Gymnázium, Praha 6, Arabská 14

Předmět Programování

MATURITNÍ PRÁCE

Knihovna pro záznam kotev v textu

Petr Chalupa IV.E

Praha, 2023/24

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V …… dne …… Petr Chalupa …….

ANOTACE

Práce se zabývá návrhem algoritmů, které by umožnili ukládání tzv. textových kotev (označení, poznámky aj.) do statického i dynamického textu (formátu XML) tak, aby je bylo možné opětovně do textu vložit i po jeho úpravě (a případně vyhodnotit chybu při vkládání). Takovýto program by pak měl být použitelný jako knihovna např. pro webové aplikace.

KLÍČOVÁ SLOVA

Algoritmus; textová kotva; XML; knihovna; webová aplikace

ABSTRACT

The thesis deals with the design of algorithms that would enable the storage of so-called text anchors (labels, notes etc.) in static and dynamic text (XML format) so that they can be re-inserted into the text even after its editing (and possibly evaluate the error during insertion). Such a program should then be usable as a library for e.g., web applications.

KEY WORDS

Algorithm; text anchor; XML; library; web application

OBSAH

1 ZADÁNÍ

Téma: Knihovna pro záznam kotev v textu

Autor: Petr Chalupa

Vedoucí práce: Mgr. Jan Lána

Popis: Knihovna s algoritmy pro zapamatování vložených kotev (označení, poznámek aj.) do textu uživatelem. Cílem je, aby fungovala i pro dynamický text, tedy aby se kotvy automaticky přizpůsobovaly změnám v konkrétním textu (v rámci možností) a případně aby poskytla „zpětnou vazbu“ ohledně chyb - např. nepovedené zařazení do textu apod. Měla by fungovat na formátu XML (HTML) - použití primárně ve webových aplikacích, jako je například projekt Digitálního učebnicového systému, kterého jsem spoluautorem.

Platforma: JS/TS, Vue.js

2 Úvod

Text a jeho podstata se v podstatě nikdy neměnila jako dnes. Texty, které byly doteď převážně fyzické, se v posledních letech začali v ohromném množství přesouvat do digitální podoby, ať už protože udržování fyzických kopií je neefektivní využití místa, nebo protože vytváření nových může být velmi nákladné a neekologické, nebo protože je to odpověď na čím dál více rostoucí poptávku po dostupnosti textů v digitální podobně, tedy převážně přes internet.

Mnoho přesouvaných textů jsou původně čistě fyzické knihy, na které se již nevztahuje vlastnické právo, ale může jít také o přesun fyzických médií (jako jsou noviny) do digitální podoby. Vzhledem k tomu, že se tyto texty vyskytují často právě na internetu, dává smysl jeho uživatelům poskytnout užitečné nástroje pro manipulaci s nimi. Motiv této práce je tedy vytvořit takový nástroj, umožňující vkládání a manipulování s textovými kotvami.

Tato práce se tedy zaměřuje na problematiku vkládání, ukládání a opětovného vkládání textových kotev do statického i dynamického textu ve formátu XML. Většina textů se nachází čistě v podmnožině HTML, ale není problém funkčnost rozšířit za hranice webového standartu. Pomocí navrhnutých algoritmů může uživatel označit klíčové body v textu a používat je i po aktualizacích původního textu. V případě, že nastane po změně původního textu problém, uživatel by se o něm měl dozvědět co nejpřívětivější cestou, aby mohl se vzniklými problémy vhodně naložit.

Cílem této práce tedy je vytvořit knihovnu, která bude sloužit jako nástroj pro manipulaci s textovými kotvami, a která bude využitelná ve webových aplikacích. Implementace této knihovny poskytne uživatelům flexibilitu a efektivitu při práci s textem, přičemž bude zajišťovat nejen správnost manipulace s kotvami, ale i detekce a řešení chyb, které by mohly nastat v průběhu procesu.

3 TERMINOLOGIE

3.1 STATICKÝ TEXT

Statický text je chápán jako řetězec znaků (jeho délka není relevantní), který se v průběhu času nemění. Takový text se dá v ideálním případě rozdělit na odstavce, věty a případně slova a znaky. Pracovat s takovým textem tedy lze předvídatelně.

3.2 DYNAMICKÝ TEXT

Dynamický text se odlišuje od statického tím, že se v průběhu času mění, což může vést ke ztížení práce s ním a ke zkomplikování operací na něm prováděných – v extrémních případech až k jejich úplnému selhání. Operace prováděné na dynamickém textu jsou funkční i na statickém texu.

3.3 TEXTOVÁ KOTVA

Textová kotva je pojem, který označuje specifický bod v textu, který je definován svojí pozicí (cesta/souřadnice apod.), a který je v ideálním případě nehybný. Kotva je charakteristická zejména svojí odlišností od textu, pokud rozšíříme její definici na právě dva sousední body, označující začátek a konec kotvy, protože poté začne mít význam i její vizuální charakteristika – např. zabarvené pozadí. Z toho vyplývá například použití pro označování částí textu, což je hlavní motivace této práce.

3.3.1 Z POHLEDU PROGRAMU

Pro algoritmické operace pozbývá význam vizuální reprezentace kotvy, ale je zcela nutné, aby byla každá kotva unikátní např. díky UUID[1]. Kotvy jsou zároveň chápány jako nejmenší možné celky, které jsou pospojované do jednolitého bloku, který může pak kotvu reprezentovat vizuálně jako celek. Důvodem pro toto rozlišení na kotvy (Anchor) a bloky kotev (AnchorBlock) je to, že ve formátu XML se každá kotva vkládá do páru tagů, který ohraničuje např. odstavec textu, ovšem označený text může přesahovat přes více než jeden takový úsek textu.

[1] UUID – Universally Unique Identifier ~ Univerzálně Unikátní Identifikátor

4 ALGORITMUS VYTVOŘENÍ KOTVY

Algoritmus pro vytvoření kotvy není omezen ani horizontálním, ani vertikálním rozsahem označeného textu, tedy textu, který má být de facto kotvami ohraničen. Jediné omezení udává přednastavený blok, který udává, se kterým textem lze takto manipulovat; tj. předek všech textových bloků, se kterými lze manipulovat – kořenový blok (rootNode). Začátku algoritmu tedy předchází impuls od uživatele, kterému v ideálním případě předcházelo označení textu. Pokud by bylo označení prázdné, nebo jiným způsobem neplatné, algoritmus skončí, protože nemůže vytvořit žádnou kotvu. Je vhodné podotknout, že takto definovaných bloků může být více a každý může operovat nezávisle na ostatních.

V případě, že je výběr validní, začne pokus o vytvoření kotvy. Označení (Selection) se v takovém případě skládá z jednoho a více objektů rozsahu (Range) – více těchto objektů je specifické pro Firefox[2], který umožňuje tzv. nesouvislý výběr. S každým rozsahem se pak pracuje zvlášť.

První krok je získání nejbližšího společného předka počátečního a koncového bloku rozsahu. Tento předek je pak zaručeně nejmenší možný blok obsahující celý rozsah (commonAncestorContainer). Následně jsou získány všechny textové bloky kořenového bloku, do kterých zároveň zasahuje rozsah. Z nich jsou vyřazeny všechny ty, které se již podílí na tvoření nějaké kotvy, čímž je zabráněno překrývání kotev – jsou nahrazeny hodnotou null. Účast na tvoření kotvy znamená, že v cestě k němu skrze DOM[3] se vyskytuje element typu Anchor. Pomocí hodnot null je pak pole těchto bloků rozděleno na menší sub-pole, která budou každé zvlášť představovat blok kotev. Tedy k rozdělení na více bloků kotev dojde pouze v případě, že označení je de facto rozděleno jednou nebo více už existujícími kotvami.

Pro každé takové sub-pole je tedy vytvořen AnchorBlok, který kromě referencí na jednotlivé menší bloky textu nese i další informace jako jsou například barva nebo data. Začátek bloku kotev je dán jeho prvním Anchorem, který přebírá odsazení svého začátku od začátku původního textového bloku z rozsahu (startOffset). Konec je pak dán jeho posledním Anchorem, který z rozsahu přebírá odsazení svého konce od začátku původního textového bloku (endOffset). Všechny případné Anchory mezi nimi mají vždy odsazení začátku nastavené na hodnotu 0 a hodnotu odsazení konce na délku textového bloku – pokrývají ho vždy celý.

Nakonec jsou všechny Anchory interně spojeny pomocí hodnot leftJoin a rightJoin, čímž algoritmus končí.

Přidat UML

[2] Zdroj: https://developer.mozilla.org/en-US/docs/Web/API/Selection/rangeCount

[3] DOM – Document Object Model ~ Objektový Model Dokumentu

5 ALGORITMUS ULOŽENÍ KOTEV

Knihovna samotná neukládá vytvořené kotvy do žádné databáze ani jiného uložiště. Implementace ukládání dat je tedy nechána na uživateli, ovšem o data samotná se nijak starat nemusí. Knihovna obsahuje metodu serialize(), která má jako návratovou hodnotu všechna data, která jsou nezbytná pro pozdější rekonstrukci kotev.

Po zavolání této metody na objektu DTA se rekurzivně volá metoda serialize() na každém AnchorBlocku, která vrací zpracovaná data právě tohoto AnchorBlocku. V těchto datech se nachází barva, objekt s daty, textová hodnota (value) celého bloku a opět rekurzivně získaná data jednotlivých objektů Anchor. Z každého z nich je získán jeho startOffset, endOffset, xPath a jeho textová hodnota. Výsledná datová struktura je tedy vrácena jako jeden objekt viz obrázek níže.

Přidat obrázek!!!

6 ALGORITMUS REKONSTRUKCE KOTEV

Opětovné vkládání kotev zajišťuje funkce deserialize(), která jako vstupní parametr předpokládá předem uložená data, která nesmí být pro správnou funkčnost algoritmu nijak porušena. Algoritmus postupuje po jednotlivých uložených AnchorBlocích – tedy vytvoří objekt AnchorBlock a v rámci něj se následně zpracovávají jednotlivé kotvy.

Prvním krokem pro rekonstrukci kotvy je nalezení rodičovského elementu. K tomu je použita uložená hodnota xPath. Pokud není požadovaný element nalezen, je kotva uložena do seznamu kotev určených k opravě a algoritmus přejde k obnově další kotvy. V opačném případě přejde algoritmus ke druhému kroku – kontrole textu elementu, který se nachází mezi uloženými hodnotami startOffset a endOffset. Text se porovnává s uloženou hodnotou striktně, kdy, pokud se shodují, je možné obnovit kotvu do původního stavu. Když ke shodě nedojde, jsou texty porovnány ještě nestriktně, v jejich normalizované podobě, tedy zbaveny veškeré diakritiky, interpunkce a nezávisle na velikosti písma. V případě, že byl text změněn jen drobnou úpravou jako například opravou diakritiky, je kotva obnovena, ale je označena jako změněná. Pokud nedojde ke shodě ani v tomto případě, je kotva zařazena do seznamu kotev určených k opravě. Pokud se podařilo obnovit alespoň jednu kotvu, jsou do AnchorBlocku vloženy uložená data a barva a je zařazen mezi aktivní AnchorBlocky.

Jestliže není seznam kotev určených k opravě prázdný, prochází tyto kotvy procesem pokusu o opravu. V tomto procesu je znám původní AnchorBlock a index dané kotvy v seznamu kotev tohoto AnchorBlocku. Pro účely opravy je vytvořen nový AnchorBlock, do kterého se opravená kotva přiřadí (přebírá také data i barvu původního AnchorBlocku). Tentokrát se místo konkrétního elementu vyhledávají veškeré výskyty uloženého textu, a to nezávisle na velikosti písma. Z těchto výskytů jsou vyloučeny všechny, jež se už nachází uvnitř nějaké kotvy. Pro případ, že by se nějaký z těchto výskytů nacházel v požadovaném uloženém elementu (existuje-li), je tento výskyt upřednostněn, jinak je použit první výskyt v textu. V určeném výskytu je dále nalezen výskyt nejblíže k uloženým hodnotám startOffset a endOffset (pro případ, že by se v daném elementu hledaný text vyskytoval vícekrát). Na tomto výskytu je následně obnovena kotva, která je dále označena za změněnou. Pokud by ovšem došlo k tomu, že by se nový AnchorBlock nacházel právě vedle původního AnchorBlocku (existuje-li), přesněji by se opravená kotva nacházela právě vedle kotvy, vedle které byla původně (a to i vzhledem ke straně), jsou tyto AnchorBlocky spojeny do jednoho, čímž je snížen vliv opravy.

7 SPECIÁLNÍ FUNKCE

V knihovně se vyskytují některé funkce, které mají velmi specifické využití a zaslouží si větší pozornost. Ty hlavní jsou v této kapitole popsané a podrobně vysvětlené.

xPath…

7.1 FUNKCE INVERTHEXCOLOR()

Tato funkce slouží k získání kontrastově obrácené barvy k zadané barvě. Funkce má jako argument barvu v HEX formátu a stejně tak vrací barvu v HEX formátu. Jako kontrastově obrácená barva se v tomto případě myslí buď černá (#000000), nebo bílá (#ffffff), jelikož funkce je využita při přebarvování Anchoru tak, aby byla barva textu vždy dostatečně kontrastivní s barvou jeho pozadí. Funkce podporuje argument v 3 nebo 6 znakovém zápisu i s možností vynechat „#“. Nejdříve je u každé barvy zajištěn 6 znakový zápis – tzn. převod v případě potřeby a poté její číselné rozložení na jednotlivé prvky R, G a B.

Nakonec…

https://stackoverflow.com/questions/3942878/how-to-decide-font-color-in-white-or-black-depending-on-background-color/3943023#3943023

8 KNIHOVNA

Celý projekt je koncipován jako knihovna pro použití ve webovém prostředí; přesněji přímo ve webových aplikacích. Od toho se také odvíjí architektura projektu a styl jeho vývoje. Důraz byl například kladen velmi na omezení využívání dalších knihoven (dependecies). Celá knihovna je pak dostupná v registru npm[4] pod názvem dynamic-text-anchors. Díky npm je jednoduché publikovat nové verze knihovny přímo z GitHub repozitáře nebo knihovnu jednoduše sémanticky verzovat (major.minor.patch). Npm ostatním vývojářům poskytuje přehledné informace o knihovně, jako je například odkaz na demo nebo README projektu.

8.1 ARCHITEKTURA

8.2 POUŽITÍ

Celý proces od instalace po užití je objasněn v README. Po instalaci je potřeba knihovnu pouze importovat:

`import DTA from "dynamic-text-anchors";`

Poté je nutné vytvořit objekt DTA, do jehož konstruktoru se vkládá element, v němž má být možné operovat s kotvami:

`const dta = new DTA(rootElement);`

Následně je již možné používat všechny veřejné metody knihovny.

[4] npm, Inc. – správce JS balíčků (knihoven) pro Node.JS

9 DEMO

9.1 FUNKCE

9.2 GENEROVÁNÍ TEXTU

10 ZÁVĚR

11 POUŽITÉ ZDROJE

Aktuální dokument neobsahuje žádné prameny.

12 SEZNAM OBRÁZKŮ

Nenalezena položka seznamu obrázků.

13 SEZNAM UKÁZEK KÓDU

Nenalezena položka seznamu obrázků.