Daily_Learning(2019-11-25)

输出以下结果

```
1 + "1"

2 * "2"

[1, 2] + [2, 1]

"a" + + "b"
```

- 1 + "1"

加性操作符：如果只有一个操作数是字符串，则将另一个操作数转换为字符串，然后再将两个字符串拼接起来

所以值为： “11”

- 2 * "2"

乘性操作符：如果有一个操作数不是数值，则在后台调用 Number()将其转换为数值

4

- [1, 2] + [2, 1]

Javascript中所有对象基本都是先调用valueOf方法，如果不是数值，再调用toString方法。

所以两个数组对象的toString方法相加，值为："1,22,1"

- "a" + + "b"

后边的“+”将作为一元操作符，如果操作数是字符串，将调用Number方法将该操作数转为数值，如果操作数无法转为数值，则为NaN。

所以值为："aNaN"

写出打印结果

```
function Foo() {
Foo.a = function() {
console.log(1)
}
this.a = function() {
console.log(2)
}
}
Foo.prototype.a = function() {
console.log(3)
}
Foo.a = function() {
console.log(4)
}
Foo.a();
let obj = new Foo();
obj.a();
Foo.a();
```

```
function Foo() {
    Foo.a = function() {
        console.log(1)
    }
    this.a = function() {
        console.log(2)
    }
}
// 以上只是 Foo 的构建方法，没有产生实例，此刻也没有执行

Foo.prototype.a = function() {
    console.log(3)
}
// 现在在 Foo 上挂载了原型方法 a ，方法输出值为 3

Foo.a = function() {
    console.log(4)
```

```
}
// 现在在 Foo 上挂载了直接方法 a ，输出值为 4

Foo.a();
// 立刻执行了 Foo 上的 a 方法，也就是刚刚定义的，所以
// # 输出 4

let obj = new Foo();
/* 这里调用了 Foo 的构建方法。Foo 的构建方法主要做了两件事：
1. 将全局的 Foo 上的直接方法 a 替换为一个输出 1 的方法。
2. 在新对象上挂载直接方法 a ，输出值为 2。
*/

obj.a();
// 因为有直接方法 a ，不需要去访问原型链，所以使用的是构建方法里所定义的 this.a，
// # 输出 2

Foo.a();
// 构建方法里已经替换了全局 Foo 上的 a 方法，所以
// # 输出 1
```