5.4 AsyncStorage(异步存储)

`AsyncStorage` 是一个简单的，异步的，持久化的 Key-Value 存储系统，它对于 App 来说是全局性的．它

用来代替 LocalStorage..

建议您在 `AsyncStorage` 的基础上做一层抽象封装，而不是直接使用 `AsyncStorage`.

在 iOS 端,`AsyncStorage` 是本地代码返回的，本地代码在字典中存储较小的值和咋文件中存储较大

的值.

在 Android 端,`AsyncStorage` 用 RocksDB 或者 SQLite 取决于哪个可用.

`AsyncStorage` 的 JS 代码提供一个更清晰的 JS API、返回真正的错误对象，以及简单的单个函数．API

中的每个方法都会返回一个 `Promise` 对象.

写入数据：

```
try {
  await AsyncStorage.setItem('@MySuperStore:key', 'I like to save it.');
} catch (error) {
  // Error saving data
```

获取数据：

```
try {
  const value = await AsyncStorage.getItem('@MySuperStore:key');
  if (value !== null){
```

```
    // We have data!!
    console.log(value);
  }
} catch (error) {
  // Error retrieving data
}
```

方法

static **getItem**(key, callback?)

通过 `key` 获得值 ， 并调用此函数. 返回一个 `Promise` 对象.

Parameters:

Name and Type	Description
key string	获得值的键.
[callback] ?(error: ?Error, result: ?string) => void	发生任何错误将回调此函数.

static **setItem**(key, value, callback?)

设置一个 `key` 值 ， 并调用此函数. 返回一个 `Promise` 对象.

Parameters:

Name and Type	Description
key string	设置 `item` 的键.
value string	`key` 的值.
[callback] ?(error: ?Error) => void	发生任何错误将回调此函数.

static **removeItem**(key, callback?)

删除一个 `key` 的值 ，并调用此函数. 返回一个 `Promise` 对象.

Parameters:

Name and Type	Description
key string	删除的 `key`.
[callback] ?(error: ?Error) => void	发生任何错误将回调此函数.

static **mergeItem**(key, value, callback?)

合并现有的 `key` 值 ，加入两个值都是 JSON 格式字符串的话. 返回一个 `Promise` 对象.

NOTE:这里不支持所有的原生实现.

Parameters:

Name and Type	Description
key string	原有的 `key`.
value string	新的 `key`.
[callback] ?(error: ?Error) => void	发生任何错误将回调此函数.

示例：

```
let UID123_object ={
  name: 'Chris',
  age: 30,
  traits: {hair: 'brown', eyes: 'brown'},
};
```

```
// You only need to define what will be added or updated
let UID123_object ={
  age: 31,
  traits: {eyes: 'blue', shoe_size: 10}
};
```

```
AsyncStorage.setItem('UID123', JSON.stringify(UID123_object), () => {
  AsyncStorage.mergeItem('UID123', JSON.stringify(UID123_delta), () => {
    AsyncStorage.getItem('UID123', (err, result) => {
      console.log(result);
    });
  });
});
// Console log result:
// => {'name':'Chris','age':31,'traits':
// {'shoe_size':10,'hair':'brown','eyes':'blue'}}
```

static **clear**(callback?)

删除所有的 `AsyncStorage` 信息. 我们可能不希望使用这个; 可以使用 `removeItem` 或者 `multiRemove`

来删除你的 key. 返回一个 `Promise` 对象.

Parameters:

Name and Type	Description
[callback] ?(error: ?Error) => void	发生任何错误将回调此函数.

static **getAllKeys**(callback?)

获得所有的 key. 返回一个 `Promise` 对象.

Parameters:

Name and Type	Description
[callback] ?(error: ?Error, keys: ?Array<string>) => void	发生任何错误将回调此函数.

static **flushGetRequests**()

Flushes any pending requests using a single batch call to get the data.

static **multiGet**(keys, callback?)

我们可以批量获取 `key` 的值，回调函数的参数是一个二维数组，数组内以键值对的形式表示:

```
multiGet(['k1', 'k2'], cb) -> cb([['k1', 'val1'], ['k2', 'val2']])
```

这个方法返回 `Promise` 对象.

Parameters:

Name and Type	Description
keys Array<string>	`key` 的数组.
[callback] ?(errors: ?Array<Error>, result: ?Array<Array<string>>) => void	参数是一个键值对的数组，另外如有任何错误也会加入数组内.

示例：

```
AsyncStorage.getAllKeys((err, keys) => {
  AsyncStorage.multiGet(keys, (err, stores) => {
    stores.map((result, i, store) => {
      // You only need to define what will be added or updated
      let key = store[i][0];
      let value = store[i][1];
    });
```

```
  });
});
```

static **multiSet**(keyValuePairs, callback?)

我们可以批量获取 `key` 的值，回调函数的参数是一个二维数组，数组内以键值对的形式表示:

```
multiSet([['k1', 'val1'], ['k2', 'val2']], cb);
```

这个方法返回 `Promise` 对象.

Parameters:

Name and Type	Description
keyValuePairs Array<Array<string>>	键值对的数组.
[callback] ?(errors: ?Array<Error>) => void	发生任何错误将回调此函数.

static **multiRemove**(keys, callback?)

批量删除 `key`，返回一个 promise 对象。

这个方法返回 `Promise` 对象.

Parameters:

Name and Type	Description
keys <Array<string>	要删除 `key` 的数组.
[callback] ?(errors: ?Array<Error>) => void	发生任何错误将回调此函数.

示例：

```
let  keys = ['k1', 'k2'];
```

```
AsyncStorage.multiRemove(keys, (err) => {
  // keys k1 & k2 removed, if they existed
  // do most stuff after removal (if you want)
});
```

static **multiMerge**(keyValuePairs, callback?)

批量合并键值对，如果值都为 JSON 字符串的话. 返回一个 Promise 对象.

NOTE:这里不支持所有的原生实现.

Parameters:

Name and Type	Description
keyValuePairs Array<Array<string>>	原有的 `key`.
value string	要合并的键值对的数组.
[callback] ?(error: ?Array<Error>) => void	发生任何错误将回调此函数.

示例：

```
// first user, initial values
let  UID234_object ={
  name: 'Chris',
  age: 30,
  traits: {hair: 'brown', eyes: 'brown'},
};
```

```
// first user, initial values
let  UID234_object ={
  age: 31,
```

```
  traits: {eyes: 'blue', shoe_size: 10}
};
```

```
// second user, initial values
let  UID345_object ={
  name: 'Marge',
  age: 25,
  traits: {hair: 'blonde', eyes: 'blue'},
};
```

```
// second user, delta values
let  UID345_object ={
  age: 26,
  traits: {eyes: 'green', shoe_size: 6}
};
```

```
let  multi_set_pairs = [['UID234', JSON.stringify(UID234_object)],
['UID345', JSON.stringify(UID345_object)]]
let  multi_merge_pairs = [['UID234', JSON.stringify(UID234_delta)], ['UID345',
JSON.stringify(UID345_delta)]]
```

```
AsyncStorage.multiSet(multi_set_pairs, (err) => {
  AsyncStorage.multiMerge(multi_merge_pairs, (err) => {
    AsyncStorage.multiGet(['UID234','UID345'], (err, stores) => {
      stores.map( (result, i, store) => {
        let  key = store[i][0];
        let  val = store[i][1];
        console.log(key, val);
```

```
      });
    });
  });
});
// Console log result:
// => UID234 {"name":"Chris","age":31,"traits":
{"shoe_size":10,"hair":"brown","eyes":"blue"}}
=> UID345 {"name":"Marge","age":26,"traits":
{"shoe_size":6,"hair":"blonde","eyes":"green"}}
```