

**jQuery - prototyp pluginu pro**

**rezervační systém**

[***KIV/BPINI***](https://portal.zcu.cz/wps/myportal/!ut/p/c5/hc7LDoIwFATQT-rQ0ofLBkmpVhCMiGwIC2MwAi6M328JiQsSde7yZDKX1MTf0L66a_vsxqG9k4rUopFZbth-TZEprEC5EHlcHOHj_SyayOgklA4wknLQlB_oTpeAZX_ap2nvd39yfInG7J8FlEkMKjV3sjABIrbwxQdGzL51scoiFcIw79aqTSBjF0wfpsnYX8ijr9DZW_gGF-KUgw!!/dl3/d3/L0lDU0lKSmdrS0NsRUpDZ3BSQ2dwUkNncFJDU0EvWU9VSUFBSUlJSU1NSUtFRUFBQ0dPR09DR0lCSkZKRkJKTkROREJOTEhMSEJMQW9FQVBQQS80QzFiOVdfTnIwZ0RFU1pJSlJERVNaTUpRaWt5VVNqRkprNGxCSWlSLUEhIS83XzdPUUczUEQyMEdWMDAwMjVWR0g4NDExME01L19VZTluODg4MDAwOTcvaWJtLmludi8yMTYzNDIxODA3OTgvUGxuZW5pUm9rVmFyaWFudHkvMjAxMi9QbG5lbmlaa3JhdGthRmFrdWx0eS9LSVYvUGxuZW5pWmtyYXRrYVByZWRtZXR1L1BSSjU!/#prohlizeniAnchor) ***- Dokumentace***

student: *Petr Kukrál*

studijní číslo: *A10B0618P*

email: kukral@students.zcu.cz

datum: 15. 12. 2012

Obsah

[1 Vytvoření pluginu 1](#_Toc350718057)

[1.1 Vytvoření pluginu a připojení ke knihovně 1](#_Toc350718058)

[1.2 Použití obalové funkce 1](#_Toc350718059)

[1.3 Význam středníku před skriptem 1](#_Toc350718060)

[1.4 Defaultní parametry pluginu 2](#_Toc350718061)

[1.5 Soukromé funkce v pluginu 2](#_Toc350718062)

[2 Konkrétní problémy 2](#_Toc350718063)

[3 Použitá literatura 2](#_Toc350718064)

[4 Příloha 2](#_Toc350718065)

[4.1 Ukázky zdrojových kódů 2](#_Toc350718066)

[4.1.1 Obalová funkce 2](#_Toc350718067)

[4.1.2 Defaultní parametry pluginu 2](#_Toc350718068)

[4.1.3 Přidávání funkcí k objektu jQuery 3](#_Toc350718069)

# Tvorba pluginu jQuery

Touto kapitolou bych chtěl sdělit nejdůležitější problémy, na které jsem při psaní pluginu narazil a popsat, jak jsem je v práci řešil.

## Verzovací systémy

Verzovací systém by měl ulehčit práci na projektu. A to hlavně v případě, že na projektu pracuje více vývojářů. To by mělo být zajištěno pomocí tkz. větví (anglicky branches). Ty by měly rozdělit projekt to více částí, na kterých vývojáři pracují samostatně. Nakonec by měl systém umět tyto větve spojit do jedné a tím sloučit projekt. Má obsahovat správu verzí projektu a nejlépe automaticky projekt verzovat. Při vývoji této práce by měl hlavně soužit pro verzování projektu. Podpora práce pro více vývojářů, bude v tomto projektu zbytečná.

Verzovacích systémů je celá řada. Já bych rád uvedl jen tři, které jsou podle mého názoru nejzákladnější.

Prvním, z těchto tří nejjednodušším, je CVS (Concurrent Version System). Podle zdroje (1), jde o pomyslné rozšíření RCS (Revision Control System). CVS je verzovací systém pro skupinu souborů. Tyto soubory mohou být uloženy v různých adresářích. Adresář zde není brán jako soubor, ale jde o zvláštní typ. Jak uvádí zdroj (2) „V CVS měl adresář speciální postavení, soubory se ukládaly a organizovaly podle toho, v jakém adresáři se nacházely. To znamená, že adresář nejde dost dobře smazat, činí to zbytečné problémy při přesouvání souborů apod. Proto by měl v našem IVS adresář pokud možno co nejvíce vystupovat jako obyčejný soubor, u kterého si pouze poznamenáme, že zároveň obsahuje i nějaké jiné soubory. Měli bychom pak možnost adresáře přesouvat, přejmenovávat, vytvářet a mazat stejně jako všechny ostatní soubory.“ CVS dokáže spravovat a slučovat větve. Provádí to ovšem neefektivním programem *diff3* a zapomíná informace o vedlejší větvi, jak uvádí zdroj (2).

SVN (Subversion) nabízí již více než CVS. Hlavní změnou, jak uvádí zdroj (3), je nakládání s adresářem. SVN totiž nerozlišuje rozdíl mezi adresářem a souborem. Také zvládá větvení projektu. Jak ale budu psát v dalším odstavci, jeho nevýhodou, oproti systému GIT, je rychlost práci s daty.

Jako poslední verzovací systém představím GIT. Tento program je napsán v jazyce C. Jak tvrdí zdroj (4), hlavním cílem při vývoji GITu byla rychlost běhu. V tabulce uvádím porovnání běhu GIT a SVN. Porovnání je pouze přibližné, jak sám autor testů uvádí. GIT umí dobře pracovat s větvemi, dokáže je vytvářet i slučovat.

Pro mojí práci jsem si vybral systém GIT. Pro veřejně přístupné projekty je možné využít systému github, který je zdarma. Celý projekt je pak dohledatelný přímo na webu github.com i se zdrojovými kódy.

Tabulka 1 zdroj (4)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation |  | Git | SVN |  |
| Commit Files (A) | Add, commit and push 113 modified files (2164+, 2259-) | 0.64 | 2.60 | 4x |
| Commit Images (B) | Add, commit and push 1000 1k images | 1.53 | 24.70 | 16x |
| Diff Current | Diff 187 changed files (1664+, 4859-) against last commit | 0.25 | 1.09 | 4x |
| Diff Recent | Diff against 4 commits back (269 changed/3609+,6898-) | 0.25 | 3.99 | 16x |
| Diff Tags | Diff two tags against each other (v1.9.1.0/v1.9.3.0 ) | 1.17 | 83.57 | 71x |
| Log (50) | Log of the last 50 commits (19k of output) | 0.01 | 0.38 | 31x |
| Log (All) | Log of all commits (26,056 commits - 9.4M of output) | 0.52 | 169.20 | 325x |
| Log (File) | Log of the history of a single file (array.c - 483 revs) | 0.60 | 82.84 | 138x |
| Update | Pull of Commit A scenario (113 files changed, 2164+, 2259-) | 0.90 | 2.82 | 3x |
| Blame | Line annotation of a single file (array.c) | 1.91 | 3.04 | 1x |

## Komentáře

V této práci komentáře hrají velkou roli. Práce obsahuje nejen plugin, ale i ukázkové použití. Proto jsem se zde snažil používat různé druhy dokumentačních postupů v závislosti na tom, v kterém jazyce jsem komentář psal.

V práci jsem při psaní Modelu databáze používal komentáře PhpDoc (celým názvem PhpDocomentor) popisované ve zdroji (5). Díky PhpDoc komentářům se dá vygenerovat přehledná dokumentace celého kódu. Dají se zde používat nejen jednoduché texty, ale mohou obsahovat i tagy jako je například seznam. Tato dokumentace se potom dá vygenerovat v různých formátech jako je HTML, PDF či CHM. Další velkou výhodou těchto komentářů je, že se dají propojit s některými vývojovými prostředími (například s vývojovým prostředím Eclipse). Při použití zdokumentovaného kódu pak může vývojové prostředí našeptávat komentáře.

Pro psaní samotného pluginu jsem se snažil co nejvíce přiblížit komentářům Jsdoc popsaným ve zdroji (6). Z takto napsané dokumentace se dá rovněž vygenerovat dokumentace ve formátu HTML.

## Vytvoření pluginu a připojení ke knihovně

Knihovna jQuery nabízí velmi jednoduchý způsob, jak k ní připojovat další funkce. Stačí rozšířit objekt *jQuery.fn*. Tedy například v případě pluginu pro rezervační systém budete ukládat do objektu *jQuery.fn.booking\_system* všechny potřebné vlastnosti pluginu. Ukázku vkládání najdete v příloze 3.1.3.

K vytvořenému pluginu pak přistoupím jednoduše tak, jako by to byla jakákoliv jiná funkce jQuery. V mém případě to bude *jQuery.booking\_system*.

## Použití obalové funkce

Při používání knihovny jQuery je zvykem, jak uvádí (3), využívat k přístupu k objektu jQuery jeho zkrácenou verzi zápisu $. Tento přístup, který jsem použil i ve své práci, má řadu výhod.

Kód je díky tomuto zápisu kratší. Jelikož se javascript stahuje spolu s načtením stránky a teprve potom je spuštěn v prohlížeči, je kladen velký důraz na to, aby byl kód co nejkratší. (4 str. 141) tvrdí, že je tím zajištěna rychlejší odezva stránky mezi tím, když se začne stránka stahovat a mezi tím, než začne plně reagovat.

Kód je čitelnější. Jelikož se objekt jQuery používá v pluginu opravdu často, zkrácený zápis může být v mnoha případech čitelnější.

Protože je ale používání zkráceného zápisu oblíbené, může přinést řadu problémů. Například knihovna, která je přidána ke stránce před touto, může zkrácený zápis $ využívat jako ukazatel na něco jiného než objekt jQuery. Z tohoto důvodu jsem v práci použil obalovou funkci. Ukázku obalové funkce najdete v příloze 3.1.1.

Jde o velmi jednoduché řešení, kdy definuji obalovou funkci, ve které se nachází celý plugin. Její parametr bude $. Přímo za dodefinováním funkce ji zavolám a jako parametr vložím objekt jQuery. Jazyk JavaScript pak vymezí rozsah platnosti této proměnné na celou funkci. V té potom můžu $ volně používat bez vzniku kolize. Další výhodou tohoto postupu je, že definováním obalové funkce vytvořím uzávěr. Ten mi pomůže s možnou kolizí názvů metod tohoto a dalších používaných pluginů.

## Význam středníku před skriptem

Jak tvrdí (3), jazyk JavaScript si dokáže poradit se zapomenutým středníkem na konci řádku. Ovšem v případě minimalizace kódu se konce řádků vypouští a může tak docházet k chybám. Proto je důležité, na začátku pluginu napsat středník. Tím ošetříte případ, kdy programátor, který psal kód přilinkovaný před Vaším na středník zapomněl.

## Defaultní parametry pluginu

Do pluginu by jsme mohli předávat argumenty jednoduše jako parametry funkce. Toto jednoduché řešení ovšem nebude stačit v momentě, chceme-li správně nabídnout uživateli pluginu defautní možnosti.

V tomto případě je dobré, definovat si vlastní defaultní hodnoty a připojit je k pluginu. Pak jednoduše pomocí metody extend porovnáte defaultní nastavení a nastavení od uživatele. Metoda sama zajistí doplňění údajů od uživatele defaultními údaji.

Výhoda tohoto přístupu je jistě čitelnější kód. Pluginy většinou mívají hodně proměnných, které je potřeba nastavit. Tímto přístupem se tedy vyvarujete tomu, kontrolovat všechny proměnné, zda byly nastaveny. Jednoduše kontrolujete jen to, co potřebujete. Použití této konstrukce v mém pluginu najdete v příloze 3.1.2.

## Soukromé funkce v pluginu

Při psaní pluginu jsem potřeboval napsat hodně funkcí, které nebudou viditelné pro uživatele pluginu. Kdybych je připojil k pluginu, mohl by uživatel tyto funkce překrýt. Tento problém jsem vyřešil právě již zmíněnou obalovou funkcí. Uvnitř obalové funkce jsem si definoval funkci, kterou jsem nepřipojil k objektu pluginu. Tím ji mohu uvnitř pluginu využívat, ale zvně pluginu zůstane skrytá.

## Načítání dat ze souboru

Při načítání dat ze souboru je možné použít Javascript a knihovnu jQuery. Slouží k tomu funkce *get* která se volá nad objektem jQuery. V případě mého pluginu jsem použil funkci *file\_get\_contents* která je v jazyce PHP. Rozhodl jsem se tak pro to, že načtení souboru vytvořeného uživatelem může být zdrojem mnoha chyb, např. že daný soubor neexistuje. Proto se mi zdálo lepší, načítat soubor do stránky, kde se dá jednoduše v náhledu zdrojového kódu ověřit, zda se opravdu data načetla a jsou správně.

## Konfigurace pluginu v formátu JSON

## Obecná použitelnost

Při psaní pluginu jsem musel dbát jeho použitelnost v různých projektech. Je to zřejmé na příkladech kina a squashového kurtu, které jsem vytvořil jako ukázky používání pluginu.

Prvnímu, čemu jsem se musel vyhnout, je přímé stylování objektů. V pluginu generuji celý systém přímo do stránky a zde nesmí být uveden žádný přímý styl. To by znemožnilo jeho použitelnost na jakýkoliv jiný případ. Je to kvůli tomu, že přímé stylování má nejvyšší prioritu vytváření vzhledu objektu a nešlo by překrýt. Všechny styly jsem tedy přesunul do vnějšího souboru, kde si je uživatel pluginu může lehce změnit.

Dalším důležitým prvkem u stylů bylo nechat třídy a identifikátory volitelné. Mohlo by totiž dojít k tomu, že by kolidovali s již použitými názvy na stránkách uživatele a ten by musel své stránky nebo plugin ručně předělat. Aby ovšem uživatel nemusel nastavovat všechny třídy ručně při zavedení pluginu do své stránky, ponechal jsem všem třídám a identifikátorům výchozí hodnoty. Uživatel si tak změní jen ty názvy, které mu na stránkách kolidují.

Jelikož plugin potřebuje pro svůj správný běh uvést cestu k obrázkům, které používá (jako je třeba obrázek sedačky v kině), i tato cesta zůstala v pluginu nastavitelná pro všechny obrázky.

Při psaní pluginu jsem dbal na jeho vizuální stránku. Při najetí kurzorem myši nad objekt, či kliknutí na objekt se spouští různé akce, které objekt graficky mění. Například ho zprůhledňují. V případě, že by uživatel chtěl s objektem vykonat jinou akci, má možnost v nastavení zadat jméno funkce, která objekt funkci v pluginu překryje. Proto i toto řadím k obecné použitelnosti, protože uživatel si může napsat funkce nakládající s objektem, jak potřebuje. Jedinou podmínkou pro uživatele je přijímat objekt *events* s parametrem *target*, přes který se dostane k objektu se kterým je manipulováno.

## Výstup pluginu

Výstup z pluginu se přímo směruje do stránky do zadaného identifikátoru. Při rozhodování o struktuře výstupu z pluginu jsem se rozhodoval, jestli použít výstup pomocí tabulky, nebo pomocí divů.

Výstup pomocí tabulky by mi pomohl v tom, že by objekty byly sami řazeny do mřížky bez toho, abych je musel jakkoliv ovlivňovat styly. To by se projevilo obzvláště u generování více elementů, jako jsou sedadla v řadě. Stačilo by totiž k sedadlům přidat patřičné HTML tagy tabulky označující začátek řádku (v případě kina řady sedadel) a dalšího prvku (sedadla). Nevýhoda tohoto přístupu by byla v tom, že nejdou elementy rozmístit nepravidelně. Tedy třeba v případě, že by chtěl uživatel rozmístit elementy do geometrického obrazce, například kruhu, by musel jednotlivé elementy vyhledávat a stylovat každý zvlášť.

Druhá možnost byla ovlivnit rozmístění objektů pomocí divů. Tedy každý div element by obsahoval vlastní objekt (v případě kina sedadlo) a uživatel by mohl objekt přesunout, kam by potřeboval. Nevýhoda tohoto přístupu je zřejmá. Musely by se udělat obecné styly pro všechny divy, nebo stylovat každý div zvlášť.

Zvolil jsem kompromis. V pluginu vytvářím skupiny objektů, ve kterých je jednotlivé objekty nachází. Můžeme si je představit například v případě kina jako řady sedadel, v kulečníkové herně by to mohly být stoly. V této skupině se pak objekty mohou stylovat libovolně. Například v případě kulečníku je můžeme umístit okolo stolu do obdélníku. Tato možnost by se s využitím tabulky řešila velmi špatně.

# Použitá literatura

1. **Baudiš, Petr.** Výlet do říše verzí: CVS. *root.* [Online] 15. duben 2003. [Citace: 11. březen 2013.] http://www.root.cz/clanky/vylet-do-rise-verzi-cvs/.

2. —. Výlet do říše verzí: ideální verzovací systém? *root.* [Online] 3. květen 2004. [Citace: 11. březen 2013.] http://www.root.cz/clanky/vylet-do-rise-verzi-idealni-verzovaci-system/.

3. **Faigl, Jan.** Systémy pro správu verzí. [Online] [Citace: 11. březen 2013.] http://lynx1.felk.cvut.cz/pte/doc/vcs.pdf.

4. **Chacon, Scott.** Small and Fast. *git.* [Online] 29. červen 2009. [Citace: 11. březen 2013.] http://git-scm.com/about/small-and-fast.

5. **Teutsch, Moshe.** Introduction to PhpDoc. *phpmaster.* [Online] 9. leden 2012. [Citace: 11. březen 2013.] http://phpmaster.com/introduction-to-phpdoc/.

6. Project Home. *jsdoc-toolkit.* [Online] [Citace: 11. březen 2013.] https://code.google.com/p/jsdoc-toolkit/.

7. **Resig, John.** *jQuery Kuchařka programátora.* Brno : Computer Press a.s., 2010. ISBN 978-80-251-3152-7.

8. **Odell, Dem.** *JavaScript Průvodce programováním ajaxových aplikací.* Brno : Computer Press a.s., 2010. ISBN 978-80-251-2733-9.

# Příloha

## Ukázky zdrojových kódů

### Obalová funkce

(fuction($) {

… celý kód pluginu

})(jQuery);

### Defaultní parametry pluginu

/\* doplneni nevyplnenych nastaveni o deafultni nastaveni \*/

var opts = $.extend({}, $.fn.booking\_system.defaults, options);

$.fn.booking\_system.defaults = {

/\* formát vstupu \*/

format: "cvs",

… další nastavení

};

### Přidávání funkcí k objektu jQuery

$.fn.booking\_system = function(options){

… zde je vše potřebné pro běh pluginu

}