一、v-for遍历数组

1.当我们有一组数据需要进行渲染时，我们就可以使用v-for来完成。

- v-for的语法类似于JavaScript的for循环
- 格式如下：item in items的形式。

2.简单案例：

```
<!-- 1.在遍历的过程中，没有使用索引值(下标值) -->
<ul>
    <li v-for="name in names">{{name}}</li>
</ul>

<!-- 2.在遍历的过程中，使用索引值 -->
<ul>
    <li v-for="(item,index) in names">{{index+1}}.{{item}}</li>
</ul>
```

3.如果在遍历的过程中不需要使用索引值

- v-for="movie in movies"
- 依次从movies中取出movie，并且在元素的内容中，我们班可以使用Mustache语法，来使用movie

4.如果在遍历的过程中，我们需要拿到元素在数组中的索引值呢?

- 语法格式：v-for=(item,index) in items
- 其中的index就代表了取出的item在原数组的索引值。

二、v-for遍历对象

1.v-for可以用户遍历对象：比如某个对象中存储着你的个人信息，我们希望以列表的形式展示出来

- why
- kobe
- james
- curry

- 1.why
- 2.kobe
- 3.james
- 4.curry

- why
- 18
- 1.88

- name:why
- age:18
- height:1.88

- 1.name:why
- 2.age:18
- 3.height:1.88

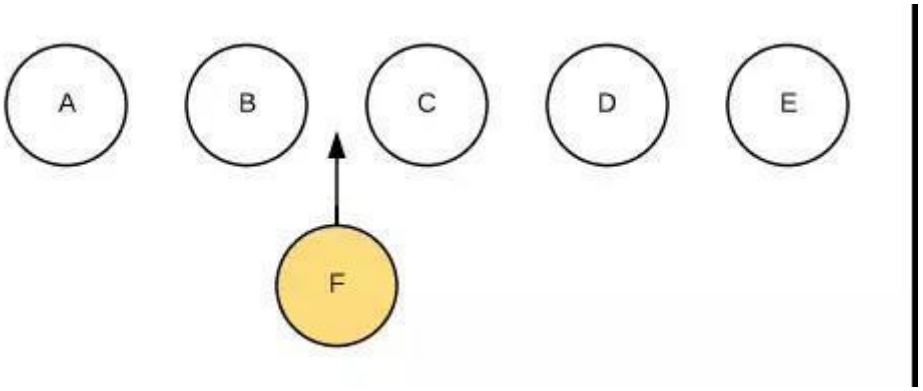
```vue
<div id="app">
    <!-- 1.在遍历的过程中，没有使用索引值(下标值) -->
    <ul>
        <li v-for="name in names">{{name}}</li>
    </ul>
    <!-- 2.在遍历的过程中，使用索引值 -->
    <ul>
        <li v-for="(item,index) in names">{{index+1}}.{{item}}</li>
    </ul>
    <!-- 3.在遍历对象的过程中，如果只是获取一个值，那么获取到的是value -->
    <ul>
        <li v-for="item in info">{{item}}</li>
    </ul>
    <!-- 4.在遍历对象的过程中，获取key，value 格式:(value,key)-->
    <ul>
        <li v-for="(value,key) in info">{{key}}:{{value}}</li>
    </ul>
    <!-- 5.在遍历对象的过程中，获取key、value和index 格式:(value,key,index)-->
    <ul>
        <li v-for="(value,key,index) in info">{{index+1}}.{{key}}:{{value}}</li>
    </ul>
</div>
<script>
    const app = new Vue({
        el: '#app',
        data: {
            names: ['why', 'kobe', 'james', 'curry'],
            info: {
                name: 'why',
                age: 18,
                height: 1.88
            }
        },
    })
</script>
```

三、组件的key属性

1.官方推荐我们在使用v-for时，给对应的元素或组件添加上一个：key属性。

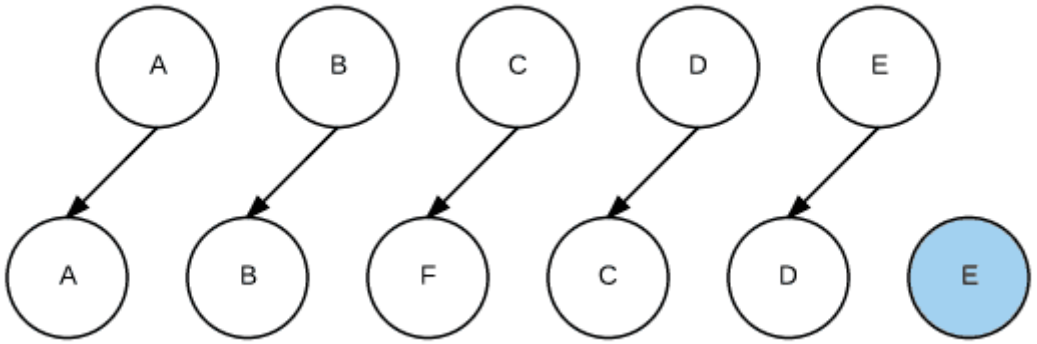
2.为什么需要这个key属性呢(了解)?

- 这个其实和Vue虚拟DOM的Diff算法有关系。
- 这里我们借用React's diff algorithm的一张图来简单说明一下：



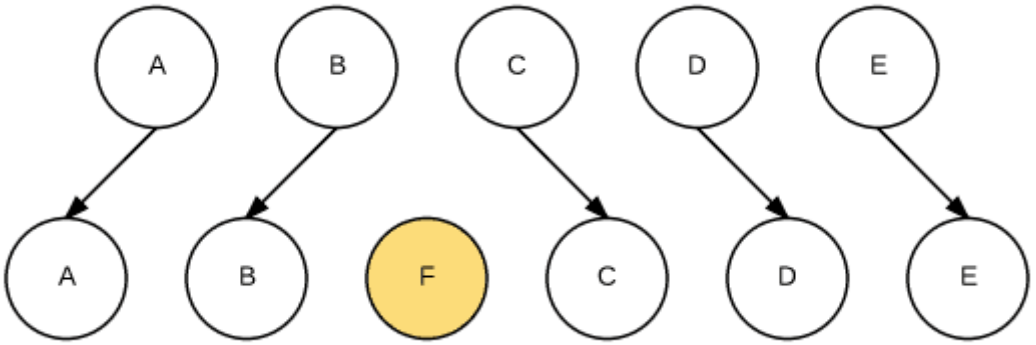
3.当某一层很多相同的节点时，也就是列表节点时，我们希望插入一个新的节点

- 我们希望可以在B和C之间加一个F，Diff算法默认执行起来是这样的：
- 即把C更新成F，D更新成C,E更新成D,最后再插入E，是不是很没有效率?



4.所以我们需要使用key来给每个节点做一个唯一标识

- Diff算法就可以正确的识别此节点
- 找到正确的位置插入新的节点



所以一句话，**key的作用主要是为了高效的更新虚拟DOM**

```
<div id="app">
    <ul>
        <li v-for="item in letters" :key="item">{{item}}</li>
    </ul>
</div>
```

四、检测数组更新

1.因为Vue是响应式的，所以当数据发生变化时，Vue会自动检测数据变化，视图会发生对应的更新

2.Vue中包含了一组观察数组编译的方法，使用它们改变数组也会触发视图的更新

- push():数组末尾追加元素
- pop():删除数组中的最后一个元素
- shift():删除数组中的第一个元素
- unshift():在数组最前面添加元素
- splice():删除元素/插入元素/替换元素
- sort():正向排序
- reverse()：翻转数组

```
//1.push方法:数组末尾追加元素
this.letters.push('aaa',"bbb");

//2.pop(): 删除数组中的最后一个元素
this.letters.pop();

//3.shift():删除数组中的第一个元素
this.letters.shift();

//4.unshift(): 在数组最前面添加元素
this.letters.unshift("aaa", "bb")

//splice作用: 删除元素/插入元素/替换元素
//删除元素: 第二个参数传入你要删除几个元素(如果没有传, 就删除后面所有的元素)
//5.splice(start):
start = 2
this.letters.splice(start, this.letters.length - 2)
//替换元素:第二个参数表示我们要替换几个元素, 后面是用于替换前面的元素
this.letters.splice(1, 3, 'm', 'n', 'l', 'x')
//插入元素: 第二个参数为0,后面是用于插入的元素
this.letters.splice(1, 0, "m", "n")

//6.sort()
this.letters.sort()

//7.reverse():翻转数组
this.letters.reverse()

//注意: 通过索引值修改数组中的元素, 不是响应式的
this.letters[0] = "bbb";
this.letters.splice(0, 1, "bbb")

//set(要修改的对象, 索引值, 修改后的值)
Vue.set(this.letters, 0, 'bbbb')
```

五、高阶函数：filter/map/reduce

编程范式：命令式编程/**声明式编程**(更常用)

编程范式：面向对象编程(第一公民:对象)/**函数式编程**(第一公民:函数)

1.filter中的回调函数有一个要求：必须返回一个boolean值

- true:当返回true时，函数内部会自动将这次回调的n加入到新的数组中
- false:当返回false时，函数内部会过滤掉这次的n

```
const nums = [10, 20, 111, 222, 444, 40, 50]
//filter函数的使用,
//1.需求：取出所有小于100的数字
//10,20,40,50
let newNums = nums.filter(function(n) {
    return n < 100
})
console.log(newNums);
```

2.map函数的使用

```
//map函数的使用
//2.需求：将所有小于100的数字进行转化：全部*2
//20,40,80,100
let new2Nums = newNums.map(function(n) {
    return n * 2
})
console.log(new2Nums);
```

3.reduce函数的使用

- reduce函数作用：对数组中所有内容进行汇总
- 参数：前一个值，当前输入值

```
//reduce函数的使用
//3.需求：将所有new2Nums数字相加，得到最终结果
//reduce函数作用：对数组中所有内容进行汇总
total = new2Nums.reduce(function(preValue, n) {
        return preValue + n
    }, 0)
    //第一次：preValue:0  n:20
    //第二次：preValue:20  n:40
    //第三次：preValue:60  n:80
    //第四次：preValue:140  n:100
    //最终结果：240
console.log(total)
```

4.案例代码汇总：

```
//使用三个高阶函数
let total = nums.filter(function(n) {
    return n < 100
}).map(function(n) {
    return n * 2
}).reduce(function(preValue, n) {
    return preValue + n
},0)
console.log(total);
```

5.使用箭头函数一行代码搞定

```
//使用箭头函数
let total = nums.filter(n => n < 100).map(n => n * 2).reduce((pre, n) => pre + n);
console.log(total);
```