React Native Storage

This is a local storage wrapper for both react native apps (using AsyncStorage) and web apps (using localStorage).

Install

```
npm install react-native-storage
npm install @react-native-community/async-storage
import Storage from 'react-native-storage';
import AsyncStorage from '@react-native-community/async-storage';

const storage = new Storage({
  // maximum capacity, default 1000
  size: 1000,

  // Use AsyncStorage for RN apps, or window.localStorage for web apps.
  // If storageBackend is not set, data will be lost after reload.
  storageBackend: AsyncStorage, // for web: window.localStorage

  // expire time, default: 1 day (1000 * 3600 * 24 milliseconds).
  // can be null, which means never expire.
  defaultExpires: 1000 * 3600 * 24,

  // cache data in the memory. default is true.
  enableCache: true,
```

```
  // if data was not found in storage or expired data was found,
  // the corresponding sync method will be invoked returning
  // the latest data.
  sync: {
    // we'll talk about the details later.
  }
});

// I suggest you have one (and only one) storage instance in global scope.

// for web
// window.storage = storage;

// for react native
// global.storage = storage;
Save & Load & Remove
// Save something with key only. (using only a keyname but no id)
// This key should be unique. This is for data frequently used.
// The key and value pair is permanently stored unless you remove it yourself.
storage.save({
  key: 'loginState', // Note: Do not use underscore("_") in key!
  data: {
```

```
    from: 'some other site',
    userid: 'some userid',
    token: 'some token'
  },

  // if expires not specified, the defaultExpires will be applied instead.
  // if set to null, then it will never expire.
  expires: 1000 * 3600
});

// load
storage
  .load({
    key: 'loginState',

    // autoSync (default: true) means if data is not found or has expired,
    // then invoke the corresponding sync method
    autoSync: true,

    // syncInBackground (default: true) means if data expired,
    // return the outdated data first while invoking the sync method.
    // If syncInBackground is set to false, and there is expired data,
    // it will wait for the new data and return only after the sync completed.
```

```
  // (This, of course, is slower)
  syncInBackground: true,

  // you can pass extra params to the sync method
  // see sync example below
  syncParams: {
    extraFetchOptions: {
      // blahblah
    },
    someFlag: true
  }
})
.then(ret => {
  // found data go to then()
  console.log(ret.userid);
})
.catch(err => {
  // any exception including data not found
  // goes to catch()
  console.warn(err.message);
  switch (err.name) {
    case 'NotFoundError':
      // TODO;
```

```
        break;
      case 'ExpiredError':
        // TODO
        break;
    }
  });

// ----------------------------------------------

// Save something with key and id.
// "key-id" data size cannot surpass the size parameter you pass in the construc-
tor.
// By default the 1001st data will overwrite the 1st data item.
// If you then load the 1st data, a catch(NotFoundError) or sync will be invoked.
var userA = {
  name: 'A',
  age: 20,
  tags: ['geek', 'nerd', 'otaku']
};

storage.save({
  key: 'user', // Note: Do not use underscore("_") in key!
  id: '1001', // Note: Do not use underscore("_") in id!
```

```
  data: userA,
  expires: 1000 * 60
});

// load
storage
  .load({
    key: 'user',
    id: '1001'
  })
  .then(ret => {
    // found data goes to then()
    console.log(ret.userid);
  })
  .catch(err => {
    // any exception including data not found
    // goes to catch()
    console.warn(err.message);
    switch (err.name) {
      case 'NotFoundError':
        // TODO;
        break;
      case 'ExpiredError':
```

```
      // TODO
      break;
    }
  });

// ------------------------------------------------

// get all ids for "key-id" data under a key,
// note: does not include "key-only" information (which has no ids)
storage.getIdsForKey('user').then(ids => {
  console.log(ids);
});

// get all the "key-id" data under a key
// !! important: this does not include "key-only" data
storage.getAllDataForKey('user').then(users => {
  console.log(users);
});

// clear all "key-id" data under a key
// !! important: "key-only" data is not cleared by this function
storage.clearMapForKey('user');
```

```
// ------------------------------------------------


// remove a single record
storage.remove({
  key: 'lastPage'
});
storage.remove({
  key: 'user',
  id: '1001'
});


// clear map and remove all "key-id" data
// !! important: "key-only" data is not cleared, and is left intact
storage.clearMap();
```

Sync remote data(refresh)

There are two ways to set the sync method. You can pass the sync method in the constructor's parameter, as a function in an object, or you can define it at any time as shown below:

```
storage.sync = {
  // The name of the sync method must be the same as the data's key name
  // And the passed params will be an all-in-one object.
  // You can return a value or a promise here
  async user(params) {
    let {
```

```
  id,
  syncParams: { extraFetchOptions, someFlag }
} = params;
const response = await fetch('user/?id=' + id, {
  ...extraFetchOptions
});
const responseText = await response.text();
console.log(`user${id} sync resp: `, responseText);
const json = JSON.parse(responseText);
if (json && json.user) {
  storage.save({
    key: 'user',
    id,
    data: json.user
  });
  if (someFlag) {
    // do something for some custom flag
  }
  // return required data when succeed
  return json.user;
} else {
  // throw error when failed
  throw new Error(`error syncing user${id}`));
```

```
    }
  }
};
```

In the following example the sync method is called, when you invoke storage.load:

```
storage.load({
    key: 'user',
    id: '1002'
}).then(...)
```

If there is no user 1002 currently in storage, then storage.sync.user will be invoked to fetch and return the remote data.

```
Load batch data
// Load batch data with an array of `storage.load` parameters.
// It will invoke each key's sync method,
// and when all are complete will return all the data in an ordered array.
// The sync methods behave according to the syncInBackground setting: (default true)
// When set to true (the default), if timed out will return the current value
// while when set to false, will wait till the sync method completes

storage.getBatchData([
    { key: 'loginState' },
    { key: 'checkPoint', syncInBackground: false },
```

```
    { key: 'balance' },
    { key: 'user', id: '1009' }
])
.then(results => {
    results.forEach(result => {
        console.log(result);
    })
})

// Load batch data with one key and an array of ids.
storage.getBatchDataWithIds({
    key: 'user',
    ids: ['1001', '1002', '1003']
})
.then( ... )
```