ES6高频面试题（30题）

1. 什么是ES6？它的主要特性有哪些？

参考答案：

ES6（ECMAScript 2015）是JavaScript的第六个版本，是JavaScript语言的重大更新。主要特性包括：

- let和const关键字
- 箭头函数
- 模板字符串
- 解构赋值
- 默认参数
- 扩展运算符
- Promise
- 类（Class）
- 模块化（import/export）
- Symbol数据类型
- Map和Set数据结构
- 生成器函数（Generator）

2. let、const和var的区别是什么？

参考答案：

- **作用域**：var是函数作用域，let和const是块级作用域
- **变量提升**：var存在变量提升，let和const存在暂时性死区
- **重复声明**：var允许重复声明，let和const不允许
- **赋值**：var和let可以重新赋值，const不能重新赋值（但对象内容可修改）
- **初始化**：var和let可以不初始化，const必须初始化

3. 什么是箭头函数？它与普通函数有什么区别？

参考答案：

箭头函数是ES6新增的函数定义方式，语法更简洁。主要区别：

- **this指向**：箭头函数没有自己的this，继承外层作用域的this
- **arguments对象**：箭头函数没有arguments对象
- **构造函数**：箭头函数不能作为构造函数使用
- **原型**：箭头函数没有prototype属性
- **变量提升**：箭头函数不存在变量提升

4. 什么是模板字符串？它有什么优势？

参考答案：

模板字符串使用反引号（`）包围，可以包含占位符（${expression}）。优势：

- 支持多行字符串
- 支持变量插值
- 支持表达式计算
- 更直观的字符串拼接
- 支持标签模板功能

5. 解构赋值是什么？请举例说明数组和对象的解构。

参考答案：

解构赋值是一种从数组或对象中提取值并赋给变量的语法。

代码块

```
// 数组解构
const [a, b, c] = [1, 2, 3];
const [x, , z] = [1, 2, 3]; // 跳过中间元素

```

```
// 对象解构
const {name, age} = {name: 'John', age: 30};
const {name: userName, age: userAge} = obj; // 重命名
```

6. 什么是扩展运算符（...）？它有哪些用途?

参考答案:

扩展运算符（...）可以将可迭代对象展开。用途包括:

- 数组展开: `[...arr1, ...arr2]`
- 对象展开: `{...obj1, ...obj2}`
- 函数参数展开: `func(...args)`
- 复制数组/对象: `[...arr]`、`{...obj}`
- 将类数组对象转为数组: `[...nodeList]`

7. 什么是剩余参数（rest parameters）？

参考答案:

剩余参数使用...语法，将多个参数收集到一个数组中:

代码块

```
function sum(...numbers) {
  return numbers.reduce((a, b) => a + b, 0);
}
sum(1, 2, 3, 4); // numbers = [1, 2, 3, 4]
```

剩余参数必须是最后一个参数，一个函数只能有一个剩余参数。

8. ES6中的默认参数是如何工作的?

参考答案:

ES6允许为函数参数设置默认值：

代码块

```
function greet(name = 'World') {
  return `Hello, ${name}!`;
}
```

- 只有当参数为undefined时才使用默认值
- 默认参数可以是表达式或函数调用
- 默认参数可以引用前面的参数
- 有默认值的参数应该放在后面

9. 什么是Promise？它解决了什么问题？

参考答案：

Promise是处理异步操作的对象，有三种状态：pending、fulfilled、rejected。

解决的问题：

- 回调地狱问题
- 错误处理困难
- 异步操作的组合和串联
- 提供了更好的异步编程模式

基本用法：

代码块

```
const promise = new Promise((resolve, reject) => {
  // 异步操作
});
promise.then(result => {}).catch(error => {});
```

10. ES6的类（Class）是如何工作的？

参考答案：

ES6的类是基于原型的语法糖：

```
代码块
1  class Person {
2    constructor(name) {
3      this.name = name;
4    }
5
6    sayHello() {
7      return `Hello, I'm ${this.name}`;
8    }
9
10   static getSpecies() {
11     return 'Homo sapiens';
12   }
13 }
```

- constructor定义构造函数
- 方法定义在原型上
- 支持静态方法
- 支持继承（extends）

11. ES6模块化（import/export）是如何工作的？

参考答案：

ES6提供了原生的模块化支持：

```
代码块
1  // 导出
2  export const name = 'John';
3  export function greet() {}
4  export default class Person {}
5
6  // 导入
7  import Person, { name, greet } from './module.js';
8  import * as utils from './utils.js';
```

- 静态导入导出
- 编译时确定依赖关系
- 支持默认导出和命名导出
- 自动严格模式

12. Symbol数据类型有什么特点和用途?

参考答案:

Symbol是ES6新增的原始数据类型，表示独一无二的值。特点:

- 每个Symbol值都是唯一的
- 不能与其他值进行运算
- 可以作为对象属性名
- 不会被常规方法遍历到

用途:

- 创建对象的私有属性
- 定义对象的元数据
- 避免属性名冲突
- 实现迭代器接口

13. Map和Set数据结构有什么特点?

参考答案:

Map特点:

- 键值对集合，键可以是任意类型
- 有序，按插入顺序遍历
- size属性获取大小
- 方法：set、get、has、delete、clear

Set特点:

- 值的集合，值唯一
- 有序，按插入顺序遍历
- size属性获取大小
- 方法：add、has、delete、clear
- 可用于数组去重

14. 什么是生成器函数（Generator）？

参考答案：

生成器函数是可以暂停和恢复执行的函数，使用function*定义：

```
代码块
function* generator() {
  yield 1;
  yield 2;
  return 3;
}

const gen = generator();
gen.next(); // {value: 1, done: false}
gen.next(); // {value: 2, done: false}
gen.next(); // {value: 3, done: true}
```

特点：

- 返回迭代器对象
- 可以暂停和恢复执行
- 用于实现异步编程
- 可以双向通信

15. 什么是迭代器（Iterator）和可迭代对象？

参考答案：

迭代器：实现了next()方法的对象，返回{value, done}格式的结果。

可迭代对象：实现了Symbol.iterator方法的对象。

内置可迭代对象：Array、String、Map、Set、arguments等。

```
代码块
1  const arr = [1, 2, 3];
2  const iterator = arr[Symbol.iterator]();
3  iterator.next(); // {value: 1, done: false}
```

16. for...of和for...in的区别是什么?

参考答案：

- **for...in**：遍历对象的可枚举属性名（包括继承的）
- **for...of**：遍历可迭代对象的值

```
代码块
1  const arr = [1, 2, 3];
2  arr.name = 'array';
3
4  for (let key in arr) {
5    console.log(key); // 0, 1, 2, name
6  }
7
8  for (let value of arr) {
9    console.log(value); // 1, 2, 3
10 }
```

17. 什么是Proxy？它有什么用途?

参考答案：

Proxy用于定义基本操作的自定义行为（如属性查找、赋值等）：

```
代码块
1   const proxy = new Proxy(target, handler);
```

用途：

- 属性访问拦截
- 函数调用拦截
- 数据验证
- 属性默认值
- 实现观察者模式
- Vue3的响应式原理

常用handler方法：get、set、has、deleteProperty等。

18. 什么是Reflect？它与Proxy有什么关系？

参考答案：

Reflect是一个内置对象，提供了拦截JavaScript操作的方法。

特点：

- 方法与Proxy handler方法一一对应
- 提供了默认的对象操作行为
- 返回值更合理（布尔值而不是抛出异常）

```
代码块
1   // 传统方式
2   delete obj.prop;
3
4   // Reflect方式
5   Reflect.deleteProperty(obj, 'prop'); // 返回boolean
```

与Proxy配合使用可以实现更好的元编程。

19. 什么是WeakMap和WeakSet？它们与Map、Set有什么区别？

参考答案：

WeakMap和WeakSet的特点：

- 键/值必须是对象
- 弱引用，不阻止垃圾回收
- 不可遍历，没有size属性
- 没有clear方法

与Map、Set的区别：

- 引用类型：弱引用 vs 强引用
- 垃圾回收：不阻止 vs 阻止
- 遍历性：不可遍历 vs 可遍历
- 用途：存储私有数据 vs 一般数据存储

20. 什么是暂时性死区（Temporal Dead Zone）？

参考答案：

暂时性死区是指let和const声明的变量在声明之前不能被访问的区域。

代码块

```
console.log(a); // ReferenceError
let a = 1;

function func() {
  console.log(b); // ReferenceError
  let b = 2;
```

```
7  }
```

特点：

- 从块级作用域开始到变量声明之前
- 访问会抛出ReferenceError
- typeof操作符也会报错
- 确保变量先声明后使用

21. ES6中的对象字面量有哪些增强？

参考答案：

ES6对象字面量增强包括：

1. **属性简写**：`{name, age}` 等价于 `{name: name, age: age}`
2. **方法简写**：`{method() {}}` 等价于 `{method: function() {}}`
3. **计算属性名**：`{[key]: value}`
4. **可以使用表达式作为属性名**
5. **super关键字**：在方法中调用原型方法

代码块

```
const name = 'John';
const obj = {
  name,                    // 属性简写
  greet() {},              // 方法简写
  [name + 'Key']: 'value' // 计算属性名
};
```

22. 什么是数组的新方法？请列举几个ES6新增的数组方法。

参考答案：

ES6为数组新增了多个方法：

- **Array.from()**：将类数组对象转为数组
- **Array.of()**：创建数组，避免Array构造函数的怪异行为
- **find()**：找到第一个满足条件的元素
- **findIndex()**：找到第一个满足条件的元素索引
- **includes()**：判断数组是否包含某个值
- **fill()**：用指定值填充数组
- **keys()、values()、entries()**：返回迭代器

23. 什么是字符串的新方法？ES6为字符串添加了哪些方法？

参考答案：

ES6为字符串新增的方法：

- **includes()**：判断是否包含子字符串
- **startsWith()**：判断是否以指定字符串开头
- **endsWith()**：判断是否以指定字符串结尾
- **repeat()**：重复字符串指定次数
- **padStart()**：在开头填充字符串到指定长度
- **padEnd()**：在末尾填充字符串到指定长度

代码块

```
'hello'.includes('ell'); // true
'hello'.startsWith('he'); // true
'hello'.repeat(2); // 'hellohello'
```

24. 什么是参数解构？请举例说明。

参考答案：

参数解构是在函数参数中直接解构传入的对象或数组：

代码块

```
// 对象参数解构
function greet({name, age = 18}) {
  console.log(`Hello ${name}, age ${age}`);
}
greet({name: 'John', age: 25});

// 数组参数解构
function sum([a, b]) {
  return a + b;
}
sum([1, 2]);

// 嵌套解构
function process({user: {name, email}}) {
  console.log(name, email);
}
```

25. 什么是尾调用优化？ES6如何支持尾调用优化？

参考答案：

尾调用是指函数的最后一步调用另一个函数。尾调用优化是指在尾调用时不增加新的栈帧，而是复用当前栈帧。

ES6在严格模式下支持尾调用优化：

代码块

```
'use strict';
function factorial(n, acc = 1) {
  if (n <= 1) return acc;
  return factorial(n - 1, n * acc); // 尾调用
}
```

条件：

- 严格模式
- 尾调用不访问当前栈帧的变量
- 尾调用是函数的最后一个操作

26. 什么是标签模板？它有什么用途？

参考答案：

标签模板是模板字符串的高级形式，允许用函数解析模板字符串：

代码块

```
function highlight(strings, ...values) {
  return strings.reduce((result, string, i) => {
    return result + string + (values[i] ? `<mark>${values[i]}</mark>` : '');
  }, '');
}

const name = 'John';
const age = 25;
const result = highlight`Hello ${name}, you are ${age} years old`;
```

用途：

- 字符串处理和转换
- 模板引擎
- 国际化
- 防止XSS攻击

27. ES6中的正则表达式有哪些新特性？

参考答案：

ES6为正则表达式添加了新特性：

1. **u标志**：Unicode模式，正确处理大于\uFFFF的Unicode字符
2. **y标志**：粘连模式，从lastIndex位置开始匹配
3. **RegExp构造函数增强**：可以接受正则表达式作为参数
4. **新方法**：flags属性获取修饰符

代码块

```
const regex = /\u{1D306}/u; // Unicode模式
const sticky = /hello/y;    // 粘连模式
regex.flags; // 'u'
```

28. 什么是二进制和八进制字面量？

参考答案：

ES6提供了二进制和八进制数值的新写法：

代码块

```
// 二进制（0b或0B开头）
const binary = 0b1010; // 10
const binary2 = 0B1010; // 10

// 八进制（0o或0O开头）
const octal = 0o755; // 493
const octal2 = 0O755; // 493

// 检测方法
Number.isFinite(binary); // true
Number.isInteger(octal); // true
```

注意：严格模式下不支持0开头的八进制写法。

29. ES6中的Number对象有哪些新方法?

参考答案:

ES6为Number对象新增了多个方法:

- **Number.isFinite()**:检测数值是否有限
- **Number.isNaN()**:检测是否为NaN
- **Number.parseInt()**:解析整数
- **Number.parseFloat()**:解析浮点数
- **Number.isInteger()**:判断是否为整数
- **Number.isSafeInteger()**:判断是否在安全整数范围内
- **Number.EPSILON**:最小精度常量
- **Number.MAX_SAFE_INTEGER/MIN_SAFE_INTEGER**:安全整数范围

30. 什么是Object.assign()?它有什么特点和用途?

参考答案:

Object.assign()用于将源对象的可枚举属性复制到目标对象:

代码块

```
const target = {a: 1};
const source = {b: 2, c: 3};
Object.assign(target, source); // {a: 1, b: 2, c: 3}
```

特点:

- 浅拷贝,不会拷贝继承属性和不可枚举属性
- 同名属性会被覆盖
- 返回目标对象
- 可以接受多个源对象

用途：

- 对象合并
- 对象克隆
- 为对象添加属性
- 为对象添加方法