
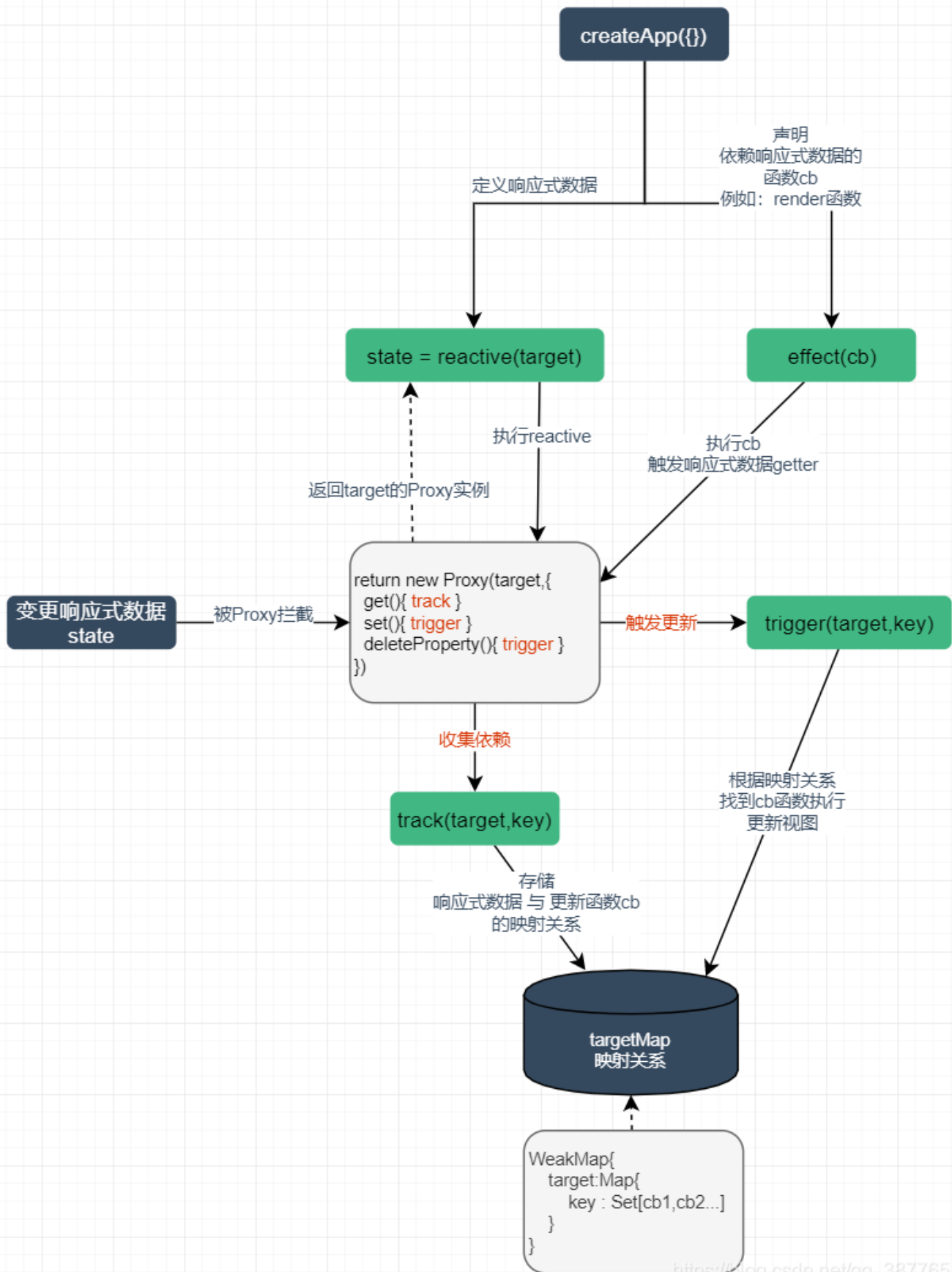


Vue3响应式原理

废话不多说,直接上图了





引导

先看这一段代码吧

```
1 let name = '小马哥', age = 22
2 let myself = `${name}今年${age}岁`
3 console.log(myself) // 小马哥今年22岁
4 age = 23 // 修改年龄
5 // 预期: 小马哥今年23岁
6 console.log(myself) // 实际: 小马哥今年22岁
```

这还用问吗?还需要解释吗?如果我想要`myself`跟随着`age`变化,如何才能完成呢?

其实,只要让`myself = `${name}今年${age}岁``再执行一次就行,如下

```
1 let name = '小马哥', age = 22, money = 20
2 let myself = `${name}今年${age}岁`
3 console.log(myself) // 小马哥今年22岁
4 age = 23 // 修改存款的金额
5 myself = `${name}今年${age}岁`
6 // 预期: 小马哥今年23岁
7 console.log(myself) // 实际: 小马哥今年23岁
```

好了,BMQ了,我就是做个引导性的代码,慢慢的把你代入到最开始的那张原理图上去

声明effect函数

```
1 let name = '小马哥', age = 22
2 let myself = ''
3 const effect = () => myself = `${name}今年${age}岁`
4 effect()
5 console.log(myself)
6 age = 23
7 effect()
8 console.log(myself) // 小马哥今年23岁
```

这段代码会引发额外的问题啊,如果我想输出打印多个,意味着我要写多个effect函数啊,这很不合理啊~

track(依赖)和trigger(触发)

再看看那张图

针对上面的问题,咱们可以这样解决:用track函数把所有依赖于age变量的effect函数都收集起来,放在dep里,dep为什么用Set呢?因为Set可以自动去重。搜集起来之后,以后只要age变量一改变,就执行trigger函数通知dep里所有依赖age变量的effect函数执行,实现依赖变量的更新。

```
1 let name = '小马哥', age = 22
2 let myself = '', otherMyself = ''
```

```
3  const effect1 = () => myself = `${name}今年
   ${age}岁`
4  const effect2 = () => otherMyself = `${age}岁的
   我名字叫${name}`
5
6  // 声明dep
7  const dep = new Set()
8  // track函数收集所有依赖于effect函数
9  function track() {
10     dep.add(effect1)
11     dep.add(effect2)
12 }
13 function trigger() {
14     dep.forEach(effect => effect())
15 }
16 // 先收集
17 track()
18 effect1()
19 effect2()
20 console.log(myself)
21 console.log(otherMyself)
22 age = 23
23 trigger() //通知变量myself和otherMyself进行更新
24 console.log(myself)
25 console.log(otherMyself)
```

上面我们只是针对基本数据类型做了依赖收集和触发更新,如果是对象呢?

对象处理怎么办?

```
1 let person = {
2   name: '小马哥',
3   age: 22
4 }
5 let nameUpdate = '', ageUpdate = ''
6 const effect1 = () => nameUpdate =
  `${person.name}是个开发也是个老师`
7 const effect2 = () => ageUpdate =
  `${person.age}岁的我名字叫小马哥老师`
8
9 let depsMap = new Map()
10 // track函数收集所有依赖于effect函数
11 function track(key) {
12   let deps = depsMap.get(key)
13   // 拿出栈顶函数
14   if (!deps) {
15     // 声明dep
16     depsMap.set(key, (deps = new Set()))
17   }
18   // 先写死,后面解决它
19   if (key === 'name') {
20     deps.add(effect1)
```

```

21     } else {
22         deps.add(effect2)
23     }
24 }
25 function trigger(key) {
26     const deps = depsMap.get(key)
27     if (deps)
28         deps.forEach(effect => effect())
29 }
30 // 先收集
31 track('name')
32 track('age')
33 effect1()
34 effect2()
35 console.log(nameUpdate)
36 console.log(ageUpdate)
37 person.name = '大马哥'
38 person.age = 30
39 trigger('name') //通知变量name进行更新
40 trigger('age') //通知变量age进行更新
41 console.log(nameUpdate)
42 console.log(ageUpdate)

```

虽然无脑,但我们还是实现了最原始的方式;如果是多个对象咋办啊

我们都知道, 每个对象会建立一个 `Map` 来存储此对象里属性的 `dep`(使用`Set`来存储), 那如果有多个对象, 该用什么来存储每个对象对应的 `Map` 呢?

```
WeakMap{  
  target: Map{  
    key : Set[cb1,cb2...]  
  }  
}
```

<https://blog.csdn.net>

```
1 let person = {  
2   name: '小马哥',  
3   age: 22  
4 }  
5 let animal = {  
6   type: "猫",  
7   age: 5  
8 }  
9 let personStr = '', animalStr = ''  
10 const effectObj = () => personStr =  
   `${person.name}今年${person.age}岁了`  
11 const effectAnimal = () => animalStr = `我家的  
   ${animal.type}叫胖胖,今年${animal.age}岁了`  
12  
13 let weakTargetObjMap = new WeakMap() //  
   [target: Map{key:Set[cb1,cb2]]]  
14
```



```
15 // let depsMap = new Map()
16 // track函数收集所有依赖于effect函数
17 function track(target, key) {
18
19     let depsMap = weakTargetObjMap.get(target)
20     if (!depsMap) {
21         weakTargetObjMap.set(target, (depsMap = new
22         Map()))
23     }
24
25     let deps = depsMap.get(key)
26     if (!deps) {
27         // 声明dep
28         depsMap.set(key, (deps = new Set()))
29     }
30
31     if (target === person) {
32         if (key === 'name') {
33             deps.add(effectObj)
34         } else {
35             deps.add(effectObj)
36         }
37     } else {
38         if (key === 'type') {
39             deps.add(effectAnimal)
40         } else {
41             deps.add(effectAnimal)
42         }
43     }
44 }
```

```
40     }
41   }
42
43 }
44 function trigger(target, key) {
45   console.log(weakTargetObjMap);
46   let depsMap = weakTargetObjMap.get(target)
47   if (depsMap) {
48     const deps = depsMap.get(key)
49     if (deps)
50       deps.forEach(effect => effect())
51   }
52
53 }
54 // 先收集
55 track(person, 'name')
56 track(person, 'age')
57 track(animal, 'type')
58 track(animal, 'age')
59 effectObj()
60 effectAnimal()
61 console.log(personStr)
62 console.log(animalStr)
63
64 person.name = '大马哥'
65 person.age = 30
```

```
66 animal.type = '狗狗'
67 animal.age = 1
68 trigger(person, 'name')
69 trigger(person, 'age')
70 trigger(animal, 'type')
71 trigger(animal, 'age')
72 console.log(personStr)
73 console.log(animalStr)
```

自动依赖收集和触发更新-Proxy

每次我们总是得自己手动去执行track函数进行依赖收集，并且当数据改变时，我们又得手动执行trigger函数去进行通知更新；Proxy可以为我们解决这个难题

```
1 let weakTargetObjMap = new WeakMap()
2
3 // track函数收集所有依赖于effect函数
4 function track(target, key) {
5
6     let depsMap = weakTargetObjMap.get(target)
7     if (!depsMap) {
8         weakTargetObjMap.set(target, (depsMap = new
9         Map()))
10    }
11    let deps = depsMap.get(key)
```

```
12     if (!deps) {
13         // 声明dep
14         depsMap.set(key, (deps = new Set()))
15     }
16     // 先写死
17     if (target === person) {
18         if (key === 'name') {
19             deps.add(effectObj)
20         } else {
21             deps.add(effectObj)
22         }
23     } else {
24         if (key === 'type') {
25             deps.add(effectAnimal)
26         } else {
27             deps.add(effectAnimal)
28         }
29     }
30
31 }
32 function trigger(target, key) {
33     let depsMap = weakTargetObjMap.get(target)
34     if (depsMap) {
35         const deps = depsMap.get(key)
36         if (deps)
37             deps.forEach(effect => effect())
```

```
38     }
39
40 }
41 function reactive(target) {
42     const handler = {
43         get(target, key, receiver) {
44             track(target, key) // 访问时收集依赖
45             return Reflect.get(target, key, receiver)
46         },
47         set(target, key, value, receiver) {
48             Reflect.set(target, key, value, receiver)
49             trigger(target, key) // 设值时自动通知更新
50         }
51     }
52
53     return new Proxy(target, handler)
54 }
55 let person = reactive({
56     name: '小马哥',
57     age: 22
58 })
59 let animal = reactive({
60     type: "猫",
61     age: 5
62 })
63 let personStr = '', animalStr = ''
```

```
64 const effectObj = () => personStr =  
  `${person.name}今年${person.age}岁了`  
65 const effectAnimal = () => animalStr = `我家的  
  ${animal.type}叫胖胖,今年${animal.age}岁了`  
66  
67  
68  
69 // 手动track收集  
70 // track(person, 'name')  
71 // track(person, 'age')  
72 // track(animal, 'type')  
73 // track(animal, 'age')  
74  
75 effectObj()  
76 effectAnimal()  
77 console.log(personStr)  
78 console.log(animalStr)  
79  
80 person.name = '大马哥'  
81 person.age = 30  
82 animal.type = '狗狗'  
83 animal.age = 1  
84 // 手动trigger触发更新  
85 // trigger(person, 'name')  
86 // trigger(person, 'age')  
87 // trigger(animal, 'type')
```

```
88 // trigger(animal, 'age')
89 console.log(personStr)
90 console.log(animalStr)
```

解决track函数中代码写死的问题

Vue3的作者们想出了一个非常巧妙的办法，使用一个全局变量`activeEffect`来巧妙解决这个问题，具体是怎么解决呢？其实很简单，就是每一个`effect`函数一执行，就把自身放到对应的`dep`里，这就可以不需要写死了。

```
1
2 let weakTargetObjMap = new WeakMap()
3 let activeEffct = null
4 function effect(fn) {
5   activeEffct = fn
6   activeEffct()
7   activeEffct = null
8 }
9 // track函数收集所有依赖于effect函数
10 function track(target, key) {
11   if (!activeEffct) return
12
13   let depsMap = weakTargetObjMap.get(target)
14   if (!depsMap) {
15     weakTargetObjMap.set(target, (depsMap = new
16     Map()))
17   }
```

```
17
18   let deps = depsMap.get(key)
19   if (!deps) {
20     // 声明dep
21     depsMap.set(key, (deps = new Set()))
22   }
23   // 先写死
24   // if (target === person) {
25   //   if (key === 'name') {
26   //     deps.add(effectObj)
27   //   } else {
28   //     deps.add(effectObj)
29   //   }
30   // } else {
31   //   if (key === 'type') {
32   //     deps.add(effectAnimal)
33   //   } else {
34   //     deps.add(effectAnimal)
35   //   }
36   // }
37   deps.add(activeEffct)
38
39 }
40 function trigger(target, key) {
41   let depsMap = weakTargetObjMap.get(target)
42   if (depsMap) {
```



```
43     const deps = depsMap.get(key)
44     if (deps)
45         deps.forEach(effect => effect())
46     }
47
48 }
49 function reactive(target) {
50     const handler = {
51         get(target, key, receiver) {
52             track(target, key) // 访问时收集依赖
53             return Reflect.get(target, key, receiver)
54         },
55         set(target, key, value, receiver) {
56             Reflect.set(target, key, value, receiver)
57             trigger(target, key) // 设值时自动通知更新
58         }
59     }
60
61     return new Proxy(target, handler)
62 }
63
64
65 // 实现ref
66 function ref(initVal) {
67     return reactive({
68         value: initVal
```

```
69   })
70 }
71 //实现computed
72 function computed(fn) {
73   const result = ref()
74   effect(() => result.value = fn()) // 执行
    computed传入函数
75   return result
76 }
77
78
79 let person = reactive({
80   name: '小马哥',
81   age: 22
82 })
83 let animal = reactive({
84   type: "猫",
85   age: 5
86 })
87 effect(() => {
88   console.log(person.name);
89   console.log(person.age);
90 })
91 effect(() => {
92   console.log(animal.type);
93   console.log(animal.age);
```

```
94  })  
95  person.name = '大马哥'  
96  person.age = 30  
97  animal.type = '狗狗'  
98  animal.age = 1  
99
```