Technical Details

胡雨軒

November 14, 2014

Items in this files are displayed in something close to lexicographic order. The first hasn't to be the most important.

Contents

1 jOOQ code generation

Refer to the jOOQ official website tutorial for further informations.

Given the modified `pom.xml` and the additionnal file `huawen.xml` (aimed at setting bindings with Java project), you may execute the following lines to update Java entities.

```
mvn compile #
mvn exec:java -Dexec.mainClass="org.jooq.util.GenerationTool" -
    Dexec.args="/huawen.xml"
```

2 LaTeX

2.1 Why using LaTeX instead of markdown?

You're right, markdown can be read directly on github with no downloads and is lighter than LaTeX but the latter is pretty easier than markdown in purpose of typesetting hypertext including links, bibliographies and so on. Documents written with LaTeX are far more beautiful and so much easier to compose instead of being a few bit slower to get typed.

Nonetheless I've been trying to automatically generate markdown files with pandoc then to not to upload 'pdf' here but just `*.tex` and `*.md`. I faced some issues (handling bibliographiy-related and `\input` commands) then I gave up. I know you more than me about it, please tell me :-)

2.2 Preamble

LaTeX files here use preambles as modules, so rather prefer to load preambles with commands from the package `import`. On the contrary, it's easier to enclose text with `\input` or even `\inputAllFiles` which import `*.tex` files from a directory and sort them lexicographically.

2.3 Babel and CJK[1]

An sinogram in standard text is rendered correctly but when typed in `\foreignlanguage` just produces a 華 character. Hum..., it seems it was a bug and it's been fixed.

2.4 Font and IDC[2]

Using `xeCJK` is bad and short-minded :

[1] Stands for *C*hinese, *J*aponese and *K*orean.

[2] Stands for *i*deographic *d*escription *c*haracter.

```
\usepackage{fontspec,xeCJK}

\setmainfont{Linux Libertine}
\setCJKmainfont{AR PL New Sung}
```

It's better to use `ucharclasses` which is buggy but do the job when nicely asked to.

```
\usepackage{fontspec}

\defaultfontfeatures{Mapping=tex-text}
\setmainfont{Linux Libertine}
\newfontfamily{\chinesefont}{AR PL New Sung}
\newfontfamily{\idcfont}{Unifont}

\usepackage[CJK, IdeographicDescriptionCharacters]{ucharclasses}
\setTransitionsForCJK{\chinesefont}{\normalfont}
\setTransitionTo{IdeographicDescriptionCharacters}{\idcfont}
\setTransitionFrom{IdeographicDescriptionCharacters}{\normalfont
    }
```

Methinks the compiler automatically embeds needed font glyphs so no need to add arguments like that `-dSubsetFonts=true -dEmbedAllFonts=true`.

2.5 Compilation

I use the XeTEX compiler.

The option `recorder` is required by the package `currfile`. More specifically, the `abspath` option loads the sub-package `currfile-abspath` and requires compiler option `recorder` to be used. Thus, as `currfile` *could be used*, your compilation line should look comething like :

```
xelatex -recorder
```

3 MariaDB

3.1 Why not MySQL ?

Because I prefer truely free and maintained software.

4 La carte de données `PairMap`

La structure de données dans laquelle on stocke les caractères est un dictionnaire qui a pour clef l'IDS d'un caractère (ou son hash). Il faut une structure qui évite complètement les redondances, c'est-à-dire que la forêt des sinogrammes n'ait pas de doublon quel que soit l'ordre dans lequel les caractères sont entrés et quel que soit le détail des IDS. `PairMap` n'a pas vocation a être générique : des structures existent déjà pour cela mais à offrir la solution la plus adaptée au stockage des sinogrammes.

Trois stratégies d'optimisation des accès mémoires :

- Un tas ;
- Un accès aléatoire ;
- Un arbre couvrant.

Stratégie du tas

- Chaque caractère possède ses propres composants ;
- La taille d'un tas est 3 ;
- Solution tributaire des IDS ;
- Explosion combinatoire pour mettre à jour ;
- Analyse de caractère rapide.

Stratégie de l'accès aléatoire

- Lecture et écriture rapide si vraiment aléatoire ;
- Chaque sinogramme contient un tableau de référence de composants.

Stratégie de l'arbre couvrant

- Hypothèse : faiblement connecté ;
- Euh… mais ce n'est vraiment pratique à parcourir !

Un sinogramme comporte trois références vers ses composants et pour l'immense majeure partie deux références. Si le sinogramme est une entrée de la table on a un caractère, un point de code et une IDS. Si c'est un sinogramme induit on n'a qu'une IDS. Certains sinogrammes d'un pan Unicode (*unicode block*) sont décomposés en sinogrammes qui n'ont pas de décomposition dans ce pan mais se décomposent dans un autre pan. L'ordre de décomposition des sinogrammes n'est pas respecté globalement mais seulement à l'intérieur d'un pan. Bien qu'un point de code et une IDS renvoient à une même réalité, il faut les voir comme deux sous-clefs, facette d'une même clef.

Tous les caractères ont une IDS mais seulement les caractères explicites ont un point de code. Les caractères implicites n'ont pas de point de code, seulement une IDS. L'IDS peut donc être vue comme une clef principale à côté de laquelle on place lorsque c'est possible un point de code. L'organisation interne de `PairMap` devra donc être un dictionnaire `(K1, V)` avec `K1` une IDS et `V` une référence vers un objet `Node`. Il y a également un dictionnaire `(K2, K1)` avec `K2` un point de code. Le dictionnaire réciproque `(K1, K2)` n'est nécessaire que pour économiser la résolution d'une référence : on peut en effet accéder à `K2` très facilement à partir de `K1` : `K1` → `V.K2`.

L'interface de `PairMap` est finalement un dictionnaire `(K1, K2, V)`.

La classe `TreeMap` peut-elle apporter quelque chose de plus que `HashMap` ?