你不知道的JS：作用域与闭包

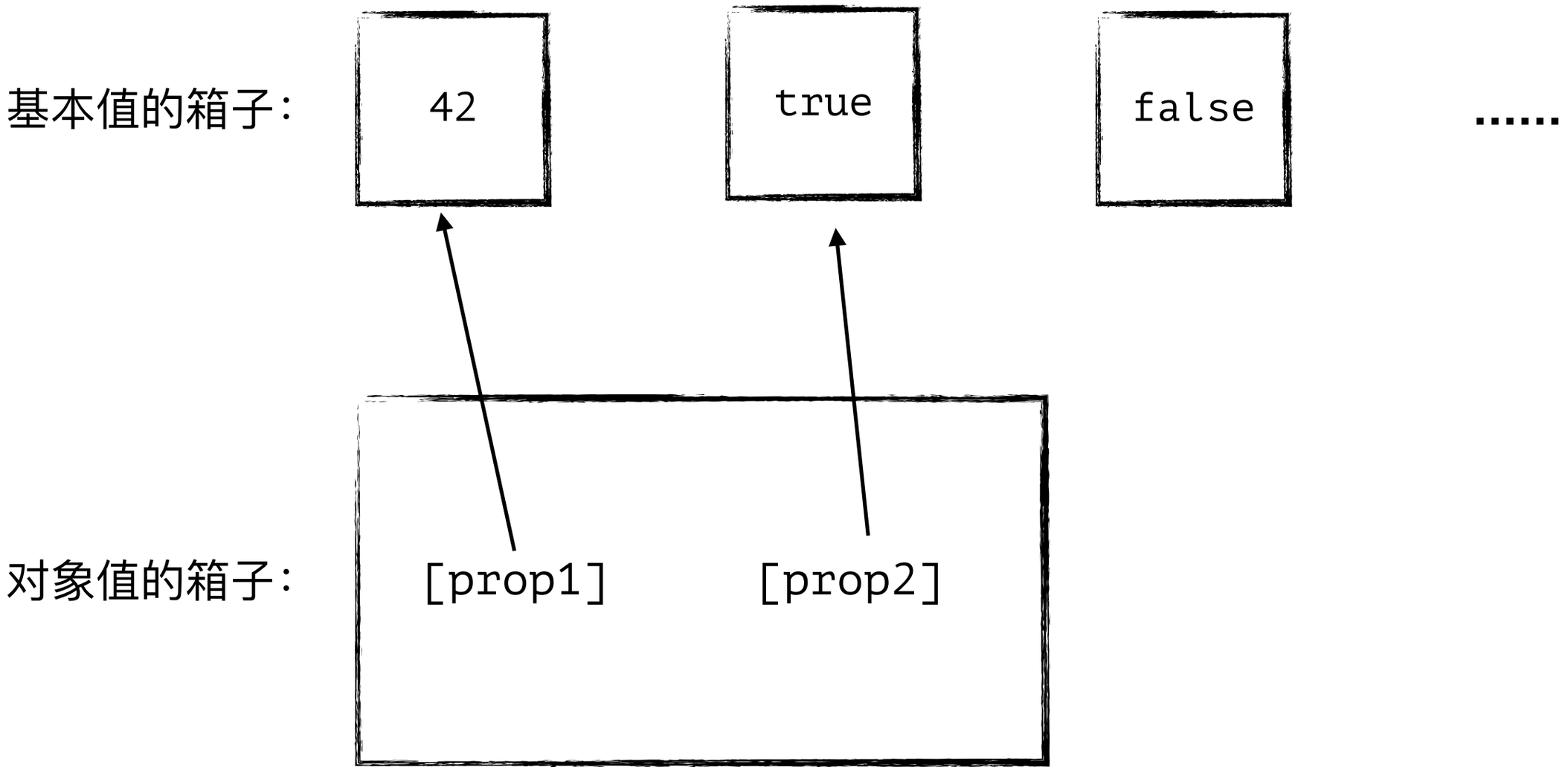
by 丁乐华

目录

变量

- 值(value)：代表的内容
- 标识符(identifier)：指代某个值的名称
- 变量(variable)：标识符 + 值

```
let var1, var2;
var1 = 42;
var2 = var1;
var1 = {
    prop1: 42,
    prop2: true
};
var1.prop2 = false;
```


基本值的箱子:
42
true
false
.....
对象值的箱子:
[prop1]
[prop2]
标识符:
var1
var2

变量可以做什么

- 复用同一个值

- 复用同一个操作（与函数一起合作）

```
1 + 2 + 3 + 4 + 5; // 15
1 + 3 + 5 + 7 + 9; // 25
```

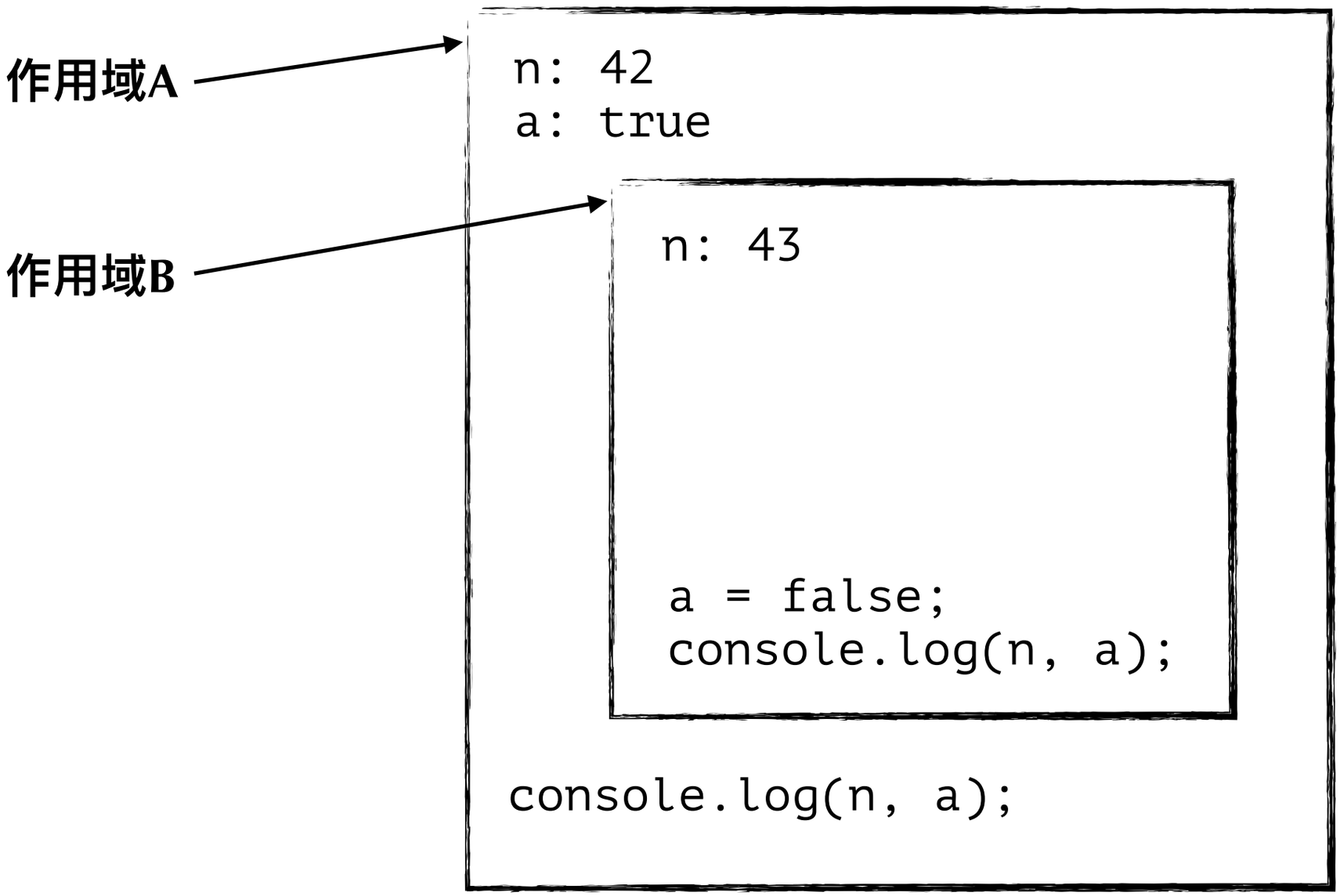
```
let getSumOf = (nArr) => {
    let sum = 0;
    let len = nArr.length;
    for (let i = 0; i < len; i += 1) {
        sum += nArr[i];
    }
    return sum;
}

getSumOf([1, 2, 3, 4, 5]); // 15
getSumOf([1, 3, 5, 7, 9]); // 25
```

作用域

- 作用域(scope)：管理变量，即管理标识符到值的映射

- 作用域可以进行层级嵌套

```
let n = 42;
let a = true;
(() => {
    let n = 43;
    a = false;
    console.log(n, a); // 43, false
})();
console.log(n, a); // 42, false
```


作用域A
n: 42
a: true
作用域B
n: 43
a = false;
console.log(n, a);
console.log(n, a);

词法作用域

- 自由变量：不能在所处的作用域中找到的变量
- 词法作用域(lexical scope)：某个变量从哪个作用域中查找是在编写代码的时候(author-time)决定的，跟代码在哪里运行无关

```
let a = 42;
let add_a = (b) => a + b;

add_a(5); // 47

{
    let a = 0;
    add_a(5); // 47
}
```

作用域A

```
a: 42
add_a: f...

add_a(5);
```

```
b: 5

return a + b;
```

定义时C的外层作用域是A

```
b: x

return a + b;
```

作用域C的模版，即add_a

作用域B

```
a: 0

add_a(5);
```

```
b: 5

return a + b;
```

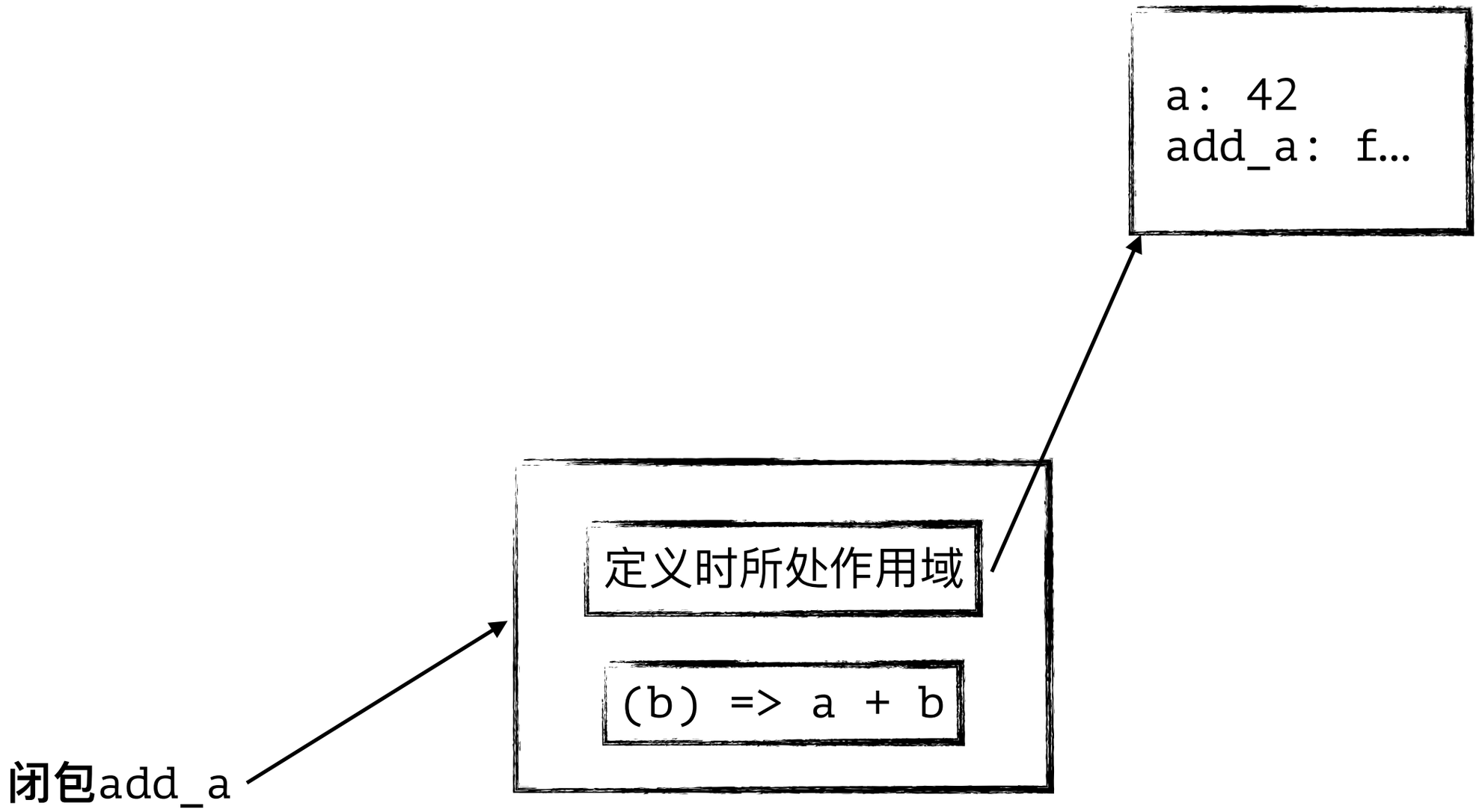
作用域解决了什么

- 变量重名(name resolution)
- 最小化暴露点(least privilege)
- 存储私密信息(private variable)

闭包

函数 + 函数定义时所处作用域 = 闭包(closure)

```
let a = 42;
let add_a = (b) => a + b;

add_a(5); // 47

{
    let a = 0;
    add_a(5); // 47
}
```


a: 42
add_a: f…
定义时所处作用域
(b) => a + b
闭包add_a

“对象是附有行为的数据，而闭包是附有数据的行为”

为什么会出现闭包

因遵守词法作用域机制所带来的结果

闭包是实现词法作用域的一种方式

资源释放问题

```
(() => {
    let alloc_256M = () => {
        // Chrome: 256M, *使用1字节
        return new Array(268435440 + 1).join('*');
    }
    let bigData = [
        alloc_256M(), alloc_256M(),
        alloc_256M(), alloc_256M()
    ];
    document.addEventListener('click', function handler() {
        console.log(bigData.length);
        setTimeout(() => {
            document.removeEventListener('click', handler);
        }, 5000);
    });
})();
```

Summary ▼ | Class filter | All objects

Profiles

HEAP SNAPSHOTS

Snapshot 1 Save
1031 MB

Constructor	Dist…	Objects Count	Shallow Size	Retained Size
▶(closure)	–	6 832 9 %	491 496 0 %	075 052 360 99 %
▶system / Con…	3	478 1 %	53 600 0 %	074 528 520 99 %
▼(string)	–	5 514 7 %	1 074 084 048 99 %	074 084 048 99 %
▶"**********	6		268 435 464 25 %	268 435 464 25 %
▶"**********	6		268 435 464 25 %	268 435 464 25 %
▶"**********	6		268 435 464 25 %	268 435 464 25 %
▶"**********	6		268 435 464 25 %	268 435 464 25 %
▶"(function(	7		22 336 0 %	22 336 0 %
▶"(function(	7		18 688 0 %	18 688 0 %
▶"(function(	7		18 080 0 %	18 080 0 %

Summary ▼ | Class filter | All objects

Profiles

HEAP SNAPSHOTS

Snapshot 1
1031 MB

Snapshot 2
1031 MB

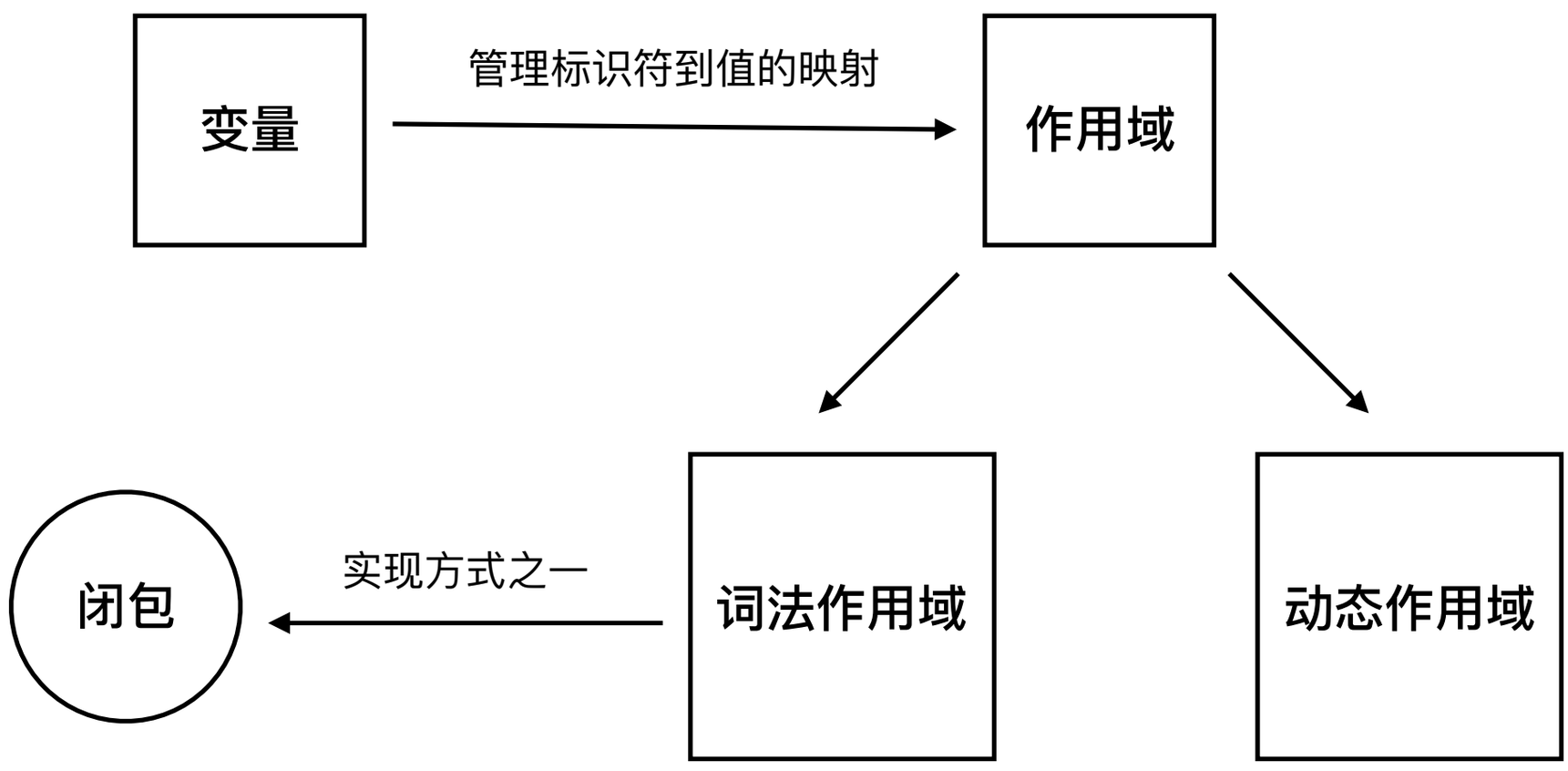
Snapshot 3
1031 MB

Snapshot 4
1031 MB

Snapshot 5 Save
7.0 MB

Constructor	Distance	Objects Count	Shallow Size	Retained Size
▶(system)	–	36 179 46 %	1 595 976 22 %	2 846 792 39 %
▶(array)	–	7 594 10 %	1 622 272 22 %	2 005 816 27 %
▶(compiled code)	3	3 532 4 %	1 088 848 15 %	1 794 992 25 %
▶(closure)	–	6 846 9 %	492 504 7 %	1 356 776 19 %
▶system / Context	3	477 1 %	53 544 1 %	857 864 12 %
▶Object	–	3 376 4 %	132 624 2 %	626 712 9 %
▼(string)	–	5 615 7 %	463 864 6 %	463 864 6 %
▶"(function(de	7		22 336 0 %	22 336 0 %
▶"(function(de	8		22 336 0 %	22 336 0 %
▶"(function(de	7		18 688 0 %	18 688 0 %
▶"(function(de	8		18 688 0 %	18 688 0 %
▶"(function(de	7		18 080 0 %	18 080 0 %

总结



参考

- https://en.wikipedia.org/wiki/Variable_(computer_science)
- https://www.quora.com/What-is-the-need-of-variables-in-a-programming-language
- http://www.cs.utah.edu/~germain/PPS/Topics/variables.html
- http://www.yinwang.org/blog-cn/2016/06/08/java-value-type
- https://www.ibm.com/developerworks/cn/linux/l-cn-closure/

Thanks