Promise

Promise

-
-
-
- Promise
- Promise

- ChatGPTChatGPT
-
- LRUEventEmitter

```javascript
function debounce(func, delay) {
  let timerId;
  return function(...args) {
    if (timerId) {
      clearTimeout(timerId);
    }
    timerId = setTimeout(() => {
      func.apply(this, args);
    }, delay);
  }
}
```

```javascript
function throttle(func, delay) {
  let lastTime = 0;
  return function(...args) {
    const currentTime = Date.now();
    if (currentTime - lastTime >= delay) {
      func.apply(this, args);
      lastTime = currentTime;
    }
  }
}
```

EventEmitter

EventEmitterNodejs

```javascript
class EventEmitter {
  constructor() {
    this.events = {}; //
  }

  //
  on(event, listener) {
    if (!this.events[event]) {
      this.events[event] = [];
    }
    this.events[event].push(listener);
  }

  //
  emit(event, ...args) {
    const listeners = this.events[event] || [];
    listeners.forEach((listener) => listener(...args));
  }

  //
  off(event, listener) {
    const listeners = this.events[event] || [];
    const index = listeners.indexOf(listener);
    if (index >= 0) {
      listeners.splice(index, 1);
    }
  }
}
```

EventEmitter

1. on(event, listener)eventlistener

2. emit(event, ...args)

3. off(event, listener)eventlistener

```javascript
const emitter = new EventEmitter();

//
emitter.on('hello', (name) => {
  console.log(`Hello, ${name}!`);
});

//
emitter.emit('hello', 'Tom'); // Hello, Tom!
```

Promise

Promise/A+ Promise

1. Promise "pending""fulfilled""rejected" Promise

2. Promise Promise

3. Promise then() Promise Promise Promise Promise Promise

4. Promise catch() then()

5. Promise Promise.all()Promise.race()Promise.resolve() Promise.reject()

6. then() Promise then() Promise then() Promise

```javascript
class MyPromise {
  constructor(executor) {
    this.state = 'pending';
    this.value = null;
    this.reason = null;
    this.onResolvedCallbacks = [];
    this.onRejectedCallbacks = [];

    const resolve = (value) => {
      if (this.state === 'pending') {
        this.state = 'fulfilled';
        this.value = value;
        this.onResolvedCallbacks.forEach(callback => callback(value));
      }
    }

    const reject = (reason) => {
      if (this.state === 'pending') {
        this.state = 'rejected';
        this.reason = reason;
        this.onRejectedCallbacks.forEach(callback => callback(reason));
      }
    }

    try {
      executor(resolve, reject);
    } catch (error) {
      reject(error);
    }
  }

  then(onResolved, onRejected) {
    onResolved = typeof onResolved === 'function' ? onResolved : value =>
value;
    onRejected = typeof onRejected === 'function' ? onRejected : reason => {
throw reason };
```

```javascript
  const promise = new MyPromise((resolve, reject) => {
    const handle = (callback, state) => {
      try {
        const result = callback(this.value);
        if (result instanceof MyPromise) {
          result.then(resolve, reject);
        } else {
          state(result);
        }
      } catch (error) {
        reject(error);
      }
    }

    if (this.state === 'fulfilled') {
      setTimeout(() => handle(onResolved, resolve), 0);
    } else if (this.state === 'rejected') {
      setTimeout(() => handle(onRejected, reject), 0);
    } else {
      this.onResolvedCallbacks.push(() => handle(onResolved, resolve));
      this.onRejectedCallbacks.push(() => handle(onRejected, reject));
    }
  });

  return promise;
}

catch(onRejected) {
  return this.then(null, onRejected);
}

static resolve(value) {
  return new MyPromise(resolve => resolve(value));
}

static reject(reason) {
  return new MyPromise((_, reject) => reject(reason));
}

static all(promises) {
  return new MyPromise((resolve, reject) => {
    const results = [];
    let count = 0;

    const handleResult = (index, value) => {
      results[index] = value;
      count++;
      if (count === promises.length) {
        resolve(results);
      }
    }
```

```
      for (let i = 0; i < promises.length; i++) {
        promises[i].then(value => handleResult(i, value), reject);
      }
    });
  }

  static race(promises) {
    return new MyPromise((resolve, reject) => {
      for (let i = 0; i < promises.length; i++) {
        promises[i].then(resolve, reject);
      }
    });
  }
}
```

LRU

```
class LRUCache {
  constructor(capacity) {
    this.capacity = capacity;
    this.cache = new Map();
  }

  get(key) {
    if (!this.cache.has(key)) {
      return -1;
    }
    const value = this.cache.get(key);
    this.cache.delete(key);
    this.cache.set(key, value);
    return value;
  }

  put(key, value) {
    if (this.cache.has(key)) {
      this.cache.delete(key);
    } else if (this.cache.size >= this.capacity) {
      const firstKey = this.cache.keys().next().value;
      this.cache.delete(firstKey);
    }
    this.cache.set(key, value);
  }
}
```

apply

```
Function.prototype.myApply = function(context, argsArray) {
  context = context || window;
  context.fn = this;
  let result;
  if (argsArray) {
    result = context.fn(...argsArray);
  } else {
    result = context.fn();
  }
  delete context.fn;
  return result;
}
```

bind

```
Function.prototype.myBind = function (context, ...args) {
  const fn = this;
  return function (...args2) {
    return fn.apply(context, [...args, ...args2]);
  };
};
```

call

```
Function.prototype.myCall = function (context, ...args) {
  const fn = Symbol("fn");
  context = context || window;
  context[fn] = this;
  const result = context[fn](...args);
  delete context[fn];
  return result;
};
```

Object.create

```
function createObject(proto) {
  function F() {}
  F.prototype = proto;
  return new F();
}
```

FprotoFFproto

instanceof

```
function myInstanceOf(obj, constructor) {
  //
  if (obj === null || typeof obj !== 'object') {
    return false;
  }

  //
  let proto = Object.getPrototypeOf(obj);

  //
  while (proto !== null) {
    if (proto === constructor.prototype) {
      return true;
    }
    proto = Object.getPrototypeOf(proto);
  }

  return false;
}
```

obj false   constructor.prototype true false

new

```
function myNew(constructor, ...args) {
  //
  const obj = Object.create(constructor.prototype);

  //  this
  const result = constructor.apply(obj, args);

  //
  return result instanceof Object ? result : obj;
}
```

Object.create this

```
function curry(fn) {
  return function curried(...args) {
    if (args.length >= fn.length) {
      return fn.apply(this, args);
    } else {
      return function(...moreArgs) {
        return curried.apply(this, args.concat(moreArgs));
      };
    }
  };
}
```

curry fn fn

Ajax

```javascript
function ajax(method, url, data, successCallback, errorCallback) {
  //  XMLHttpRequest
  const xhr = new XMLHttpRequest();

  //  readyState
  xhr.onreadystatechange = function() {
    if (xhr.readyState === 4) {
      if (xhr.status === 200) {
        //
        successCallback(xhr.responseText);
      } else {
        //
        errorCallback(xhr.status);
      }
    }
  };

  //
  xhr.open(method, url, true);

  //
  xhr.setRequestHeader("Content-Type", "application/json;charset=UTF-8");

  //
  xhr.send(data);
}
```

1. ajax methodurldatasuccessCallbackerrorCallback

2. XMLHttpRequest readyState readyState 4 200

3. open setRequestHeader send

1. Set

```javascript
function uniqueBySet(arr) {
  return [...new Set(arr)];
}
```

2. Array.reduce()

```
function uniqueByReduce(arr) {
  return arr.reduce((acc, cur) => {
    if (!acc.includes(cur)) {
      acc.push(cur);
    }
    return acc;
  }, []);
}
```

3. filter

```
function unique(arr) {
  return arr.filter((item, index, array) => {
    return array.indexOf(item) === index;
  });
}
```

JS

```
function flatten(arr) {
  return arr.reduce((prev, curr) => {
    return prev.concat(Array.isArray(curr) ? flatten(curr) : curr);
  }, []);
}
```

reduce flatten

```
function printTrafficLight() {
  const colors = ['' , '' , '' ];
  let index = 0;
  setInterval(() => {
    console.log(colors[index]);
    index = (index + 1) % colors.length;
  }, 1000);
}

printTrafficLight();
```

colors index setInterval 1 index 1 index 3 0

,

```javascript
function inheritPrototype(subType, superType) {
  const prototype = Object.create(superType.prototype);
  prototype.constructor = subType;
  subType.prototype = prototype;
}

function Animal(name) {
  this.name = name;
  this.colors = ['white', 'black'];
}

Animal.prototype.eat = function() {
  console.log(this.name + ' is eating.');
};

function Dog(name) {
  Animal.call(this, name);
  this.type = 'dog';
}

inheritPrototype(Dog, Animal);

Dog.prototype.bark = function() {
  console.log(this.name + ' is barking.');
};

const dog = new Dog('Snoopy');
dog.eat(); //  "Snoopy is eating."
dog.bark(); //  "Snoopy is barking."
```