RFC 8977 : Registration Data Access Protocol (RDAP) Query Parameters for Result Sorting and Paging

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 24 janvier 2021

Date de publication du RFC : Janvier 2021

https://www.bortzmeyer.org/8977.html

Le protocole RDAP, normalisé notamment dans les RFC 9082[1] et RFC 9083, permet de récupérer des informations structurées sur des trucs (oui, j'ai écrit « trucs ») enregistrés auprès d'un registre, par exemple des domaines auprès d'un registre de noms de domaine. Voici un RFC tout juste publié qui ajoute à RDAP la possibilité de trier les résultats et également de les afficher progressivement (*"paging"*). C'est évidemment surtout utile pour les requêtes de type « recherche », qui peuvent ramener beaucoup de résultats.

Avec des requêtes « exactes » (*"lookup"* dans le RFC 9082), le problème est moins grave. Ici, je cherche juste de l'information sur un nom de domaine et un seul :

```
% curl https://rdap.nic.bzh/domain/chouchen.bzh
...
   "events" : [
      {
         "eventDate" : "2017-07-12T10:18:12Z",
         "eventAction" : "registration"
      },
      {
         "eventDate" : "2020-07-09T09:49:06Z",
         "eventAction" : "last changed"
      },
...
```

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc9082.txt

Mais si je me lançais dans une recherche plus ouverte (”*search*” dit le RFC 9082), avec par exemple la requête `domains` (notez le S à la fin, cf. RFC 9082, section 3.2.1), le nombre de résultats pourrait être énorme. Par exemple, si je demandais tous les domaines en `.bzh` avec `https://rdap.nic.bzh/rdap/domains?n` j'aurais une réponse d'une taille conséquente `<https://ntldstats.com/tld/bzh>`. (Et je ne vous dis pas pour `.com`...)

En pratique, cette requête ne fonctionnera pas car je ne connais aucun registre qui autorise les recherches RDAP aux utilisateurs anonymes. Ceux-ci ne peuvent faire que des requêtes exactes, à la fois pour épargner les ressources informatiques (cf. la section 7 du RFC sur la charge qu'impose les recherches), et pour éviter de distribuer trop d'informations à des inconnus pas toujours bien intentionnés. Pour effectuer ces recherches, il faut donc un compte et une autorisation. Autrement, vous récupérez un code 401, 403 ou bien carrément une liste vide.

Et si vous avez une telle autorisation, comment gérer une masse importante de résultats ? Avec le RDAP originel, vous récupérez la totalité des réponses, que vous devrez analyser, et ce sera à vous, client, de trier. (Si le serveur n'envoie qu'une partie des réponses, pour épargner le client et ses propres ressources, il n'a malheureusement aucun moyen de faire savoir au client qu'il a tronqué la liste.) C'est tout le but de notre RFC que de faire cela côté serveur. Des nouveaux paramètres dans la requête RDAP vont permettre de mieux contrôler les réponses, ce qui réduira les efforts du client RDAP, du serveur RDAP et du réseau, et permettra d'avoir des résultats plus pertinents.

La solution ? La section 2 de notre RFC décrit les nouveaux paramètres :

— `count` : le client demande à être informé du nombre de trucs que contient la liste des réponses.
— `sort` : le client demande à trier les résultats.
— `cursor` : ce paramètre permet d'indiquer un endroit particulier de la liste (par exemple pour la récupérer progressivement, par itérations successives).

Par exemple, `https://example.com/rdap/domains?name=example*.com&count=true` va récupérer dans `.com` (si le registre de `.com` acceptait cette requête...) tous les noms de domaine dont le nom commence par `example`, et indiquer leur nombre. Une réponse serait, par exemple :

```
"paging_metadata": {
       "totalCount": 43
},
"domainSearchResults": [
       ...
]
```

(`paging_metadata` est expliqué plus loin.)

Pour le paramètre `sort`, le client peut indiquer qu'il veut un tri, sur quel critère se fait le tri, et que celui-ci doit être dans l'ordre croissant (`a` pour ”*ascending*”) ou décroissant (`d` pour ”*descending*”). Ainsi, `https://example.com/rdap/domains?name=*.com&sort=name` demande tous les noms en `.com` triés par nom. `https://example.com/rdap/domains?name=*.com&sort=registrationDate:d` demanderait tous les noms triés par date d'enregistrement, les plus récents en premier.

L'ordre de tri dépend de la valeur JSON du résultat (comparaison lexicographique pour les chaînes de caractères et numérique pour les nombres), sauf pour les adresses IP, qui sont triées selon l'ordre des adresses et pour les dates qui sont triées dans l'ordre chronologique. Ainsi, l'adresse `9.1.1.1` est inférieure à `10.1.1.1` (ce qui n'est pas le cas dans l'ordre lexicographique). Le RFC fait remarquer que tous les SGBD sérieux ont des fonctions pour traiter ce cas. Ainsi, dans PostgreSQL, la comparaison des deux chaînes de caractères donnera :

```
=> SELECT '10.1.1.1' < '9.1.1.1';
 t
```

Alors que si les adresses IP sont mises dans une colonne ayant le type correct (`INET`), on a le bon résultat :

```
=> SELECT '10.1.1.1'::INET < '9.1.1.1'::INET;
 f
```

Après le `sort=`, on trouve le nom de la propriété sur laquelle on trie. C'est le nom d'un membre de l'objet JSON de la réponse. Non, en fait, c'est plus compliqué que cela. Certains membres de la réponse ne sont pas utilisables (comme `roles`, qui est multi-valué) et des informations importantes (comme `registrationDate` cité en exemple plus haut) ne sont pas explicitement dans la réponse. Notre RFC définit donc une liste de propriétés utilisables, et explique comment on les calcule (par exempe, `registrationDate` peut se déduire des `events`). Plutôt que ces noms de propriétés, on aurait tout pu faire en JSONpath <`http://goessner.net/articles/JsonPath/`> ou JSON Pointer (RFC 6901) mais ces deux syntaxes sont complexes et longues (`$.domainSearchResults[*].events[?(@.eventActi` est le JSONPath pour `registrationDate`). La mention en JSONPath du critère de tri est donc facultative.

Et, bien sûr, si le client envoie un nom de propriété qui n'existe pas, il récupérera une erreur HTTP 400 avec une explication en JSON :

```
{
       "errorCode": 400,
       "title": "Domain sorting property 'unknown' is not valid",
       "description": [
           "Supported domain sorting properties are:"
           "'aproperty', 'anotherproperty'"
       ]

}
```

Et le troisième paramètre, `cursor` ? Ce RFC fournit deux méthodes pour indiquer où on en est dans la liste des résultats, la pagination par décalage ("*offset pagination*") et celle par clé ("*keyset pagination*", qu'on trouve parfois citée <`https://developers.facebook.com/docs/graph-api/using-graph-api`> sous le nom ambigu de "*cursor pagination*", qui désigne plutôt une méthode avec état sur le serveur). Ces deux méthodes ont en commun de ne **pas** nécessiter d'état du côté du serveur. La pagination par décalage consiste à fournir un décalage depuis le début de la liste et un nombre d'éléments désiré, par exemple « donne-moi 3 éléments, commençant au numéro 10 ». Elle est simple à mettre en œuvre, par exemple avec SQL :

```
SELECT truc FROM Machins ORDER BY chose LIMIT 3 OFFSET 9;
```

Mais elle n'est pas forcément robuste si la base est modifiée pendant ce temps : passer d'une page à l'autre peut faire rater des données si une insertion a eu lieu entretemps (cela dépend aussi de si la base est relue à chaque requête paginée) et elle peut être lente (surtout avec RDAP où la construction des réponses prend du temps, alors que celles situées avant le début de la page seront jetées). L'autre méthode est la pagination par clé où on indique une caractéristique du dernier objet vu. Si les données sont triées, il est facile de récupérer les N objets suivants. Par exemple en SQL :

```
SELECT truc FROM Machins WHERE chose > [la valeur] ORDER BY chose LIMIT 3;
```

Un inconvénient de cette méthode est qu'il faut un champ (ou un ensemble de champs) ayant un ordre (et pas de duplicata). RDAP rend cela plus difficile, en agrégeant des informations provenant de différentes tables (cf. l'annexe B du RFC). (Voir des descriptions de cette pagination par clé dans « *"Paginating Real-Time Data with Keyset Pagination"* <https://www.sitepoint.com/paginating-real-time-data-> » ou « *"Twitter Ads API"* <https://developer.twitter.com/en/docs/twitter-ads-api/pagination> », pour Twitter.) RDAP permet les deux méthodes, chacune ayant ses avantages et ses inconvénients. L'annexe B du RFC explique plus en détail ces méthodes et les choix faits. (Sinon, en dehors de RDAP, un bon article sur le choix d'une méthode de pagination, avec leur mise en œuvre dans PostgreSQL est « *"Five ways to paginate in Postgres, from the basic to the exotic"* <https://www.citusdata.com/blog/2016/03/30/five-ways-to-paginate/> ».) Pour RDAP, un `https://example.com/rdap/doma` récupérerait une page de trois éléments, commençant au dixième, et `https://example.com/rdap/domains?name` trouverait les noms qui suivent `foobar.com`. Notez qu'en réalité, vous ne verrez pas directement le décalage, la clé et la taille de la page dans l'URL : ils sont encodés pour permettre d'utiliser des caractères quelconques (et aussi éviter que le client ne les bricole, il est censé suivre les liens, pas fabriquer les URL à la main). Les exemples du RFC utilisent Base64 pour l'encodage, en notant qu'on peut certainement faire mieux.

La capacité du serveur à mettre en œuvre le tri et la pagination s'indiquent dans le tableau `rdapConformance` avec les chaînes `sorting` et `paging` (qui sont désormais dans le registre IANA <https://www.iana.org/assignments/rdap-extensions/rdap-extensions.xml#rdap-extensions-1>). Par exemple (pagination mais pas tri) :

```
"rdapConformance": [
           "rdap_level_0",
           "paging"
     ]
```

Il est recommandé que le serveur documente ces possibilités dans deux nouveaux éléments qui peuvent être présents dans une réponse, `sorting_metadata` et `paging_metadata`. Cela suit les principes d'auto-découverte de HATEOAS. Dans la description `sorting_metadata`, on a `currentSort` qui indique le critère de tri utilisé, et `availableSorts` qui indique les critères possibles. Chaque critère est indiqué avec un nom (`property`), le fait qu'il soit le critère par défaut ou pas, éventuellement une expression JSONPath <https://goessner.net/articles/JsonPath/> désignant le champ de la réponse utilisé et enfin une série de liens (RFC 8288) qui vont nous indiquer les URL à utiliser. Pour `paging_metadata`, on a `totalCount` qui indique le nombre d'objets sélectionnés, `pageSize` qui indique le nombre récupérés à chaque itération, `pageNumber` qui dit à quelle page on en est et là encore les liens à suivre. Ce pourrait, par exemple, être :

```
"sorting_metadata": {
        "currentSort": "name",
        "availableSorts": [
          {
          "property": "registrationDate",
          "jsonPath": "$.domainSearchResults[*].events[?(@.eventAction==\"registration\")].eventDate",
          "default": false,
          "links": [
            {
            "value": "https://example.com/rdap/domains?name=example*.com&sort=name",
            "rel": "alternate",
            "href": "https://example.com/rdap/domains?name=example*.com&sort=registrationDate",
            "title": "Result Ascending Sort Link",
```

```
            "type": "application/rdap+json"
            },
            {
            "value": "https://example.com/rdap/domains?name=example*.com&sort=name",
            "rel": "alternate",
            "href": "https://example.com/rdap/domains?name=example*.com&sort=registrationDate:d",
            "title": "Result Descending Sort Link",
            "type": "application/rdap+json"
            }
          ]
  ...
```

Ici, `sorting_metadata` nous indique que le tri se fera sur la base du nom, mais nous donne les URL à utiliser pour trier sur la date d'enregistrement. Quant à la pagination, voici un exemple de réponse partielle, avec les liens permettant de récupérer la suite :

```
"paging_metadata": {
       "totalCount": 73,
       "pageSize": 50,
       "pageNumber": 1,
       "links": [
         {
         "value": "https://example.com/rdap/domains?name=example*.com",
         "rel": "next",
         "href": "https://example.com/rdap/domains?name=example*.com&cursor=wJlCDLIl6KTWypN7T6vc6nWEmEYe99Hjf1XY
         "title": "Result Pagination Link",
         "type": "application/rdap+json"
         }
       ]
```

L'idée de permettre le contrôle des réponses via des nouveaux paramètres (`count`, `sort` et `cursor`, présentés ci-dessus), vient entre autre du protocole OData. Une autre solution aurait été d'utiliser des entêtes HTTP (RFC 7231). Mais ceux-ci ne sont pas contrôlables depuis le navigateur, ce qui aurait réduit le nombre de clients possibles. Et puis cela rend plus difficile l'auto-découverte des extensions, qui est plus pratique via des URL, cette auto-découverte étant en général considérée comme une excellente pratique REST.

Il existe à l'heure actuelle une seule mise en œuvre de ce RFC dans le RDAP **non public** de `.it`. La documentation est en ligne <`https://rdap.pubtest.nic.it/doc/README.html`>.

Notre RFC permet donc de ne récupérer qu'une partie des objets qui correspondent à la question posée. Si on veut plutôt récupérer une partie seulement de chaque objet, il faut utiliser le RFC 8982.