Vue源码探秘之

mustache模板引擎

课程简介

为什么要学Vue源码?

- **面试变得越来越难**
  - 2016年~2019年，面试主要考察Vue的**使用**
  - 2020年~2021年，面试越来越爱考Vue**底层**
- **中高级前端、leader职位必会底层知识**
  - 为企业“造轮子”，开发通用组件
  - 解决编程中遇见的问题
  - 解决效率问题

Vue组件由哪三部分组成?
Vue父子组件如何通信?
Vuex的作用是什么?

Vue是怎么实现双向绑定的?
Vue的最小化更新过程是怎样的?
Vue如何实现指令系统的?

- **内容循序渐进：**Vue源码非常庞大，各种机理很多：模板技术、数据劫持、虚拟节点、最小量更新、抽象语法树.....我们将它们一一拆解，从简单知识到复杂，循序渐进；
- **提升编程能力：**手写底层源码，拒绝纸上谈兵，让同学能实打实的提升编程能力；
- **讲解图文并茂：**老师用丰富的图示、例子进行讲解，力求让同学们彻底理解每一个核心机理；
- **新老版本兼顾：**兼顾Vue2和3，它们的底层区别老师上课会着重讲解。核心机理是共通的、永恒的。

本次课程内容

- 什么是模板引擎 — 介绍宏观背景、历史沿革
- mustache基本使用 — 先学会怎么用
- mustache的底层核心机理 — 再研究它底层机理
- 带你手写实现mustache库 — 最后手写它，彻底掌握它！

循序渐进

学习本课程的**知识储备前提**：

- 会写一些JavaScript常见算法，比如递归、二维数组遍历等；
- 熟悉ES6的常用特性，比如let、箭头函数等；
- 了解webpack和webpack-dev-server
- 实际开发经验不限，应届生也可学习

什么是模板引擎

模板引擎是将数据要变为视图最优雅的解决方案

数据：

```
[
    { "name": "小明", "age": 12, "sex": "男" },
    { "name": "小红", "age": 11, "sex": "女" },
    { "name": "小强", "age": 13, "sex": "男" }
]
```

Vue的解决方法：

```
<li v-for="item in arr"></li>
```

这实际上就是一种模板引擎

视图：

```
<ul>
    <li>
        <div class="hd">小明的基本信息</div>
        <div class="bd">
            <p>姓名：小明</p>
            <p>年龄：12</p>
            <p>性别：男</p>
        </div>
    </li>
    <li>
        <div class="hd">小红的基本信息</div>
        <div class="bd">
            <p>姓名：小红</p>
            <p>年龄：11</p>
            <p>性别：女</p>
        </div>
    </li>
    ……
</ul>
```

历史上曾经出现的数据变为视图的方法

- **纯DOM法**:
- **数组join法**:
- **ES6的反引号法**:
- **模板引擎**:

非常笨拙，没有实战价值

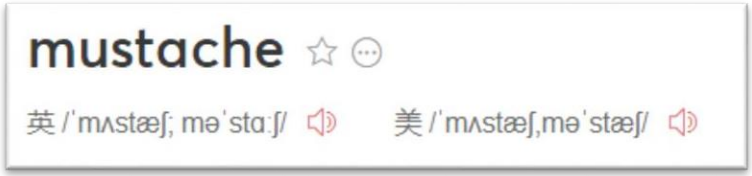
曾几何时非常流行，是曾经的前端必会知识

ES6中新增的`${a}`语法糖，很好用

解决数据变为视图的最优雅的方法

mustache的基本使用

- mustache官方git： https://github.com/janl/mustache.js
- mustache是“胡子”的意思，因为它的嵌入标记{{ }}非常像胡子
- 没错，{{ }}的语法也被Vue沿用，这就是我们学习mustache的原因



- mustache是最早的模板引擎库，比Vue诞生的早多了，它的底层实现机理在当时是非常有创造性的、轰动性的，为后续模板引擎的发展提供了崭新的思路

- 必须要引入mustache库，可以在bootcdn.com上找到它
- mustache的模板语法非常简单，比如前述案例的模板语法如下：

```
<ul>
    {{#arr}}
        <li>
            <div class="hd">{{name}}的基本信息</div>
            <div class="bd">
                <p>姓名：{{name}}</p>
                <p>性别：{{sex}}</p>
                <p>年龄：{{age}}</p>
            </div>
        </li>
    {{/arr}}
</ul>
```

mustache库基本使用 - 循环对象数组

```
Mustache.render(templateStr, data);
```

```
<ul>
    {{#arr}}
        <li>
            <div class="hd">{{name}}的基本信息</div>
            <div class="bd">
                <p>姓名：{{name}}</p>
                <p>性别：{{sex}}</p>
                <p>年龄：{{age}}</p>
            </div>
        </li>
    {{/arr}}
</ul>
```

```
[
    { "name": "小明", "age": 12, "sex": "男" },
    { "name": "小红", "age": 11, "sex": "女" },
    { "name": "小强", "age": 13, "sex": "男" }
]
```

```
<ul>
    <li>
        <div class="hd">小明的基本信息</div>
        <div class="bd">
            <p>姓名：小明</p>
            <p>年龄：12</p>
            <p>性别：男</p>
        </div>
    </li>
    <li>
        <div class="hd">小红的基本信息</div>
        <div class="bd">
            <p>姓名：小红</p>
            <p>年龄：11</p>
            <p>性别：女</p>
        </div>
    </li>
    ……
</ul>
```

mustache库基本使用 - 不循环

```
Mustache.render(templateStr, data);
```

```
<h1>我买了一个华为手机，好开心</h1>
```

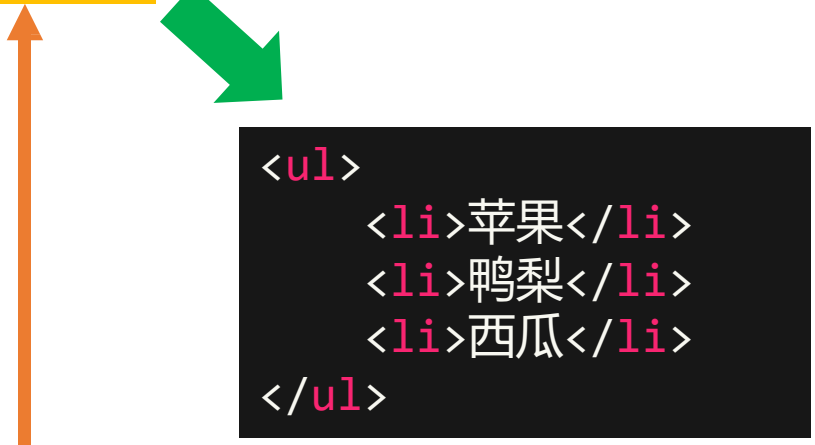
```
<h1>我买了一个{{thing}}，好{{mood}}</h1>
```

```
{
    thing: '华为手机',
    mood: '开心'
}
```

mustache库基本使用 - 循环简单数组

```
Mustache.render(templateStr, data);
```

```
<ul>
    {{#arr}}
        <li>{{.}}</li>
    {{/arr}}
</ul>
```



```
<ul>
    <li>苹果</li>
    <li>鸭梨</li>
    <li>西瓜</li>
</ul>
```



```
['苹果', '鸭梨', '西瓜']
```

mustache库基本使用 -数组可以嵌套

```
Mustache.render(templateStr, data);
```

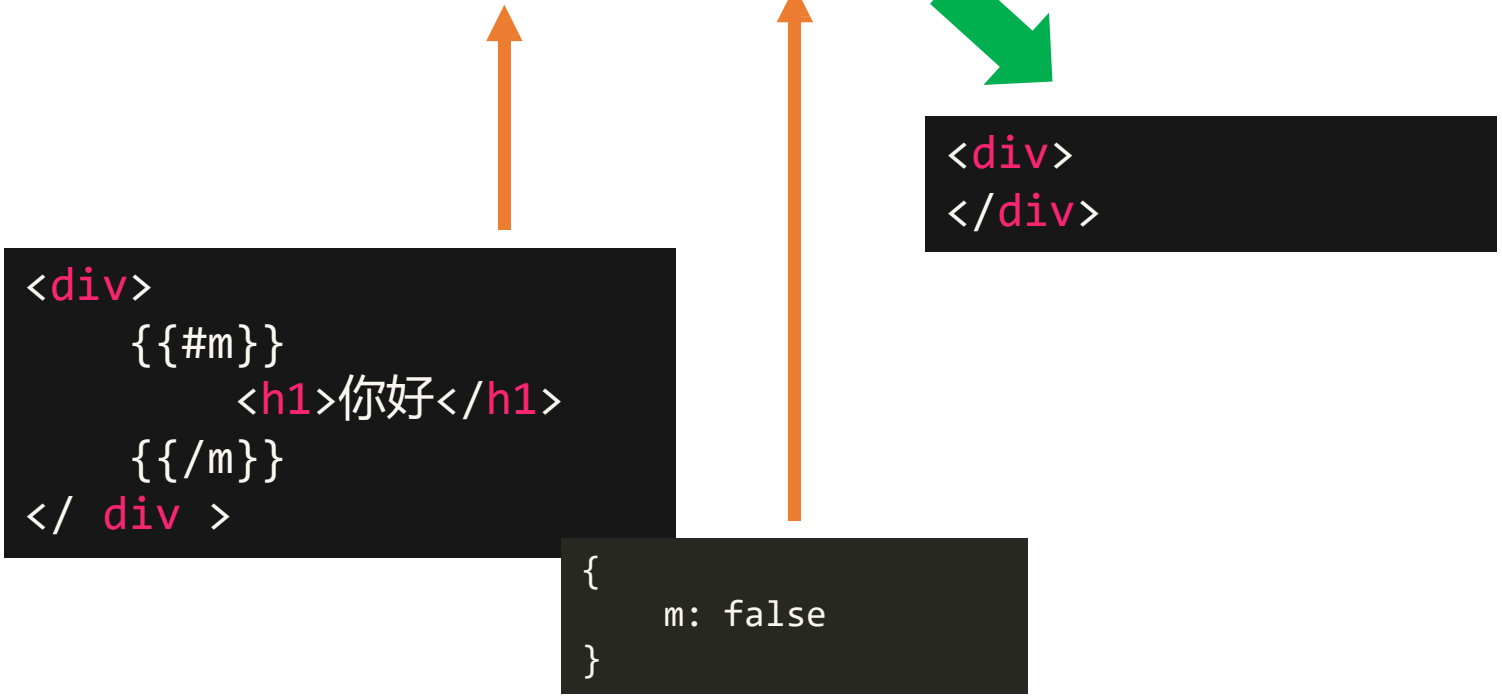
- 小明的爱好是：
  1. 游泳
  2. 羽毛球
- 小红的爱好是：
  1. 编程
  2. 写作文
  3. 看报纸
- 小强的爱好是：
  1. 打台球

```
var templateStr = `
    <ul>
        {{#arr}}
            <li>
                {{name}}的爱好是:
                <ol>
                    {{#hobbies}}
                        <li>{{.}}</li>
                    {{/hobbies}}
                </ol>
            </li>
        {{/arr}}
    </ul>
`;
```

```
var data = {
    arr: [
        {'name': '小明', 'age': 12, 'hobbies': ['游泳', '羽毛球']},
        {'name': '小红', 'age': 11, 'hobbies': ['编程', '写作文', '看报纸']},
        {'name': '小强', 'age': 13, 'hobbies': ['打台球']},
    ]
};
```

mustache库基本使用 - 布尔值

```
Mustache.render(templateStr, data);
```



```
<div>
    {{#m}}
        <h1>你好</h1>
    {{/m}}
</ div >
```

```
{
    m: false
}
```

```
<div>
</div>
```

mustache的底层核心机理

mustache库不能用简单的正则表达式思路实现

- 在较为简单的示例情况下，可以用正则表达式实现

模板字符串 `<h1>我买了一个{{thing}}，好{{mood}}</h1>`

数据

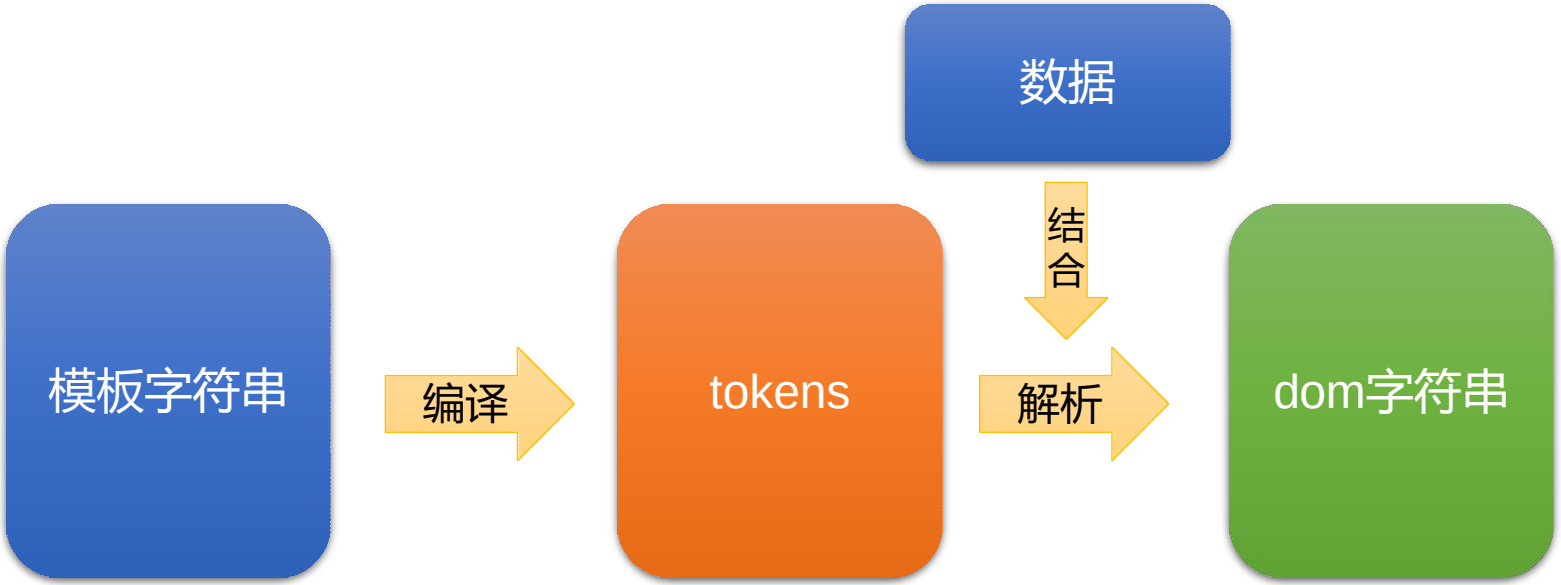
```
{
    thing: '华为手机',
    mood: '开心'
}
```

- 但是当情况复杂时，正则表达式的思路肯定就不行了。比如这样的模板字符串，是不能用正则表达式的思路实现的

模板字符串

```
<div>
    <ul>
        {{#arr}}
        <li>{{.}}</li>
        {{/arr}}
    </ul>
</div>
```

mustache库的机理



什么是tokens?

- tokens是一个JS的嵌套数组，说白了，就是模板字符串的JS表示
- 它是“抽象语法树”、“虚拟节点”等等的开山鼻祖

模板字符串 `<h1>我买了一个{{thing}}，好{{mood}}啊</h1>`

tokens

```
[
    ["text", "<h1>我买了一个"],
    ["name", "thing"],
    ["text", "，好"],
    ["name", "mood"],
    ["text", "啊</h1>"],
]
```

token
token
token
token
token

非常有创造性的、轰动性的模板解析原理

模板字符串

```
<h1>我买了一个{{thing}}，好{{mood}}啊</h1>
```

编译

tokens

```
[
    ["text", "<h1>我买了一个"],
    ["name", "thing"],
    ["text", "好"],
    ["name", "mood"],
    ["text", "啊</h1>"],
]
```

数据

```
{
    thing: '华为手机',
    mood: '开心'
}
```

结合

解析

```
<h1>我买了一个华为手机，好开心</h1>
```

循环情况下的tokens

- 当模板字符串中有循环存在时，它将被编译为嵌套更深的tokens

```
<div>
    <ul>
        {{#arr}}
        <li>{{.}}</li>
        {{/arr}}
    </ul>
</div>
```


```
[
    ["text", "<div><ul>"],
    ["#", "arr", [
        ["text", "<li>"],
        ["name", "."],
        ["text", "</li>"]
    ]],
    ["text", "</ul></div>"]
]
```

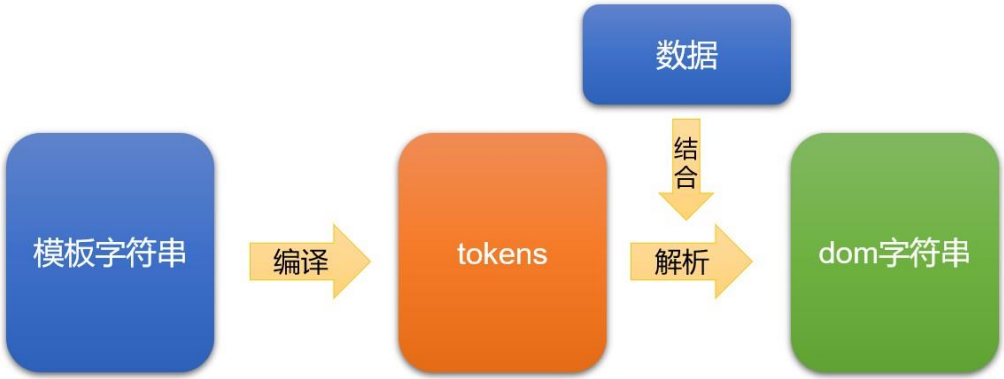
双重循环情况下的tokens

- 当循环是双重的，那么tokens会更深一层

```
<div>
    <ol>
        {{#students}}
        <li>
            学生
            <ol>
                {{#hobbies}}
                <li>{{.}}</li>
                {{/hobbies}}
            </ol>
        </li>
        {{/students}}
    </ol>
</div>
```


```
[
    ["text", "<div><ol>"],
    ["#", "students", [
        ["text", "<li>学生"],
        ["name", "name"],
        ["text", "的爱好是<ol>"],
        ["#", "hobbies", [
            ["text", "<li>"],
            ["name", "."],
            ["text", "</li>"],
        ]],
        ["text", "</ol></li>"],
    ]],
    ["text", "</ol></div>"]
]
```

mustache库的机理



mustache库底层重点要做两个事情：

① 将模板字符串编译为tokens形式

② 将tokens结合数据，解析为dom字符串

观察tokens

把256行这里

```
253         if (openSection)
254             throw new Error('Unclosed section "' + openSectio
255
256
257         return nestTokens(squashTokens(tokens));
258     }
```

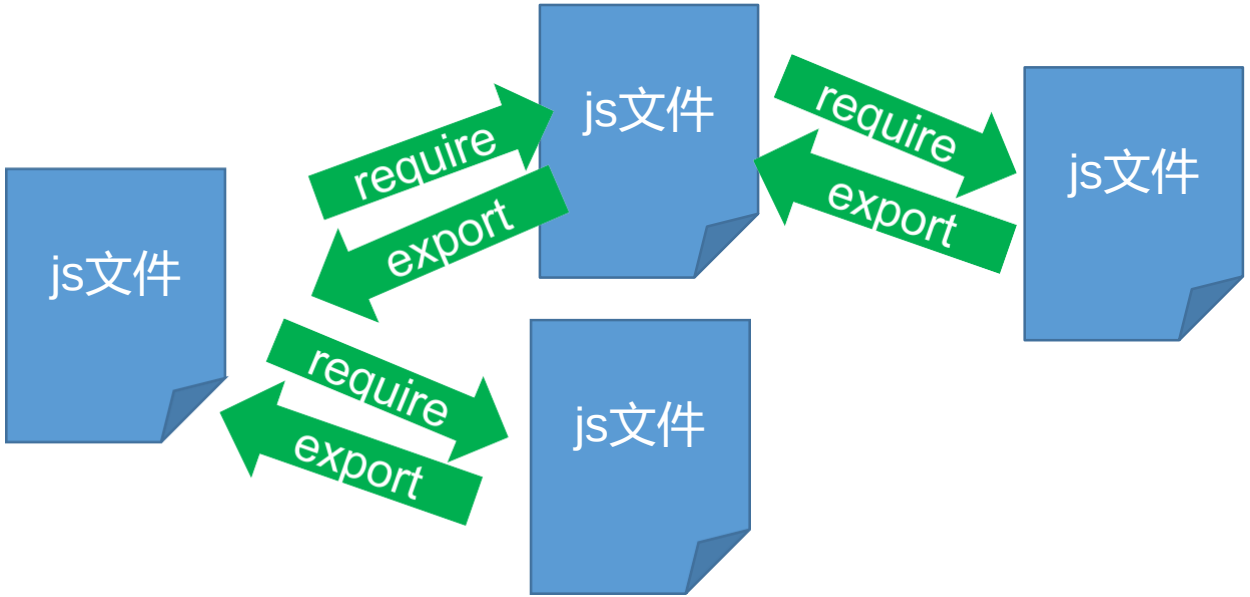
改为

```
var tokens = nestTokens(squashTokens(tokens));
console.log(tokens);
return tokens;
```

带你手写实现mustache库

- 模块化打包工具有webpack（webpack-dev-server）、rollup、Parcel等
- mustache官方库使用rollup进行模块化打包，而我们今天使用webpack（webpack-dev-server）进行模块化打包，这是因为webpack（webpack-dev-server）能让我们更方便地在浏览器中（而不是nodejs环境中）实时调试程序，相比nodejs控制台，浏览器控制台更好用，比如能够点击展开数组的每项。
- 生成库是UMD的，这意味着它可以同时在nodejs环境中使用，也可以在浏览器环境中使用。实现UMD不难，只需要一个“通用头”即可。

- 模块化打包工具有webpack（webpack-dev-server）、rollup、Parcel等



注意webpack和webpack-dev-server的版本

- webpack最新版是5，webpack-dev-server最新版是4，但是目前它们的最新版兼容程度不好，建议大家使用这样的版本：

```
"webpack": "^4.44.2",
"webpack-cli": "^3.3.12",
"webpack-dev-server": "^3.11.0"
```

```
const path = require('path');

module.exports = {
    mode: 'development',
    entry: './index.js',
    output: {
        filename: 'bundle.js'
    },
    devServer: {
        contentBase: path.join(_dirname, "www"),
        compress: false,
        port: 8080,
        // 虚拟打包的路径，bundle.js文件没有真正的生成
        publicPath: "/xuni/"
    }
};
```

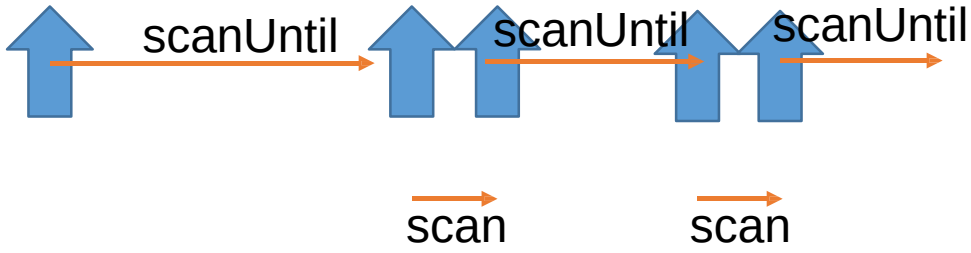
- 学习源码时，源码思想要借鉴，而不要抄袭。要能够发现源码中书写的精彩的地方；
- 将独立的功能拆写为独立的js文件中完成，通常是一个独立的类，每个单独的功能必须能独立的“单元测试”；
- 应该围绕中心功能，先把主干完成，然后修剪枝叶；
- 功能并不需要一步到位，功能的拓展要一步步完成，有的非核心功能甚至不需实现；

将模板字符串变为tokens


```
<div>
    <ol>
        {{#students}}
        <li>
            学生{{name}}的爱好是
            <ol>
                {{#hobbies}}
                <li>{{.}}</li>
                {{/hobbies}}
            </ol>
        </li>
        {{/students}}
    </ol>
</div>
```

```
[
    ["text", "<div><ol>"],
    ["#", "students", [
        ["text", "<li>学生"],
        ["name", "name"],
        ["text", "的爱好是<ol>"],
        ["#", "hobbies", [
            ["text", "<li>"],
            ["name", "."],
            ["text", "</li>"],
        ]],
        ["text", "</ol></li>"],
    ]],
    ["text", "</ol></div>"]
]
```

我买了一个{{thing}}，好{{mood}}啊



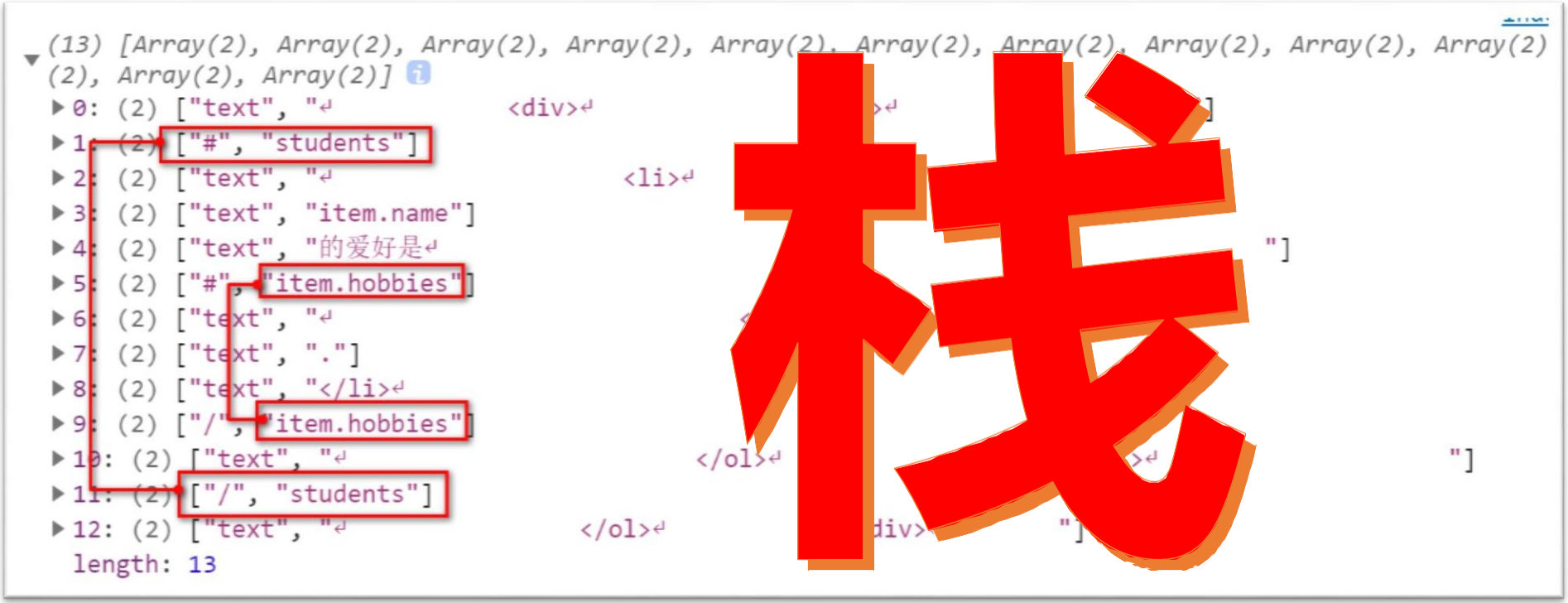
我买了一个{{thing}}，好{{mood}}啊

生成tokens数组

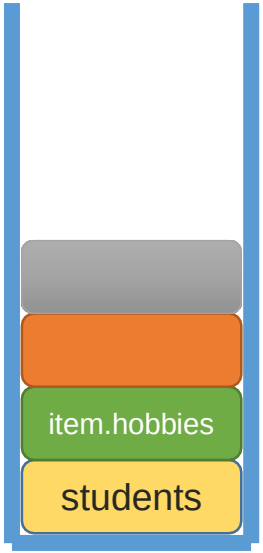
```
<h1>我买了一个{{thing}}, 好{{mood}}啊</h1>
```

```
[
    ["text", "<h1>我买了一个"],
    ["name", "thing"],
    ["text", "好"],
    ["name", "mood"],
    ["text", "啊</h1>"],
]
```

将零散的tokens嵌套起来



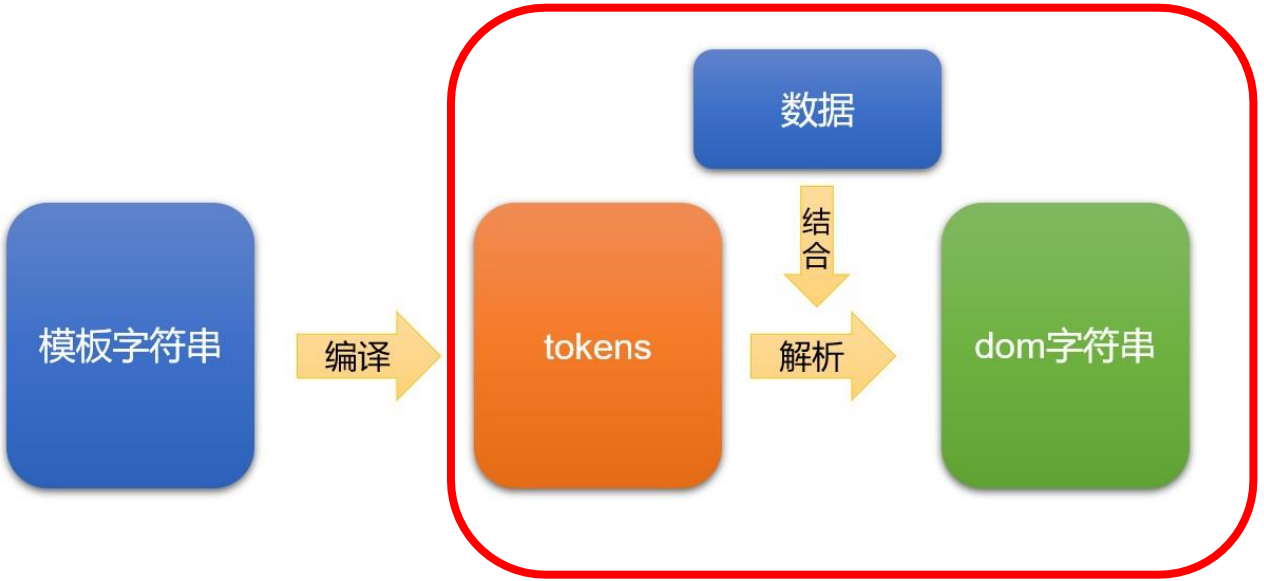
栈是什么



first in last out
FILO

遇见#就进栈
遇见/就出栈

将tokens结合数据，解析为dom字符串



粗浅的完成简单的目标：

```
var templateStr = '我爱{{somebody}}, {{somebody}}也爱我';
var data = {
    somebody: '尚硅谷'
};
```

不带有#标记

```
export default function renderTemplate(tokens, data) {
    console.log(tokens, data);
    // 结果字符串
    var resultStr = '';
    // 遍历tokens
    for (let i = 0; i < tokens.length; i++) {
        let token = tokens[i];

        // 看类型
        if (token[0] == 'text') {
            // 拼起来
            resultStr += token[1];
        } else if (token[0] == 'name') {
            resultStr += data[token[1]];
        }
    }

    console.log(resultStr);
}
```

- #标记的tokens，需要递归处理它的下标为2的小数组
- 现在遇见一个问题，JS不认识点符号

```
var mmm = {
    a: {
        b: {
            c: 100
        }
    }
}

mmm['a.b.c']    //不行
```

课程总结

- Mustache底层太美了！tokens的意义也不言自明了。如果没有token，那么数组的循环形式，就很难处理。我们通过本课，确实学到了很多，视野面变得更广，感觉肚子里的东西更多了；

- 在Mustache源码中，还有Context类和Writer类，在我们的代码演示中，都将它们进行了简化，但是不影响主干功能的实现。我们的这个“简化版本的”代码非常值得大家进行手写，你会受益良多的！当然，如果有精力，可以再研究透彻这个“简化版本的”代码后，自己对Mustache原包进行学习；