JavaScript a client-side návrh programu Ptakopět

Vilém Zouhar

May 2019

1 JavaScript

1.1 Úvod

1.1.1 Historie

Jazyk JavaScript začal v roce 1995 ve společnosti Netscape Communications. Původně se jmenoval Mocha a jméno obsahující Java vzniklo až kvůli partnerství se Sun Microsystems (JavaScript jako partnerský jazyk k Javě). Toto spojení dvou firem mělo rozdělit domény jazyků na Java - velké komerční systémy, JavaScript - malé skriptování, spíše pro amatérské tvůrce.

Z časových důvodů ohledně uzavírání smlouvy byla první verze Mocha/JavaScript navrhnuta velmi rychle. Autor chtěl původně udělat něco podobného Scheme (v podstatě LISP), ale bylo mu řečeno, že to musí vypadat jako Java.[1] Z těchto důvod má syntaxi blízkou Javě. Funguje však implementačně principiálně odlišně. K k běhu potřeba JavaScript engine. V té době tedy vznikly dva konkurenční: SpiderMonkey na straně Netscape a JScript na straně Microsoftu. Později jeho kodifikaci přejal ECMA pod názvem ECMAScript.

Postupně se s vývojem jazyka rozkvétaly i standardní funkce v implementacích a napojilo se tak z hlediska programátora více funkcionalit (nyní lze např. získat přístup ke webkameře, či omezenému lokálnímu úložišti) a na webovou Javu se zapomnělo.[2][3]

V roce 2012 byl zveřejněn (do té doby používán pro interní účely Microsoftu) projekt TypeScript, který se snaží napravit spoustu chyb JavaScriptu a soustředí se zejména na škálovatelnost a údržbu, což je v čistém JavaScriptu problém.

1.1.2 Rysy a paradigma

JavaScript je multiparadigmní interpretovaný jazyk vyšší úrovně. Narozdíl od ostatních jazyků té doby je JavaScript velmi permisivní (věc, kterou se snaží řešit projekt TypeScript). Jistým způsobem se dá v JavaScriptu programovat objektově (za pomocí prototypů), ale je dovolený také funkcionální přístup (funkce je obyčejný volatelný objekt, který může mít vlastní proměnné, closure funkce apod). Ta objektovost je oproti jazykům Java/C#/C++ v JavaScriptu spíše dekorativní. Rozhodně se tam nenajde žádný memory management a RAII vázaný na životnost objektů. I přesto, že obsahuje klíčové slovo *class*, tak to je jen syntaktická vrstva nad prototypem.

[1] Is ECMAScript really a dialect of Lisp? stackoverflow.com/questions/5030285/is-ecmascript-really-a-dialect-of-lisp

[2] Za to možná může licensování a distribuce, neboť pro webovou Javu je třeba plugin třetí strany, zatímco JavaScript je "nativně" podporován.

[3] Oracle Announces End Of Java Applet Support, i-programmer.info/news/80-java/9391-oracle-annouces-the-end-of-java-applet-support.html

Je dynamicky typovaný. Obsahuje primitiva typu: boolean, NaN, undefined, array, string, object.

Jelikož je to otevřený interpretovaný jazyk, měl občas bizarní problémy, jako například že volání funkcí s dlouhým jménem trvá déle než s krátkým. Většina moderních implementací však používá Just In Time kompilaci, která tyto problémy řeší.

Funkcionální rysy využívá třeba pro event driven programování, což je v prostředí uživatelského rozhraní na stránce přirozené.

Slouží zejména pro skriptování webů, WPA a části serverů. Webové implementace proto přímo obsahují funkce určené pro editaci stránkového DOM. V dnešní době se však používají různé knihovny, které tuto práci usnadňují. Známé jsou jQuery, Angular a Vue, ale každý rok vyjde něco nového. Tyto knihovny jsou známé zejména tím, jak krátkou životnost mají. Ve srovnání např. s C++ knihovnou Boost, která je tu stabilně od roku 1999, jsou tyto frameworky aktivní pouze 3-4 roky, než se přejde na něco nového.[4] Na portálu jobs.cz v květnu 2019 mezi 369 pracovními nabídkami na pozici programátora JavaScriptu 31 požadovalo znalost vue.js, 66 požadovalo React a 43 Angular.[5] To vypovídá o tom, že v průmyslu se programuje zejména za pomocí knihoven a nikoliv v čistém JavaScriptu.

1.1.3 Alternativy

Bohužel jako frontend má we webovém prostředí monopolní roli. Dávno se opustilo od Java appletů a Adobe Flash pluginů[6]. V rámci HTML5 není doporučováno kromě JavaScriptu nic jiného. Konkurence nyní probíhá zejména na úrovni JavaScript enginů a WebAssembly. Např. Google Chrome udělal malou revoluci[7] v rychlosti exekuce kódu svým projektem V8, ale různé prohlížeče používají obecně různé (Nitro, Carakan). Některé tyto engines (např. V8 jako NodeJS) umí běžet nativně a např. odpovídat na HTTP dotazy, což z nich dělá ideální prostředky pro rychlé prototypování webů.

O dominantní roli se dá přesvědčit tím, že výrobce procesorových čipů ARM v roce 2016 přidalo[8] speciální instrukce, které mají za cíl zrychlit implementaci JavaScriptového engine V8.

1.1.3.1 Dart

Co se týče samotných jazyků, tak před několika lety vypadal nadějně projekt Dart, který vypadal podobně Javě, ale měl propracované vazby s DOM stromem a asynchronnost a event-driven programování řešil velmi elegantně za pomocí Future objektů. Byly velké plány, že tento jazyk za pár let bude interpretován prohlížeči a nebude se muset kompilovat do JavaScriptu, ale od toho se nakonec, bohužel, opustilo. Díky hlubší inspiraci Javou měl klasické objektové prvky a působil mnohem robustněji.

Ve verzi 2 Dart SDK se už počítá s tím, že výsledný kód bude kompilován do JavaScriptu a bude se tak kvůli tomu optimalizovat.

1.1.3.2 CoffeeScript

[4] infoq. com/news/2018/01/javascript-lifespan-limited/

[5] Klíčová slova: *vue.js, angular js, react js*

[6] Adobe Flash is nearly dead as 95% of websites ditch the software ahead of its retirement alphr.com/software/1009184/adobe-flash-dead-websites-ditch-software

[7] Celebrating 10 years of V8 v8.dev/blog/10-years

[8] Armv8-A architecture: 2016 additions community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/armv8-a-architecture-2016-additions, instrukce `FJCVTZS`, `FCVTZS`

Další alternativou je CoffeeScript, který se také transpiluje do JavaScriptu, ale je to jen syntaktické pozlátko a nemyslím si, že přináší něco revolučního, kromě statické type kontroly. Zlepšuje čitelnost kódu a zjednodušuje kód. Nevýhodou je, že programátor stejně musí umět JavaScript, aby mohl psát v CoffeeScriptu a moc profesionálních programátorů tento jazyk nepoužívá.

1.1.3.3 TypeScript

TypeScript vznikl v roce 2012 ve společnosti Microsoft. Je nadmnožinou JavaScriptu, tj. veškerý JavaScript kód (až na klíčová slova) je validní TypeScript kód. Přináší pořádné objektové programování ve smyslu, jaké známe třeba z Java a statické typování proměnných. Zní to jako málo, ale pro větší projekty je toto stěžejní, aby se např. daly rozdělit na více modulů apod. Aktuálně se transpiluje do JavaScriptu, což je občas nepohodlné.

Napsat kód v TypeScriptu je náročnější než ekvivalent v JavaScriptu, ovšem ve druhém případě je mnohem snazší vytvořit technický dluh.

1.1.3.4 Elm

Elm je plně funkcionální programovací jazyk, který se používá pro programování webových uživatelských rozhraní. Kompiluje se do JavaScriptu a bohužel není dostatečně rozšířen.

1.1.3.5 ClojureScript

ClojureScript je implementace programovacího jazyka Clojure, který se kompiluje do JavaScriptu. Má syntaxi podobnou LISPu.

1.1.3.6 WebAssembly

Nadějnou alternativou je WebAssembly, což je formát instrukcí pro abstraktní stroj běžící v prohlížeči. Je podporován moderními prohlížeči a dovoluje být "kompilován" z velké škály jazyků (Rust, C/C++, C#/.Net, Java, Python, Elixir, Go). Jelikož se neprogramuje přímo v tomto jazyce, ale v těch ze kterých se kompiluje, je vysoce škálovatelný a oproti JavaScriptu mnohem rychlejší.[9] Nevýhodou stále zůstává, že integrace s webovým prostředím je stále minimální, tudíž se používá v kombinaci s JavaScriptem s tím, že výpočetní výkony se předávají do části kódu běžící ve WebAssembly.

1.1.3.7 Newebové alternativy

Kromě webových jazyků by bylo možné použít i nějaký z klasických jazyků. Toto je rozebráno v analýze projektu Ptakopět.

1.1.4 Popularita

JavaScript je lehký, snadno na naučení (jde ruku v ruce s designem HTML) a nemá pokročilé dekorace. Kód lze psát skoro kamkoliv (tag v HTML, atributy v HTML, samostatné soubory,

[9] Replacing JavaScript: How eBay made a web app 50x faster by switching programming languages techrepublic.com/article/replacing-javascript-with-webassembly-how-ebay-made-a-web-app-50x-faster-by-switching-programming-languages/

dynamický eval). Používá se pro výuku základů programování,[10] neboť ve spojení s HTML dokáže s minimální prací vizualizovat průměr programu a výpočty.

Jazyk se hodí na prototypování různých ukázek (viz Ptakopět jako demo pro Bergamot). Nepříjemnou vlastnosti v komerčním prostředí pak je, že manažeři nutí použít tyto různé mock-upy pro opravdovou implementaci z důvodu zrychlení dodání. Může se tak stát, že po roce vývoje je třeba velkou část kódu reinženýrovat (udělat nový návrh a implementovat).

Nemá žádný unifikovaný standard kódu, což je velký kontrast např. proti Javě, která má toto jako hlavní přednost. Ve větších projektech proto občas bývá problém s komunikací ve vývojovém týmu, poněvadž při pohledu do libovolného zdrojáku bez dobrých komentářů není moc zjevné, k čemu se vlastně vztahuje, kdy se tento kód začne provádět apod.

Jazyk sám o sobě kromě plytké křivky učení není nijak extra zajímavý. Důležité jsou technologie, knihovny a frameworky, které jej obklopují a dovolují programátorům rychlý a snadný vývoj.

Podobně jako Java běží na virtuálním stroji, tak je kód interpretován pouze v sandboxu stránky. Proto tedy uživateli nevadí, že na jeho stroji běží během surfování po internetu cizí kód, neboť může poškodit nejvýše samotný sandbox (tj. stránku). Ve webovém prostředí tomu mohl konkurovat jen Adobe Flash, který měl však licenční a výkonostní problémy a Java Applets, které byly ke konci příliš krkolomné na používání.

1.1.5 Špatné použití

JavaScript je v místech používán až příliš. Jde to vidět třeba na Progresivních Webových Aplikacích, které nahrazují robustní nativní programy. Například monitor ukazatele příští zastávky v autobusech funguje tak, že při nastartování autobusu se nabootuje nějaký Linux, tam se přihlásí defaultní uživatel, zapne se prohlížeč Firefox v celoobrazovém módu a načte se stránka, která ukazuje ty zastávky. Výhodou tohoto přístupu je, že je nesmírně rychlé nějaký takový systém vyvinout. Cena však je neskutečná ztráta výkonu (viz. software disenchantement [11]) a chybovost. Na takovém místě není důvod aby nebyl nějaký embedded system, který by bylo sice zprvu dražší vyvinout, ale samotný provoz a údržba by byla levnější a mělo by to víc pozitivních vlastností.

Konkrétní detaily běžných programátorských chyb v JavaScriptu viz další sekce.

[10] Teaching Kids JavaScript joezimjs.com/javascript/teaching-kids-javascript/

[11] Software disenchantement tonsky.me/blog/disenchantment/

1.2 Programování v JavaScriptu

Běžně by bylo potřeba uvažovat o tom, kde se spouští daný kus JavaScriptu. V případě webové stránky nezáleží, zdali je celý napsaný uvnitř tagu, nebo v souboru, kam se odkazuje `src` atribut, ale už záleží na tom, zdali se nachází uvnitř tagu `<head>`, nebo `<body>`. V prvním případě se při načítání hlavičky počká na načtení celého kódu, který se pak spustí a až posléze se začne zpracovávat `<body>`. Ve druhém se začne vykonávat jednoduše při načtení daného tagu (díky asynchronosti HTTP může být kdykoliv). Toto lze upravit zaregistrováním eventu v objektu `window`.

V ukázkách můžeme předpokládat, že kód se načte vždy jako poslední objekt na stránce.

1.2.1 Základy

```
1 console.log(5555)
2 let a = 5
3 console.log(a*1111)
```

JavaScript dovoluje, ale nepožaduje používání středníků. Základní syntax je totožná s libovolným moderním C/C++ jazykem. Funkce `log` v objektu `console` se používá pro debugovací výpisy.

```
1 let a = document.createElement('p')
2 a.innerHTML = 'lorem ipsum'
3 a.style.color = 'blue'
4 document.body.append(a)
```

Druhá ukázka demonstruje propojení s HTML (vytváření prvků v DOM).

```
1 let a = $("<p style='color: blue'>lorem ipsum</p>")
2 $(document).append(a)
```

V případě, že bychom měli načtenou populární knihovnu jQuery, byl by úkon ještě snazší.

1.2.2 Objekty

```
1 let arr = []
2 arr.push_back({name: 'a', value: 5})
3 arr.push_back({name: 'ab', value: 15})
4 arr.push_back({name: 'abc', value: -5})
5 arr.push_back({name: 'lorem', value: 2})
6 console.log(arr.map(x => x.value).reduce((a, b) => a + b, 0)) // 17
```

Objekty zde nejsou typově definované. Páry se jednoduše napíší do složených závorek. Díky dynamickému typování lze návrh objektu v průběhu programu měnit:

```
1 let obj = {}
2 obj.q = 15
3 console.log(q) // 15
4 delete obj.q
5 console.log(q) // undefined
```

1.2.3 Konstruktor

Jako konstruktor slouží libovolná funkce. Pomocí klíčového slova `this` získávají funkce objektu přístup ke svým členským proměnným.

```
1 function Lorem() {
2   this.text = "dolor sat"
3   this.getText = function() { return "ipsum " + this.text; }
4 }
5 let tmp = new Lorem()
6 console.log(tmp.getText()) // "ipsum dolor sat"
```

1.2.4 Dědičnost

Dědičnost se v JavaScriptu děje pomocí zavolání dalších konstruktorů uvnitř toho nejnižšího.

```
function AddA() {
  this.a = 2
}
function AddB() {
  this.b = 5
}
function AddAalternative() {
  this.a = 15
}
function Lorem() {
  AddA.call(this)
  AddB.call(this)
  AddAalternative.call(this)
}
let l = new Lorem()
console.log(l.b) // 5
console.log(l.a) // 15
```

Příjemnou výhodou je, že takto se *konstruktory* mohou volat kdykoliv a dynamicky se přepisují.

1.2.5 Prototypy

Prototypy lze využít pro konstrukci objektů, které by se jinak musely komplikovaně vnořovat do konstruktoru.

```
function Lorem() {
    this.text = 'dolor sat'
}
Lorem.prototype.getText = function() { return this.text; }
let tmp = new Lorem()
console.log(tmp.getText()) // "dolor sat"
```

Tady bohužel objektovost jazyka JavaScript končí. Jazyk je však stále ve vývoji. Například v roce 2015 bylo přidané klíčové slovo *class*, které zjednodušuje práci s prototypy.

1.2.6 Funkcionální programování

I přesto, že je JavaScript principálně imperativní jazyk, obsahuje některé prvky z funkcionálního programování.

Lambda pure funkce:

```
const a = (b, c) => Math.sqrt(b*b+c*c)
```

Neměnnost (immutablity):

```
let b = "LoremX"
let c = b.split('X').join(' dolor sat amet').toLowerCase()
```

Funkce jako objekty:

```
let evokeWith5 = function(a) {
    a(5);
}
evokeWith5(function(c) { console.log(c*c); })
```

1.2.7 Běžné chyby

Jelikož nemá JavaScript nativní podporu modifikátorů přístupu členů objektů (`private, public, protected`), často se kód degeneruje do používání globálních proměnných, což je obecně považováno za špatný návrh.

Další častou chybou je špatné používání operátorů porovnávání:[12]

```
5 === 5 // true
'hello world' === 'hello world' // true
77 === '77' // false
77 == '77' // true

false === 0 // false
false == 0 // true
0 == "" // true
"" == false // true

null == null // true
undefined == undefined // true
null == undefined // true

NaN == null // false
NaN == undefined // false
NaN == NaN // false
```

Trojité rovnítko totiž znamená rovnost jak v hodnotě, tak v typu. Zároveň se pro nulové hodnoty dají použít i třeba prázdné řetězce, existují `null` a `undefined`, které celou situaci komplikují.

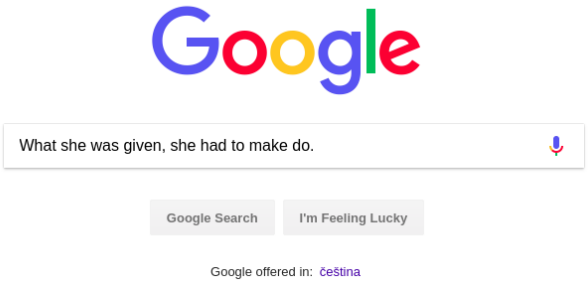
[12] https://codeburst.io/javascript-double-equals-vs-triple-equals-61d4ce5a121a

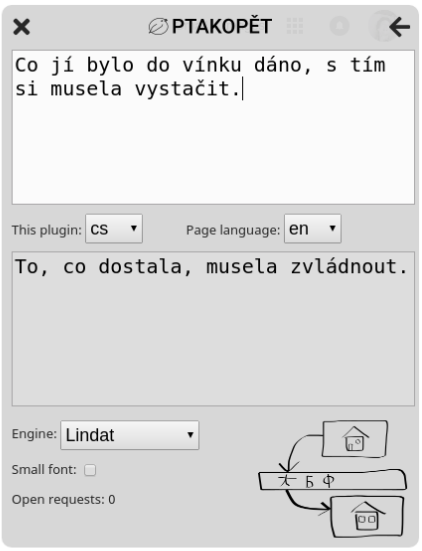
2 Ptakopět

2.1 Verze 1

2.1.1 Úvod

Ptakopět v1[13] vznikl jako práce pro předmět Soutěžní strojový překlad[14] vedeným Ondřejem Bojarem. Cílem bylo vytvořit první prototyp pro Outbound translation (překlad do jazyka, u kterého uživatel není schopen zhodnotit kvalitu překladu). Nyní již nefunkční ukázka (přesunul se překladový backend) připravená pro Den otevřených dveří zde.





Po konci předmětu se od tohoto přístupu opustilo a přešlo se k druhé verzi, která běží na konkrétní stránce a využívá metody odhadu kvality překladu.

2.1.2 Návrh

Zdrojový kód je dostupný na GitHubu. První verze byla vyvíjena jako plugin do prohlížečů (Google Chrome, Mozilla Firefox, Opera). Zároveň však bylo cílem, aby se dal plugin připojit na libovolnou stránku při přidání speciálního `script` tagu. Spouštění ze dvou kontextů (background script vs content script) přineslo spoustu technických potíží. Kód v souboru `ptakopet_background_bootstrap.js` je proveden v případě, že se spouští Ptakopět jako plugin a je přeskočen pokud jako content script ze stránky.

[13] Dokumentace k projektu github.com/zouharvi/ptakopet/blob/master/v1/meta/report.pdf
[14] Soutěžní strojový překlad, NPFL101 is.cuni.cz/studium/predmety/index.php?do=predmet&kod=NPFL101

Soubor `ptakopet_init_user.js` obsahuje spádový boodstrap, který zesynchronizuje soubory, které se mohly načíst v libovolném pořadí. Čistější alternativou by bylo *kompilovat/skládat* celý zdroják do jednoho souboru, čímž by se pořadí vyřešilo, ale za cenu snížení rychlosti asynchronních HTTP požadavků.

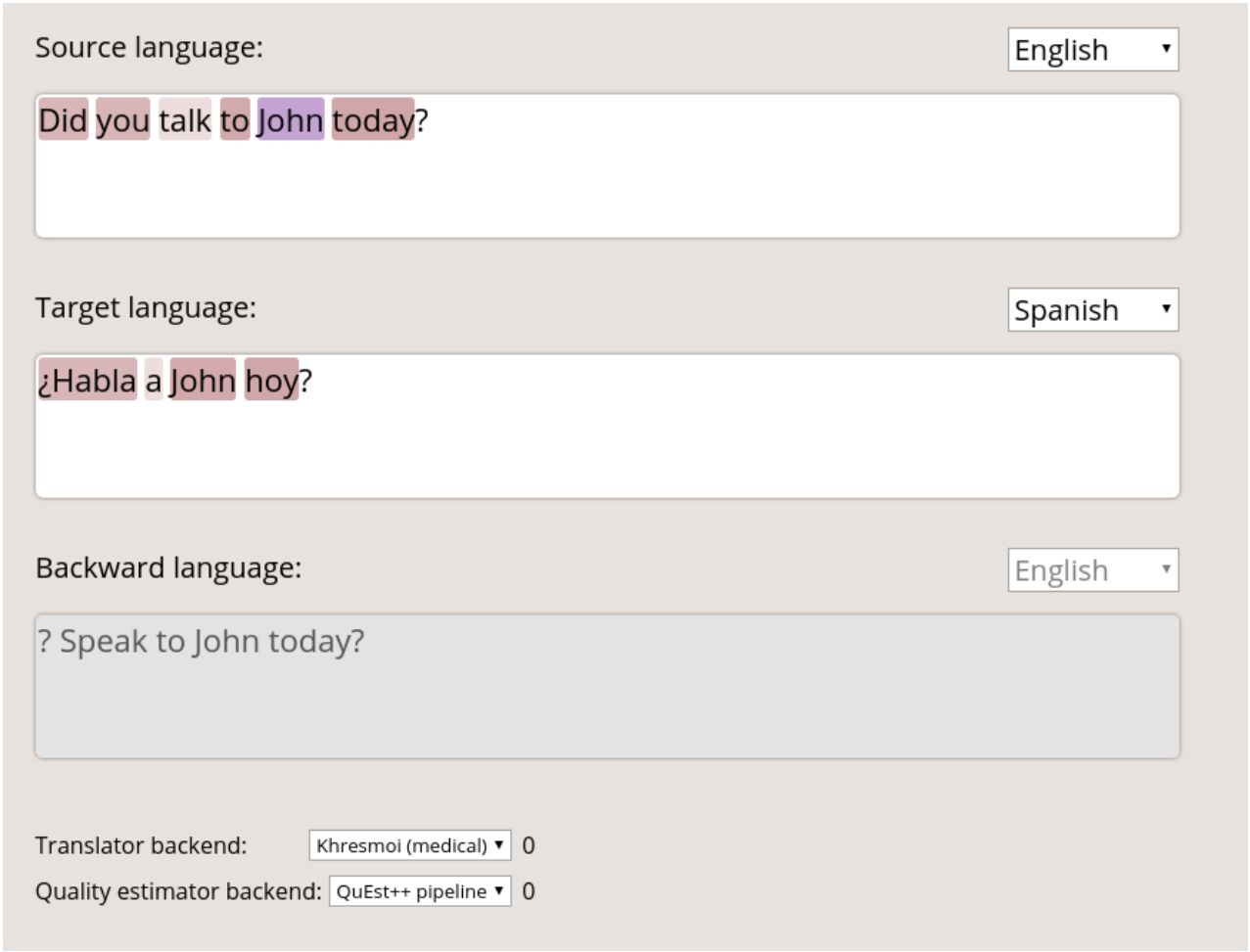
```
1  var ptakopet = ptakopet || {};
2
3  function defer_loading(method) {
4      if ((typeof $ !== "undefined") && (typeof PTAKOPET_ARCH_LOADED != '
       undefined') && (typeof PTAKOPET_TRANSLATOR_LOADED != 'undefined'))
5          method();
6      else
7          setTimeout(function() { defer_loading(method) }, 500);
8  }
9
10 defer_loading(function() {
11     ptakopet_translator_ready();
12     ptakopet_arch_ready();
13     ptakopet_translator_strap();
14
15     // finished loading
16     ptakopet_arch_show();
17 });
```

Zbylé části kódu ve verzi 1 jsou obsáhlé ve verzi 2 a navíc rozšířené.

2.2 Verze 2

2.2.1 Úvod

Verze 2 je aktivně ve vývoji (doposud jako ročníkový projekt) a je přístupná na ptakopet.vilda.net, dokumentace ptakopet.vilda.net/docs. Kód obdobně na GitHubu. Využívá dva backendy pro quality estimation (deepQuest, QuEst++), které nejsou ještě zcela funkční, ovšem front-end je již stabilní.



2.2.2 Návrh

Frontend používá knihovnu jQuery pro lehčí manipulaci s DOM a upravený (forknutý) modul highlight-within-textare. Kód je členěn do následujících souborů:

2.2.2.1 utils.js

Obsahuje třídu Utils, která má několik pomocných členských proměnných (pro úpravu řetězců, tokenizace, vyhledávání permutace).

2.2.2.2 main.js

Tento soubor je vstupním bodem frontendu. Váže proměnné k jejich HTML ekvivalentům (do globálních proměnných) a boostrapuje jiné části kódu. Začíná kontrolou HTTPS požadavku, který je přeložen do HTTP, kvůli problémům spojeným s pravidly smíšeného obsahu (ptakopet.vilda.net má vaild SSL certifikovaný, ale některé připojené komponenty nejsou).

2.2.2.3 arch.js

V tomto souboru se registrují eventy pro překlad, alignment a quality estimation požadavky. Díky jQuery je toto velmi minimalistické, neboť stačí použít funkci `on` daného HTML objektu, např. `input_source.on('keydown', event_handler_function)`.

2.2.2.4 translator.js

Tato část byla z velké části transplantovaná z první verze. Obsahuje konfigurace pro překladače (v tuto chvíli Khresmoi a Lindat MT Transformer) a systém pro předávání požadavků. Přesměrování přes `cors.io` používáme kvůli chybějící hlavičce Access Control Allow Origin v Khresmoi. Toto je potenciální bezpečnostní riziko.

Obsahuje dva hlavní objekty `translator.transformer` a `translator.khresmoi`, od kterých se očekává, že budou obsahovat funkce `translate_target` a `translate_target`. V libovolném jazyce se silnějšími objektovými rysy by se toto řešilo pomocí interface.

2.2.2.5 estimator.js

Tento soubor obsahuje velmi podobnou strukturu jako `translator.js`, tj. obsahuje konfigurace pro posílání požadavků. Vyskytuje se zde stejný problém, který by šel elegantně řešit pomocí interface, které v JavaScriptu chybí. V tomto bodě začíná struktura kódu již chátrat, což by se s větší objektovostí a typovostí nestalo.

2.2.2.6 indicator.js

Jedná se pouze o jednu malou funkci, která sleduje počet aktivních requestů na překlady a odhady kvality.

2.2.3 Alternativy

Při sledování cíle prototypu dynamického zobrazování odhadu kvality překladu se nativní aplikace a s tím spojené tradiční jazyky (Java/C#/C++) nehodí z více důvodů. Předně se má jednat o prototyp, tudíž pečlivý vývoj např. v prostředí Java by nebyl žádoucí kvůli zvýšené časové investici. Také je uživatel zvyklý, že služby tohoto typu jsou k dispozici instantně, tj. nemusí nic stahovat a instalovat. Nutit uživatele, aby si musel stahovat program, který jen komunikuje s backendem a něco vykresluje a k tomu si např. instalovat Java Runtime Environment je v roce 2019 UX nesmysl. Nejdůležitější však je, že klíčová byla rychlost nasazení a dostupnost. Při vývoji v tradičním prostředí by bylo potřeba řešit záležitosti ohledně distribuce a platformy. Dostat něco spustitelného a uživatelsky přívětivého na Linux a Windows není velký problém, ovšem očekáváme, že uživatelé budou používat systém Android, nebo iOS, což by situaci značně komplikovalo.[15]

Jelikož se ve front-endu nedějí žádné vypočetně náročné operace (veškerá machine learning/deep learning inference je v backendu), bylo by zbytečné použít WebAssembly.

Jako prototyp byl Ptakopět verze 2 dostatečný a bude se v něm pokračovat součástí projektu Bergamot, nebo jako bakalářská práce. Aktuální struktura kódu je však nedostačující pro přidávání nových komponent, proto se v další fázi bude přepisovat do TypeScriptu, který nabídne potřebnou škálovatelnost. Dlouhodobý plán je provádět inferenci, zejména místo deepQuest backendu, na straně uživatele. K tomu bude vhodná práce s WebAssembly, která zajistí potřebný výkon.

[15] Před několika lety byla v rozvoji platforma Xamarin, která je postavená na C#/.net a dovoluje vyvíjet aplikace na Linux/Windows/Android/iOS zároveň. Takový vývoj je však zdlouhavý a neřeší předchozí problémy

2.2.4 Závěr

Backend je složen z více jazyků (Python3, Python2, C++, Java, Perl), kvůli různým integrovaným projektům, server samotný běží v Python3 a posílá žádosti do těchto modulů. Použití JavaScriptu v tomto bodě (např. za pomocí NodeJS) by nebylo žádoucí, neboť je vzdálenější od způsobu komunikace, který se aktuálně používá mezi moduly (soubory a pipes).

Frontend Ptakopětu byl vyvíjen v jazyce JavaScript, což odpovídá moderním trendům ve vývoji webových technologiích. Umožnilo to rychlé prototypování a dostupnost. Pro další použití je však třeba tuto část přepsat do více škálovatelného jazyka, což je např. TypeScript, který zvýší přehlednost a udržovatelnost.