhá část kódu
        if ($chyba2) {
            throw new FillFileException;
        }
        return $fp;
    } catch (FillFileException $e) {
        fclose($fp);
        unlink($filename);
        return false;
    }
}
?>
```

Jazykové konstrukce

161 Univerzálnost cyklu for



Je dobré si uvědomit, že příkaz `for` má univerzální syntaxi a dá se použít i pro složitější věci než jen pro předem známý počet průchodů. Jednotlivé výrazy znamenají (inicializace; podmínka ukončení; inkrementace). Inicializace se provádí před prvním průchodem, podmínka se vyhodnocuje před každým průchodem a inkrementuje se na konci každého průchodu.

```
<?php
// delší řešení
$queue->first();
while (!$queue->last()) {
    // zde bude kód cyklu
    $queue->next();
}

// využití for cyklu
for ($queue->first(); !$queue->last(); $queue->next()) {
    // zde bude kód cyklu
}
?>
```

162 Došli jsme na konec cyklu?



Pokud při iteraci předčasně vyskočíme za určité podmínky z cyklu, můžeme chtít následně zjistit, zda se cyklus dokončil. To se dá poznat podle hodnoty proměnné, přes kterou iterujeme:

```
<?php
for ($i=0; $i < $count; $i++) {
    if (!processData($i)) {
        break;
    }
}
if ($i == $count) {
    // došli jsme na konec cyklu
}
?>
```

163 Počet průchodů cyklem



Pokud si u podmíněného cyklu `while` potřebujeme počítat počet průchodů, můžeme využít obecnost cyklu `for`:

```
<?php
for ($count = 1; $row = $result->fetchArray(); $count++) {
    // v proměnné $count je uložen počet průchodů počítaný od 1
}
?>
```

164 Jak použít operátor čárka



Především ve for cyklu se používá operátor čárka, který jednoduše vyhodnotí více výrazů:

```
<?php
for ($i=0, $x->first(); $i < 10 && $x->valid(); $i++, $x->next()) {
    // prochází zároveň přes proměnné $i a $x
}
?>
```

165 Alternativní syntaxe řídicích konstrukcí



Většina jazykových konstrukcí PHP umožňuje zápis v alternativní syntaxi:

```
<?php
if ($condition):
    // kód
endif;
?>
```

Tato syntaxe může být přehlednější především v situaci, kdy se hodně kombinuje PHP a HTML kód. Sám ji ale nepoužívám, protože programátorské editory umožňují snadno přeskakovat mezi otevírací a uzavírací závorkou, takže klasická syntaxe dovoluje lepší orientaci v kódu.

166 Řetězení podmíněného příkazu



Kvůli alternativní syntaxi je v PHP povoleno psaní častého obratu `else if` také jako `elseif`. Důvodem je, aby po takto zřetězených příkazech nebylo potřeba uvádět několikrát `endif`. Tento obrat lze ale použít i v obvyklé syntaxi:

```
<?php
if ($a == 2) {
} elseif ($b == 3) {
}
?>
```

167 Vyvolání chyby v podmíněném příkazu



Pokud má podmíněný příkaz vyvolat chybu (např. vrácením hodnoty `false` nebo vyvoláním výjimky), je vhodné podmínku zkonstruovat tak, aby nebylo vůbec potřeba uvádět blok `else`:

Toto je pouze náhled elektronické knihy. Zakoupení její plné verze je možné v elektronickém obchodě společnosti eReading.