January 2023

Data Structure for Supporting In-Memory Storage with Multiple Keys

Theodore Obbard

Jonathan Katzman

Data Structure for Supporting In-Memory Storage with Multiple Keys

ABSTRACT

The Object data structure in JavaScript is simply a set of key-value pairs in which the values for each key must be one of the primitive data types. Developers can add multiple value fields for a given key within a JavaScript Object by using an array as the value. Developers can also use an array as a key and retrieve any value via the array. However, developers must query Objects with whole matching key values which prevents execution of queries with partial or multi-key matching. Overcoming these limitations by storing data in a client-side database on the local disk of a user device is an asynchronous, high latency, and potentially risky operation. This disclosure describes a higher-level data structure called InMemoryStorage that can be used to store data with multiple keys in memory. The values for the keys within the InMemoryStorage data structure can be JavaScript objects, each of which can store its own key-value pairs as usual. Developers can retrieve data from the InMemoryStorage data structure by formulating queries based on keys within the objects stored within it. The techniques can be used for any application that requires synchronous, in-memory, client-side storage and access of data with more than one key, such as batching logs in memory prior to transmitting them to remote persistent storage. The techniques extend JavaScript by adding support for partial and multi-key matching.

KEYWORDS

- JavaScript
- Key value pair
- In-Memory Storage (IMS)
- Interaction log
- Batched logging

BACKGROUND

JavaScript is commonly used to implement various operational capabilities within a website. As a scripting language, JavaScript lacks support for the more sophisticated features of full-fledged object-oriented programming languages. For instance, the Object data structure in JavaScript is simply a set of key-value pairs. For example, the values for various relevant user attributes of a user interacting with the results of a search can be stored within an Object that contains the query and the top two search results:

```
const user = {
"query": "classical music",
"video1": "The Blue Danube",
"video2": "The Four Seasons",
}
```

Fig. 1

Developers can use the JavaScript Object data structure to check the values of any of the keys stored within it. For instance, for the example shown in Fig. 1, the second-ranked video corresponding to the user's query can be obtained by checking the key `user["video2"]`to obtain its value "`The Four Seasons.`" However, developers cannot add multiple value fields for a given key within a JavaScript Object which prevents execution of queries with partial or multi-key matching. Developers can add multiple value fields for a given key within a JavaScript Object by using an array as the value. Developers can also use an array as a key and retrieve any value via the array. However, developers must query Objects with whole matching key values which prevents execution of queries with partial or multi-key matching. Therefore, Object data structures in JavaScript are not well-suited if developers wish to store multiple attributes for a given key and

maintain the ability to query the attributes with a subset of the keys struct, such as whether the user clicked on the first or the second video in the list of search results.

The limitation of JavaScript Objects keys requiring exact matches prevents developers from seeking values based on partial or multi-key matching. For instance, in the example above, developers cannot create queries to seek all videos within the search results that the user clicked and played. An approach to overcome these limitations is to store the data in a database on the local disk of the user device via the JavaScript IndexedDB database library. However, such an asynchronous approach is slower than data structures stored only in short-term working memory because of the latency involved in confirming successful storage of the data to disk. Moreover, the approach can potentially create security and privacy risks since the data stored on the user device persists even after the working memory allocated to JavaScript is garbage collected after the user closes the browser tab. As a result, users may choose not to grant permission for JavaScript code to access data on the local disk which makes it infeasible to use the approach for implementing the desired operation.

Developers can use the JavaScript Object data structure as temporary storage for logs, such as various interactive actions of the user while browsing the website. Such logs stored in the working memory of the user device can serve as a cache that is periodically flushed to persistent remote storage, such as the cloud. Developers cache the logs instead of sending each log item individually to avoid generating unreasonably high network traffic which is inefficient and can create latency. Instead, logs cached in working memory are flushed at various appropriate times determined by one or more appropriate parameters, such as number of items in the cache, time elapsed since the last flush, idle times during system operation, etc.

However, limitations of the Object data structure mentioned above can make it infeasible to record logs in the desired form. For instance, the inability to query complex partial keys can make it infeasible to distinguish between user actions performed via different accounts on the system as well as between those of different users who share a device and have at least one other delineating factor. Not being able to associate specific log items with specific user accounts with certainty can limit the utility of the logs.

DESCRIPTION

This disclosure describes techniques that augment the capabilities of JavaScript by enabling developers to store data with multiple keys in short-term working memory. The augmentation is achieved with a higher-level data structure for in memory storage, referred to herein as InMemoryStorage.

Similar to the JavaScript Object data structure, the JavaScript InMemoryStorage data structure is designed to store key-value pairs. The values for each key stored within the JavaScript Object data structure must be one of the primitive data types, such as strings, numbers, Booleans, etc. In contrast, the values for each of the keys within the InMemoryStorage data structure can be JavaScript Objects, each of which can store its own key-value pairs as usual. Developers can retrieve data from the InMemoryStorage data structure by formulating queries based on one or more keys within the Objects stored within it.

```
const aliceKeys = {
authenticatedUser: "Alice",
query: "classical music",
"video1": "The Blue Danube",
"video2": "The Four Seasons",
playedTop2Videos: true,
watchDuration: 743
};

const bobKeys = {
authenticatedUser: "Bob",
"query": "top rock songs",
"video1": "Living on a Prayer",
"video2": "We Will Rock You",
playedTop2Videos: false,
watchDuration: 0
}

const logs = new InMemoryStorage();
logs.set(aliceKeys, "Alice");
logs.set(bobKeys, "Bob");
```

Fig. 2: Storing JavaScript Objects as values for keys in the InMemoryStorage data structure

Fig. 2 shows example code snippets that illustrate operational implementation of the techniques for the case of two users Alice and Bob who share a device for watching online videos via their respective user accounts with the content provider. With appropriate permissions, the logs of their respective online interaction within the content platform can be stored as relevant key-value pairs within corresponding JavaScript Object data structures which are in turn stored as values within the higher level InMemoryStorage data structure.

With such a storage scheme, developers can query the InMemoryStorage data structure to retrieve results based partial matches for the values of keys in the Objects contained within it. For instance, in the example illustrated in Fig. 2, the query `logs.get({playedTop2Videos: false})` will return the result `“Bob.”` In addition, developers can retrieve data based on queries

that use more than one key in the Objects stored within the InMemoryStorage data structure. In the case illustrated in Fig. 1, for instance, the query `logs.get({playedTop2Videos: true, watchDuration > 900})` would return no matches since `watchDuration` for Alice is below 900 even though the value of `playedTop2Videos` is true. If users permit, developers can additionally use the InMemoryStorage data structure for queries to perform relevant operations such as counting the number of entries that match a given key.

The described techniques can be used for any applications that require synchronous, in-memory, client-side storage and access of data with more than one key. For example, developers can use the InMemoryStorage data structure to batch logs of user interactions in memory prior to sending the logs as a collective payload to remote persistent storage. Developers can also use the InMemoryStorage data structure to create a single payload with the logs for multiple user accounts, distinguished by varying keys.

For example, developers can store the currently authenticated user account as a varying key, `authenticatedUser`, to detect authentication changes and assign each logged action with the user account associated with it. For instance, Alice performs three logged actions before logging out, followed by Bob logging in to perform a couple of logged actions, the first three logged entries within the payload can be associated with Alice and the next two with Bob. As a result, the stored logs can later be queried directly to retrieve all log entries for a particular user account, such as Alice or Bob. In addition, developers can choose to include a prioritization key that can help prioritize the logs by the type of information being logged by the log entry.

The techniques described in this disclosure can be implemented via any suitable mechanism, such as a client-side JavaScript library. The techniques can extend JavaScript by adding support for

partial and multi-key matching, neither of which are currently supported by the native JavaScript Object data structure.

CONCLUSION

This disclosure describes a high level data structure called InMemoryStorage that can be used to store data with multiple keys in memory. The values for the keys within the InMemoryStorage data structure can be JavaScript objects, each of which can store its own key-value pairs as usual. Developers can retrieve data from the InMemoryStorage data structure by formulating queries based on keys within the objects stored within it. The techniques can be used for any application that requires synchronous, in-memory, client-side storage and access of data with more than one key, such as batching logs in memory prior to transmitting them to remote persistent storage. The techniques extend JavaScript by adding support for partial and multi-key matching.