29. 什么是原型？什么是原型链?

1. 核心概念 (Core Concept)

在 JavaScript 中，原型（Prototype）是一种机制，通过它对象可以从其他对象继承属性和方法。原型链（Prototype Chain）是这种继承关系形成的链式结构，用于实现属性查找和继承。

2. 为什么需要它？ (The "Why")

1. **实现继承:** 它是 JavaScript 实现对象继承的主要方式，使得对象可以访问父级/原型对象上的属性和方法，避免代码重复。
2. **节省内存:** 共享原型对象上的方法可以有效地节省内存，特别是当有大量相同类型的对象实例时。
3. **动态性:** 原型链是动态的，可以在运行时修改原型对象，从而影响所有继承自该原型的实例。

3. API 与用法 (API & Usage)

JavaScript 中与原型相关的关键 API 和属性包括：

- **`[[Prototype]]` (或 `__proto__`)**: 对象的内部属性，指向创建该对象的构造函数的 `prototype` 属性。它是构成原型链的链接。虽然 `__proto__` 广泛使用，但在 ES6 后推荐使用 `Object.getPrototypeOf()` 和 `Object.setPrototypeOf()`。
- **`Constructor.prototype`**: 这是一个常规属性，每个函数（包括构造函数）都会自动获得一个 `prototype` 属性，这个属性的值是一个对象，所有由这个构造函数创建的实例都会将它们的 `[[Prototype]]` 指向这个对象。
- **`Object.getPrototypeOf(obj)`**: ES6 推荐的方法，用于获取指定对象的 `[[Prototype]]`（即它的原型）。
- **`Object.setPrototypeOf(obj, prototype)`**: ES6 推荐的方法，用于设置一个对象的 `[[Prototype]]`。不推荐在性能敏感的代码中使用，因为它可能导致优化问题。
- **`Object.create(prototype, propertiesObject)`**: 创建一个新对象，并将该新对象的 `[[Prototype]]` 指定为 `prototype` 参数。可以用来实现基于原型的纯粹继承。

代码示例:

```
// 构造函数
function Person(name) {
  this.name = name;
}

// 在 Person 的原型上添加方法
Person.prototype.sayHello = function() {
```

```
  console.log(`Hello, my name is ${this.name}`);
};

// 创建 Person 的实例
const person1 = new Person('Alice');
const person2 = new Person('Bob');

// 实例可以访问原型上的方法
person1.sayHello(); // 输出: Hello, my name is Alice
person2.sayHello(); // 输出: Hello, my name is Bob

// 访问对象的原型
console.log(
  Object.getPrototypeOf(person1) === Person.prototype // true
);

// 原型链示例: Person.prototype 的原型是 Object.prototype
console.log(
    Object.getPrototypeOf(Person.prototype) === Object.prototype // true
);

// Object.prototype 的原型是 null, 原型链的终点
console.log(
    Object.getPrototypeOf(Object.prototype) === null // true
);

// 使用 Object.create 创建对象并指定原型
const obj = Object.create( { x: 1, y: 2 } );
console.log(obj.x); // 输出: 1 (从原型继承)
console.log(Object.getPrototypeOf(obj)); // 输出: { x: 1, y: 2 }
```

4. 关键注意事项 (Key Considerations)

1. **属性查找顺序:** 当访问对象的属性时，JavaScript 引擎会首先在该对象自身查找。如果找不到，就会沿着 `[[Prototype]]` 指向的原型对象继续查找，直到找到该属性或到达原型链的终点 (`null`)。如果在原型链的任何位置找到属性，就返回该属性的值。如果整个链条上都找不到，则返回 `undefined` 。
2. **`this` 的指向:** 原型方法中的 `this` 仍然指向调用该方法的对象实例，而不是原型对象本身。
3. **不要直接修改 `__proto__` :** 尽管可以使用 `__proto__` 来访问或设置对象的原型，但直接修改它的性能开销较大，且可能导致难以预测的行为。ES6 提供了 `Object.getPrototypeOf()` 和 `Object.setPrototypeOf()` 作为更规范的方式，但 `setPrototypeOf` 在频繁使用时仍需谨慎考虑性能。
4. **原型链的终点:** 所有内置对象和大部分自定义对象的原型链最终都会指向 `Object.prototype` ，而 `Object.prototype` 的 `[[Prototype]]` 是 `null` ，标志着原型链的结束。

5. 参考资料 (References)

- MDN Web Docs - Inheritance and the prototype chain
- JavaScript.info - Prototypal inheritance