师资培训课件 （ES6 题目）

1. 十三届省赛灯的颜色变化

```
let trafficlightsTimer = 3000;
// 红灯
function red() {
  return new Promise(function (resolve, reject) {
    setTimeout(() => {
      defaultlight.style.display = "none";
      redlight.style.display = "inline-block";
      resolve();
    }, trafficlightsTimer);
  });
}
// 绿灯
function green() {
  return new Promise(function (resolve, reject) {
    setTimeout(() => {
      redlight.style.display = "none";
      greenlight.style.display = "inline-block";
      resolve();
    }, trafficlightsTimer);
  });
}

// 红绿灯
async function trafficlights() {
  await red();
  await green();
```

```
}

trafficlights();
module.exports = { trafficlights };
```

2.收集帛书碎片

- 考察：数组方法、数组去重、扩展运算符
- 解题思路：入参是一个二维数组，转换成一维数组，去重

答案 1：

```
const result = [...new Set(puzzles.flat())];
return result;
```

答案 2：

```
let result = [];
for (let index = 0; index < puzzles.length; index++) {
  const item = puzzles[index];
  result.push(...item);
}
result = [...new Set(result)];
return result;
```

- 知识点解析： `flat()` 是 `JavaScript` 数组的一个方法，用于将多维数组扁平化为一维数组。

该方法可以接收一个整数参数，表示要扁平化的嵌套层数。例如，如果传递参数 2 ，则会将二维数组扁 平化为一维数组，但不会将三维及以上的数组扁平化。

如果不传递参数，则默认值扁平化一层。如果数组中有空位（即未定义的元素），则 `flat()` 方法默认会将其删除，返回一个新的不含空位的数组。

以下是 `flat()` 方法的示例用法：

```
const arr1 = [1, 2, [3, 4]];

arr1.flat(); // [1, 2, 3, 4]

const arr2 = [1, 2, [3, 4, [5, 6]]];
```

```
arr2.flat(); // [1, 2, 3, 4, [5, 6]]

const arr3 = [1, 2, [3, 4, [5, 6]]];

arr3.flat(2); // [1, 2, 3, 4, 5, 6]
```

在上面的示例中，`arr1` 和 `arr2` 数组中的嵌套数组都被扁平化为了一维数组。在 `arr3` 中，`flat(2)` 方法将嵌套数组扁平化了两层，生成了一个包含所有元素的一维数组。`Set` 是 `JavaScript` 中的一种数据结构，它类似于数组，但是它的值是唯一的，不会有重复的值。

在ES6中，"..."（三个连续的点）是一个扩展运算符（Spread Operator）的语法。它有以下几种主要的作用：

- 数组的展开：使用扩展运算符可以将一个数组展开为独立的元素。例如：

```
const arr = [1, 2, 3];
console.log(...arr); // 输出: 1 2 3
```

- 函数调用时的参数传递：扩展运算符可以用于将一个数组作为参数传递给函数，并展开为独立的参数。例如：

```
function sum(a, b, c) {
  return a + b + c;
}

const numbers = [1, 2, 3];
console.log(sum(...numbers)); // 输出: 6
```

- 对象的展开：扩展运算符可以用于将一个对象展开为另一个对象的属性。例如：

```
const obj1 = { x: 1, y: 2 };
const obj2 = { ...obj1, z: 3 };
console.log(obj2); // 输出: { x: 1, y: 2, z: 3 }
```

- 数组和对象的浅拷贝：通过扩展运算符可以创建数组和对象的浅拷贝。例如：

```
const arr = [1, 2, 3];
const arrCopy = [...arr];
```

```
const obj = { x: 1, y: 2 };
const objCopy = { ...obj };
```

需要注意的是，扩展运算符只能在可迭代对象（如数组）或具有可枚举属性的对象上使用。它提供了一种方便的方式来处理数组和对象的展开、合并和复制操作，使代码更加简洁和易读。

3.萌宠小玩家

- 考察： `class`、数组、dom 操作

```
// 验证是否已经起名
 verifyName() {
   this.name = nickname.value;
   if (!this.name) {
     vail_name.style.display = "block";
   } else {
     vail_name.style.display = "none";
   }
 }
 showLog(record) {
   if (this.logList.length == 10) {
     this.logList.pop();
     this.logList.unshift(record);
   } else {
     this.logList.unshift(record);
     console.log(this.logList);
   }
   let logHtmlStr = this.logList
     .map((item) => {
       return `<div>${item}</div>`;
     })
     .join("");
   list.innerHTML = logHtmlStr;
 }
```

高频考点知识点讲解：

高频考点知识点详细讲解带案例：

1. Promise:

Promise 是 JavaScript 中用于处理异步操作的一种机制。它可以更优雅地处理异步代码，避免了回调地狱。Promise 有三种状态：Pending（进行中）、Fulfilled（已成功）和 Rejected（已失败）。一旦 Promise 的状态发生变化，就会调用相应的处理程序。

```
// 创建一个简单的 Promise 示例
const myPromise = new Promise((resolve, reject) => {
  // 模拟异步操作
  setTimeout(() => {
    const randomNumber = Math.random();
    if (randomNumber > 0.5) {
      resolve(randomNumber);
    } else {
      reject('Error: Random number is less than or equal to 0.5');
    }
  }, 1000);
});

// 使用 Promise 的 then() 方法处理成功情况
myPromise.then((result) => {
  console.log('Promise resolved with result:', result);
}).catch((error) => {
  console.error('Promise rejected with error:', error);
});
```

2. class:

ES6 引入了 class 关键字，用于定义类。类可以包含构造函数和类方法，提供了一种更加面向对象的编程方式。类可以被继承，通过 `extends` 关键字实现继承。类方法可以使用 `static` 关键字定义为静态方法。

```
// 定义一个 Animal 类
class Animal {
  constructor(name) {
    this.name = name;
  }

  // 类方法
  speak() {
    console.log(`${this.name} makes a noise.`);
  }

  // 静态方法
  static info() {
    console.log('This is the Animal class.');
  }
}

// 继承 Animal 类并重写 speak() 方法
class Dog extends Animal {
  speak() {
```

```
    console.log(`${this.name} barks.`);
  }
}

// 创建 Dog 实例
const dog = new Dog('Rex');

// 调用方法
dog.speak(); // Output: Rex barks.
Animal.info(); // Output: This is the Animal class.
```

3. Set:

Set 是一种 JavaScript 的数据结构，用于存储唯一值，即不允许重复值。Set 中的值是无序的。通过 `add()` 方法可以向 Set 中添加值，通过 `delete()` 方法可以删除值，通过 `has()` 方法可以检查值是否存在。

```
// 创建一个 Set 示例
const mySet = new Set();

// 添加值
mySet.add(1);
mySet.add(2);
mySet.add(3);

console.log(mySet); // Output: Set(3) { 1, 2, 3 }

// 检查值是否存在
console.log(mySet.has(2)); // Output: true

// 删除值
mySet.delete(2);

console.log(mySet); // Output: Set(2) { 1, 3 }
```

4. Map:

Map 是一种 JavaScript 的数据结构，类似于对象，但是键可以是任意数据类型。Map 中的键值对是有序的。通过 `set()` 方法可以向 Map 中添加键值对，通过 `get()` 方法可以获取指定键的值，通过 `delete()` 方法可以删除指定键的键值对。

```
// 创建一个 Map 示例
const myMap = new Map();

// 添加键值对
myMap.set('a', 1);
myMap.set('b', 2);
myMap.set('c', 3);
```

```
console.log(myMap.get('b')); // Output: 2

// 删除指定键的键值对
myMap.delete('b');

console.log(myMap); // Output: Map(2) { 'a' => 1, 'c' => 3 }
```

5. 扩展运算符 `...`:

扩展运算符 `...` 可以将一个数组转为用逗号分隔的参数序列，或者将一个类数组对象转为真正的数组。它还可以用于对象的展开操作，将一个对象展开成多个键值对。

```
// 数组扩展
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5];

console.log(arr2); // Output: [1, 2, 3, 4, 5]

// 对象扩展
const obj1 = { a: 1, b: 2 };
const obj2 = { ...obj1, c: 3 };

console.log(obj2); // Output: { a: 1, b: 2, c: 3 }
```

6. proxy

`Proxy` 是 ES6 中引入的一种代理机制，用于创建一个对象的代理，可以拦截并修改该对象的底层操作。`Proxy` 对象接收两个参数：目标对象和处理程序对象（handler），处理程序对象中定义了拦截的行为，它是一个包含了在目标对象上定义拦截操作的方法的对象。

下面我们将详细讲解 `Proxy` 的使用方法，并提供一些案例来说明它的应用。

6. 创建 Proxy 对象

```
const target = {
  message: "Hello, world!"
};

const handler = {
  get: function(target, prop) {
    // 拦截读取属性的操作
    console.log(`Getting property "${prop}"`);
    return target[prop];
  },
  set: function(target, prop, value) {
```

```
    // 拦截设置属性的操作
    console.log(`Setting property "${prop}" to ${value}`);
    target[prop] = value;
    return true;
  }
};

const proxy = new Proxy(target, handler);

console.log(proxy.message); // Output: Getting property "message" // Hello,
world!

proxy.message = "Goodbye, world!"; // Output: Setting property "message" to
Goodbye, world!

console.log(proxy.message); // Output: Getting property "message" //
Goodbye, world!
```

2. Proxy 拦截操作

- `get(target, prop, receiver)`: 拦截对象属性的读取操作。
- `set(target, prop, value, receiver)`: 拦截对象属性的设置操作。
- `apply(target, thisArg, argumentsList)`: 拦截函数的调用、call 和 apply 操作。
- `construct(target, argumentsList, newTarget)`: 拦截类的构造函数，即使用 new 操作符创建实例的操作。
- 等等，共有 13 种拦截操作，详见 MDN 文档。

3. 应用案例

数据验证

```
const validator = {
  set: function(target, prop, value) {
    if (prop === 'age') {
      if (!Number.isInteger(value)) {
        throw new TypeError('Age must be an integer');
      }
      if (value < 0) {
        throw new TypeError('Age must be a positive integer');
      }
    }
    target[prop] = value;
    return true;
  }
};
```

```
const person = new Proxy({}, validator);

person.age = 30;
console.log(person.age); // Output: 30

person.age = 'thirty'; // Throws TypeError: Age must be an integer

person.age = -30; // Throws TypeError: Age must be a positive integer
```

记录日志

```
const logger = {
  get: function(target, prop) {
    console.log(`Reading property "${prop}"`);
    return target[prop];
  },
  set: function(target, prop, value) {
    console.log(`Setting property "${prop}" to ${value}`);
    target[prop] = value;
    return true;
  }
};

const data = {
  name: 'John',
  age: 30
};

const monitoredData = new Proxy(data, logger);

console.log(monitoredData.name); // Output: Reading property "name" // John

monitoredData.age = 35; // Output: Setting property "age" to 35
```

通过上面的例子，你可以看到 `Proxy` 的强大之处，它可以让你对目标对象的访问和修改进行控制和监视，从而实现更加灵活的数据操作。

ES6 考点，重点考点 promise（必考点）、class，扩展运算符, ES6 考题有可能单纯出现，另一种情况就是伴随别的题目出现，因为 es6 要掌握的知识点不多，备考的时候一定要把所有的知识点掌握好。