The REBOL Documentation Project

-- FR - Documentation REBOL - Divers --

Divers

Les erreurs désarmées, la fonction "cause"

Philippe Le Goff

Première publication : 4 mai 2006, et mis en ligne le jeudi 4 mai 2006

Résumé :

La traduction du blog 7 de Rebol 3.0, sur la fonction cause

Carl Sassenrath, CTO REBOL Technologies 9-Apr-2006 19:13 GMT Article #0007

En REBOL 3.0, les valeurs d'erreur sont désarmées (disarmed) par défaut.

Les versions précédentes de REBOL utilisaient les erreurs "à chaud". Cependant, vous deviez traiter ces valeurs d'erreur de façon spécifique, ou elles déclenchaient automatiquement la gestion d'erreur. Ce comportement avait été implémenté à l'origine pour éviter que les erreurs se propagent trop loin de leurs origines (le principe était de préserver autant que possible la localisation d'une erreur).

Ces erreurs "à chaud" finissaient par être réduites à néant, et l'intérêt de pouvoir localiser une erreur était compensé par la difficulté de gérer les valeurs d'erreur en général. (Voir les articles de Ladislav Mecir qui a écrit d'excellentes notes sur ce sujet). C'était très souvent vraiment épineux.

Avec ceci à l'esprit, sachez que REBOL 3.0 supprimera les erreurs à chaud.

Les valeurs d'erreurs (objets) sont désarmées par défaut, et peuvent être traitées de la même manière que tous les autres objets. Les fonctions comme **try** peuvent être utilisées pour capturer les erreurs et les traiter comme des valeurs normales.

Dans les cas où il est nécessaire de ré-armer une erreur, et de forcer son retour dans l'état "à chaud", la fonction **cause** a été ajoutée. Vous "provoquez" la reconsidération de l'erreur pour être traitée. (voir la description de Ladislav sur "fire".)

Voici un simple exemple de manipulation d'une erreur spécifique :

```
result: try [... do something ...]
if all [
        error? result
        result/id = 'zero-divide
] [
        cause result
]
```

L'exemple évalue un bloc et capture toutes les erreurs. Le résultat (erreur ou non) est stocké dans le mot *result*. Le code vérifie si l'erreur est celle d'une division par zéro (zero-divide), et si oui, renvoie l'erreur à un niveau de traitement supérieur (non montré). Remarquez que disarm n'est pas utilisée.

Pour la compatibilité avec REBOL 2.0, la fonction disarm sera encore fournie, mais elle ne fait plus grand chose. (renvoie l'objet error comme un object ! plutôt que comme une error !, c'est tout.)

Notes pour les experts :

En interne, la fonction cause provoque un "stack throw" (un long saut en C) qui est très rapide, mais des parties importantes du contexte d'exécution sont perdues dans ce processus.

Par comparaison, les fonctions REBOL break, return, et throw libèrent la pile d'une manière ordonnée, ce qui permet de rendre possible des options de déboguage durant le processus.

Plus d'informations sur la gestion des erreurs,bientôt.

(Traduction : Philippe Le Goff)