

```
$ hexo n vue-skill render-JSX
```

title: 渲染函数 & JSX

- [前言](#)
- [render](#)
 - [基础](#)
 - [节点、树以及虚拟 DOM](#)
 - [最简单的使用方式](#)
 - [基础示例](#)
 - [深入使用](#)
 - [render: h => h\(App\)](#)
 - [小结](#)
- [JSX](#)
 - [什么是JSX](#)
 - [安装配置](#)
 - [实现](#)
 - [vue实现css-in-js](#)
- [尾声](#)

前言

官方文档介绍的[渲染函数](#)似乎有点太过偏重表现而放轻原因。已经看过这部分好几次，还是有点不知所以然。这也不是第一次看不明白文档的介绍了，还是需要自己整理下知识点。所以，这里的内容是对文档做降级处理，好让这一块知识点更容易理解一点。文档有的东西，这里不会再重复。

建议阅读顺序：先粗略浏览下官方文档；精读这篇文章同时结合文档。

render

先讨论render，再讨论JSX。因为JSX是为了render的编写更加方便。

基础

[渲染函数 & JSX - 基础](#) 这一小节官网介绍的是render的使用的场景。还举了个例子，真的不尽人意，起码看完我没能回答自己，render是啥。

[render](#)是一个实例方法，跟字符串模板/template选项一样可以创建DOM。比起字符串模板，render的强大之处在于：字符串模板创建的DOM是静态的；render可以借助js的能力动态创建DOM。（不要忘了render本质上是一个被vue框架封装的js函数）

静态/动态DOM如何理解？template不是也可以使用v-if v-show v-for之类的指令吗？

虽然template可以使用以上指定，对DOM进行修改，可是其无法改变HTML tag。以上的指令只是切换/移除/新增固定的HTML tag罢了。（举个反例，随机生成HTML tag 试试？）

官网基础这一小节，我们只需知道render是可以使用js能力创建HTML，是template的升级版就够了。

接下来的三个小节都是在讲如何使用render，我们先从[虚拟DOM](#)这个知识点入手。**【基础】**小节的例子，也会相应的介绍。

- [\[节点 树以及虚拟 DOM\]\(https://cn.vuejs.org/v2/guide/render-function.html#节点、树以及虚拟 DOM\)](#)
- [createElement 参数](#)
- [使用 JavaScript 代替模板功能](#)

节点、树以及虚拟 DOM

HTML DOM 节点的知识点相信基础扎实的你烂熟于心了，也看过文档复习了，这里不重复。

虚拟节点（virtual node）：描述 DOM 节点的JS对象，该对象保存着节点的相应信息。别名 VNode。

那 DOM 是由节点组成，虚拟DOM的概念也自然很好理解：由Vnode组成的节点树结构。

而render返回的值就是一个虚拟DOM，虚拟DOM最终也会被转换为真实的DOM挂载在页面上。

最简单的使用方式

我学东西喜欢由简单到复杂，如果文档一开始就写个Hello World该多好。

```
<div class="skill-render">skill-render</div>
```

我们用`template`，`render`分别创建以上相同的dom。

```
// skill-template.vue
<template>
  <div class="skill-render">
    {{options.name}}
  </div>
</template>
<script>
export default {
  name: 'skill-render'
}
</script>
```

自行与下面的版本找不同。

```
// skill-render.vue
<script>
export default {
  name: "skill-render",
  render(createElement) {
    return createElement(
      "div",
      {
        class: "skill-render"
      },
      this.$options.name
    );
  }
};
</script>
```

终于我们知识点足够使用最简单的render，可是这里还是要说明下。`render`返回的是`createElement`函数运行后的返回值。所以其实是**`createElement`有能力创建虚拟DOM，而`render`依赖`createElement`的能力渲染为HTML**。两者的作用很符合函数名（笑）。

你已经知道`createElement`可以创建虚拟DOM，现在可以好好看看 [createElement 参数](#)了。

基础示例

到现在，我们才要开始讲文档的基础示例，说实话，这个示例真的挺基础。

文档要解决的问题其实就是要设计一个`anchored-heading`组件。

- 这个组件里面的HTML tag (`h1~h6`) 可以使用prop动态决定，并且标题内部使用了插槽。
- 而使用`template`方案，代码冗长又存在重复问题。
- 所以使用了`render`函数动态创建HTML tag。
- 在基础小节的最后还进行了一波劝退：深入渲染函数之前推荐阅读[实例属性 API](#)。

深入使用

只要在原生的 JavaScript 中可以轻松完成的操作，Vue 的渲染函数就不会提供专有的替代方法。

文档也写得很详细了，深入底层的代价可以更好地控制交互细节。

这里略过不重复。

render: h => h(App)

当你认真真把关于渲染函数的资料看完，你也自然理解了`main.js`中，`render`为什么这么写。

```
new Vue({
  render: h => h(App)
}).$mount('#app')
```

小结

render函数并不复杂，理解其需要返回虚拟DOM，最后会挂载在页面上就可以了。编写会比较繁琐，具体的体现就是vue提供的某些指令需要自己实现。用途会比较少，不过有助于我们研究vue，如果你想之后阅读vue源码的话。

JSX

vue为了render编写更加方便，createElement可以使用JSX替换之。vue也支持JSX语法，不过需要一个[Babel 插件](#)支持。

什么是JSX

JSX的定义，我们引用[下官网](#)的描述。

```
JSX - a faster, safer, easier JavaScript
...
...
...
```

好吧，那我按照自己的理解来介绍了。

JSX是JavaScript的语法扩展。类似模版语言，本质上是js语法。

安装配置

安装配置也很简单，使用也有介绍。

```
npm install @vue/babel-preset-jsx @vue/babel-helper-vue-jsx-merge-props
```

如果是vue-cli 2.x创建的vue项目，需要.babelrc配置，vue-cli 3.x+安装后忽略配置，可直接使用。

```
{
  "presets": ["@vue/babel-preset-jsx"]
}
```

实现

新建一个vue文件。

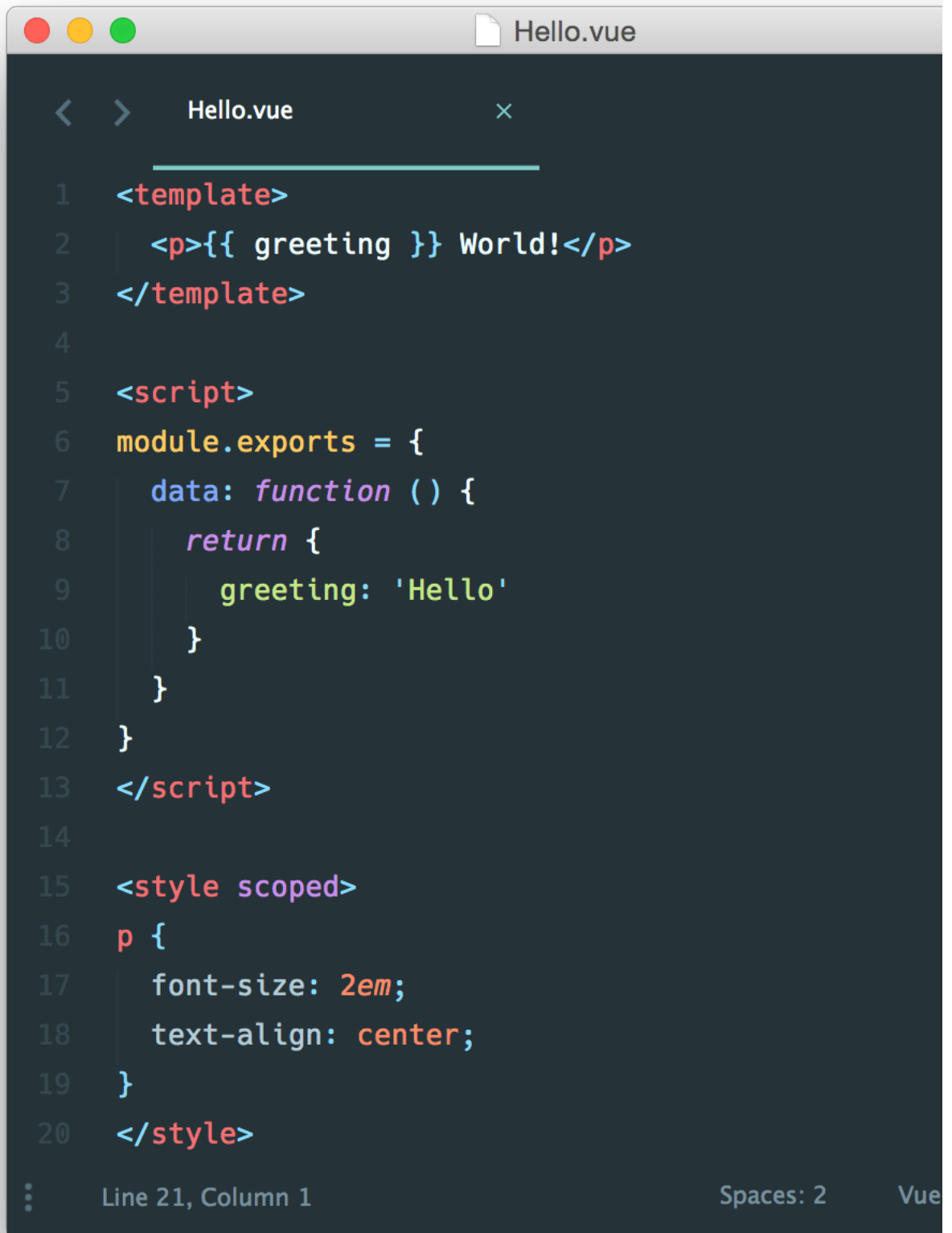
```
<!-- JSX.vue -->
<script>
export default {
  name: "skill-jsx-vue",
  data() {
    return {
      type: "vue"
    };
  },
  render() {
    return (
      <p>
        hello
        <b> {this.type}</b>
      </p>
    )
  }
}
</script>
```

然后发现，这里甚至可以使用去掉<script>标签，直接用js文件编写，不更方便？于是我们删除<script>标签，另存为JSX.js，相应该改变组件名还有type数据就能得到以下文件。

```
// JSX.js
export default {
  name: 'skill-jsx-js',
  data() {
    return {
      type: 'js'
    }
  }
}
```

```
    }  
  },  
  render() {  
    return <p>hello <b>{this.type}</b></p>  
  }  
}
```

美哉美哉。以后可以抛弃*.vue文件直接使用*.js编写组件了，可把自己高兴坏了。我也不喜欢*.vue的格式。



```
1 <template>
2   <p>{{ greeting }} World!</p>
3 </template>
4
5 <script>
6   module.exports = {
7     data: function () {
8       return {
9         greeting: 'Hello'
10      }
11    }
12  }
13 </script>
14
15 <style scoped>
16   p {
17     font-size: 2em;
18     text-align: center;
19   }
20 </style>
```

Line 21, Column 1 Spaces: 2 Vue

经常要考虑当前行的作用域问题，是vue示例呢，还是原生js的作用域呢。
but!!

vue实现css-in-js

```
// JSX.vue
<script>
export default {
  // ..
  render() {
    return (
      <p class="skill-jsx-vue">
        hello
        <b> {this.type}</b>
      </p>
    )
  }
}

<style lang="stylus">
.skill-jsx-vue {
  color: green;
}
</style>
```

JSX.js是原生js文件，不支持<style lang="stylus">的语法解析。那也没关系，css-in-js就是为了解决在js中编写css样式而出现的一种方案。

虽然隔壁框架也有成熟的JSX还有css-in-js解决方案，不过这与本文无关，相关内容不会讨论。

vue的css-in-js 解决方案是[vue-styled-components](#)，当然不止这一个，还有其他的解决方案等待我们去探索。

安装完后，我们需要声明一个组件，这个组件只有样式，内容是使用插槽传入的，插槽自动定义的，这个组件供render使用即可。

```
// JSX.js
import styled from 'vue-styled-components';

const StyledP = styled.p`
  color: wheat;
`

export default {
  name: 'skill-jsx-js',
  data() {
    return {
      type: 'js'
    }
  },
  render() {
    return <StyledP class="skill-jsx-js">hello <b>{this.type}</b></StyledP>
  }
}
```

StyledP也是一个vue组件，而render的<StyledP></StyledP> 相当于使用这个vue组件，组件名是js变量，大小写敏感。

思考一下，写一个带有样式的DOM，就需要声明一个vue组件，这样情况会不会造成组件滥用？

当然你可能会觉得，这是不是违背了关注点分离的原则。对于组件化系统的框架的，组件是组成项目的基本单位，更加关注的是，以组件为单位，而一个组件又包括HTML、CSS、JS，vue觉得将其所有组合起来比较合理。

[如何看待关注点分离？](#)

当然我也会提供这一章的源码还有示例给你，还顺便实现了vue-styled-components官网的例子。

- JSX.vue [源码](#) [示例](#)
- JSX.js [源码](#) [示例](#)
- css-in-js.js [源码](#) [示例](#)

尾声

今天我们从render出发，知道了虚拟DOM这一很重要的概念。途中认识了JSX，最后还结合了css-in-js实现JSX+css-in-jss。可是这些东西实际作用比较低。render函数示例开发中应用场景并不多，而JSX，css-in-js，vue对其的支持又有点不知道说什么好。不过如果你是抱着研究vue的精神而不是使用vue的想法看待以上内容，相信还是对你有点帮助的。