Vue源码探秘之

AST抽象语法树

讲师：邵山欢

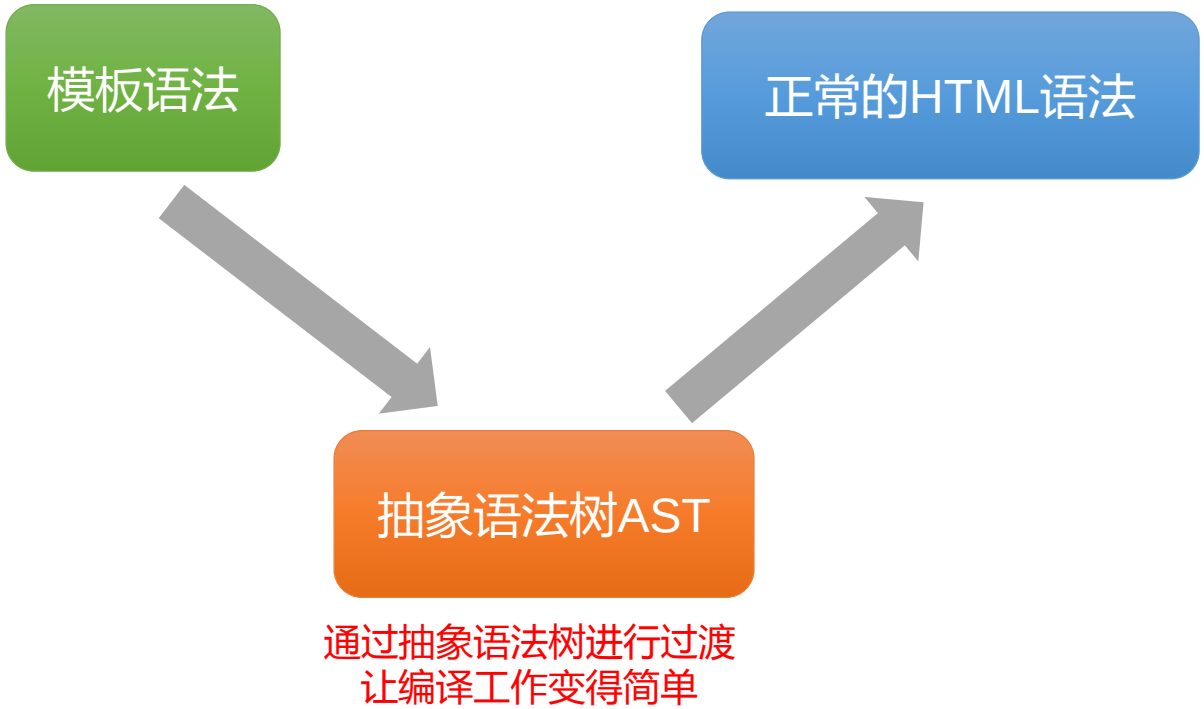
课程简介

抽象语法树是什么


```
<div class="box">
    <h3 class="title">我是一个标题</h3>
    <ul>
        <li v-for="item in arr" :key="index">
            {{item}}
        </li>
    </ul>
</div>
```

```
<div class="box">
    <h3 class="title">我是一个标题</h3>
    <ul>
        <li>牛奶</li>
        <li>咖啡</li>
        <li>可乐</li>
    </ul>
</div>
```

抽象语法树是什么



抽象语法树本质上就是一个JS对象

```
<div class="box">
    <h3 class="title">我是一个标题</h3>
    <ul>
        <li v-for="item in arr" :key="index">
            {{item}}
        </li>
    </ul>
</div>
```

以字符串的视角

```
<div class="box">
    <h3 class="title">我是一个标题</h3>
    <ul>
        <li v-for="item in arr" :key="index">
            {{item}}
        </li>
    </ul>
</div>
```

解析为AST

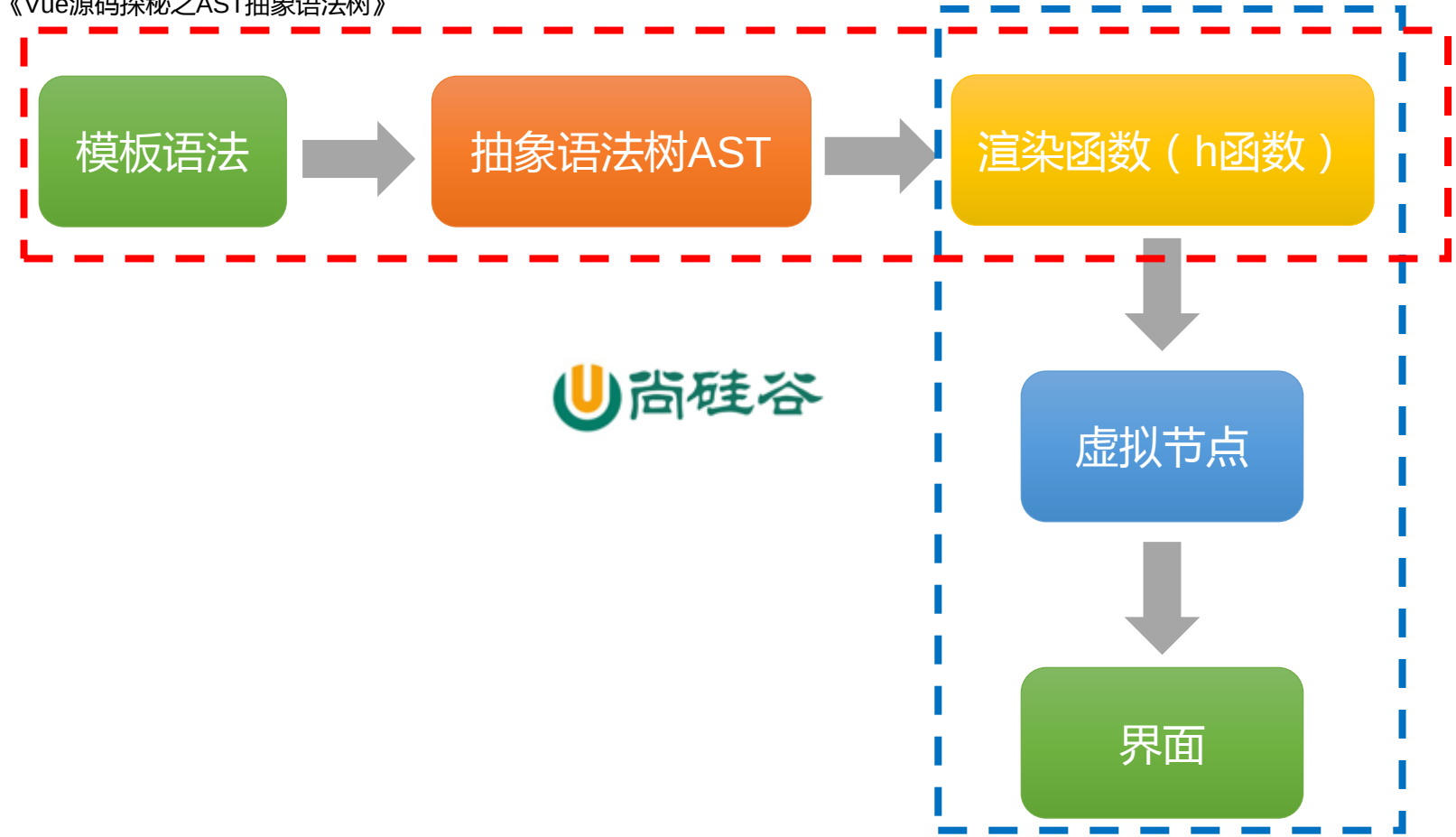
```
{
    tag: "div",
    attrs: [{ name: "class", value: "box" }],
    type: 1,
    children: [
        {
            tag: "h3",
            attrs: [{ name: "class", value: "title" }],
            type: 1,
            children: [{ text: "我是一个标题", type: 3 }]
        },
        {
            tag: "ul",
            attrs: [],
            type: 1,
            children: [
                {
                    tag: "li",
                    for: "arr",
                    key: "index",
                    alias: "item",
                    type: 1,
                    children: []
                }
            ]
        }
    ]
}
```

Abstract Syntax Tree
抽象语法树，简称AST

抽象语法树和虚拟节点的关系

《Vue源码探秘之AST抽象语法树》

《Vue源码探秘之虚拟节点和DIFF算法》



- 相关算法储备
- AST形成算法
- 手写AST编译器
- 手写文本解析功能
- AST优化
- 将AST生成h()函数

学习本课程的**知识储备前提**：

- 会Vue（Vue2.x和Vue3.x均可）
- 简单了解webpack和webpack-dev-server
- 实际开发经验不限，应届生也可学习

相关算法储备 - 指针思想

指针思想题目

试寻找字符串中，连续重复次数最多的字符。

'aaaabbbbbccccccccccccccddddddd'

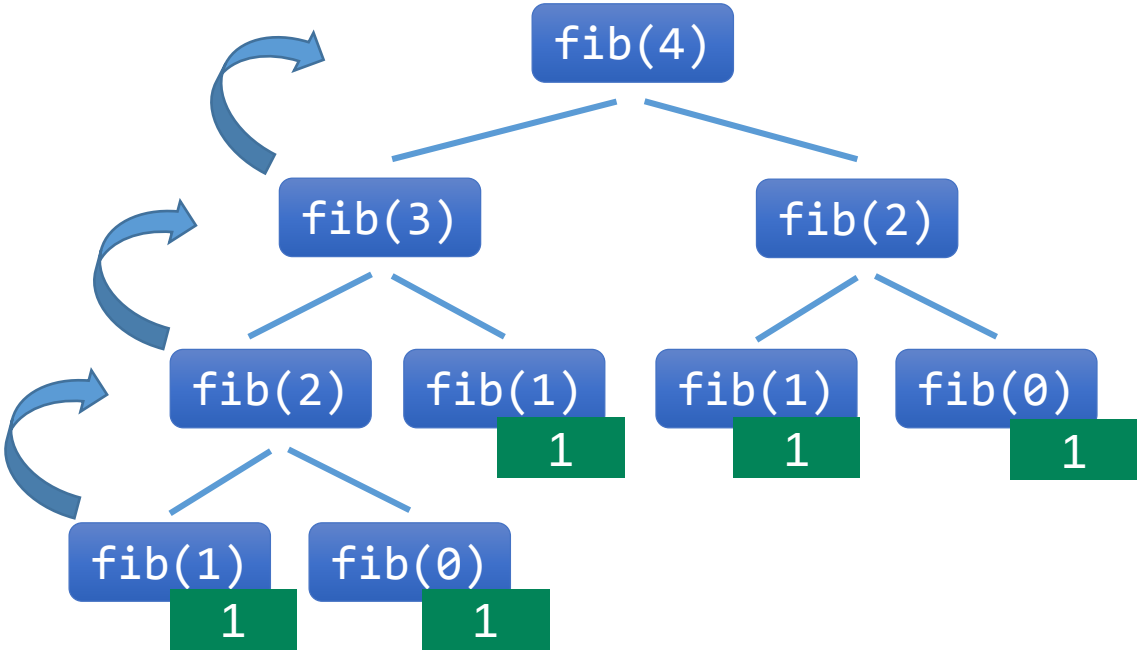
指针就是下标，不是C语言中的指针，C语言中的指针可以操作内存。JS中的指针就是一个下标位置。

```
i: 0
j: 1
```

- 如果i和j指向的字一样，那么i不动，j后移
- 如果i和j指向的字不一样，此时说明它们之间的字都是连续相同的，让i追上j，j后移

相关算法储备 - 递归深入

试输出斐波那契数列的前10项，即1、1、2、3、5、8、13、21、34、55。然后请思考，代码是否有大量重复的计算？应该如何解决重复计算的问题？



cache思想

```
{
    '0': 1,
    '1': 1,
    '2': 2,
    '3': 3,
    '4': 5,

}
```

形式转换：试将高维数组[1，2，[3，[4，5]，6]，7，[8]，9]变为图中所示的对象

小技巧：只要出现了“规则复现”就要想到用递归。

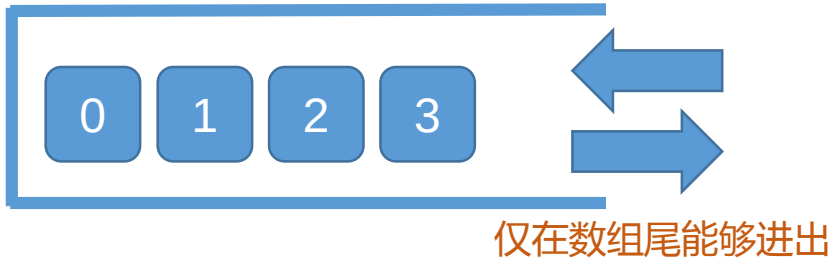
```
{
    children: [
        { value: 1 },
        { value: 2 },
        { children: [
            { value: 3 },
            { children: [
                { value: 4 },
                { value: 5 }
            ]},
            { value: 6 }
        ]},
        { value: 7 },
        { children: [
            { value: 8 },
            { value: 9 }
        ]},
    ]
}
```

相关算法储备 - 栈

栈

- 栈（stack）又名堆栈，它是一种运算受限的线性表，仅在表尾能进行插入和删除操作。这一端被称为栈顶，相对地，把另一端称为栈底。
- 向一个栈插入新元素又称作进栈、入栈或压栈；从一个栈删除元素又称作出栈或退栈。
- 后进先出（LIFO）特点：栈中的元素，最先进栈的必定是最后出栈，后进栈的一定会先出栈

- JavaScript中，栈可以用数组模拟。需要限制只能使用push()和pop()，不能使用unshift()和shift()。即，数组尾是栈顶。
- 当然，可以用面向对象等手段，将栈封装的更好。



试编写“智能重复”smartRepeat函数，实现：

- 将3[abc]变为abcabcabc
- 将3[2[a]2[b]]变为aabbaabbaabb
- 将2[1[a]3[b]2[3[c]4[d]]]变为abbbcccddddcccddddabbbcccddddcccddddd

不用考虑输入字符串是非法的情况，比如：

- 2[a3[b]]是错误的，应该补一个1，即2[1[a]3[b]]
- [abc]是错误的，应该补一个1，即1[abc]

- 词法分析的时候，经常要用到栈这个数据结构；
- 初学者大坑：栈的题目和递归非常像，这类题目给人的感觉都是用递归解题。信心满满动手开始写了，却发现递归怎么都递归不出来。此时就要想到，不是用递归，而是用栈。

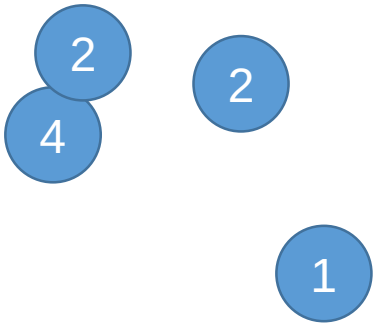
利用“栈”的题目1

333[2[abc]2[d]]

存放数字

存放字符串

''



遍历每一个字符

- 如果这个字符是数字，那么就把数字压栈，把空字符串压栈
- 如果这个字符是字母，那么此时就把栈顶这项改为这个字母
- 如果这个字符是]，那么就将数字弹栈，就把字符串栈的栈顶的元素重复刚刚的这个次数，弹栈，拼接到新栈顶上

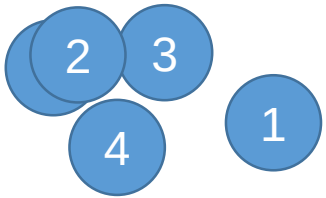
利用“栈”的题目1

2[1[a]3[b]2[3[c]4[d]]]

存放数字

存放字符串

' abbbcccddddcccddddabbbcccddddcccdddd '



遍历每一个字符

- 如果这个字符是数字，那么就把数字压栈，把空字符串压栈
- 如果这个字符是字母，那么此时就把栈顶这项改为这个字母
- 如果这个字符是]，那么就将数字弹栈，就把字符串栈的栈顶的元素重复刚刚的这个次数，弹栈，拼接到新栈顶上

正则表达式的相关方法

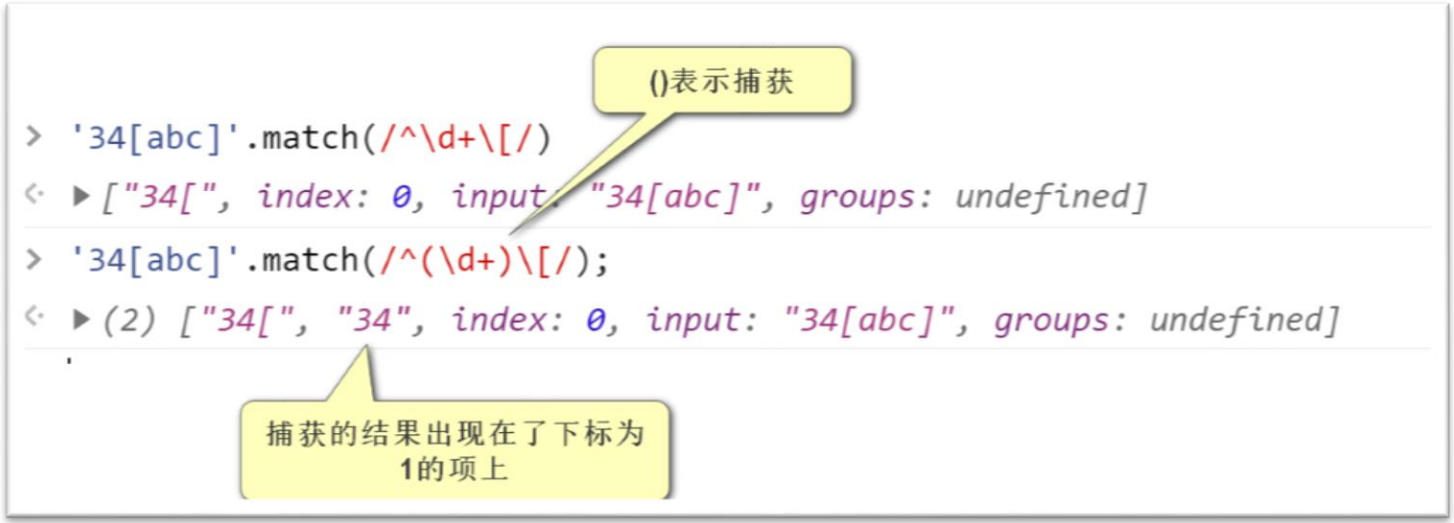
```
> 'abc666defg123mnp'.replace(/\d/g, '');
"abcdefgmnp"
```

```
> 'abc666defg123mnp'.search(/\d/)
3
> 'abc666defg123mnp'.search(/\d/g)
3
> 'abc666defg123mnp'.match(/\d/)
▶ ["6", index: 3, input: "abc666defg123mnp", groups: undefined]
> 'abc666defg123mnp'.match(/\d/g)
▶ (6) ["6", "6", "6", "1", "2", "3"]
```

search()方法加g修饰符没有用

match()方法加g非常好用，能够寻找到所有匹配的字符

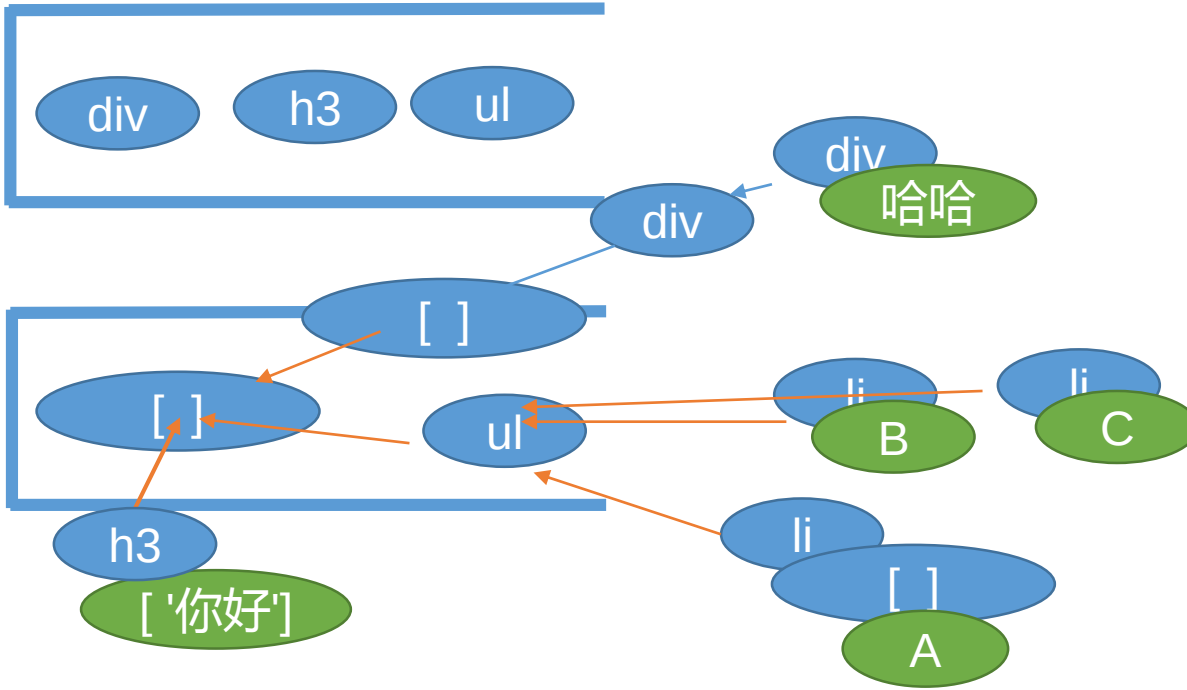
```
> /^\d/.test('abc')
false
> /^\d/.test('5abc')
true
```


()表示捕获
'34[abc]'.match(/^\d+\[/)
▶ ["34[", index: 0, input: "34[abc]", groups: undefined]
'34[abc]'.match(/^(\d+)\[/);
▶ (2) ["34[", "34", index: 0, input: "34[abc]", groups: undefined]
捕获的结果出现在了下标为1的项上

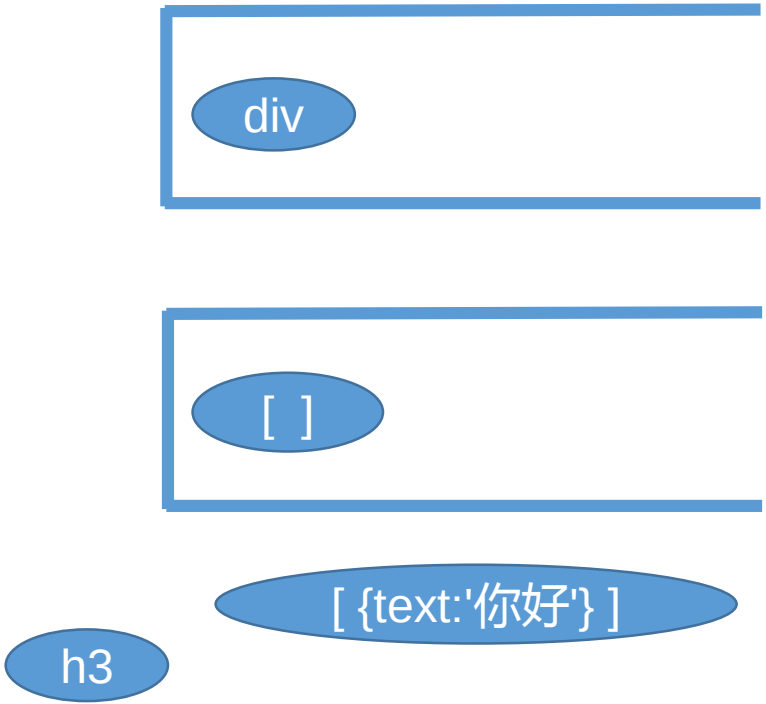
手写实现AST抽象语法树

- 学习源码时，源码思想要借鉴，而不要抄袭。要能够发现源码中书写的精彩的地方；
- 将独立的功能拆写为独立的js文件中完成，通常是一个独立的类，每个单独的功能必须能独立的“单元测试”；
- 应该围绕中心功能，先把主干完成，然后修剪枝叶；
- 功能并不需要一步到位，功能的拓展要一步步完成，有的非核心功能甚至不需实现；

```
<div>
    <h3>你好</h3>
    <ul>
        <li>A</li>
        <li>B</li>
        <li>C</li>
    </ul>
</div>
```



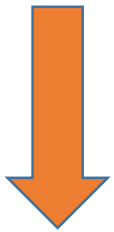
```
<div>
    <h3>你好</h3>
    <ul>
        <li>A</li>
        <li>B</li>
        <li>C</li>
    </ul>
</div>
```



识别attrs


```
isYinhao = false
point = 16

['class="aa bb cc"', 'id="mybox"']
```



class="aa bb cc" id="mybox"