传统函数与箭头函数区别

1. 基本写法区别

传统函数声明

```
function foo() {
  // 函数体
}
```

箭头函数

```
const foo = () => {
  // 函数体
}
```

2. 语法/特性对比

特性	function foo() {}	const foo = () => {}
this 绑定	动态，调用时决定	静态，定义时决定（继承外层）
arguments 对象	有	没有（需用 ...rest）
构造函数（new）	可以作为构造函数	不能作为构造函数
prototype 属性	有	没有
重名提升	有函数提升	没有提升
书写简洁	普通	更简洁
适合场景	需要自己的 this/arguments	需要继承外层 this 的回调等

3. 重点差异详细讲解

① this 指向

- **function** 声明的函数，`this` 会根据调用方式动态绑定；
- **箭头函数** 没有自己的 `this`，它的 `this` 取决于定义它时的外层作用域。

示例：

```
const obj = {
  value: 42,
  foo: function() {
    console.log(this.value);
  },
  bar: () => {
    console.log(this.value);
  }
};
obj.foo(); // 42
obj.bar(); // undefined （因为箭头函数的 this 不是 obj，而是外部作用域的 this）
```

---

② arguments

- **function** 拥有自己的 `arguments` 对象；
- **箭头函数** 没有 `arguments`，可用 rest 参数代替。

示例：

```
function test() {
  console.log(arguments);
}
test(1, 2, 3); // [1, 2, 3]

const test2 = (...args) => {
  console.log(args);
};
test2(1, 2, 3); // [1, 2, 3]
```

---

③ 不能用作构造函数

```
function Foo() {}
const foo = new Foo(); // OK

const Bar = () => {};
const bar = new Bar(); // TypeError: Bar is not a constructor
```

④ 没有 prototype

```
function Foo() {}
console.log(Foo.prototype); // 有 prototype

const Bar = () => {};
console.log(Bar.prototype); // undefined
```

⑤ 不能提升（Hoisting）

- function 声明会提升；
- 箭头函数（变量赋值）不会提升。

4. 什么时候用哪种?

- **普通 function**：需要自己的 `this`、`arguments` 或用作构造函数时。
- **箭头函数**：用作回调、内部函数，不需要自己的 this，喜欢简洁写法时。

5. 总结一句话

> 箭头函数没有自己的 this、arguments，不能用作构造函数，也没有 prototype，语法更简洁，适合回调和闭包；传统 function 则适合需要自己 this 的场景。