RedisJSON

Applications developed with the open source version of RedisJSON are 100% compatible with RedisJSON in Redis Enterprise Software (RS).

RedisJSON path syntax

Since there is no standard for JSON path syntax, RedisJSON implements its own. RedisJSON’s syntax is a subset of common best practices and resembles JSONPath not by accident.

Paths always begin at the root of a RedisJSON value. The root is denoted by the period character (\.). For paths referencing the root’s children, prefixing the path with the root is optional.

Dotted and square-bracketed, single-or-double-quoted-child notation are both supported for object keys, so the following paths all refer to bar, child of foo under the root:

- `.foo.bar`
- `foo["bar"]`
- `['foo']["bar"]`

Array elements are accessed by their index enclosed by a pair of square brackets. The index is 0-based, with 0 being the first element of the array, 1 being the next element and so on. These offsets can also be negative numbers, indicating indices starting at the end of the array. For example, -1 is the last element in the array, -2 the penultimate, and so on.

JSON key names and path compatibility

By definition, a JSON key can be any valid JSON String. Paths, on the other hand, are traditionally based on JavaScript’s (and in Java in turn) variable naming conventions. Therefore, while it is possible to have RedisJSON store objects containing arbitrary key names, accessing these keys via a path is only possible if they respect these naming syntax rules: Names must begin with a letter, a dollar ($) or an underscore (_) character Names can contain letters, digits, dollar signs and underscores Names are case-sensitive

Time complexity of path evaluation

The complexity of searching (navigating to) an element in the path is made of: Child level - every level along the path adds an additional search Key search - O(N)**, where N is the number of keys in the parent object Array search - O(1) This means that the overall time complexity of searching a path is O(N*M), where N is the depth and M is the number of parent object keys.

** While this is acceptable for objects where N is small, access can be optimized for larger objects, and this is planned for a future version.

Example

Log into the RedisJSON enabled database with redis-cli and the --raw switch so we can see the raw output:

```
$ redis-cli --raw
127.0.0.1:6379>
```

I have a document called example1 I want to store and it looks like so:

```
{"foo": "bar"}
```
Add a new entity to the example1 document called test2 and set it to an array called test3

```
127.0.0.1:6379> json.set example1 .test2 'test3':[1,2,3]'
OK
```

Get the object to see what it looks like:

```
127.0.0.1:6379> JSON.GET example1
{foo:"bar",test2:{test3:[1,2,3]}}
```

Get the second array value from the test2 sub-document in the test3 object's array:

```
127.0.0.1:6379> JSON.GET example1 .test2.test3[1]
2
```

As you can see, you can easily traverse the document structure by path to get only the data you want, when you need it. You could also use this to increment or another operation on this object.

**More info**

- RedisJSON commands
- RedisJSON source

**Updated:** April 26, 2022