TAlone

Zeste de savoir

Le Markdown et la gestion des échappements

11 août 2019

Table des matières

Salut à tous,

une question est passée il n'y a pas longtemps sur Discord et portait sur l'échappement des caractères au sein des blocs de code dans ZMarkdown ; je tente ici d'apporter une réponse au problème, en expliquant pourquoi la solution initialement voulue n'est pas viable. Plutôt qu'une courte réponse sur le canal, il m'a semblé intéressant d'informer la communauté dans son ensemble de cette possibilité offerte par le Markdown et de développer un peu sur le choix technique effectué.

1. Une histoire de syntaxe

Le Markdown est un langage de balisage léger, dont les premières implémentations remontent à 2004 ; il s'agit du langage utilisé pour la rédaction sur Zeste de Savoir. Quand je parle d'implémentation, je parle ici de code, puisque le langage n'a jamais été standardisé, ce qui a très vite posé des problèmes dans les implémentations, qui divergeaient dans leurs façon de traiter la syntaxe.

Un standard *de facto* a plus ou moins émergé de `Markdown.pl`, mais l'implémentation a toujours été buguée, et les différents parseurs ont commencé à mettre en place chacun de leurs côté des règles concernant :

— la précédence des opérateurs (le titre l'emporte sur le gras, par exemple) ;
— les nouvelles fonctionnalités.

En 2012, CommonMark tente d'unifier ces syntaxes et d'en établir une spécification universelle. Cette spécification prendra peu au début, mais va connaître son essor en 2017 grâce à Github Flavored Markdown , une nouvelle spécification, basée sur CommonMark, et n'en changeant qu'une très petite partie. Fait standard pour la rédaction sur Github, le format prends très vite, et la grande majorité des parseurs sont basés sur cette syntaxe aujourd'hui.

2. Les choix techniques initiaux

Le nouveau standard en place tente de répondre aux questions suivantes :

— quelle sont les espacement à utiliser pour le rendu ; en particulier, quelle indentation défini une sous-liste, quelle indentation démarque un bloc de code et faut-il des espaces entre les titres et leur syntaxe (`#`) ;
— les sauts de lignes doivent-ils être interprétés strictement ? Si oui, dans quels cas ? Par exemple, un saut de ligne est-il requis avant une citation ?
— Le rendu des paragraphes peut-il casser le rendu des autres syntaxes, typiquement les listes à puces ?
— Les listes peuvent-elles contenir un ou plusieurs éléments vides ?

Comme vous pouvez le voir, ces problématique ont pour la plupart attrait à la **précédence**, c'est-à-dire l'ordre dans lequel doivent être traités les éléments. GFM y réponds en définissant une distinction simple entre les éléments :

— les blocs, qui sont toujours parsés de façon prioritaire, il s'agit par exemple des citations, qui forment alors un nouveau bloc de sous-parsage, dans lequel les éléments autres pourront être rendus ;
— les éléments en ligne (*inline*), qui n'ont pas de priorité, et sont donc toujours parsé ensuite, c'est ce qui fait qu'un titre dans un bloc de code ne s'affiche pas comme un titre, car le bloc de code (bloc) à la priorité sur le titre (*inline*) ;
— dans les cas où l'on veut s'abstraire de cette précédence en faisant en sorte que le parseur ne prenne pas en compte certains caractères, il faut **échapper** le caractère, en le faisant précéder d'un *antislash* `\` ; les caractères *inline* seront alors rendus dans le bloc parent.

3. Blocs de code, mes amours

Le problème relevé initialement par @**A-312** concerne le code suivant :

```
`WHERE \`personnes\` LIKE %machin%`
```

devrait selon lui s'afficher `WHERE `personnes` LIKE %machin%`, alors que le rendu actuel est `WHERE \personnes‘ LIKE %machin%‘`, l'échappement n'étant pas rendu. Alors, pourquoi ce comportement quand `\*italique*` ne rends pas l'italique ?

D'abord vu comme un problème avec le rendu de Markdown par `ZMarkdown`, et surtout sa couche inférieure, `remark`, il s'avère en fait... que ce problème n'en est pas un. Les *antislashs*, comme mentionné précédemment, ne se voient attribuer précédence (ou plutôt, retirent la précédence d'une syntaxe) que lors d'une ambiguïté de parsage entre bloc et *inline*. **Il n'y en a aucune ici** ; le seul caractère échappable est le début du bloc de code : `\`test`` rend donc ‘test‘, puisqu'il donne la priorité au bloc de paragraphe, plutôt qu'au bloc *inline* de code pour le rendu du caractère.

4. Une solution simple

Alors, oui, c'est bien beau, cette histoire de priorité, mais comment alors peut-on rendre le code initial en *inline* ? La spécification mentionne pour le code *inline* la structure suivante :

> Une chaîne d'accents graves (`) est une chaîne d'un ou plusieurs accents graves qui n'est ni précédée, ni suivie d'un accent grave.
> Un bloc en ligne de code commence par une chaîne d'accents graves, et se termine par une chaîne d'accents graves **de la même longueur**.
>
> *GFM §6.3 (traduction libre)*

En résumé, il s'agit de la plus longue chaîne possible d'accents. Un bloc de code *inline* n'est donc **pas nécessairement** commencée et terminée par un unique caractère `; si l'on souhaite avoir dans son code un de ces caractères, il suffit alors de choisir une chaîne de longueur supérieure à 1 ; le problème était donc résolu avant même qu'il ne se soit posé. Pour rendre la chaîne initiale, on pourra donc faire :

```

```

Notez bien les **deux** accents au lieu d'un seul, qui permet le bon rendu de la chaîne : `` WHERE `personnes` LIKE %machin% ``. Notez qu'il n'est pas nécessaire d'avoir n+1 accents préliminaires pour une chaîne contenant n accents qui se suivent, ainsi ``` est rendu par :

```

```

Qui est une chaîne d'un seul accent, en contenant trois à l'intérieur. L'espace est une autre astuce, permettant au Markdown d'éviter de confondre le code avec une chaîne de 5 accents.

La prochaine fois que vous verrez un nouveau membre poster un code non-coloré, vous saurez maintenant comment l'informer correctement à propos du problème, sans vous prendre la tête avec la syntaxe.