Optimalizační techniky v JS

aneb Kdo se vleče, neuteče!

Marek Fojtl, Programátor UI, marek.fojtl@firma.seznam.cz

SEZNAM.CZ

Po dnešku budu mít přehled o

- základních výkonnostních problémech v JavaScriptu a jejich řešení
- optimalizacích na úrovni zápisu JS kódu a na úrovni prohlížeče (interpretu)

Výkon obecně ovlivňuje

- architektura PC uživatele
- JavaScript = interpretovaný jazyk
- prohlížeč
- zápis kódu
- použité algoritmy

Co mi to přinese?

- **User eXperience!**
- **šetření systémových prostředků klientských PC**
- **vytváření výkonnostních rezerv pro náročnější výpočty**

Jak testovat a měřit?

- http://jsperf.com
- vlastní měření - new Date().getTime()
  - často nepřesné
- hledat nástroje, pluginy prohlížečů, atd …

Optimalizovat lze na úrovni

- JS kódu
- prohlížeče

- produktu

Optimalizace na úrovni JS kódu

Když začínáme optimalizovat

http://funny-pix.co/lols/funny-fear-pictures/

Když dokončíme optimalizaci

http://www.last.fm/music/Lord+Darth+Vader/+images/26322553

Zaměřte se na

- volání funkcí
- cykly a jiný často se opakující kód
  - práce s polem a objekty
- rendering stránky

Obecný proces volání funkcí

- dle jména najít tělo funkce
- zvýšit scope
- zpracovat parametry a vytvořit z nich lokální proměnné
- vykonat tělo funkce
- snížit scope
- zpracovat návratovou hodnotu

Optimalizace volání funkcí

- funkce nepoužívat => nevolají se
  - snižuje se čitelnost kódu
  - obtížně implementovatelné rekurzivní problémy
  - nedoporučováno!!!
- omezit volání funkcí
  - „hot“ kód nedávat do funkcí
- optimalizovat pro JIT

Procházení pole

```
var DATA=[10, 9, 8, 7, 6, 5, 4, 3, 2, 1];

/* neoptimalizovano */
var min=DATA[0];
for(var i=1; i<DATA.length; i++) {
  if (DATA[i] < min) { min = DATA[i]; }
}
```

VS

```
/* optimalizovano */
var min=DATA[0];
for(var i=1, len=DATA.length; i<len; i++) {
  var data = DATA[i];
  if (data < min) { min = data; }
}
```

Vestavěné metody pole

- Array.prototype.filter, Array.prototype.forEach, Array.prototype.sort
- jsou mnohonásobně pomalejší než totožný kód napsaný v JS

Objekt

- hash table (dvojice klíč – hodnota)
  - pomalé procházení
  - lze cachovat podobně jako prvky pole
  - moderní prohlížeče řeší už na úrovni interpretu
- v JS je vše objekt!!

Rendering stránky zpomaluje

- neustálé překreslování stránky po změně v DOMu
  - nastavení hodnoty atributu style
  - přidání, odebrání, změna elementu
- animace
- použití doplňků ovlivňujících strukturu DOMu

Rendering stránky lze zrychlit

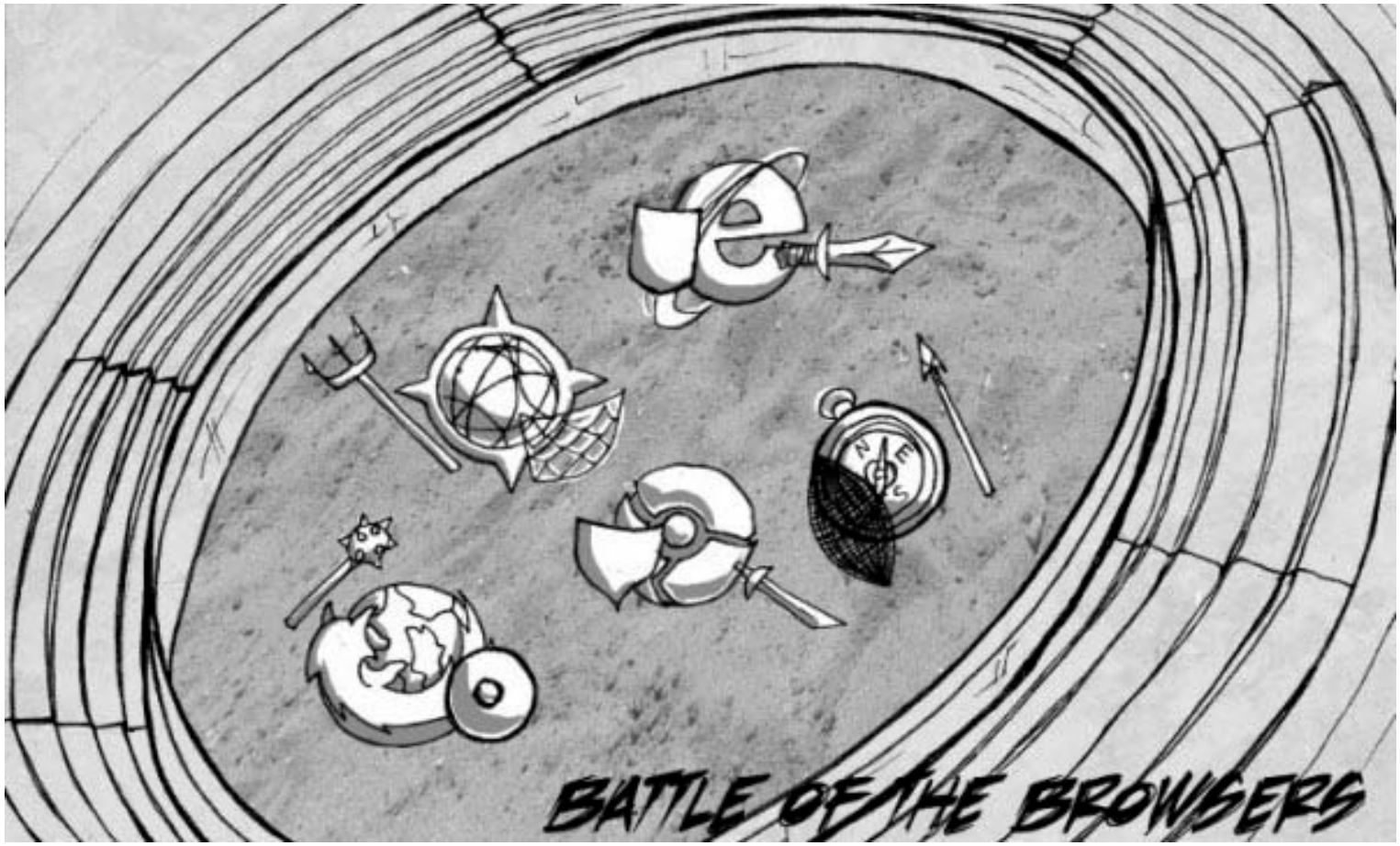
- **vytvářením elementů mimo DOM a připnutím větší části stránky najednou**
  - **document.createDocumentFragment**
- **vytvářením elementů pomocí stringu**
  - **node.innerHTML = “foo“;**
- **vykreslením viditelné části stránky ihned a zbytek pomocí funkcí *setTimeout* nebo *setInterval***

Optimalizace na úrovni prohlížeče

Situace před třemi lety

http://themetapicture.com/when-browsers-have-a-meeting/

Situace dnes



http://www.hongkiat.com/blog/20-awesome-battle-of-the-browsers-artworks/

Inlining

- vkládání těla funkce na místo jejího volání

Just In Time Compilation

- zkratka JIT
- je proces, při kterém dochází k pozorování chování vykonávaných funkcí a v případě splnění určitých podmínek následný překlad do nativního kódu

JIT - Terminologie

- **Hot** – často vykonávaný kód
- **Stable** – fce aspirující na kompilaci, tzv. „hodná“ funkce
- **Bailout** – návrat do interpretovaného režimu

JIT - Jak na „hodnou“ funkci?

- neměnit za běhu datový typ proměnné nebo atributu objektu
- neměnit počty atributů v objektech
- nezaměňovat celá a desetinná čísla
- volat fci vždy se stejným počtem a dat. typy parametrů
- vyhnout se eval
- v try-catch volat celou funkci

JIT - Pozorovací proces

- často se vykonávající kód budeme pozorovat
- pokud se ukáže, splňuje podmínky pro kompilaci, zkompilujeme ho do nativního kódu
- heuristika

JIT - Další postup

- pokud po zkompilování dojde k porušení podmínek, vrátíme funkci zpět do interpretovaného režimu
- v případě „hodných“ funkcí může být provedena i optimalizace v podobě **inliningu** – nesouvisí s JIT

JIT - Podpora v prohlížečích

- Google Chrome
- Firefox 3.5+
- IE 9+
- Opera 11+
- Safari 4+

Ostatní optimalizace

- hidden classes
- Inline caches (neplést si s Inlining)
- loop-invariant code motion
- asm.js
- …

Resumé

Tipy

- zaměřte se cykly a často se opakující události
- používejte profilování
- testujte na méně výkonných strojích!
- optimalizace vs. čitelnost kódu
- zvýšený zájem o optimalizace mějte zejména v mapových aplikacích, hrách ...

...najdu tam, co neznám!

Marek Fojtl, Programátor UI, marek.fojtl@firma.seznam.cz