Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Využití regulárních výrazů při opravách HTML kódu

Kamil Štědrý

Bakalářská práce

2008

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra informačních technologií
Akademický rok: **2007/2008**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Kamil ŠTĚDRÝ**

Studijní program: **B2646 Informační technologie**

Studijní obor: **Informační technologie**

Název tématu: **Využití regulárních výrazů při opravách HTML kódu**

Zásady pro vypracování:

Problém různých programů, které automaticky generují html kód je, že ve výsledném html souboru je zbytečně moc html tagů.
Cílem teoretické části bude:
* popis regulárních výrazů a jejich možností,
* studium a návrh vhodných regulárních výrazů pro čištění html kódu,
* navrhnout různé úrovně čištění kódu.

Úkolem aplikační části bude vytvořit aplikaci využívající regulární výrazy na zjednodušení a zpřehlednění html kódu. Z html kódu by měli zůstat jen základní prvky - odkazy, obrázky, tabulky, odstavce. Uživatel bude moci nastavovat různou úroveň čištění kódu.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

Pavel Satrapa - Regulární výrazy - seriál pro root.cz - http://www.kit.vslib.cz/ satrapa/docs/regvyr
Dokumentace k php - http://cz.php.net/docs.php

Vedoucí bakalářské práce: **RNDr. Josef Rak**
Ústav elektrotechniky a informatiky

Datum zadání bakalářské práce: **30. listopadu 2007**

Termín odevzdání bakalářské práce: **16. května 2008**

Univerzita Pardubice Česká republika

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

V Pardubicích dne 29. dubna 2008

SOUHRN

Tato bakalářská práce se zabývá podrobným vysvětlením regulárních výrazů. Zaměřuje se na možnost jejich použití při opravách HTML kódu. Druhou částí je implementace webové aplikace na základě nabytých znalostí.

KLÍČOVÁ SLOVA

regulární výrazy, opravy html, validita, kódy

TITLE

Useful using of regular expressions in HTML code corrections

ABSTRACT

This bachelor work deals with a detailed explanation of regular expressions. The focus is on possibilities of their use in HTML code corrections. The second part is web application implementation based on gained knowledge.

KEYWORDS

regular expressions, html corrections, validity, codes

Obsah

Seznam tabulek

Seznam obrázků

Seznam zkratek

ASP	Active Server Pages (aktivní serverové stránky)
CSS	Cascading Style Sheets (tabulky kaskádových stylů)
HTML	HyperText Markup Language (značkovací jazyk pro hypertext)
Mac OS	Macintosh Operating System (operační systém počítačů Macintosh)
OOP	Object-Oriented Programming (objektově orientované programování)
PDA	Personal Digital Assistant (osobní digitální pomocník)
PHP	PHP: Hypertext Preprocesor (PHP: Hypertextový preprocesor)
RE	Regular Expressions (regulární výrazy)
SGML	Standard Generalized Markup Language (standardizovaný zobecněný značkovací jazyk)
UTF-8	Universal Character Set Transformation Format (transformační formát univerzální znakové sady)
W3C	World Wide Web Consortium (společenství celosvětové pavučiny)
WWW	World Wide Web (celosvětová pavučina)
WYSIWYG	What You See Is What You Get (co vidíš, to dostaneš)
XHTML	eXtensible HyperText Markup Language (rozšiřitelný hypertextový značkovací jazyk)
XML	eXtensible Markup Language (rozšiřitelný značkovací jazyk)

1 UVEDENÍ DO PROBLEMATIKY OPRAV HTML KÓDU

1.1 Nadbytečnost kódu

V dnešní době může vytvářet internetové stránky téměř každý uživatel, který vlastní počítač s přístupem na Internet. Nejsnazší cestou k vytvoření statické internetové prezentace je použití některého z nabízených WYSIWYG editorů. Uživatel tak interaktivně navrhuje obsah i vzhled celého dokumentu, jak je zvyklý např. z textových editorů. Příkladem takových editorů jsou programy *Microsoft FrontPage* nebo *Macromedia Dreamweaver*. Extrémním řešením tvorby internetových stránek je jejich vytvoření v textovém editoru (např. *Microsoft Word*) a uložení jako HTML dokumentu. Problémem těchto i jiných pomocných nástrojů, které automaticky generují HTML kód je, že výsledný soubor obsahuje nadbytečně mnoho HTML *tagů* a *atributů*. Zejména časté opakování stejných hodnot zbytečně zvyšuje velikost samotného souboru a snižuje přehlednost kódu zkušenějším uživatelům.

1.2 Cíl práce

Cílem práce je prostudování možností pro čištění HTML kódu a získání teoretických znalostí, nezbytných pro návrh vhodných postupů při implementaci dané aplikace. Vytvořená aplikace by měla opravovat HTML soubory a čistit přebytečný kód dle zadání uživatele. Dále by měla nabízet několik možností nastavení úrovní čištění kódu, aby tím uživatel mohl řídit výstup aplikace podle svých potřeb. Důraz je kladen také na zachovávání *validity* opravených souborů, tzn., že musí být zachována struktura zápisu kódu a nesmí být vymazáváno více *tagů* a *atributů* než je požadováno. Výsledná aplikace je přístupná na adrese *http://opravy-html.kvalitne.cz*.

1.3 Účel aplikace

Hlavním účelem takovéto aplikace je zjednodušení a zpřehlednění výsledného HTML kódu pro jeho další použití. To přináší další výhodu, a to zmenšení velikosti výsledného souboru. Úspora velikosti může být značná a sama je motivem pro používání aplikace. Dále je aplikaci možné použít pro opravy zastaralých kódů a jejich snadné použití pro nové projekty. Zajímavě lze aplikaci použít pro výběr pouze požadovaných elementů dokumentu a tím filtrovat obsah. Toto použití ovšem závisí na konkrétním zápisu kódu daného dokumentu. Aplikace má potenciál k rozšíření o další funkce, např. nahrazování zvolených *tagů* za jiné atd.

1.4 Komu je aplikace určena

Cílem aplikace není pokrytí co nejširší skupiny uživatelů. Jsou uživatelé, kteří přispívají obsahu Internetu i bez znalosti *syntaxe* jazyka HTML. Takovým jistě prospěšná nebude, ani jí nebudou vyhledávat. Aplikace je tedy určena spíše zkušenějším uživatelům, kteří vytvářejí internetové stránky a mají alespoň základní znalost HTML kódu. Pro uživatele, píšící přímo zdrojový kód, bude prospěšná zejména při úpravách cizích dokumentů nebo stránek stažených z Internetu.

2 VYSVĚTLENÍ ZÁKLADNÍCH POJMŮ

2.1 HTML (HyperText Markup Language)

2.1.1 Historie HTML

HTML je značkovací jazyk pro hypertext, který se používá pro popis struktury WWW stránek. Jeho vývoj začal v roce 1989 odvozením od jazyka SGML a ve verzi 4.01 byl v roce 1999 ukončen. Předpokládalo se přejití na XHTML, ale v roce 2007 byla započata práce na vývoji verze 5.

2.1.2 Popis jazyka HTML

Jazyk HTML je charakterizován množinou značek a jejich atributů. Mezi značky se zapisují části textu čímž je označen jejich význam. Názvy značek se uzavírají mezi úhlové závorky (< a >) a nazývají se *tagy*. Obvykle jsou párové a název ukončovacího *tagu* začíná lomítkem. Otevírací *tag*, obsah *elementu* a ukončovací *tag* tvoří dohromady *element* dokumentu a za pomoci *atributů* lze upřesňovat vlastnosti *elementů*. Příkladem je *element* označující hypertextový odkaz na internetovou stránku www.upce.cz s *atributem target*, jehož hodnota označuje otevření odkazu v novém okně prohlížeče:

```
<a href=”http://www.upce.cz” target=”_blank”>text odkazu</a>
```

2.1.3 Druhy značek

Značky můžeme rozdělit do tří skupin, podle významu jde o:

- **strukturální značky**, které určují strukturu dokumentu a dávají mu formu.
- **Popisné (sémantické) značky**, které popisují povahu elementu. Jejich koncepce se blíží jazyku XML, což usnadňuje vyhledávání a zpracovávání.
- **Stylistické značky**, které určují vzhled. Trendem je stylování pomocí CSS.

2.2 XHTML (Extensible HyperText Markup Language)

Jazyk XHTML, aktuálně ve verzi 1.1, byl původně vyvíjen jako nástupce jazyka HTML, ze kterého vychází, a byl přepracován tak, aby vyhovoval standardu XML a byla zachována zpětná kompatibilita. Nyní se paralelně pracuje na XHTML 2 a HTML 5.

Oproti HTML musejí být všechny *tagy* ukončeny, a to včetně nepárových *tagů*. XHTML dále rozlišuje velikosti písmen, proto musejí být všechny *tagy* a *atributy* zapsány malými písmeny. Atributy musejí být navíc uzavřeny do uvozovek a dokument musí začínat XML deklarací s uvedeným kódováním (nepovinné pro UTF-8). XHTML nepodporuje takové možnosti formátovaní textu jako HTML, pro tyto účely se používá CSS.

2.3 CSS (Cascading Style Sheets)

Jazyk CSS se používá pro popis zobrazení webových stránek. Hlavním smyslem je oddělení vzhledu od struktury a obsahu dokumentu. CSS nabízí oproti HTML rozsáhlejší možnosti pro formátování. Výhodou je také jednotný styl pro všechny stránky webové prezentace, kdy je formátování zapsáno do jednoho externího souboru a všechny HTML stránky si odtud načítají potřebné styly. Díky načtení do paměti je tento způsob také rychlejší. Jednotný styl přináší také jednoduchou změnu vzhledu na jednom místě. Kaskádové styly umožňují vytvoření několika souborů a změnu stylu podle požadavku uživatele. Vytvořeny mohou být i styly pro různá zařízení, např. styl pro tiskárny, mobilní zařízení nebo zařízení pro zrakově postižené.

Definice stylu se skládá ze selektoru, což může být název elementu, třídy nebo identifikátoru, a deklarací vlastností s příslušnými hodnotami, uzavřenými ve složených závorkách. Příkladem je nastavení červeného písma nadpisům první úrovně:

```
h1 {color: red ;}
```

Třídy umožňují určit různé formátování pro stejné elementy. Zapisují se tečkou a jejím názvem. Pokud před tečku uvedeme název elementu, bude třída spojena jen s daným elementem. V opačném případě je možné třídu použít u všech elementů. Zápis do HTML elementu se provádí pomocí *atributu* `class` a názvu třídy. Podobně fungují identifikátory. Ty se zapisují křížkem (#) a názvem. Rozdílem je, že každý identifikátor může být na stránce použit pouze jednou. Identifikátor se zapisuje do HTML elementu pomocí *atributu* `id` a názvu identifikátoru.

Pro hromadnou deklaraci se jednotlivé selektory oddělují čárkou. Zápis může vypadat takto: `H1, H2, H3 {color: blue}`. Lze použít také kontextovou deklaraci, kdy se styl aplikuje pouze na element uvnitř konkrétního elementu. Zapíše se nejprve vnější element, následuje mezera a vnitřní element. Ovlivnění odkazu pouze v nadpisu: `H1 A {font-size: small}`. Pseudotřídy jsou speciálním typem tříd, které jsou aktivovány obvykle nějakou speciální událostí nebo speciálním stavem. Příkladem je navštívený a nenavštívený odkaz. Pseudotřída se zapisuje dvojtečkou a jejím názvem.

Dědičnost v kaskádových stylech znamená, že element přebírá definované vlastnosti od nadřazených elementů. Platná je ta hodnota, která je definovaná nejblíže danému elementu. Některé vlastnosti (barva, typ písma , ad.) se dědí, jiné ne. Pokud tedy definujeme vlastnost elementu `body`, zdědí tuto vlastnost všechny elementy. Později je však možné tuto vlastnost pro libovolné elementy předeklarovat. Platná je vždy ta vlastnost, která je v zápisu deklarována nejdále.

Mezi kaskádovými styly a HTML dokumenty existuje určitá spojitost. Je vhodné brát na tuto spojitost ohled i při opravách HTML kódu. Například není nutné definovat styl pro elementy nebo třídy, které jsou aplikací odstraněny. Problém ovšem nastává u kaskádových stylů definovaných v externím souboru, který navíc mohou používat i jiné dokumenty. Jedná se pouze o zamyšlení se nad návazností, ale tato funkce by mohla být v aplikaci také implementována.

2.4 PHP (PHP: Hypertext Preprocessor)

Jazyk PHP je vyvíjen od roku 1995, byl původně vytvořen pro zpracování záznamů o přístupech k webu. Jednoduché skripty vycházeli z jazyka *Perl*, později byl rozsáhleji implementován v programovacím jazyce C. Již od počátku byla přidána podpora pro práci s databázemi. Od verze 3 už nebylo použití PHP omezeno jen na operační systém Linux (Unix), ale jsou podporovány operační systémy *Microsoft Windows* a *Mac OS*. Verze 4 znamenala zvýšení výkonu pro složité aplikace a podporu OOP. Nejnovější verze PHP 5.2.5 přináší především vylepšenou práci s objekty.

Skriptovací jazyk PHP slouží ke generování dynamických internetových stránek na straně serveru. Do HTML stránek je vkládán PHP kód, který se provede pokaždé, když má být stránka zobrazena. Uživatel tak uvidí pouze výsledek tohoto skriptu, ne samotný zdrojový PHP kód. Ve zdrojovém kódu je zapsána např. funkce pro zjištění času:

```
Aktuální čas: <?php echo Date ("H.i:s"); ?>
```

Po vykonání skriptu vypadá kód HTML stránky stažené uživatelem takto:

```
Aktuální čas: 10.57:09
```

PHP nabízí celou řadu funkcí pro práci s různými databázemi, soubory, obrázky nebo regulárními výrazy. Obsahuje také matematické a různé další funkce, spojené především s internetovými technologiemi.

2.5 Webová aplikace

Webové aplikace využívají některého ze skriptovacích jazyků. Jde vlastně o dynamicky generované HTML stránky, které poskytují uživateli požadovanou funkcionalitu. Výhodou webových aplikací je, že uživatel nemusí takovou aplikaci instalovat na svém počítači, je dostupná po celém světě a nemusí být implementována pro různé platformy. Strukturou aplikace může být standardní třívrstvý model. Prezentační vrstvou, starající se o zobrazení dat uživateli, je webový prohlížeč. Logickou vrstvou je některý z nástrojů pro dynamické generování stránek (PHP, ASP, ad.) a pro ukládání dat slouží databáze, jako datová vrstva. Webové rozhraní aplikací stále častěji nahrazuje klasické aplikace, příkladem je i mnou vytvořená aplikace sloužící k opravě HTML kódu.

3 REGULÁRNÍ VÝRAZY

3.1 Co jsou regulární výrazy

Regulární výraz bývá často označován jako regex, regexp nebo RE, podle anglického regular expression. Jde o řetězec, popisující celou množinu řetězců, dle regulárního jazyka. Nejčastěji se regulárních výrazů využívá pro vyhledávání a úpravu textů pomocí počítačových programů nebo skriptovacích jazyků. Uživatel tak může díky regulárním výrazům v textu vyhledávat řetězec, který nezná celý, nebo může obsáhnout všechny varianty řetězce, které ho zajímají.

Znalost regulárních výrazů Vám dává do rukou mimořádně silný nástroj pro práci s textem. Jejich prostřednictvím můžete:

- vytahovat z textových dat údaje, které vás zajímají,
- přetvářet je do podoby, kterou potřebujete,
- vyhledávat a nahrazovat v textových editorech a dalších programech.

3.2 Jednoduché výrazy

Nejjednodušším regulárním výrazem je jakékoliv písmeno. Výsledkem vyhledávání budou všechny jeho výskyty. Užitečnější a používanější je zřetězení několika písmen, čímž dostaneme slovo. Vyhledávání slov je nejběžnější činností, ale neukazuje sílu regulárních výrazů.

3.3 Metaznaky

Regulární výraz se skládá z posloupnosti znaků nebo symbolů odpovídajících skutečným znakům hledaného řetězce, a ze speciálních znaků, které slouží pro popis alternativ, počtu výskytů a přepínačů. Tyto speciální znaky nejsou součástí textu. V tabulce č. 1 uvádím význam jednotlivých metaznaků.

Tabulka č. 1: Význam metaznaků regulárních výrazů,
zdroj: http://www.regularnivyrazy.info/shrnuti-syntaxe.html

Metaznak	Význam
.	odpovídá libovolnému znaku
\	vrací metaznaku původní význam
?	předcházející znak se musí vyskytovat 0x až 1x
*	předcházející znak se musí vyskytovat 0x až neomezeně krát
+	předcházející znak se musí vyskytovat 1x až neomezeně krát
{*n*}	předcházející znak se musí vyskytovat *n* krát
{*m*, *n*}	předcházející znak se musí vyskytovat *m* krát až *n* krát
[ ]	odpovídá jednomu ze znaků v závorkách
[^]	odpovídá jednomu znaku, neuvedenému v závorkách
|	odděluje několik dílčích výrazů
( )	subřetězec na nějž je možno aplikovat kvantifikátor
^	na začátku řetězce či řádku
$	na konci řetězce či řádku
(?=)	kladné tvrzení o následujícím
(?!)	záporné tvrzení o následujícím
(?<=)	kladné tvrzení o předcházejícím
(?<!)	záporné tvrzení o předcházejícím

3.3.1 Zastoupení skupiny znaků

Nejobecnějším zástupným znakem je znak tečka (.). Té vyhoví právě jeden libovolný znak. Například regulárnímu výrazu **k.s** vyhoví všechny řetězce začínající „*k*“, následované jedním znakem a končící „*s*“ (kus, kos, k3s, k s, atd.).

Někdy není žádoucí povolit výskyt všech znaků, proto můžeme použít hranaté závorky, pro deklaraci přípustných znaků. Náš výraz pozměníme tak, aby mu vyhovovaly pouze slova kos a kus. Výsledkem bude regulární výraz **k[ou]s**. Do hranatých závorek můžeme uvést libovolný počet znaků nebo intervalů (např. „*k-p*“). Celý regulární výraz vyhoví, pokud bude znak zastoupen ve hranatých závorkách. Pro negování znaků uvedených v závorkách, stačí na začátku závorky uvést znak stříška (^). Výrazu **k[^ou]s** pak vyhoví řetězce začínající „*k*“, následované jedním znakem (kromě „*o*“ a „*u*“) a končící „*s*“.

3.3.2 Původní význam metaznaků

Pokud budeme chtít vyhledávat tečku, otazník nebo některý jiný metaznak, stačí před tímto znakem uvést zpětné lomítko (\). Tím mu vrátíme jeho původní význam. Uvnitř hranatých závorek mají všechny znaky svůj normální význam, výjimkou je stříška (negace) a pomlčka (interval). Původní význam stříšce vrátíme jejím uvedením kdekoliv jinde než na začátku závorky. U pomlčky stačí neuvést jednu z hranic intervalu (zapsat pomlčku na začátek nebo konec závorky).

3.3.3 Opakování výrazu

Při vyhledávání řetězců s různou délkou můžeme využít opakování výrazu. Základní konstrukcí pro opakování je hvězdička (*), která znamená, že výraz před ní se může libovolně mnohokrát opakovat. Libovolně znamená i žádné opakování, tedy takový výraz se nemusí v textu vůbec vyskytovat. Striktněji se chová znak plus (+). Při jeho použití se musí předcházející výraz vyskytovat alespoň jednou. Otazník (?) označuje nepovinný výskyt předcházejícího výrazu, může se tedy vyskytovat jednou nebo vůbec. Další konstrukce označuje přesný počet opakování, uvádí se do složených závorek {*n*}, kde *n* značí požadovaný počet opakování. Tato direktiva může být rozšířena na interval možných počtů opakování {*min*, *max*} od počtu *min* do *max*.

Nejběžnějším výrazem pro opakování je kombinace tečky a hvězdičky (.*). Je nutné si pamatovat důležité skutečnosti:

- do libovolného počtu opakování se počítá i nula,
- opakování se týká regulárního výrazu, nikoli řetězce, který je s ním porovnáván,
- opakování je hladové - snaží se pojmout co nejvíce znaků.

Nulový počet opakování má za následek, že regulárnímu výrazu vyhoví i prázdné řetězce, proto je nutné takový výraz doplnit o povinnou část, před nebo za výrazem.

3.4 Princip srovnávání

Srovnávání znamená vyhledávání, zda řetězec odpovídá regulárnímu výrazu či ne. Při samotném vyhledávání není třeba znát princip srovnávání, ale je to nutné zejména při nahrazování textu.

Regulární výraz začíná prohledávat od začátku řetězce. Po nalezení první vyhovující části výrazu se snaží přiřadit co nejdelší část zkoumaného textu, teprve poté pokračuje srovnáváním dalších částí. Pokud nevyhoví další část výrazu, vrátí se zpět a pokusí se přiřadit řetězec o jeden znak kratší. Pokud se řetězec zkrátí na minimum a přesto nevyhovuje, posune se k dalšímu znaku a začne znovu.

3.5 Nenasytnost opakování

Je třeba si dávat pozor na vlastnost opakování regulárních výrazů, která má za následek pohlcení co největšího počtu znaků ve vyhledávaném textu. Díky tomu může dojít k pohlcení části textu, se kterou jsme nepočítali. Nejčastější chybou začínajících uživatelů je přílišná obecnost výrazu, určeného k opakování. Například pro vyhledání textu v uvozovkách použijí výraz: **".*"**. Tento výraz vybere vše mezi prvními a posledními uvozovkami v celém řetězci. Pokud se tedy v daném řetězci vyskytuje více částí s uvozovkami (více než dvoje uvozovky), bude tento výraz vybírat i text mezi nimi. Proto je nutné omezit libovolnost znaku na jakýkoliv jiný kromě ukončovacího znaku, v tomto případě uvozovek. Správný výraz, kterému vyhovuje pouze text v uvozovkách, bude vypadat takto: **"[^"]*"**.

3.6 Paměť regulárních výrazů

Jednou z nejsilnějších částí regulárních výrazů je jejich paměť. Regulární výraz si dokáže zapamatovat řetězec, vyhovující části výrazu, a později jej použít. Což je velmi výhodné, zejména při nahrazování.

Jednotlivé části výrazu, které chceme uložit do paměti, uzavřeme do závorek. Při zpětném použití stačí zapsat zpětné lomítko a číslo (\číslo), které značí o kolikátou část v paměti se jedná. Protože závorky mohou být i vnořené, rozhoduje o pozici otevírací závorka.

3.6.1 Použití při vyhledávání

Použití paměti regulárních výrazů při vyhledávání je docela vzácné. K využití může dojít, pokud část hledaného řetězce je libovolná, ale některá další část se s ní má shodovat. Při vyhledávání v HTML kódu můžeme paměť použít pro doplnění ukončovacího *tagu* podle otevíracího, který je vybrán z více možností. Např. výraz **<(b|i|u)[^>]*>[^<]*</\1>**, vyhledá všechen text, zvýrazněný pomocí *tagů* <b>, <i> a <u> (tučné písmo, kurzíva a podtržení). Pro doplnění uzavíracího *tagu* je použita paměť regulárních výrazů, která si zapamatuje otevírací *tag*.

3.6.2 Použití při nahrazování

Daleko častější použití paměti regulárních výrazů je při nahrazování. Díky ní můžeme zachovat některá vstupní data, nebo je uspořádat do jiného tvaru. Ve složitých případech můžeme daný řetězec vyhledat jen pomocí uvedení části řetězce, kterou vlastně nechceme vybrat. Pomocí paměti můžeme danou část řetězce vrátit zpět do původního textu. Příkladem může být vyhledání všech *atributů width*, kromě těch obsažených v *tagu* <*img*>. Toho docílíme regulárním výrazem:

(<(?!img)[^>]{1,10}[^w>]*) (width)=["']?[^"']*["']?.

Díky zpětné referenci (paměti regulárních výrazů), můžeme při náhradě vložit zpět první část řetězce, která obsahuje počáteční *tag* a případné další atributy. Protože se jedná o první část výrazu, bude mít nahrazovací výraz jednoduchou podobu: **\1**.

3.7 Modifikátory

Modifikátory jsou speciální značky, které již nejsou součástí regulárního výrazu, ale určují, jak se má daný regulární výraz chovat v rámci celého textu. V PHP se odděluje začátek a konec regulárního výrazu speciálním znakem (běžně se používá např. @), který není obsažen v samotném regulárním výrazu. Za tento koncový znak se uvádějí jednotlivé modifikátory. V tabulce č. 2 uvádím kompletní přehled modifikátorů a jejich význam. Kompletní výraz s modifikátory pak vypadá např. takto: **@"[^"]*"@si**

Tabulka č. 2: Význam modifikátorů,
zdroj: http://www.regularnivyrazy.info/shrnuti-syntaxe.html

Modifikátor	Název	Význam
i	ignore case	nerozlišování malých/velkých písmen
s	single line	`.` odpovídá i znaku `\n` (nový řádek)
m	multiple lines	`^/$` odpovídá i začátku/konci každého řádku
x	extended	bílé znaky a komentáře jsou ignorovány
g	global match	hledány všechny části řetězce, odpovídající RE

3.8 Speciality Perlu

Programovací jazyk *Perl* a další na něm založené jazyky (např. PHP), nabízejí uživateli řadu konstrukcí, které usnadňují zápis regulárních výrazů. Jedná se o nejběžněji používané kategorie znaků, které se jinak zapisují zbytečně složitě. V tabulce č. 3 uvádím jejich seznam s odpovídajícím původním regulárním výrazem.

Tabulka č. 3: Speciální skupiny znaků,
zdroj: http://www.kit.vslib.cz/~satrapa/docs/regvyr/cast7.html

Zápis	Význam	Odpovídající regulární výraz
\d	číslice	[0-9]
\D	nečíslice	[^0-9]
\w	alfanumerický znak	[a-zA-Z_0-9]
\W	nealfanumerický znak	[^a-zA-Z_0-9]
\s	prázdný znak	[\ \t\n\r]
\S	neprázdný znak	[^\ \t\n\r]

3.9 Vyhlížení

Vyhlížení (anglicky look ahead) je nástroj regulárních výrazů, který zjistí, zda předchozí nebo následující část vyhovuje části regulárního výrazu, ale nezařadí tento výběr do výsledného řetězce. Toho se dá využít při vyhledávání podle charakteristické části textu, kterou ale nechceme zařadit do výběru. Vyhlížení není nezbytné a vždy se může nahradit zpětnou referencí (obráceně tomu tak není). Existují hned čtyři druhy vyhlížení:

- kladné tvrzení o následujícím **(?=výraz)**,
- záporné tvrzení o následujícím **(?!výraz)**,
- kladné tvrzení o předchozím **(?<=výraz)**,
- záporné tvrzení o předchozím **(?!=výraz)**.

Vyhlížení musí být vždy použito v kombinaci s dalším regulárním výrazem, aby bylo jasné od jaké části textu se má vyhlížet.

4 VALIDITA DOKUMENTŮ

4.1 Co je validita

Jazyk HTML, jako každý jazyk, má svá pravidla. Tato pravidla zápisu určuje konsorcium W3C. Validní dokument je takový, který dodržuje popsané doporučení zápisu HTML kódu. Samotná validita není nutnou ani postačující podmínkou pro dobře napsané internetové stránky. Určuje pouze zápis kódu, kterým by se měli tvůrci webových stránek řídit.

4.2 Validátor

Validátor je nástroj, sloužící ke kontrole kódů podle popsaných pravidel. Nejběžnější je online způsob pomocí webové stránky *http://validator.w3.org/*. Tento validátor je produktem přímo společnosti W3C. Existují i jiné online validátory. Další možností jsou offline validátory. Např. programy s integrovaným *HTML Tidy* (nástroj pro automatickou opravu kódu).

4.3 Výhody validního kódu

4.3.1 Správné zobrazení

Existuje celá řada internetových prohlížečů, ale žádný se přesně neřídí popsanými standardy. Různé prohlížeče mohou interpretovat nevalidní stránky rozdílně. Pokud jsou kódy validní, máme vyšší šanci, že se budou ve všech prohlížečích zobrazovat správně. Validní dokumenty se také lépe zobrazují při vypnutých kaskádových stylech nebo v přenosných zařízeních, jako jsou např. PDA, mobilní telefony atd. V neposlední řadě je validní kód lépe čitelný pro roboty vyhledávačů. Proto se budou validní webové stránky zobrazovat při vyhledávání na předních pozicích.

4.3.2 Kontrola chyb

Při psaní kódu dokumentu se může stát, že zapomeneme uzavřít nějaký *tag*. Toho si nemusíme všimnout hned, ale tato chyba může mít vliv na zobrazování dalších částí dokumentů. Validátor takovou chybu snadno odhalí a upozorní na ní.

4.3.3 Další práce s kódem

Běžnému uživateli, který si přeje pouze přečíst obsah internetových stránek, nebude na „nějaké“ validitě vůbec záležet. Pokud chce s kódem pracovat zkušenější uživatel (nebo Vy sami), je dodržování standardů vhodné. V budoucnu si tím uživatelé ušetří spoustu práce. Pokud se tvorbě webu věnují na profesionální úrovni, je validní kód jejich vizitkou.

5 NÁVRH VHODNÝCH REGULÁRNÍCH VÝRAZŮ

5.1 Postup při tvorbě

Pro tvorbu vhodných regulárních výrazů lze použít přímo programovací jazyk, který je podporuje. To je zdlouhavé a nepohodlné. Dále je možné použít některý ze standardních programů. Při tvorbě aplikace jsem zjistil, že regulární výrazy podporuje např. *Total Commander*, *Notepad++*, ad. Ani tyto aplikace nejsou vhodnými prostředky pro návrh a ladění regulárních výrazů. Proto k tomuto účelu existuje celá řada specializovaných programů. Jedním z nich je program *The Regex Coach*, který jsem při tvorbě aplikace využíval.

5.2 The Regex Coach

Tento program nabízí jednoduché prostředí pro vytváření regulárních výrazů. Do horní části okna se zapisuje regulární výraz. Ve spodní části může být vložen libovolný text. V tomto textu program vyhledává a graficky zvýrazňuje vyhovující části, což je vidět na obrázku č. 1. Dále nabízí použití modifikátorů, zpětných referencí nebo krokování postupu srovnávání regulárního výrazu s textem.

Obrázek č. 1: Program The Regex Coach

Program *The Regex Coach* byl dobrým pomocníkem pro návrh konkrétních výrazů. Posloužil také při jejich odlaďování. Dále usnadnil hlubší pochopení fungování regulárních výrazů. Program je možné pro nekomerční použití používat zdarma. Jeho nejnovější verze je k dispozici na adrese: *http://weitz.de/regex-coach/#install*.

5.3 Úrovně čištění

Úrovně čištění představují základní volbu čištění kódu. Určují postup nahrazování vhodných řetězců v textu. Pro vytvářenou aplikaci byli navrženy čtyři základní úrovně čištění:

- standardní opravy zápisu elementů,
- převedení dokumentu na text,
- zachování pouze vybraných *tagů*,
- zachování vybraných *atributů tagů*,
- přednastavená úroveň čištění,
- možnost volby kategorií uživatelem.

5.4 Kategorie regulárních výrazů

Nejdříve je nutné určit, jaké části kódu mají být odstraňovány. Ty je pak vhodné rozdělit do několika kategorií, aby měl uživatel možnost nastavovat úroveň čištění podrobněji.

5.4.1 CSS (kaskádové styly)

Zruší veškeré formátování pomocí kaskádových stylů, včetně pozicování na stránce. Vše mezi *tagy* `<style>` a `</style>`, atributy `style`, `class`, `id` a připojení externího souboru pomocí `<link ...>`.

5.4.2 Div tagy, které mají ID

Odstraňuje *tagy* `<div>` a `</div>` včetně obsahu, pokud mají zadané `id` a neobsahují další *tag* `div`. Jedná se většinou o reklamy nebo nezobrazovaný obsah, ale někdy může tato volba vymazat i část textu, záleží jak je dokument zapsán.

5.4.3 Java Script

Odstraní skriptování na straně klienta, může se jednat i o různá menu atd. Vše mezi *tagy* `<script>` a `</script>`. Dále jsou odstraněny veškeré události, které *Javascript* aktivují.

5.4.4 Komentáře (poznámky)

Vymaže komentáře tj. kódy, které se na výsledné stránce nezobrazují. Vše mezi *tagy* `<!--` a `-->`, `/*` a `*/` a od `//` do konce řádku.

5.4.5 HTML atributy tagů

Tato volba zruší atributy *tagů* upravující vzhled. Místo nich se používají kaskádové styly. Odstraní atributy `face`, `background`, `text`, `align`, `valign`, `scroll`, `bgcolor`, `border`, `bordercolor`, `bgproperties`, `bordercolorlight`, `bordercolordark`, `leftmargin`, `rightmargin`, `topmargin`, `bottommargin`, `marginwidth`, `marginheight`, `alink`, `vlink`, `link`, `cellspacing`, `cellpadding`. Dále odstraní prázdné *tagy* `<font>` a `</font>`. Speciálně jsou odstraněny všechny atributy `width` a `height` ze všech elementů kromě těch, které jsou uvnitř *tagu* `img`.

5.4.6 Prázdné tagy

Touto volbou jsou smazány všechny počáteční a ukončovací *tagy*, pokud mezi nimi není žádný obsah (např. `<p></p>`). Pokud se Vám v kódu kříží *tagy*, nemusí tato volba správně fungovat.

5.4.7 Prázdné řádky

Pokud se v původním zdrojovém kódu nacházejí prázdné řádky, tedy dva a více řádkových zlomů, jsou tyto nahrazeny jedním. Tato volba je univerzálně navržena tak, aby nalezla různě definované řádkové zlomy, které jsou používány na různých platformách.

5.5 Popis použitých regulárních výrazů

V HTML kódu je vhodné vyhledávat podle některé z jeho charakteristických částí. Přímo se nabízejí špičaté závorky, do kterých se *tagy* uzavírají. V některých případech je potřeba vyhledávat podle složitějších konstrukcí.

5.5.1 Jednoduchý výběr tagů

Nejjednodušší regulární výraz může vypadat takto: **<[^>]*>**. Nejprve se nalezne otevírací špičatá závorka. Části výrazu v hranatých závorkách vyhoví díky negaci jakýkoliv znak, kromě pravé špičaté závorky. Hvězdička, zapsaná za ním, určuje, že může být tento znak libovolněkrát zopakován. Vyhoví tedy všechny znaky následující za otevírací špičatou závorkou až po uzavírací špičatou závorku. Ta je na konci výrazu doplněna. Výsledkem výběru jsou všechny otevírací i uzavírací *tagy* včetně obsažených atributů. Tento konkrétní regulární výraz je v aplikaci použit při převodu dokumentu na text. Pokud za otevírací špičatou závorku zadáte název *tagu*, omezíte výběr jen na určené *tagy*.

5.5.2 Výběr tagů včetně obsahu

Vybírat všechny *tagy* včetně obsahu by nemělo žádný smysl, protože bychom vybraly celý kód dokumentu. Předchozí výraz tedy upravíme pro vyhledávání pouze konkrétního *tagu* a doplníme další část pro výběr obsahu a ukončovací značky. Takovým použitým výrazem je např. tento: **<script[^>]*>.*?</script>**. První část vychází z předchozího výrazu, je pouze omezena na *tagy* `<script>`. Ty mohou obsahovat libovolné atributy. Následují znaky tečka a hvězdička, které dohromady označují libovolný znak, libovolněkrát zopakovaný. Následující otazník je speciální konstrukcí, která ruší nenasytnost hvězdičky. Bez něho by vyhověl řetězec až po poslední ukončovací *tag* `</script>`. Otazník zaručí výběr pouze po nejbližší ukončovací *tag*. Je tedy vybírán každý skript zvlášť a nemůže dojít k nechtěnému vymazání jiného obsahu.

5.5.3 Podmíněný výběr atributů

Vyhledávat samotné atributy je obtížnější, protože se v jejich blízkosti nevyskytují nějaké charakteristické značky. Situace je ještě složitější pokud chceme vybrat konkrétní atributy ze všech *tagů*, kromě jednoho vybraného. Toto je použito při vyhledávání všech atributů `width` a `height`, kromě těch, vyskytujících se v *tagu* `<img>`. Výsledný výraz: **(<(?!img)[^>]{1,10}[^w>]*) (width)=["']?[^"']*["']?**. Vychází se od počáteční špičaté závorky, za kterou nesmí následovat řetězec „img“. Protože je použito vyhlížení, musí být doplněn název samotného *tagu*. To je řetězec o jednom až deseti znacích, neobsahující ukončovací špičatou závorku nebo mezeru. Dále mohou následovat libovolné znaky kromě počátečního písmene hledaného atributu nebo ukončovací špičaté závorky. Tato první část regulárního výrazu je později vrácena pomocí zpětné reference: **\1**. Za mezerou následuje název samotného atributu. Jeho hodnota může být uvedena v uvozovkách, apostrofech nebo přímo. To zajišťují hranaté závorky s otazníky. Výsledkem tohoto výrazu je vybrání pouze atributu `width` s jeho hodnotou.

5.6 Výběr PHP funkcí

V PHP existuje celá řada funkcí pro práci s regulárními výrazy. Jednoznačné bylo zaměření na skupinu funkcí *preg*, které pracují s *Perl* kompatibilními výrazy. To umožňuje použití konstrukcí jako jsou speciální skupiny znaků nebo otazník, uvedený za kvalifikátorem opakování. Ten zde ruší nenasytnost opakování regulárního výrazu. Pro složitější část aplikace, opravující atributy HTML elementů, byly použity funkce *preg_split* a *preg_grep*. Funkce *preg_split* rozděluje text dle regulárního výrazu a jeho části uloží do pole. *Preg_grep* zachová v poli jen prvky, které splňují regulární výraz. Ostatním druhům oprav nejlépe vyhověla funkce *preg_replace*. Tato funkce má tři parametry. Prvním je výraz, který se má vyhledávat. Druhý parametr je řetězec pro náhradu. Posledním parametrem je proměnná obsahující text, který se má prohledávat. Použití funkce může vypadat takto:

```
for ($i=0; $i<4; $i++)
  $text = PReg_Replace($this->styl[$i],"", $text);
```

V cyklu je funkce postupně volána s uloženými regulárními výrazy. Druhým parametrem je zde prázdný řetězec. Nalezené výrazy budou tedy pouze vymazány. Výsledek je uložen zpět do původní proměnné, ve které se vyhledávalo.

6 ARCHITEKTURA APLIKACE

6.1 Výchozí podmínky

Ze zadání bakalářské práce vycházejí technologie, které mají být pro tvorbu aplikace použity. Jsou jimi regulární výrazy, skriptovací jazyk PHP. Je tedy jasné, že se bude jednat o webovou aplikaci. Pro tvorbu rozhraní aplikace jsem zvolil technologie HTML a CSS. Okrajově jsem použil i technologii JavaScript.

6.2 Požadavky na aplikaci

Prvním krokem při tvorbě jakékoliv aplikace by mělo vždy být získání informací o tom, jaké vlastnosti by měla aplikace mít. Tyto informace jsou důležité pro návrh aplikace a vymezení jejích schopností. Kromě daných podmínek je tedy nutné zjistit další požadavky. Požadavky na aplikaci:

- jednoduché rozhraní,
- jednoduché používání i pro méně zkušené uživatele,
- funkčnost aplikace ve všech internetových prohlížečích,
- více úrovní čištění HTML kódu,
- správnost oprav (důraz na validaci),
- případné hlášení chyb,
- snadné rozšíření aplikace.

6.3 Metoda programování

Takto jednoduchou aplikaci by bylo možné naprogramovat za pomoci strukturovaného programování. Pro implementaci jsem ale použil objektově orientované programování, a to hned z několika důvodů. Objekt pro opravy HTML kódu je jeden modul, který lze snadno implementovat v rámci jiné aplikace. Nebo současnou aplikaci rozšířit o další modul. Dalším důvodem je přehlednost kódu. V neposlední řadě byla důvodem i možnost vyzkoušet si OOP v PHP.

6.4 Rozhraní

Rozhraní aplikace vychází z běžných zvyklostí vzhledu internetových stránek. Důraz byl kladen především na jednoduchost aplikace. To se týká jak vzhledu, tak jejího používání. Po grafické stránce tedy neobsahuje žádné zbytečné prvky, které jen odpoutávají pozornost. Grafické rozhraní obsahuje:

- jednoduché logo, aby si uživatelé aplikaci lépe zapamatovali,
- navigaci nutnou pro procházení jednotlivých stránek aplikace,
- hlavní část s obsahem stránek,
- patu stránky s informacemi o aplikaci a tvůrci aplikace.

Hlavní stránka aplikace obsahuje jednoduchý formulář, ve kterém uživatel zvolí cestu k souboru a požadovanou úroveň čištění. Další stránky mají pouze informativní charakter. Jsou zde základní informace o aplikaci a návod k jejímu používání.

6.5 Adresářová struktura

Jednotlivé části aplikace jsou rozděleny do adresářové struktury. To zaručuje vyšší přehlednost a modularitu celé aplikace. Strukturu adresářů názorně zobrazuje obrázek č. 2.

Obrázek č. 2: Adresářová struktura

Přímo v hlavním adresáři je uložen soubor *index.php*, který je standardně jako první načítán. Rozděluje stránku na jednotlivé části, do kterých je načítán obsah. Jsou zde načteny používané třídy a menu. To je uloženo také v hlavním adresáři v souboru *menu.php*.

6.5.1 Classes

Adresář *classes* je určen pro uložení tříd používaných v aplikaci. Nyní obsahuje pouze dva soubory s třídami, ale při rozšíření aplikace zde mohou být další. Každá třída je uložena ve vlastním souboru s označením, že se jedná o třídu. Najdeme zde tedy soubor *c.opravy.php*, zapouzdřující metody určené pro opravy HTML kódu. Druhý soubor s názvem *c.soubory.php* slouží pro práci se soubory.

6.5.2 CSS

V adresáři CSS jsou uloženy kaskádové styly pro úpravu vzhledu aplikace. Jsou vytvořeny pouze dva základní styly. První, v souboru *screen.php*, slouží k určení vzhledu na obrazovce počítače. Druhý, s názvem *print.php*, je určen pro tisk stránek. Dále by zde mohly být uloženy styly pro další zařízení, jako jsou mobilní telefony, PDA, atd.

6.5.3 Img

V adresáři *img* jsou uloženy veškeré obrázky použité v aplikaci. Je zde logo aplikace a další obrázky doprovázející text aplikace. Při vyšším počtu rastrových obrázků bych doporučil ještě jejich další rozdělení.

6.5.4 Include

Adresář *include* obsahuje soubory s hlavním textem aplikace, které jsou zobrazovány podle požadavků uživatele. Jako první je zobrazován soubor *index.php*. Ten obsahuje formulář pro zvolení souboru k opravě a volby nastavení úrovně čištění. Po potvrzení volby je načtena stránka *oprav.php*. Ta pracuje s připravenými třídami a volá jejich metody dle zaslaných voleb. Na vyžádání si může uživatel nechat zobrazit stránku *o-aplikaci.php* nebo *navod.php*. V těch je uložen jen doprovodný text aplikace, odpovídající názvům souborů.

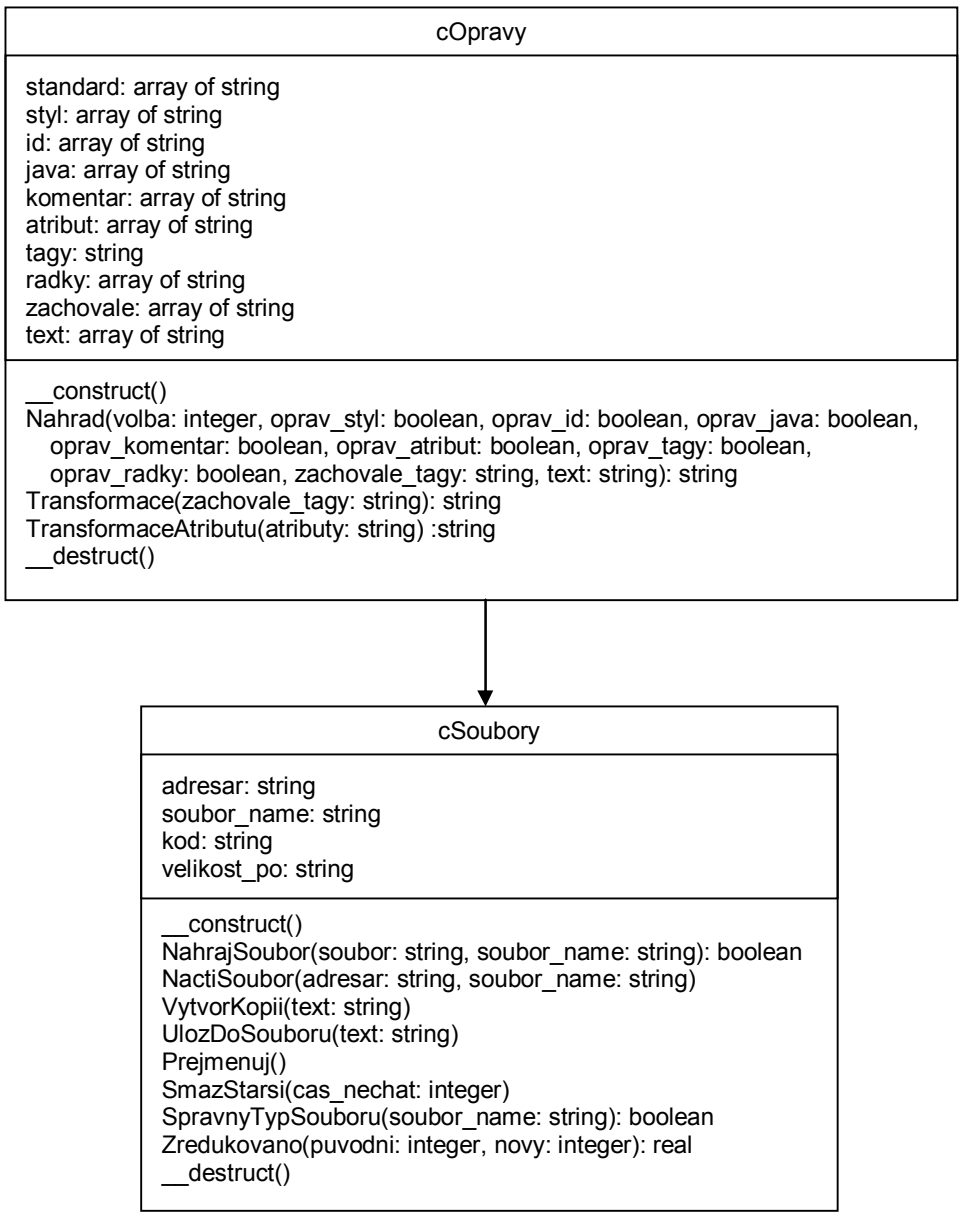
6.5.5 Temp

Do adresáře temp jsou ukládány dočasné soubory, které nahrávají na server uživatelé. Ty jsou opraveny a ponechány zde k dispozici. Soubory jsou strukturovány do podadresářů a po určité době jsou smazány. Tyto procesy popíši podrobněji přímo na daných kódech programu.

7 POPIS KÓDU

7.1 Diagram tříd

Aplikace obsahuje dvě třídy. Třída *cOpravy* slouží k opravám samotného kódu. Pro vytváření souborů, načítání jejich obsahu a zapisování do nich využívá třídu *cSoubory*. Na obrázku č. 3 je znázorněna vazba mezi použitými třídami.



Obrázek č. 3: Diagram tříd

7.2 Třída cSoubory

Třída ve svých atributech uchovává informace o souboru, se kterým pracuje. Má vytvořené metody, pomocí kterých se soubory pracuje. Některé metody nyní popíši podrobněji.

7.2.1 Metoda NahrajSoubor

Metoda převezme jméno souboru z cesty zadané ve formuláři a uloží si ho pro další použití. Vytvoří adresář se jménem získaným z funkce *time* (čas v sekundách od roku 1970). Do tohoto adresáře je nahrán soubor s původním názvem. Jedinečné jméno adresáře zaručuje minimální riziko konfliktů (nahrání více souborů se stejným jménem).

```
$this->soubor_name = $soubor_name;
if (is_uploaded_file($soubor))
{
  $this->adresar = time();
  MkDir($cesta="./temp/$this->adresar
  if (move_uploaded_file ($soubor, $cesta.$soubor_name))
    return true;
  else
    return false;
}
```

7.2.2 Metoda SmazStarsi

Tato metoda prochází adresář s dočasnými soubory (*temp*) a čte názvy adresářů. Porovná, zda je název nižší než čas, po který mají být soubory uchovávány. Pokud je starší, otevře ho a načítá postupně soubory v něm uložené, které maže. Po smazání celého jeho obsahu, smaže i samotný adresář. Metoda je volána při každém načtení hlavní stránky aplikace. Tím je zaručeno, že se na serveru nebudou hromadit staré soubory.

```
if ($cesta = OpenDir('./temp/'))
{
  while ($adresar = ReadDir($cesta))
  {
    if ($adresar != "." && $adresar != ".." && $adresar < Time()-
$cas_nechat)
    {
      if ($cesta2 = OpenDir('./temp/'.$adresar.'/'))
      {
        while ($soubor = ReadDir($cesta2))
        {
          if ($soubor != "." && $soubor != "..")
            UnLink('./temp/'.$adresar.'/'.$soubor);
        }
        CloseDir($cesta2);
        RmDir('./temp/'.$adresar);
      }
    }
  }
  CloseDir($cesta);
}
```

7.2.3 Ostatní metody

Zbylé metody třídy *cSoubory* jsou jednoduché. Metoda *NactiSoubor* pouze načte celý obsah souboru do jedné proměnné. *Vytvorkopii* vytvoří před opravami kopii nahraného souboru pro pozdější porovnání a validaci. Metoda *UlozDoSouboru* zapíše opravený kód zpět do původního souboru. Metoda *Prejmenuj* je volána jen při volbě převodu dokumentu na text a slouží k přejmenování přípony souboru na textový soubor. *SpravnyTypSouboru* je metoda, zajišťující možnost nahrání na server jen povolených typů souborů. Poslední metoda, s názvem *Zredukovano*, spočítá o kolik byl kód dokumentu zmenšen. Toho docílí z velikosti souboru před opravou a po ní.

7.3 Třída cOpravy

Třída *cOpravy* má ve svých atributech uloženy potřebné regulární výrazy. Ty jsou podle zvolené úrovně aplikovány na HTML kód. Atributy jsou rozděleny do několika kategorií dle zvolených úrovní čištění kódu. Metodě *Nahrad* jsou pomocí parametrů předány volby čištění. Ta použije jen potřebné regulární výrazy. Metoda *Transformace* je použita u volby *Zachovat tagy*. Zde přeměňuje řetězec *tagů* pro zachování do formy vhodné pro regulární výrazy.

7.4 Ostatní skripty

7.4.1 Skript zajišťující vkládání obsahu správné stránky

Tento skript nejdříve zjistí, zda byla v adrese předána hodnota určující název souboru, který se má vložit. Dále zjistí, zda soubor s takovým názvem existuje ve složce */include*. Pokud jsou obě podmínky splněny, vloží jeho obsah a zobrazí ho, pokud nejsou, vloží a zobrazí obsah hlavního souboru */include/index.php*.

```
if (isset($_GET['stranka']) and
File_Exists('./include/'.$_GET['stranka'].'.php'))
{
      include('./include/'.$_GET['stranka'].'.php');
}
else
{
      include('./include/index.php');
}
```

Odkaz lze pak zapsat takto (na začátku cesty je možné vynechat zápis *index.php*):

```
<a href="index.php?stranka=navod">Návod</a>
```

7.4.2 JavaScript

JavaScript je použit u vstupního formuláře k zamezení nebo povolení částí formuláře. Událostí je kliknutí myší na některou z hlavních voleb čištění kódu. Tou je aktivována funkce pro (ne)zpřístupnění dalších voleb a textového pole. To závisí na dané volbě. Skriptování na straně klienta bylo zvoleno, aby se zpřehlednila práce s formulářem a nebylo nutné znovu načítat stránku.

8 ZÁVĚR

Při vytváření aplikace sloužící k opravám HTML kódu jsem rozšířil své znalosti a schopnosti v mnoha oblastech. Nejpřínosnější byla však část týkající se regulárních výrazů. Po jejich hlubším nastudování jsem poznal jejich sílu a zároveň jednoduchost. Nyní je používám při běžných činnostech, jako je např. hromadné přejmenování souborů. Návrh vhodných regulárních výrazů byl sice nejsložitější částí, ale zřejmě právě proto byl určitou výzvou. Při tvorbě vlastní aplikace jsem čerpal ze svých dřívějších zkušeností. Díky této práci jsem je obohatil o objektový přístup v PHP. Dále jsem získal lepší přehled o správném zápisu a validitě kódu.

Domnívám se, že jsem splnil požadavky kladené na tuto práci. V průběhu tvorby aplikace jsem odladil jednotlivé regulární výrazy a aplikace je proto připravena k běžnému používání. Při jejím vyšším používání mohou být nalezeny další možnosti pro rozšíření. Jednou z nabízených možností je volba zachovávaných *tagů* a upřesněním dle atributů. Tato volba není jednoduchá na implementaci a vyžaduje zvážení správného přístupu.

9 SEZNAM POUŽITÝCH ZDROJŮ

[1] *Cascading Style Sheets : Wikipedie, otevřená encyklopedie* [online]. 2008 [cit. 2008-05-05]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Cascading_Style_Sheets>.

[2] DRUSKA, Peter. *CSS a XHTML : tvorba dokonalých webových stránek krok za krokem*. [s.l.] : Grada Publishing, 2006. 200 s. ISBN 80-2471-382-9.

[3] *Extensible HyperText Markup Language : Wikipedie, otevřená encyklopedie* [online]. 2008 [cit. 2008-05-05]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Extensible_HyperText_Markup_Language>.

[4] *HyperText Markup Language : Wikipedie, otevřená encyklopedie* [online]. 2008 [cit. 2008-05-05]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/HyperText_Markup_Language>.

[5] KOSEK, Jiří. *PHP : tvorba interaktivních internetových aplikací*. [s.l.] : Grada Publishing, 1999. 490 s. ISBN 80-7169-373-1.

[6] PECKA, Miroslav. *Regulární výrazy* [online]. c2005-2008 [cit. 2008-05-05]. Dostupný z WWW: <http://www.regularnivyrazy.info>.

[7] *PHP : Wikipedie, otevřená encyklopedie* [online]. 2008 [cit. 2008-05-05]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Php>.

[8] *Proč psát validní kód* [online]. [2006] [cit. 2008-05-05]. Dostupný z WWW: <http://www.hellish.cz/weblog/proc_psat_validni_kod>.

[9] SATRAPA, Pavel. *Regulární výrazy* [online]. c2000 [cit. 2008-05-05]. Dostupný z WWW: <http://www.kit.vslib.cz/~satrapa/docs/regvyr/all.html>.

[10] *Webová aplikace : Wikipedie, otevřená encyklopedie* [online]. 2008 [cit. 2008-05-05]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Webová_aplikace>.

[11] *World Wide Web Consortium : Wikipedie, otevřená encyklopedie* [online]. 2008 [cit. 2008-05-05]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/World_Wide_Web_Consortium>.