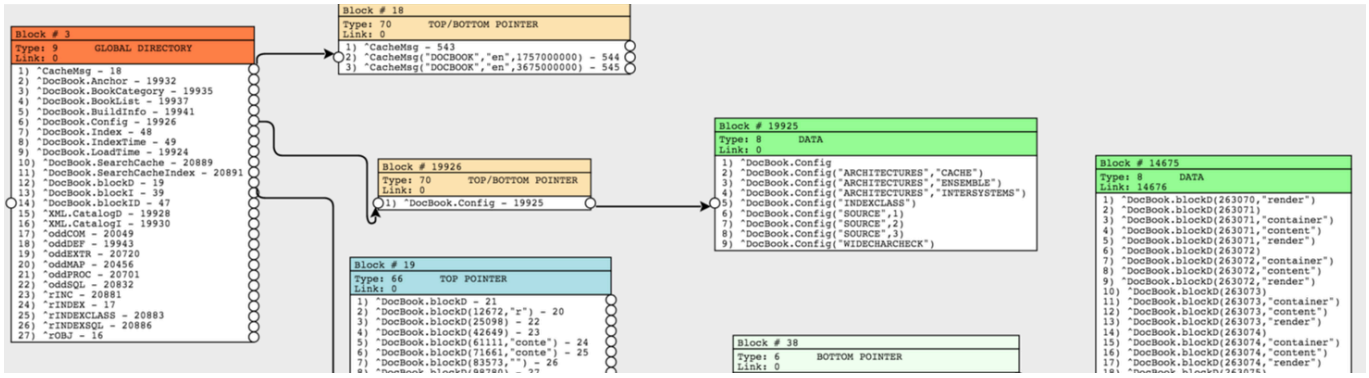
Article
Guillaume Rongier · Avr 13, 2022 7m de lecture

Structure interne des blocs de la base de données Caché, partie 2

Ce texte est la suite de mon article où j'ai expliqué la structure d'une base de données Caché. Dans cet article, j'ai décrit les types de blocs, les connexions entre eux et leur relation avec les globales. L'article est purement théorique. J'ai fait un projet qui aide à visualiser l'arbre des blocs - et cet article explique comment il fonctionne en détail.



Pour les besoins de la démonstration, j'ai créé une nouvelle base de données et l'ai débarrassée des globales que Caché initialise par défaut pour toutes les nouvelles bases de données. Créons une globale simple :

```
set ^colors(1)="red"
 set ^colors(2)="blue"
 set ^colors(3)="green"
? set ^colors(4)="yellow"
```

Notez l'image illustrant les blocs du global créé. Celui-ci est simple, c'est pourquoi nous voyons sa description dans le bloc de type 9 (bloc catalogue des globales). Il est suivi par le bloc "pointeur supérieur et inférieur" (type 70), car l'arbre des globales n'est pas encore profond, et vous pouvez utiliser un pointeur vers un bloc de données qui tient encore dans un seul bloc de 8 Ko.

Maintenant, écrivons tant de valeurs dans une autre globale qu'elles ne peuvent pas être placées dans un seul bloc - et nous verrons de nouveaux nœuds dans le bloc de pointeurs pointant vers de nouveaux blocs de données qui ne pouvaient pas être placés dans le premier.

Écrivons 50 valeurs, de 1000 caractères chacune. Rappelez-vous que la taille du bloc dans notre base de données est de 8192 octets.

```
set str=""
for i=1:1:1000 {
    set str=str_"1"
}
for i=1:1:50 {
    set ^test(i)=str
}
```

```
?   quit
```

Regardez l'image suivante :

Nous avons plusieurs nœuds au niveau du bloc de pointeurs pointant vers des blocs de données. Chaque bloc de données contient des pointeurs vers le bloc suivant ("lien correct"). Offset - pointe vers le nombre d'octets occupés dans ce bloc de données.

Essayons de simuler une division de bloc. Ajoutons tellement de valeurs au bloc que la taille totale du bloc dépasse 8 Ko, ce qui provoquera la division du bloc en deux.

Exemple de code

```
   set str=""
   for i=1:1:1000 {
       set str=str_"1"
   }
   set ^test(3,1)=str
   set ^test(3,2)=str
?   set ^test(3,3)=str
```

Le résultat est présenté ci-dessous :

Le bloc 50 est divisé et rempli de nouvelles données. Les valeurs remplacées se trouvent maintenant dans le bloc 58 et un pointeur vers ce bloc apparaît maintenant dans le bloc des pointeurs. Les autres blocs sont restés inchangés.

Un exemple avec de longues chaînes de caractères

Si nous utilisons des chaînes plus longues que 8 Ko (la taille du bloc de données), nous obtiendrons des blocs de "données longues". Nous pouvons simuler une telle situation en écrivant des chaînes de caractères de 10000 octets, par exemple.

Exemple de code

```
   set str=""
   for i=1:1:10000 {
       set str=str_"1"
   }
   for i=1:1:50 {
       set ^test(i)=str
?   }
```

Voyons le résultat :

En conséquence, la structure des blocs de l'image est restée la même, puisque nous n'avons pas ajouté de nouveaux nœuds globaux, mais seulement modifié les valeurs. Cependant, la valeur Offset (nombre d'octets occupés) a changé pour tous les blocs. Par exemple, la valeur Offset du bloc #51 est maintenant 172 au lieu de 7088. Il est clair que maintenant, lorsque la nouvelle valeur ne peut pas être insérée dans le bloc, le pointeur vers le dernier octet de données devrait être différent, mais où sont nos données ? Pour le moment, mon projet ne supporte pas la possibilité d'afficher des informations sur les "grands blocs". Utilisons l'outil ^REPAIR pour obtenir des informations sur le nouveau contenu du bloc #51.

Laissez-moi vous expliquer le fonctionnement de cet outil. Nous voyons un pointeur sur le bloc correct #52, et le même numéro est spécifié dans le bloc du pointeur parent dans le noeud suivant. Le collatéral de la globale est défini sur le type 5. Le nombre de noeuds avec des chaînes longues est de 7. Dans certains cas, le bloc peut contenir à la fois des valeurs de données pour certains noeuds et des chaînes longues pour d'autres, le tout dans

un seul bloc. Nous voyons également quelle référence de pointeur suivante doit être attendue au début du bloc suivant.

Concernant les blocs de longues chaînes de caractères : nous voyons que le mot clé "BIG" est spécifié comme valeur du global. Cela nous indique que les données sont en fait stockées dans des "gros blocs". La même ligne contient la longueur totale de la chaîne contenue, et la liste des blocs stockant cette valeur. Jetons un coup d'oeil au "bloc de chaînes longues", le bloc #73.

Malheureusement, ce bloc est montré encodé. Cependant, nous pouvons remarquer que les informations de service de l'en-tête du bloc (qui font toujours 28 octets) sont suivies de nos données. Connaître le type de données rend le décodage du contenu de l'en-tête assez facile :

Position	Value	Description	Comment
0-3	E4 1F 00 00	Offset pointant vers la fin des données	Nous avons 8164 octets, plus 28 octets d'en-tête pour un total de 8192 octets, le bloc est plein.
4	18	Type de bloc	Comme on s'en souvient, 24 est l'identifiant de type pour les longues chaînes de caractères.
5	05	Collate	Collate 5 signifie "Caché standard"
8-11	4A 00 00 00	Lien correct	Nous obtenons 74 ici, car nous nous souvenons que notre valeur est stockée dans les blocs 73 et 74

Je vous rappelle que les données du bloc 51 n'occupent que 172 octets. Cela s'est produit lorsque nous avons enregistré de grandes valeurs. Il semble donc que le bloc soit devenu presque vide avec seulement 172 octets de données utiles, et pourtant il occupe 8ko ! Il est clair que dans une telle situation, l'espace libre sera rempli de nouvelles valeurs, mais Caché nous permet également de compresser une telle globale. Pour cela, la classe %Library.GlobalEdit dispose de la méthode CompactGlobal. Pour vérifier l'efficacité de cette méthode, utilisons notre exemple avec un grand volume de données - par exemple, en créant 500 noeuds.

Voici ce que nous avons obtenu.

```
kill ^test
for l=1000,10000 {
    set str=""
    for i=1:1:l {
        set str=str_"1"
    }
    for i=1:1:500 {
        set ^test(i)=str
    }
}
quit
```

Nous n'avons pas montré tous les blocs ci-dessous, mais le résultat devrait être clair. Nous avons beaucoup de blocs de données, mais avec un petit nombre de noeuds.

Exécution de la méthode CompactGlobal :

```
write ##class(%GlobalEdit).CompactGlobal("test","c:\intersystems\ensemble\mgr\test")
```

Jetons un coup d'oeil au résultat. Le bloc des pointeurs ne compte plus que 2 nœuds, ce qui signifie que toutes nos valeurs sont allées à deux nœuds, alors que nous avions initialement 72 nœuds dans le bloc des pointeurs. Nous nous sommes donc débarrassés de 70 nœuds et avons ainsi réduit le temps d'accès aux données lors du

passage par la globale, puisque cela nécessite moins d'opérations de lecture de bloc.

CompactGlobal accepte plusieurs paramètres, comme le nom du global, la base de données et la valeur de remplissage cible, 90% par défaut. Et maintenant nous voyons que Offset (le nombre d'octets occupés) est égal à 7360, ce qui est autour de ces 90%. Quelques paramètres de sortie de la fonction : le nombre de mégaoctets traités et le nombre de mégaoctets après compression. Auparavant, les globales étaient compressés à l'aide de l'outil ^GCOMPACT qui est maintenant considéré comme obsolète.

Il convient de noter qu'une situation où les blocs ne sont que partiellement remplis est tout à fait normale. De plus, la compression des globales peut parfois être indésirable. Par exemple, si votre globale est principalement lue et rarement modifiée, la compression peut s'avérer utile. Mais si la globale change tout le temps, une certaine sparsité dans les blocs de données permet d'éviter de diviser les blocs trop souvent, et l'enregistrement de nouvelles données sera plus rapide.

#Administration du système #Bases de données #Caché

URL de la source:https://fr.community.intersystems.com/post/structure-interne-des-blocs-de-la-base-de-donn%C3%A9es-cach%C3%A9-partie-2