

K-SUWABE 2012/02/26



## 实现一个前端模板引擎



有只猫

前端工程师

关注他

8 人赞同了该文章

前端模板引擎相信大家都不会陌生了吧，尤其是注重前后端分离的今天(除非你还在用拼接字符串)。

**引擎**一词总让人感觉很高端的样子，其实归根结底也只是处理字符串的一种方式而已。本文总结了3种实现模板引擎的方式，最后将编写一个 类似于 underscore.template 的模板插件。

### 一、replace

#### 介绍

replace 是字符串提供的一个超级强大的方法，这里只介绍简单的使用，更多详细的语法参见下面提供的链接。

- 一参可为`字符串`或`正则`：
  - 为正则时有两种情况: `普通匹配模式` 和 `全局匹配模式`：
    - 全局匹配模式下，若二参为函数，则该函数在每次匹配时都会被调用
- 二参可为`字符串`或`一个用于生成字符串的函数`：
  - 当为字符串时：
    - 可在字符串中使用 特殊替换字符 (\$n ...)
  - 当为函数时：
    - 函数中不能用 特殊替换字符
    - 一参为正则匹配的文本
    - 倒数第二参为匹配到的子字符串在原字符串中的偏移量
    - 最后一参为被匹配的原始字符串
    - 其余参数为正则中每个分组匹配到的文本

```
function replacer(match, p1, p2, p3, offset, string) {  
  // match 为 'abc12345#$*%'      : 正则匹配的文本  
  // p1,p2,p3 分别为 abc 12345 #$*% : 即每个小组匹配到的文本, pn 表示有 n 个小组  
  // offset 为 0 : 匹配到的子字符串在原字符串中的偏移量。  
  // (比如, 如果原字符串是'abcd', 匹配到的子字符串时'bc', 那么这个参数将是1)  
  // string 为 'abc12345#$*%'      : 被匹配的原始字符串  
  return [p1, p2, p3].join(' - ');  
}
```

// 注意正则中的 括号 , 这里分有 3个组

```
var newString = 'abc12345#$*%'.replace(/([^\d]*) (\d*) ([^\w]*)/. replacer):
```

▲

赞同 8

▼

分享

以下是各种情况下的具体案例:

```
// 最基础的使用
'123'.replace('1', 'A') // 'A23'
'lalala 2Away0x2'.replace(/2(.*)2/, '$1') // 'LaLaLa Away0x'

// trim
const trim = str => str.replace(/(^\\s*)|(\\s*$)/g, '')

trim('  abc  ') // 'abc'

// format
const format = str =>
  (...args) =>
    str.replace(/{{(\\d+)}}/g, (match, p) => args[p] || '')

format('lalal{0}wowowo{1}hahah{2}')( '-A-', '-B-', '-C') // LaLa-A-wowow-B-haha-C
```

原理

先在模板中预留占位( {{填充数据}} ), 再将对应的数据填入

实现

要求1: 可填充简单数据

```
const tpl = (str, data) => str.replace(/{{(\\.*)}}/g, (match, p) => data[p])
tpl('<div>{{data}}</div>', {data: 'tpl'}) // '<div>tpl</div>'
```

要求2: 可填充嵌套数据

```
// 可根据占位符 {{data.a}} 中的 "." 来获得数据的依赖路径, 从而得到对应的数据
// 由于 使用 "." 连接, 所以其前后应为合法的变量名, 因此需重新构造正则
/* 合法变量名
 *   - 开头可为字符和少量特殊字符: [a-zA-Z$_]
 *   - 余部还可是数字: [a-zA-Z$_0-9]
 */
// 除开头外还需匹配 连接符 ".", 因此最终正则: /{{([a-zA-Z$_][a-zA-Z$_0-9\\.]*)}}/g
function tpl (str, data) {
  const reg = /{{([a-zA-Z$_][a-zA-Z$_0-9\\.]*)}}/g

  // 全局匹配模式下, replace 的回调在每次匹配时都会执行,
  // p 为占位符中的变量, 该例为 data.a
  return str.replace(reg, (match, p) => {
    const paths = p.split('.') // ['data', 'a']
    let result = data

    while (paths.length > 0)
      result = result[ paths.shift() ] // 得到路径最末端的数据
    return String(result) || match // 需转成字符串, 因为可能遇到 0, null 等数据
  })
}
tpl('<div>{{data.a}}</div>', {data: {a: 'tpl'}}) // '<div>tpl</div>'
```

最终代码

```
function tpl (str, data) {
  const reg = /{{([a-zA-Z$_][a-zA-Z$_0-9\\.]*)}}/g

  return str.replace(reg, (match, p) => {
    const paths = p.split('.')
    let result = data

    while (paths.length > 0)
```

▲

赞同 8

▶

分享

优缺点

- 优点: 简单
- 缺点: 无法在模板中使用表达式，所有数据都得事先计算好再填入，且填充的数据应为基础类型，灵活性差，难以满足复杂的需求

资料

详细语法

二、es6 模板字符串

介绍

- 模板字符串是 es6 中我最爱的特性啦！比起传统模板引擎，我更喜欢用模板字符串来编写组件
- 模板字符串包裹在 反引号(Esc按钮下面那个) 中，其中可通过 `${}` 的语法进行插值

```
// 特性一: 多行
`123123
 23213`

// 特性二: 字符串中可插值（强大的不要不要的）
/* 作为一门伪函数式编程语言，js 的很多语法都可以返回数据:
 *   - 表达式: 各种运算符表达式，三目( 可用来替代简单的判断语句)
 *   - 函数: 封装各种复杂的逻辑，最后返回一个值即可
 *   - 方法: 如一些有返回值的数据方法
 *   - 最强大的如数组的 map, filter ...
 */
// 以下字符串都等于 '123tpl456'
const str1 = `123${'tpl'}456`
const str2 = `123${false || 'tpl'}456`
const str3 = `123${true ? 'tpl' : ''}456`
const str4 = `123${ (function () {return 'tpl'})() }456`
const fn = () => 'tpl'
const str5 = `123${ fn() }456`
const str6 = `123${
  ['T', 'P', 'L'].map(s => s.toLowerCase()).join('')
}456`
console.log([str1, str2, str3, str4, str5, str6].every(s => s === '123tpl456'))

// 特性三: 模板函数 (个人很少用到)
var a = 5, b = 10
function tag (strArr, ...vals) {
  console.log(strArr, vals)
}
tag`Hello ${ a + b } world ${a * b}`
// strArr => ['Hello ', ' world ', '']
// vals   => [15, 30]  (${ }里的值)
```

案例

- - 由于直接用模板字符串当模板引擎了，所以就直接写个组件吧
  - [演示](#)
  - [代码](#)
- - 用这种方法写模板需注意的是一定要细分组件(很函数式，有种写 jsx 的既视感)

资料

▲

赞同 8

▶

分享

### 三、Function

#### 介绍

- Function 是 js 提供的一个用于构造 Function 对象的构造函数
- 使用

- ```
// 普通函数
function log (user, msg) {
  console.log(user, msg)
}
log('Away0x', 'lalala')
// Function 构造函数
const log = new Function('user', 'msg', 'console.log(user, msg)')
log('Away0x', 'lalala')
```

#### 实现

- 大多数前端模板引擎都是用这种方式实现的，其原理在于运用了 js Function 对象可将字符串解析为函数的能力。
- 一个普通模板引擎的工作步骤大致如下:

- ```
// 1. 编写模板
{@ if( data.con > 20 ) { @}
  <p>ifififififif</p>
{@ } else { @}
  <p>elseelseelseelse</p>
{@ } @}
// 2. 由模板生成函数体字符串
const functionbody = `
  var tpl = ''
  if (data.con > 20) {
    tpl += '<p>ifififififif</p>'
  } else {
    tpl += '<p>elseelseelseelse</p>'
  }
  return tpl
`
// 3. 通过 Function 解析字符串并生成函数
new Function('data', functionbody)(data)
```

- 由此可见，只要将 {@ @} 里的字符串内容生成 js 语句，而其余内容之前加上 一个 'tpl += ' 即可。
- 实现代码如下

- ```
const tpl = (str, data) => {
  const tplStr = str.replace(/\n/g, '')
    .replace(/{{(.+?)}}/g, (match, p) => `'+${p}'+`)
    .replace(/{@(.+?)@}/g, (match, p) => `'; ${p}; tpl += ``)
  // console.log(tplStr)
  return new Function('data', `var tpl='${tplStr}'; return tpl;`)(data)
}
// 测试
const str = `
  {@ if( data.con > 20 ) { @}
    <p>ifififififif</p>
  {@ } else { @}
```

▲

赞同 8

▶

分享

```
{@ for(var i = 0; i < data.list.length; i++) { @}
  <p>{{i }} : {{ data.list[i] }}</p>
{@ } @}
、

const data = {con:21, list: [1,2,3,4,5,76,87,8]}
console.log( tpl(str, data) )
// <p>ifififififif</p>
// <p>0 : 1</p><p>1 : 2</p><p>2 : 3</p><p>3 : 4</p><p>4 : 5</p><p>5 : 76</p><p>6 :
87</p><p>7 : 8</p>
```

ok, 一个最最简单的模板引擎就已经完成了，支持在模板中嵌入 js 语句，虽然只有不到10行，但还是挺强大的对不。

拓展

1. 实现模板类

为了能够更好的使用，将前面的代码抽成一个类。

- 需求:
  - 标识符格式有可能和后端模板引擎冲突，因此应实现成可配置的
    - {@ @} : 用于嵌套逻辑语句
    - {{ }} : 用于嵌套变量或表达式
  - 在模板中应能添加注释，注释有两种:
    - <!-- --> : 会输出
    - {# #} : 这种注释会在编译时被忽略，即只在模板中可见

```
class Tpl {
  constructor (config) {
    const defaultConfig = {
      signs: {
        varSign:      ['{{', '}}'],    // 变量/表达式
        evalSign:     ['{@', '@}'],    // 语句
        commentSign:  ['<!--', '-->'], // 普通注释
        noCommentSign: ['{#', '#}']    // 忽略注释
      }
    }
    // 可通过配置来修改标识符
    this.config = Object.assign({}, defaultConfig, config)
    // ['{{', '}}'] => /{([\\s\\S]+?)} /g 构造正则
    Object.keys(this.config.signs).forEach(key => {
      this.config.signs[key].splice(1, 0, '(.+?)')
      this.config.signs[key] = new RegExp(this.config.signs[key].join(''), 'g')
    })
  }
  // 模板解析
  _compile (str, data){
    const tpl = str.replace(/\n/g, '')
    // 注释
    .replace( this.config.signs.noCommentSign, () => '' )
    .replace( this.config.signs.commentSign, (match, p) => `'+<!-- ${p} -`
    // 表达式/变量
    .replace( this.config.signs.varSign, (match, p) => `'+(${p})+'`)
    // 语句
    .replace( this.config.signs.evalSign, (match, p) => {
      let exp = p.replace('&gt;', '>').replace('&lt;', '<')
      return `'; ${exp}; tpl += `
    })

    return new Function('data', `var tpl='${tpl}'; return tpl;`)(data)
  }
}
// 入口
```

▲

赞同 8

▼

分享



```
return new Tpl(config)
}

console.log( tpl().compile(str, data) ) // 得到
```

2. 注释的 BUG

上面的代码在解析一些特殊模板注释(如下)时会出错

```
<!-- {{a}} -->
// 由于注释中有标识符，因此会将 a 作为变量解析，会报未定义错误
```

解决: 在解析注释时，如注释里有标识符，则将其先替换成其他符号，等语句变量的解析完成时，再替换回来

```
.replace( this.config.signs.commentSign, (match, p) => {
    const exp = p.replace(/\{\<\|\>\}/g, match => `&*&${match.charCodeAt()}&*&`)
    return `'+<!-- ${exp} -->'+`
})
// ... 解析变量和语句
.replace(/\&\*\&(.*?)\&\*\&/g, (match, p) => String.fromCharCode(p))
```

3. 语法模式

在模板里写 js 好烦呀，各种 '{' 乱飞，有些模板提供了更好看的语法：

```
{@ if data.con > 20 @} // if (data.con > 20) {
  <p>ifififififif</p>
{@ elif data.con === 20 @} // } else if (data.con === 20) {
  <p>elseelseelseelseififififififif</p>
{@ else @} // } else {
  <p>elseelseelseelse</p>
}/{@ if @} // }
// for (var index = 0; index < data.list.length; index++) { var item = data.list[index]
{@ each data.list as item @}
  <p>循环 {{ index + 1 }} 次: {{ item }}</p>
}/{@ each @}

```

其实就是在解析语句时多做一些处理而已:

```
// 配置中增加 syntax 属性，默认 false，其为 true 是开启 语法模式
// 配置中增加语法模式结束语句的标识符: endEvalSign: ['{/@', '@}']
// 给 Tpl 类添加方法，用于 语法模式 的语句解析
_syntax (str) {
  const arr = str.trim().split(/\s+/)
  let exp = str

  if (arr[0] === 'if') {
    // if (xx) {
    exp = `if ( ${arr.slice(1).join(' ')} ) {`
  } else if (arr[0] === 'else') {
    // } else {
    exp = `} else {`
  } else if (arr[0] === 'elif') {
    // } else if (xx) {
    exp = `} else if ( ${arr.slice(1).join(' ')} ) {`
  } else if (arr[0] === 'each') {
    // for (var index = 0, len = xx.length; index < len; index++) { var item = xx[
    exp = `for (var index = 0, len = ${arr[1]}.length; index < len; index++) {var
  }

  return exp
}
```

▲

赞同 8

➤

分享

```
// 语法模式
exp = this.config.syntax ? this._syntax(exp) : exp
return `'; ${exp}; tpl += `
})
// 增加结束标识的解析 {@ if @} {@ each @}
.replace( this.config.signs.endEvalSign, () => `"} tpl += `)
```

4. 过滤器

很多模板引擎中都有提供很多好用的过滤器:

```
// 字符串转大写的过滤器
<p>{{ 'tpl' | upper }}</p> => <p>TPL</p>
// 支持流式
<p>{{ 'tpl' | f1 | f2 }}</p>
```

现在我们来编写这个拓展:

```
// 由于过滤器是对于变量的操作, 所以只需在解析变量标识符{{}}的过程中做一下处理即可
// {{}} => 无过滤器直接返回,  {{ xx | xx }} 有过滤器则调用 Filters 类中对应的过滤器函数

// 过滤器函数
const Filters = {
  upper: str => str.toUpperCase()
}

// 解析 {{}}
.replace( this.config.signs.varSign, (match, p) => {
  const filterIndex = p.indexOf('|')
  let val = p

  if (filterIndex !== -1) { // 有过滤器
    const
      arr      = val.split('|').map(s => s.trim()),
      filters = arr.slice(1) || [],
      oldVal   = arr[0]

    val = filters.reduce((curVal, filterName) => {
      if ( ! Filters[filterName] ) {
        throw new Error(`没有 ${filterName} 过滤器`)
        return
      }
      return `Filters['${filterName}'](${curVal})`
    }, oldVal)
  }

  return `'+(${val})+'`
})

// 使用
// <h1>{{ 'tpl' | upper | reverse }}</h1> // => 'LPT'
```

- 至此该模板引擎就完成了, 总结下功能:
  - 支持在模板中使用 js 语句
  - 支持自定义标识符
  - 支持更简洁的语法模式
  - 支持过滤器

其他

虽然这个模板工具还是有点简陋, 比如没有个很友善的报错机制, 不支持 include 等功能, 但日常跑跑小项目什么的已经足够强大了。测试了下, 我们这个 模板工具 的性能略优于 underscore.template, 但比 Mustache 要差一些。

▲

赞同 8

▶

分享

资料  
详细语法

只有20行Javascript代码！手把手教你写一个页面模板引擎

最后

如还有其他的实现方式，欢迎提意见  
github 完整代码

编辑于 2017-04-11

前端开发   JavaScript   前端工程师

推荐阅读



学web前端，一定要知道这些实用的JavaScript片段

前端一粒维...   发表于前端开发学...



菜鸟前端的咸鱼之旅-第一天：参数传值

lost




编写一个简!板引擎


凯斯

2 条评论


切换为时间排序


写下你的评论...





 颜海镜

github.com/yanhaijing/t... template.js 了解下

 1

 tjwyz



 赞

2018-11-16

09-17