JSON-数据存储

王红元 coderwhy

JSON的由来

- 在目前的开发中，JSON是一种非常重要的**数据格式**，它并**不是编程语言**，而是一种可以在服务器和客户端之间传输的数据格式。
- JSON的全称是JavaScript Object Notation（JavaScript对象符号）：
  - JSON是由Douglas Crockford构想和设计的一种轻量级资料交换格式，算是JavaScript的一个子集；
  - 但是虽然JSON被提出来的时候是主要应用JavaScript中，但是目前已经独立于编程语言，可以在各个编程语言中使用；
  - 很多编程语言都实现了将JSON转成对应模型的方式；
- **其他的传输格式：**
  - XML：在早期的网络传输中主要是使用XML来进行数据交换的，但是这种格式在解析、传输等各方面都弱于JSON，所以目前已经很少在被使用了；
  - Protobuf：另外一个在网络传输中目前已经越来越多使用的传输格式是protobuf，但是直到2021年的3.x版本才支持JavaScript，所以目前在前端使用的较少；
- 目前JSON被使用的场景也越来越多：
  - 网络数据的传输JSON数据；
  - 项目的某些配置文件；
  - 非关系型数据库（NoSQL）将json作为存储格式；

小程序的app.json

app.json

app.json > ...

```json
{
  "pages":[
    "pages/index/index",
    "pages/logs/logs"
  ],
  "window":{
    "backgroundTextStyle":"light",
    "navigationBarBackgroundColor": "#fff",
    "navigationBarTitleText": "Weixin",
    "navigationBarTextStyle":"black"
  },
  "style": "v2",
  "sitemapLocation": "sitemap.json"
}
```

JSON基本语法

- JSON的顶层支持三种类型的值：
  - 简单值：数字（Number）、字符串（String，不支持单引号）、布尔类型（Boolean）、null类型；
  - 对象值：由key、value组成，key是字符串类型，并且必须添加双引号，值可以是简单值、对象值、数组值；
  - 数组值：数组的值可以是简单值、对象值、数组值；

```
1   123
```

```
{
  "name": "why",
  "age": 18,
  "friend": {
    "name": "kobe"
  }
}
```

```
[
  123,
  "abc",
  {
    "name": "kobe"
  }
]
```

JSON序列化

- 某些情况下我们希望将JavaScript中的复杂类型转化成JSON格式的字符串，这样方便对其进行处理：
  - 比如我们希望将一个对象保存到localStorage中；
  - 但是如果我们直接存放一个对象，这个对象会被转化成 [object Object] 格式的字符串，并不是我们想要的结果；

```
const obj = {
  name: "why",
  age: 18,
  friend: {
    name: "kobe"
  },
  hobbies: ["篮球", "足球", "乒乓球"]
}
```

Key	Value
info	[object Object]

JSON序列化方法

- 在ES5中引用了JSON全局对象，该对象有两个常用的方法：
  - stringify方法：将JavaScript类型转成对应的JSON字符串；
  - parse方法：解析JSON字符串，转回对应的JavaScript类型；
- 那么上面的代码我们可以通过如下的方法来使用：

```
// 转成字符串保存
const objString = JSON.stringify(obj)
localStorage.setItem("info", objString)

// 获取字符串转回对象
const itemString = localStorage.getItem("info")
const info = JSON.parse(itemString)
console.log(info)
```

Stringify的参数replace

- **JSON.stringify()** 方法将一个 JavaScript 对象或值转换为 JSON 字符串：
  - 如果指定了一个 replacer 函数，则可以选择性地替换值；
  - 如果指定的 replacer 是数组，则可选择性地仅包含数组指定的属性；

```
// 转成字符串
const objString1 = JSON.stringify(obj)
// {"name":"why","age":18,"friend":{"name":"kobe"},"hobbies":["篮球","足球","乒乓球"]}
console.log(objString1)

// replace参数是一个数组
const objString2 = JSON.stringify(obj, ["name", "age"])
// {"name":"why","age":18}
console.log(objString2)

// replace参数是一个函数
const objString3 = JSON.stringify(obj, (key, value) => {
  console.log(key, value)
  if (key === "name") {
    return "coderwhy"
  }
  return value
})
// {"name":"coderwhy","age":18,"friend":{"name":"coderwhy"},"hobbies":["篮球","足球","乒乓球"]}
console.log(objString3)
```

Stringify的参数space

- 当然，它还可以跟上第三个参数space：

```
// space参数
const objString4 = JSON.stringify(obj, null, 2)
console.log(objString4)
```

- 如果对象本身包含toJSON方法，那么会直接使用toJSON方法的结果：

```
const obj = {
  name: "why",
  age: 18,
  friend: {…
  },
  hobbies: ["篮球", "足球", "乒乓球"],
  toJSON: function() {
    return "coderwhy"
  }
}
```

```
const objString5 = JSON.stringify(obj)
console.log(objString5) // coderwhy
```

parse方法

- **JSON.parse()** 方法用来解析JSON字符串，构造由字符串描述的JavaScript值或对象。
  - 提供可选的 **reviver** 函数用以在返回之前对所得到的对象执行变换(操作)。

```
// 转回对象，并且转换某些值
const info2 = JSON.parse(objString, (key, value) => {
  if (key === "time") {
    return new Date(value)
  }
  return value
})
console.log(info2)
```

使用JSON序列化深拷贝

- 另外我们生成的新对象和之前的对象并不是同一个对象：
  - 相当于是进行了一次深拷贝；

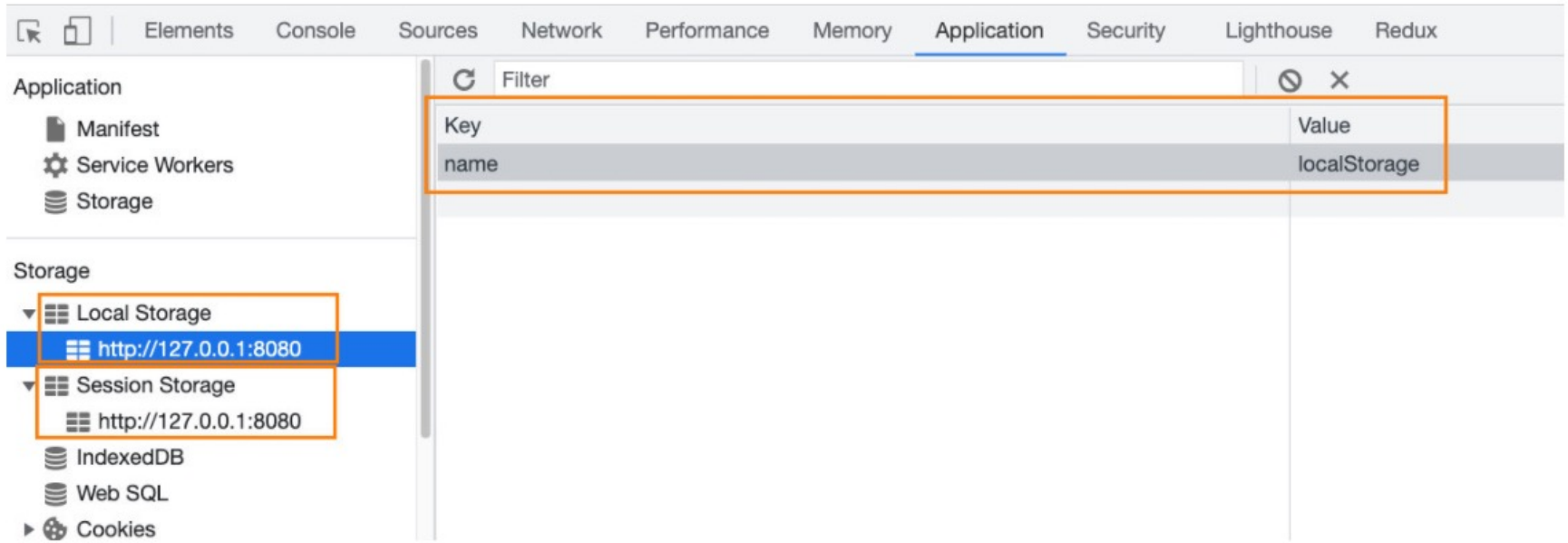
```
const objString = JSON.stringify(obj)
const info = JSON.parse(objString)
// { name: 'why', age: 18, friend: { name: 'kobe' } }
console.log(info)

console.log(info === obj) // false
info.friend.name = "james"
console.log(obj.friend.name) // kobe
```

- 注意：这种方法它对函数是无能为力的
  - 创建出来的info中是没有foo函数的，这是因为stringify并不会对函数进行处理；
  - 我们后续会讲解如何编写深拷贝的工具函数，那么这样就可以对函数的拷贝进行处理了；

认识Storage

- WebStorage主要提供了一种机制，可以让浏览器提供一种比cookie更直观的key、value存储方式：
  - localStorage：本地存储，提供的是一种永久性的存储方法，在关闭掉网页重新打开时，存储的内容依然保留；
  - sessionStorage：会话存储，提供的是本次会话的存储，在关闭掉会话时，存储的内容会被清除；

```
localStorage.setItem("name", "localStorage")
sessionStorage.setItem("name", "sessionStorage")
```



localStorage和sessionStorage的区别

- 我们会发现localStorage和sessionStorage看起来非常的相似。
- 那么它们有什么区别呢？
  - 验证一：关闭网页后重新打开，localStorage会保留，而sessionStorage会被删除；
  - 验证二：在页面内实现跳转，localStorage会保留，sessionStorage也会保留；
  - 验证三：在页面外实现跳转（打开新的网页），localStorage会保留，sessionStorage不会被保留；

Storage常见的方法和属性

- **Storage有如下的属性和方法：**
- 属性：
  - Storage.length：只读属性
    - ✓ 返回一个整数，表示存储在Storage对象中的数据项数量；
- 方法：
  - Storage.key()：该方法接受一个数值n作为参数，返回存储中的第n个key名称；
  - Storage.getItem()：该方法接受一个key作为参数，并且返回key对应的value；
  - Storage.setItem()：该方法接受一个key和value，并且将会把key和value添加到存储中。
    - ✓ 如果key存储，则更新其对应的值；
  - Storage.removeItem()：该方法接受一个key作为参数，并把该key从存储中删除；
  - Storage.clear()：该方法的作用是清空存储中的所有key；

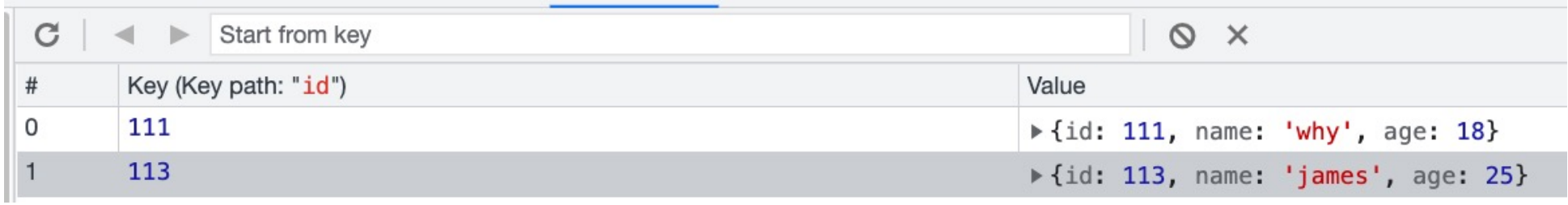
封装Storage

- 在开发中，为了让我们对Storage使用更加方便，我们可以对其进行一些封装：

```
class HYCache {
  constructor(isLocal) {
    this.storage = isLocal ? localStorage: sessionStorage
  }

  setItem(key, value) {
    this.storage.setItem(key, JSON.stringify(value))
  }

  getItem(key) {
    let value = this.storage.getItem(key)
    if (value) {
      value = JSON.parse(value)
    }
    return value
  }

  removeItem(key) {
    this.storage.removeItem(key)
  }

  removeItem(key) {
    this.storage.removeItem(key)
  }

  clear() {
    this.storage.clear()
  }

  key(index) {
    return this.storage.key(index)
  }

  length() {
    return this.storage.length
  }
}

const localCache = new HYCache(true)
const sessionCache = new HYCache(false)

export {
  localCache,
  sessionCache
}
```

认识IndexedDB

- 什么是IndexedDB呢？
  - 我们能看到DB这个词，就说明它其实是一种数据库（Database），通常情况下在服务器端比较常见；
  - 在实际的开发中，大量的数据都是存储在数据库的，客户端主要是请求这些数据并且展示；
  - 有时候我们可能会存储一些简单的数据到本地（浏览器中），比如token、用户名、密码、用户信息等，比较少存储大量的数据；
  - 那么如果确实有大量的数据需要存储，这个时候可以选择使用IndexedDB；
- IndexedDB是一种底层的API，用于在客户端存储大量的结构化数据。
  - 它是一种事务型数据库系统，是一种基于JavaScript面向对象数据库，有点类似于NoSQL（非关系型数据库）；
  - IndexDB本身就是基于事务的，我们只需要指定数据库模式，打开与数据库的连接，然后检索和更新一系列事务即可；



IndexDB的连接数据库

- 第一步：打开indexDB的某一个数据库；
  - 通过indexDB.open(数据库名称, 数据库版本)方法；
  - 如果数据库不存在，那么会创建这个数据；
  - 如果数据库已经存在，那么会打开这个数据库；
- 第二步：通过监听回调得到数据库连接结果；
  - 数据库的open方法会得到一个**IDBOpenDBRequest**类型
  - 我们可以通过下面的三个回调来确定结果：
    - ✓ onerror：当数据库连接失败时；
    - ✓ onsuccess：当数据库连接成功时回调；
    - ✓ onupgradeneeded：当数据库的version发生变化并且高于之前版本时回调；
      - ➢ 通常我们在这里会创建具体的存储对象：db.createObjectStore(存储对象名称, { keypath: 存储的主键 })
  - 我们可以通过onsuccess回调的event获取到db对象：event.target.result

IndexedDB的数据库操作

- 我们对数据库的操作要通过事务对象来完成：
  - 第一步：通过db获取对应存储的事务 db.transaction(存储名称, 可写操作)；
  - 第二步：通过事务获取对应的存储对象 transaction.objectStore(存储名称)；
- 接下来我们就可以进行增删改查操作了：
  - 新增数据 store.add
  - 查询数据
    - ✓ 方式一：store.get(key)
    - ✓ 方式二：通过 store.openCursor 拿到游标对象
      - ➢ 在request.onsuccess中获取cursor：event.target.result
      - ➢ 获取对应的key：cursor.key；
      - ➢ 获取对应的value：cursor.value；
      - ➢ 可以通过cursor.continue来继续执行；
  - 修改数据 cursor.update(value)
  - 删除数据 cursor.delete()

IndexedDB操作的代码

```
const transaction = db.transaction("students", "readwrite")
const store = transaction.objectStore("students")
```

```
for (const stu of students) {
  store.add(stu)
}
transaction.oncomplete = function(event) {
  console.log("添加完成", event)
}
```

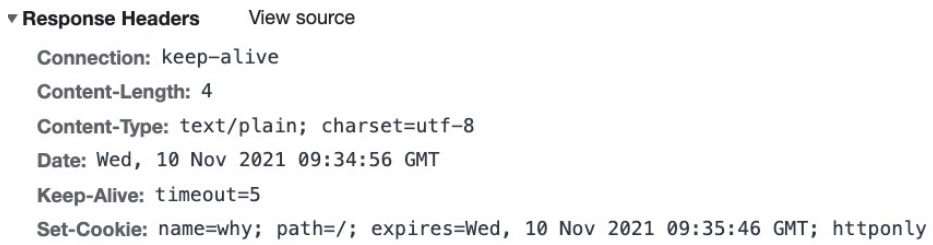
```
const deleteRequest = store.openCursor()
deleteRequest.onsuccess = event => {
  const cursor = event.target.result
  if (cursor) {
    if (cursor.key === 112) {
      cursor.delete()
    } else {
      cursor.continue()
    }
  }
}
```

```
const updateRequest = store.openCursor()
updateRequest.onsuccess = event => {
  const cursor = event.target.result
  if (cursor) {
    if (cursor.key === 112) {
      const value = cursor.value
      value.age = 20
      cursor.update(value)
    } else {
      cursor.continue()
    }
  }
}
```

```
// 1.单个查询
const request = store.get(111)
request.onsuccess = (event) => {
  console.log(event.target.result)
}

// 2.多个查询
const request = store.openCursor()
request.onsuccess = event => {
  const cursor = event.target.result
  if (cursor) {
    console.log(cursor.key, cursor.value)
    cursor.continue()
  } else {
    console.log("查询数据完成")
  }
}
```

认识Cookie

- **Cookie**（复数形态Cookies），又称为“小甜饼”。类型为“**小型文本文件**，某些网站为了辨别用户身份而存储在用户本地终端（Client Side）上的数据。
  - 浏览器会在特定的情况下携带上cookie来发送请求，我们可以通过cookie来获取一些信息；
- Cookie总是保存在客户端中，按在客户端中的存储位置，Cookie可以分为内存Cookie和硬盘Cookie。
  - 内存Cookie由浏览器维护，保存在内存中，浏览器关闭时Cookie就会消失，其存在时间是短暂的；
  - 硬盘Cookie保存在硬盘中，有一个过期时间，用户手动清理或者过期时间到时，才会被清理；
- 如果判断一个cookie是内存cookie还是硬盘cookie呢？
  - 没有设置过期时间，默认情况下cookie是内存cookie，在关闭浏览器时会自动删除；
  - 有设置过期时间，并且过期时间不为0或者负数的cookie，是硬盘cookie，需要手动或者到期时，才会删除；



cookie常见的属性

- **cookie的生命周期：**
  - 默认情况下的cookie是内存cookie，也称之为会话cookie，也就是在浏览器关闭时会自动被删除；
  - 我们可以通过设置*expires*或者*max-age*来设置过期的时间；
    - expires：设置的是Date.toUTCString()，设置格式是;expires=date-in-GMTString-format；
    - max-age：设置过期的秒钟，;max-age=max-age-in-seconds (例如一年为60*60*24*365)；
- **cookie的作用域：（允许cookie发送给哪些URL）**
  - Domain：指定哪些主机可以接受cookie
    - 如果不指定，那么默认是 origin，不包括子域名。
    - 如果指定Domain，则包含子域名。例如，如果设置 Domain=mozilla.org，则 Cookie 也包含在子域名中（如developer.mozilla.org）。
  - Path：指定主机下哪些路径可以接受cookie
    - 例如，设置 Path=/docs，则以下地址都会匹配：
      - /docs
      - /docs/Web/
      - /docs/Web/HTTP

客户端设置cookie

- js直接设置和获取cookie：

```
console.log(document.cookie);
```

- 这个cookie会在会话关闭时被删除掉；

```
// 设置过期时间就是本地cookie,不设置就是内存cookie
document.cookie = "name=coderwhy";
document.cookie = "age=18";
```

- 设置cookie，同时设置过期时间（默认单位是秒钟）

```
document.cookie = "name=coderwhy;max-age=10";
```