

# 前端新宠 Svelte 带来哪些新思想？赶紧学起来！

[Svelte](#) 是我用过最爽的框架，就算 Vue 和 React 再强大，生态再好，我还是更喜欢 Svelte，因为它开发起来真的很爽。

其实在很久之前我就注意到 [Svelte](#)，但一直没把这个框架放在心上。

因为我之前的工作主要使用 Vue，偶尔也会接触到一些 React 项目，但完全没遇到过使用 Svelte 的项。

直到 Vite 的出现，我才开始开始重视 Svelte。

从 [Vite文档](#) 里可以看到它支持这些模板：

JavaScript	TypeScript
<a href="#">vanilla</a>	<a href="#">vanilla-ts</a>
<a href="#">vue</a>	<a href="#">vue-ts</a>
<a href="#">react</a>	<a href="#">react-ts</a>
<a href="#">preact</a>	<a href="#">preact-ts</a>
<a href="#">lit</a>	<a href="#">lit-ts</a>
<a href="#">svelte</a>	<a href="#">svelte-ts</a>

能让祖师爷也重视的框架，不简单不简单~

我喜欢用 Demo 的方式学习新技术，[Svelte 官方入门教程](#) 就提供了这种方式。

这是我觉得入门比较舒服且方便日后搜索的学习方式。

虽然 [Svelte 官方入门教程](#) 已经给出很多例子，而且 [Svelte中文网](#) 也有对应的翻译，但有些翻译看上去是机译，而且部分案例可能不太适合新手学习~

本文的目的是把 Svelte 的学习流程梳理出来，让第一次接触 Svelte 的工友能顺利上手。

**本文适合人群：**有 **HTML** 、 **CSS** 、 **JS** 基础，知道并已经安装了 **Node** 。

如果你是打算从0开始学习前端，那本文暂时还不适合你阅读。

## Svelte 简介

[Svelte](#) 是一个构建 web 应用程序的工具。

传统框架如 React 和 Vue 在浏览器中需要做大量的工作，而 Svelte 将这些工作放到构建应用程序的编译阶段来处理。

需要注意，Svelte 是一款编译器。它可以将按照规定语法编写的代码打包成浏览器能运行的项目。

和其他前端框架一样，同样也是使用 **HTML** 、 **CSS** 和 **JavaScript** 进行开发。

## 作者

在学习 Svelte 之前先了解一下它的父亲（作者）。

Svelte 的作者叫 [Rich Harris](#)，正在吃东西的这位就是他。



可能国内大多数工友对他不是很熟悉（我也完全不熟），但应该听过 [Rollup](#)。

没错，他也是 [Rollup](#) 的爸爸。

他在开发 Svelte 之前还开发过 [Ractive.js](#)，听说 Vue 的部分实现也是受到了 Ractive 的启发。

关于 **Rich Harris** 的介绍还有很多，我搜到的资料上这样介绍到：

- 大学专业是学哲学的
- 在纽约时报调查组工作的图形编辑，身兼记者和开发者职位

还有更多关于他和 Svelte 的介绍，可以看看 [《Svelte - The magical disappearing UI framework - Interview with Rich Harris》](#)

## | Svelte 的优势

Svelte 翻译成中文就是“苗条”的意思，侧面表明它打包出来的包非常小。

Svelte 主要优势有以下几点。

## 1. 编译器

在打开[Svelte官网](#)时就能看到这样的介绍。

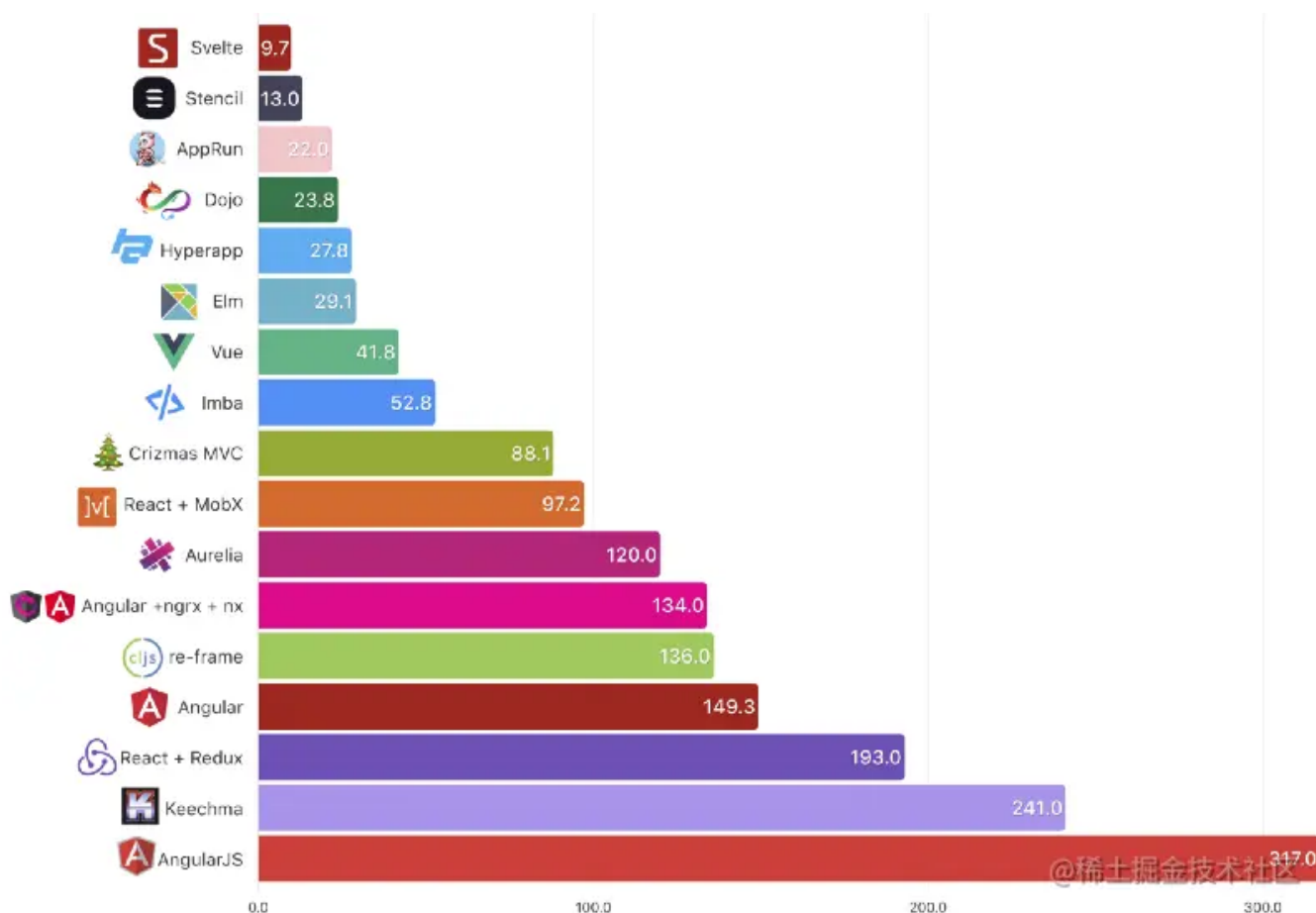
Svelte 是一种全新的构建用户界面的方法。传统框架如 React 和 Vue 在浏览器中需要做大量的工作，而 Svelte 将这些工作放到构建应用程序的编译阶段来处理。

Svelte 组件需要在 `.svelte` 后缀的文件中编写，Svelte 会将编写好的代码翻编译 `JS` 和 `CSS` 代码。

## 2. 打包体积更小

Svelte 在打包会将引用到的代码打包起来，而没引用过的代码将会被过滤掉，打包时不会加入进来。

在 [《A RealWorld Comparison of Front-End Frameworks with Benchmarks \(2019 update\)》](#) 报告中，对主流框架进行了对比。



在经过 `gzip` 压缩后生成的包大小，从报告中可以看出，Svelte 打包出来的体积甩开 Vue、React 和 Angular 几条街。

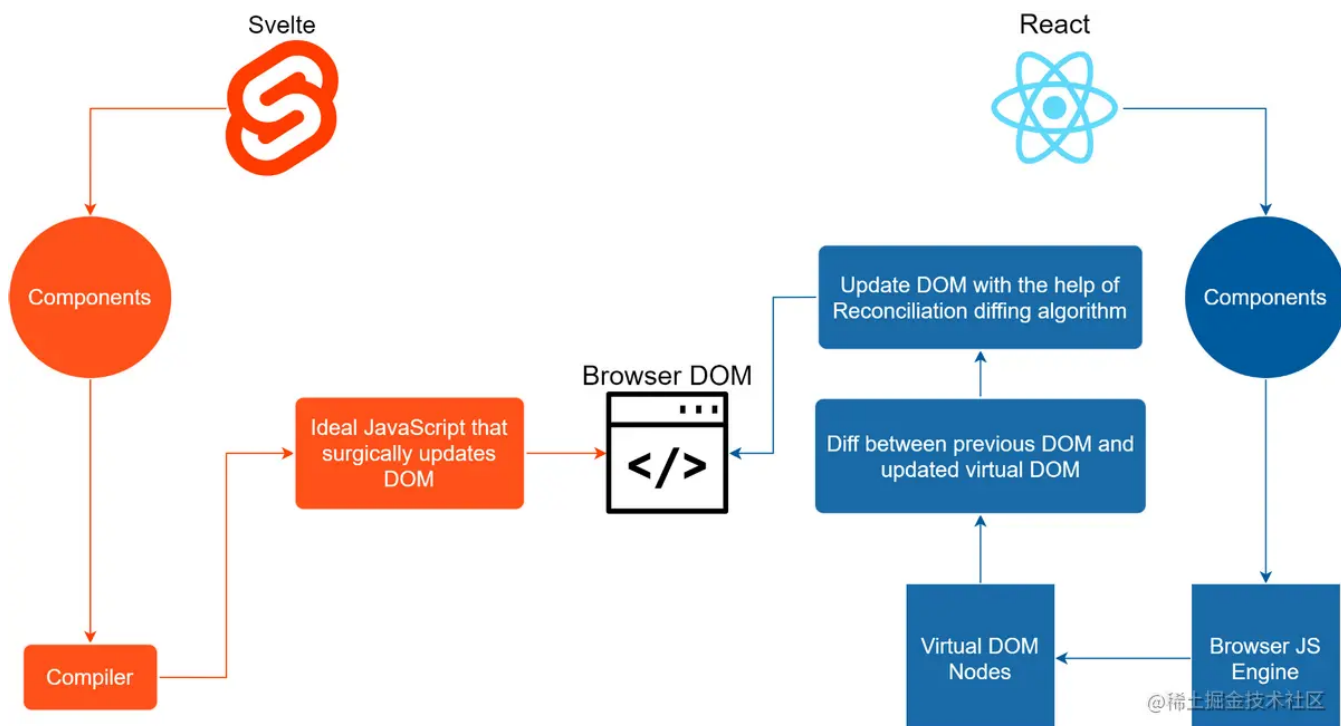
这是因为经过 Svelte 编译的代码，仅保留引用到的部分。

### 3. 不使用 Virtual DOM

Virtual DOM 就是 **虚拟DOM**，是用 JS 对象描述 DOM 节点的数据，由 React 团队推广出来的。

**虚拟DOM** 是前端的网红，因此也有很多开发者开始研究和搞辩论赛。

网上有一张图对比了 Svelte 和 React 在数据驱动视图的流程



其实主要对比了使用虚拟DOM和直接操作真实DOM的区别。

在 React 中实现数据驱动视图大概流程是这样的：

数据发生变化 -> 通过diff算法判断要更新哪些节点 -> 找到要更新的节点 -> 更新真实DOM rust 复制代码

Vue 的数据更新原理其实也差不多，只是实现方式和使用语法会有所不同。

**diff算法** 会根据数据更新前和更新后生成的虚拟DOM进行对比，只有两个版本的虚拟DOM存在差异时，才会更新对应的真实DOM。

使用虚拟DOM对比的方式会比直接对比真实DOM的效率高。

而且真实DOM身上挂载的属性和方法非常多，使用虚拟DOM的方式去描述DOM节点树会显得更轻便。

但这也意味着每次数据发生变化时都要先创建一个虚拟DOM，并使用 **diff算法** 将新虚拟DOM与旧虚拟DOM进行比对，这个步骤会消耗一点性能和需要一点执行时间。

而 Svelte 在未使用虚拟DOM的情况下实现了响应式设计。

我以粗暴的方式理解：Svelte 会监听顶层组件所有变量，一旦某个变量发生变化，就更新使用过该变量的组件。这就仅仅只需更新受影响的那部分DOM元素，而不需要整个组件更新。

综上所述，在我的理解力，虚拟DOM的思想很优秀，也是顺应时代的产物，但虚拟DOM并不是最快的，JS 直接操作 DOM 才是最快。

[《Virtual DOM is pure overhead》](#) 是 Svelte 官网上的一篇博客，专门讨论虚拟DOM。有兴趣的工友可以看看~

## 4. 更自然的响应式

这也是我刚接触 Svelte 时立刻喜欢上的理由。

这里说的响应式设计是只关于数据的响应，而不是像 Bootstrap 的响应式布局。

现在流行的前端框架基本都使用 **数据驱动视图** 这个概念，像 Vue 和 React 这些框架，都有响应式数据的概念。

但 Vue 和 React 在数据响应方面还是有点“不那么自然”，我简单举几个例子：

- 在 React 中，如果需要更新数据并在视图中响应，需要使用 `setState` 方法更新数据。
- 在 Vue2 中，响应式数据要放在 `data` 里，在 `methods` 中使用 `this.xxx` 来更新数据。
- 在 Vue3 的 Composition API 语法中，需要使用 `ref` 或者 `reactive` 等方法包裹数据，使用 `xxx.value` 等方式修改数据。

上面这几种情况，感觉多少都添加了点东西才能实现响应式数据功能（至少在普通开发者开发时是这样）。

在 Svelte 的理念中，响应式应该给开发者一种无感体验，比如在 Excel 中，当我规定 C1 单元格的值是 A1 + B1 的和，设置好规则后，用户只需要修改 A1 和 B1 即可，C1 会自动响应，而不需再做其他操作。

	A	B	C	D
1	+		0	
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				金猿王猴金猿王猴

在这方面，Svelte 我认为在现阶段是做得最自然的。

# 雷猴

金猿王猴金猿王猴

html 复制代码

```
<h1>{name}</h1>

<script>
  let name = '雷猴'

  setTimeout(() => {
    name = '鲨鱼辣椒'
  }, 1000)
</script>
```

上面的代码中，1秒后修改 `name` 的值，并更新视图。

从代码就能看出，在使用 Svelte 开发项目时，开发者一般无需使用额外的方法就能做到和 Vue、React 的响应式效果。

如果你对 Svelte 响应式原理感兴趣，推荐阅读 [FESKY](#) 的 [《Svelte 响应式原理剖析 —— 重新思考 Reactivity》](#)

也可以看看 [《Rethinking reactivity》](#)，看看官方对 reactivity 的思考。



## 5. 性能强

Stefan Krause 给出一份 [性能测试报告（点击可查看）](#) 对比里多个热门框架的性能。从 Svelte 的性能测试结果可以看出，Svelte 是相当优秀的。

## 6. 内存优化

[性能测试报告（点击可查看）](#) 也列出不同框架的内存占用程度，Svelte 对内存的管理做到非常极致，占用的内存也是非常小，这对于配置不高的设备来说是件好事。

第5和6点，由于测试报告比较长，我没截图放进文中。大家有兴趣可以[点开链接查看测试报告](#)。

## 7. 更关注无障碍体验

在使用 Svelte 开发时会 **自动对无障碍访问方面的体验进行检测**，比如 `img` 元素没有添加 `alt` 属性，Svelte 会向你发出一条警告。无障碍体验对特殊人事来说是很有帮助的，比如当你在 `img` 标签中设置好 `alt` 属性值，使用有声浏览器会把 `alt` 的内容读出来。

在此我还要推荐2本关于设计体验的书。

- 《点石成金：访客至上的Web和移动可用性设计秘笈》
- 《包容性Web设计》

它们的封面长分别这个样子



Steve Krug



# DON'T MAKE ME THINK


原书第3版  
Third Edition

*Revisited*

## 点石成金

[美] Steve Krug 著 蒋芳 译  
访客至上的Web和移动可用性设计秘笈

A Common Sense Approach to Web Usability  
and Mobile

 机械工业出版社  
China Machine Press

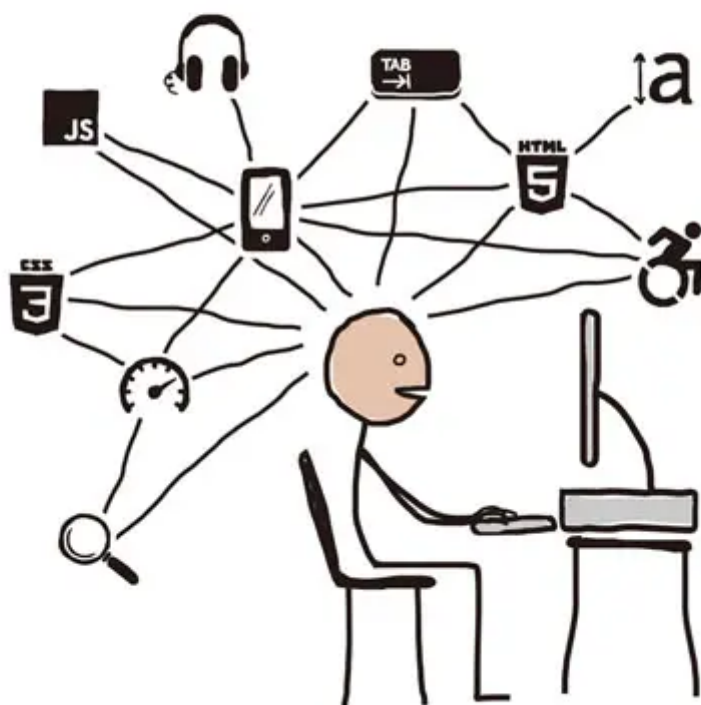
@稀土掘金技术社区

# 包容性Web设计

## INCLUSIVE DESIGN PATTERNS

Coding Accessibility Into Web Design

[美] Heydon Pickering 著 于坤 译



中国工信出版集团

人民邮电出版社  
POSTS & TELECOM PRESS

@稀土掘金技术社区

Svelte 的优势肯定还有很多，但由于我开发经验不足，只能总结出以上这些了。如果你对 Svelte 有更多理解，欢迎在评论区补充~

## | Svelte 的不足

1. Svelte 对 IE 是非常不友好的，但我并不把这放在眼里。如果想兼容 IE 我还是推荐使用 jQuery。
2. Svelte 的生态不够丰富。由于是“新宠”，生态方面肯定是不如 Vue 和 React 的。

## | 与 Svelte 相关的库

### Sapper

[Sapper 官网地址](#)

Sapper 是构建在 Svelte 上的框架，Sapper 提供了页面路由、布局模板、SSR等功能。

### Svelte Native

[Svelte Native 官网地址](#)

Svelte Native 是建立在 [NativeScript](#) 之上的产物，可以开发安卓和iOS应用，是一个跨端技术。

有点类似于 React Native 和 Weex 之类的东西。

### svelte-gl

[svelte-gl 仓库](#)

svelte-gl 还没正式发布，但这是个很有趣的工具，它和 [three.js](#) 类似，专门做 3D应用的。

虽然现在 github 上的 Star 还不是很多，但也可以写些 demo 玩玩。

## 创建项目

在开始之前，你需要在电脑上安装 [Node](#) 环境。

编辑工具我使用了 VS Code ，同时安装了 [Svelte for VS Code 扩展插件](#)。

使用 Svelte 前，必须有一个开发环境。

创建或使用开发环境有以下几种方式：

1. REPL
2. Rollup 版
3. Webpack 版
4. Parcel 版
5. Vite 版

本文使用的是 Vite 创建项目，但上面列出的所有方式我都会逐一说说。

## REPL

REPL 是 Svelte 提供的一个线上环境，打开 [Svelte 官网](#) 可以看到顶部导航栏上面有个 [REPL](#) 的选项。点击该选项就可以跳转到 Svelte 线上开发环境了。



The screenshot shows the Svelte website homepage. At the top, the navigation bar includes links for '入门教程', 'API', '实例', 'REPL', '博客', 'FAQ', 'Sapper', and '英文官网'. Red arrows point from the 'REPL' link in the navigation bar to the 'REPL' link in the main content area. The main content area features the Svelte logo and the tagline 'Cybernetically enhanced web apps'. Below this, there are three colored boxes with text: '减少代码量' (Reduce code volume), '无虚拟 DOM' (No virtual DOM), and '真正的反应能力' (True reactivity). Each box has a '了解更多' (Learn more) link. At the bottom, there is a section titled 'Svelte 是一种全新的构建用户界面的方法。' (Svelte is a brand new way to build user interfaces.) followed by a paragraph and a '阅读此博文' (Read this article) link. On the right side, there is a code block with installation instructions and a '@稀土掘金技术社区' (Xinshu Juejin Technology Community) watermark.

SVELTE

入门教程 API 实例 **REPL** 博客 FAQ Sapper 英文官网

# SVELTE

Cybernetically enhanced web apps

### 减少代码量

重复利用你所掌握的编程语言 - HTML、CSS 和 JavaScript，构建的组件无需依赖模板文件。

[了解更多 →](#)

### 无虚拟 DOM

Svelte 将你的代码编译成体积小、不依赖框架的普通 JS 代码，让你的应用程序无论启动还是运行都变得迅速。

[了解更多 →](#)

### 真正的反应能力

无需复杂的状态管理库，Svelte 为 JavaScript 自身添加反应能力。

[了解更多 →](#)

Svelte 是一种全新的构建用户界面的方法。传统框架如 React 和 Vue 在浏览器中需要做大量的工作，而 Svelte 将这些工作放到构建应用程序的编译阶段来处理。

与使用虚拟（virtual）DOM 差异对比不同。Svelte 编写的代码在应用程序的状态更改时就能像做外科手术一样更新 DOM。

[阅读此博文](#) 了解更多信息。

```
npx degit sveltejs/template my-svelte-project
# or download and extract this .zip file
cd my-svelte-project
# to use TypeScript run:
# node scripts/setupTypeScript.js

npm install
npm run dev
```

@稀土掘金技术社区

Hello world

App.svelte +

```
1 <script>
2   let name = 'world';
3 </script>
4
5 <h1>Hello {name}!</h1>
```

Result JS output CSS output

# Hello world!

Console @稀土掘金技术社区 CLEAR

REPL 是 **read**(读取)、**evaluate**(执行)、**print**(打印) 和 **loop**(循环) 这几个单词的缩写。

如果你只是想尝试 Svelte 的某些功能或者测试小型代码，可以使用这款线上工具。

REPL 还提供了多组件开发，按左上角的 **+** 号 可以创建新组件。组件的内容稍后会说到。

界面右侧，顶部有3个选项：

- **Result**：运行结果。
- **JS output**：Svelte 编译后的 **JS** 代码。
- **CSS output**：Svelte 编译后的 **CSS** 代码。

Hello world

App.svelte +

```
1 <script>
2   let name = 'world';
3 </script>
4
5 <h1>Hello {name}!</h1>
```

Result JS output CSS output

# Hello world!

Console @稀土掘金技术社区 CLEAR

在 REPL 界面右上角还有一个下载按钮。

入门教程 API 实例 REPL 博客 FAQ Sapper 英文官网

Download icon (downward arrow) is highlighted with a red arrow.

Result JS output CSS output

# Hello world!

@稀土掘金技术社区

当你在线上环境写好代码，可以点击[下载按钮](#)把项目保存到本地，下载的文件是一个 **zip**，需要自己手动解压。

然后使用以下命令初始化项目并运行即可。

shell 复制代码

```
# 1、初始化项目
npm install

# 2、运行项目
npm run dev

# 3、在浏览器访问 http://localhost:5000
```

运行结果：

# Hello world!

© 林士豪 2021 年 10 月 10 日

## | Rollup 版

Svelte 官方也提供了一个命令，可以下载 Svelte 项目到本地。

命令最后需要输入你的项目名称。

复制代码

```
# 1、下载模板
npx degit sveltejs/template 项目名称

# 2、安装依赖
npm install

# 3、运行项目
npm run dev

# 4、在浏览器访问 http://localhost:8080
```



运行结果：

# HELLO WORLD!

Visit the [Svelte tutorial](#) to learn how to build Svelte apps.

@稀土掘金技术社区

这是官方提供的创建项目方式，这个项目是使用 Rollup 打包的。

Rollup 和 Svelte 都是同一个作者 ([Rich Harris](#)) 开发的，用回自家东西很正常。

## | Webpack 版

如果你不想使用 Rollup 打包项目，可以尝试使用 Webpack。

复制代码

```
# 1、下载模板
npx degit sveltejs/template-webpack 项目名称

# 2、安装依赖
npm install

# 3、运行项目
npm run dev

# 4、在浏览器访问 http://localhost:8080/
```

运行结果：

# HELLO WORLD!

Visit the [Svelte tutorial](#) to learn how to build Svelte apps.

@稀土掘金技术社区

## | Parcel 版

我并 **不推荐使用** 该方法创建项目，因为 Svelte 并没有提供使用 Parcel 打包工具的模板。但 [GitHub 上有第三方的解决方案（点击访问仓库）](#)。

将 [DeMoorJasper/parcel-plugin-svelte](#) 的代码下载下来。

shell 复制代码

```
# 1、进入 `packages/svelte-3-example` 目录
```

```
# 2、安装依赖
```

```
npm install
```

```
# 3、运行项目
```

```
npm run start
```

```
# 4、在浏览器访问 http://localhost:1234/
```

运行结果：

# Hello Anonymous,

the time is 23:25:22



@稀土掘金技术社区

## | Vite 版

本文接下来所有例子都是使用 Vite 创建 Svelte 项目进行开发的。

使用 Vite 创建项目的原因是：**快!**

# 1、下载模板的命令  
`npm init vite@latest`

# 2、输入项目名

# 3、选择 **Svelte** 模板（我没选 **ts**）

# 4、进入项目并安装依赖  
`npm install`

# 5、运行项目  
`npm run dev`

# 6、在浏览器访问 `http://127.0.0.1:5173/`

csharp 复制代码

运行结果：



# Vite + Svelte

count is 0

Check out [SvelteKit](#), the official Svelte app framework powered by Vite!

Click on the Vite and Svelte logos to learn more

@稀土掘金技术社区

本文使用 Vite 创建项目，目录结构和 Rollup 版 创建出来的项目结构稍微有点不同，但开发逻辑是一样的。

## 起步

`index.html`、`src/main.js` 和 `src/App.svelte` 这三个是最主要的文件。

`index.html` 是项目运行的入口文件，它里面引用了 `src/main.js` 文件。

`src/main.js` 里引入了 `src/App.svelte` 组件，并使用以下代码将 `src/App.svelte` 的内容渲染到 `#app` 元素里。

```
const app = new App({
  target: document.getElementById('app')
})
```

`target` 指明目标元素。

我们大部分代码都是写在 `.svelte` 后缀的文件里。

`.svelte` 文件主要包含 多个 HTML 元素、1个 `script` 元素 和 1个 `style` 元素。这3类元素都是可选的。

我们主要的工作目录是 `src` 目录。

为了减轻学习难度，我们先做这几步操作。

## 1、清空全局样式

如果你使用 Rollup版 创建项目，不需要做这一步。

在使用 Vite 创建的 Svelte 项目中，找到 `src/app.css` 文件，并把里面的内容清空掉。

## 2、改造 `src/App.svelte`

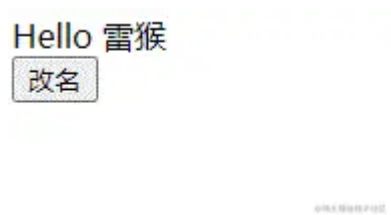
将 `src/App.svelte` 文件改成以下内容

```
<script>
  let name = '雷猴'

  function handleClick() {
    name = '鲨鱼辣椒'
  }
</script>

<div>Hello {name}</div>
<button on:click={handleClick}>改名</button>
```

此时点击按钮，页面上的“雷猴”就会变成“鲨鱼辣椒”



上面的代码其实和 Vue 有点像。

- 变量和方法都写在 `<script>` 标签里。
- 在 `HTML` 中使用 `{}` 可以绑定变量和方法。
- 通过 `on:click` 可以绑定点击事件。

只需写以上代码，Svelte 就会自动帮我们做数据响应的操作。一旦数据发生改变，视图也会自动改变。

是不是非常简单！

## 基础模板语法

Svelte 的模板语法其实和 Vue 是有点像的。如果你之前已经使用过 Vue，那本节学起来就非常简单。

### 插值

在“起步章节”已经使用过 **插值** 了。在 Svelte 中，使用 `{}` 大括号将 `script` 里的数据绑定到 `HTML` 中。

雷猴

雷猴

html 复制代码

```
<script>
  let name = '雷猴'
</script>

<div>{name}</div>
```

此时页面上就会出现 `name` 的值。

这种语法和 Vue 是有点像的，Vue 使用双大括号的方式 `{{}}` 绑定数据。Svelte 就少一对括号。

## 表达式

在 `HTML` 中除了可以绑定变量外，还可以绑定表达式。

雷猴 世界!  
1 + 2 = 3  
鲨鱼辣椒

雷猴 世界!

html 复制代码

```
<script>
  let name = '雷猴'

  function sayHi() {
    return `${name} 世界!`
  }

  let a = 1
  let b = 2

  let state = false
</script>

<div>{sayHi()}</div>

<div>{a} + {b} = {a + b}</div>
```

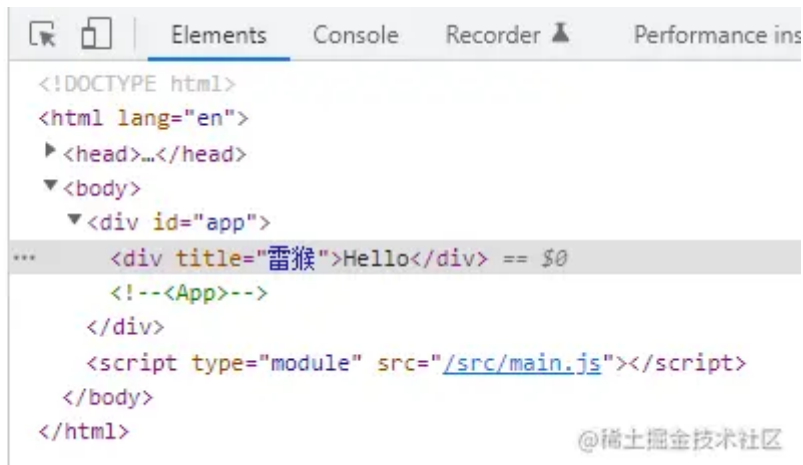
```
<div>{state ? '雷猴' : '鲨鱼辣椒'}</div>
```

## 属性绑定

HTML 的属性需要动态绑定数据时，也是使用 `{}` 语法。

Hello

雷猴



@稀土掘金技术社区

html 复制代码

```
<script>
```

```
  let name = '雷猴'
```

```
</script>
```

```
<div title={name}>Hello</div>
```

当鼠标放到 `div` 标签上时，会出现 `title` 里的提示信息。

## 渲染 HTML 标签 @html

如果只是使用插值的方法渲染带有 HTML 标签的内容，Svelte 会自动转义 `<`、`>` 之类的标签。

```
<h1 style="color: pink;">雷猴</h1>
```

@稀土掘金技术社区

html 复制代码

```
<script>
```

```
  let h1El = '<h1 style="color: pink;">雷猴</h1>'
```

```
</script>
```



```
<div>{h1El}</div>
```

这种情况多数出现在渲染富文本。

在 Vue 中有 `v-html` 方法，它可以将 `HTML` 标签渲染出来。在 Svelte 中也有这个方法，在插值前面使用 `@html` 标记一下即可。

雷猴

前端全栈技术社区

html 复制代码

```
<script>
  let h1El = '<h1 style="color: pink;">雷猴</h1>'
</script>

<div>{@html h1El}</div>
```

但此方法有可能遭受 `XSS` 攻击。

我在 [《NodeJS 防止xss攻击》](#) 中简单演示过 `XSS` 攻击，有兴趣的可以看看。

## 样式绑定

在日常开发中，给 `HTML` 标签设置样式主要通过 `行内 style` 和 `class` 属性。

基础的 `HTML` 写法和原生的一样，这里不过多讲解。

下面主要讲动态设置样式，也就是将 `JS` 里的变量或者表达式绑定到 `style` 或者 `class` 里。

### 行内样式 style

雷猴

雷猴

html 复制代码

```
<script>
  let color = 'red'

  setTimeout(() => {
    color = 'blue'
  }, 1000)
</script>

<div style="color: {color}">雷猴</div>
```

1秒后，文字从红色变成蓝色。

## 绑定 class

雷猴

雷猴

html 复制代码

```
<script>
  let foo = true

  setTimeout(() => {
    foo = false
  }, 1000)
</script>

<div class:active={foo}>雷猴</div>

<style>
  .active {
    color: red;
  }
</style>
```

在 HTML 里可以使用 `class:xxx` 动态设置要激活的类。这里的 `xxx` 是对应的类名。

语法是 `class:xxx={state}`，当 `state` 为 `true` 时，这个样式就会被激活使用。

## 条件渲染 #if

使用 `{#if}` 开头，`{/if}` 结尾。

### 基础条件判断

```
{#if 条件判断}
...
{/if}
```

复制代码

举个例子

雷猴

```
<script>
  let state = true

  setTimeout(() => {
    state = false
  }, 1000)
</script>

{#if state}
  <div>雷猴</div>
{/if}
```

html 复制代码

1秒后改变状态

## 两种条件

复制代码

```
{#if 条件判断}
...
{:else}
...
{/if}
```

举个例子

雷猴



html 复制代码

```
<script>
  let state = true

  setTimeout(() => {
    state = false
  }, 1000)
</script>

{#if state}
  <div>雷猴</div>
{:else}
  <div>鲨鱼辣椒</div>
{/if}
```

## 多种条件

复制代码

```
{#if 条件判断}
...
{:else if 条件判断}
...
{/if}
```

举个例子

雷猴

金明士 趙世瑜 尹 衡

html 复制代码

```
<script>
  let count = 1

  setInterval(() => {
    count++
  }, 1000)
</script>

{#if count === 1}
  <div>雷猴</div>
{:else if count === 2}
  <div>鲨鱼辣椒</div>
{:else}
  <div>蟑螂恶霸</div>
{/if}
```

条件渲染的用法比较简单，只要 JS 基础就能看得懂。

## 列表渲染 #each

如果你有一堆数据需要展示出来，可以使用 `#each` 方法。

使用 `{#each}` 开头, `{/each}` 结尾。

## 遍历数组

```
{#each expression as name}  
...  
{/each}
```

举个例子

- a
- b
- c
- d
- e
- f

[Svelte 遍历](#)

html 复制代码

```
<script>  
  let list = ['a', 'b', 'c', 'd', 'e', 'f']  
</script>  
  
<ul>  
  {#each list as item}  
    <li>{item}</li>  
  {/each}  
</ul>
```

要注意，Svelte 和 Vue 的遍历在写法上有点不同。

Vue的方式是：

html 复制代码

```
<div v-for="元素 in 源数据">  
  <span>{{元素}}</span>  
</div>
```

Svelte的方式是：

html 复制代码

```
<div>  
  {#each 源数据 as 元素}  
    <span>{元素}</span>  
  {/each}  
</div>
```

## 遍历数组（带下标）

- 0 -- a
- 1 -- b
- 2 -- c
- 3 -- d
- 4 -- e
- 5 -- f

js 遍历数组带下标

html 复制代码

```
<script>
  let list = ['a', 'b', 'c', 'd', 'e', 'f']
</script>

<ul>
  {#each list as item, index}
    <li>{index} -- {item}</li>
  {/each}
</ul>
```

注意：as 后面首先跟着元素，然后才是下标。而且元素和下标不需要用括号括起来。

## 如果元素是对象，可以解构

- 雷猴
- 鲨鱼辣椒

js 遍历数组带下标

html 复制代码

```
<script>
  let list = [
    {name: '雷猴'},
    {name: '鲨鱼辣椒'}
  ]
</script>

<ul>
  {#each list as {name}}
    <li>{name}</li>
```

```
    {/each}  
</ul>
```

## 默认内容

如果源数据没有内容，是空数组的情况下，还可以组合 `{:else}` 一起使用。

暂无数据

由本系统自动生成

html 复制代码

```
<script>  
  let list = []  
</script>  
  
<div>  
  {#each list as {name}}  
    <div>{name}</div>  
  {:else}  
    <div>暂无数据</div>  
  {/each}  
</div>
```

## 事件绑定 on:event

使用 `on:` 指令监听 DOM 事件，`on:` 后面跟随事件类型

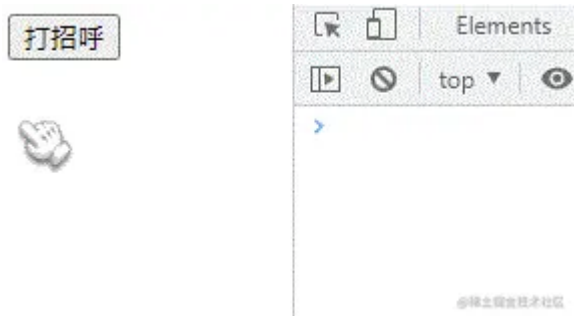
语法：

`on: 事件类型={事件名}`

csharp 复制代码

举个例子，点击按钮时在控制台输出“雷猴”。





html 复制代码

```
<script>
  function sayHi() {
    console.log('雷猴')
  }
</script>

<button on:click={sayHi}>打招呼</button>
```

绑定其他事件（比如change等）也是同样的道理。

## 事件修饰符

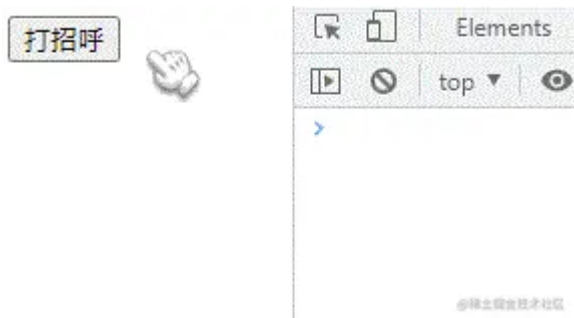
如果你只希望某些事件只执行一次，或者取消默认行为，或者阻止冒泡等，可以使用事件修饰符。

语法：

csharp 复制代码

**on:** 事件类型 | 修饰符={事件名}

举个例子，我希望点击事件只能执行一次，之后再点击都无效，可以使用官方提供的 **once** 修饰符。



html 复制代码

```
<script>
  function sayHi() {
    console.log('雷猴')
  }
</script>

<button on:click|once={sayHi}>打招呼</button>
```

从上图可以看出，多次点击都只是输出1次“雷猴”。

除了 `once` 之外，还有以下这些修饰符可以用：

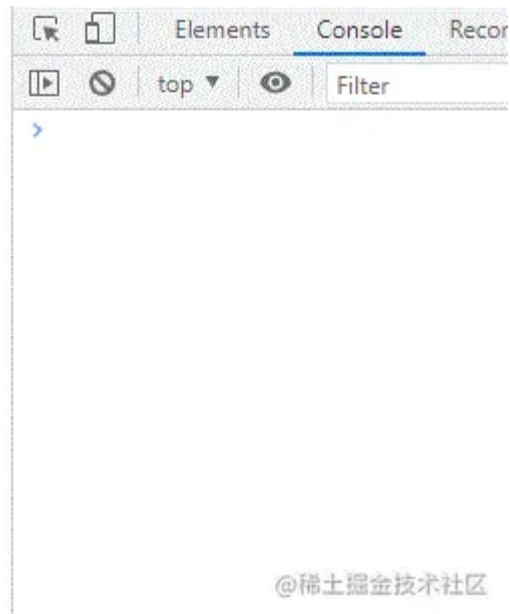
- `preventDefault`：禁止默认事件。在程序运行之前调用 `event.preventDefault()`
- `stopPropagation`：调用 `event.stopPropagation()`，防止事件到达下一个标签
- `passive`：改善了 touch/wheel 事件的滚动表现（Svelte会在合适的地方自动加上它）
- `capture`：表示在 *capture*阶段而不是*bubbling*触发其程序
- `once`：程序运行一次后删除自身

## 串联修饰符

修饰符还可以串联起来使用，比如 `on:click|once|capture={...}`

但需要注意，有些特殊的标签使用修饰符会出现“意想不到”的结果，比如 `<a>` 标签。

去学习Canvas ?



@稀土掘金技术社区

html 复制代码

```
<script>
  function toLearn() {
    console.log('还在思考要不要学Canvas')
  }
</script>

<a
  href="https://juejin.cn/post/7116784455561248775"
  on:click|once|preventDefault={toLearn}
>去学习Canvas ? </a>
```

本来是想给 `<a>` 标签绑定一个点击事件，第一次点击时在控制台输出一句话，并且禁止 `<a>` 标签的默认事件。

所以使用了 `once` 和 `preventDefault` 修饰符。

但实际上并非如此。上面的代码意思是 `once` 设定了只执行一次 `toLearn` 事件，并且只有一次 `preventDefault` 是有效的。

只有点击时就不触发 `toLearn` 了，而且 `preventDefault` 也会失效。所以再次点击时，`<a>` 元素就会触发自身的跳转功能。

## 数据绑定 bind

数据绑定通常会和表单元素结合使用。

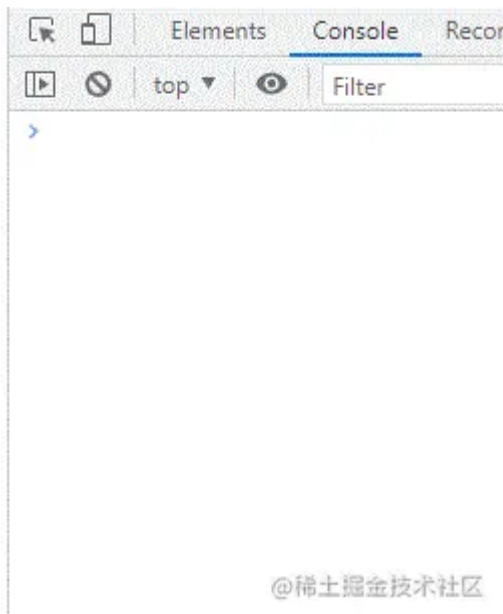
`bind` 可以做到双向数据绑定的效果。我觉得 Svelte 里的 `bind` 有点像 Vue 的 `v-model`。

语法：

ini 复制代码

```
bind:property={variable}
```

## | input 单行输入框



@稀土掘金技术社区

html 复制代码

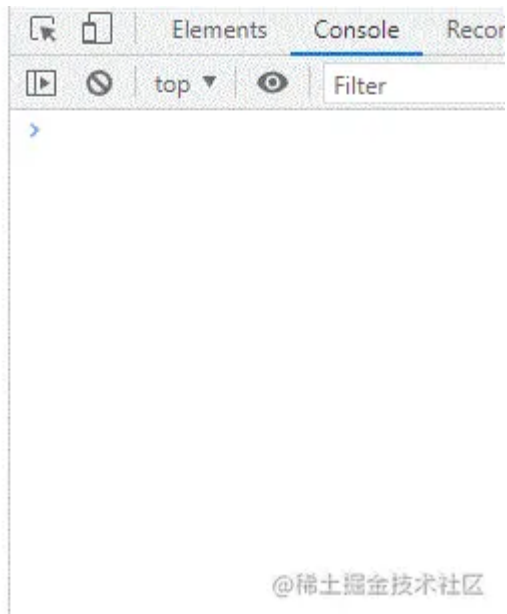
```
<script>
  let msg = 'hello'

  function print() {
    console.log(msg)
  }
</script>

<input type="text" value={msg} />
<button on:click={print}>打印</button>
```

如果只是使用 `value={msg}` 的写法，`input` 默认值是 `hello`，当输入框的值发生改变时，并没有把内容反应回 `msg` 变量里。

此时就需要使用 `bind` 了。



@稀土掘金技术社区

html 复制代码

```
<!-- 省略部分代码 -->
<input type="text" bind:value={msg} />
```

## | textarea 多行文本框

多行文本框同样绑定在 `value` 属性上。



hello

@稀土掘金技术社区

html 复制代码

```
<script>
  let msg = 'hello'
</script>

<textarea type="text" bind:value={msg} />
<p>{msg}</p>
```

## | input range 范围选择

因为都是 `input` 元素，只是 `type` 不同而已。所以范围选择元素同样需要绑定 `value` 。



©稀土掘金技术社区

html 复制代码

```
<script>
  let val = 3
</script>

<input type="range" bind:value={val} min=0 max=10 />
<p>{val}</p>
```

## | radio 单选

单选框通常是成组出现的，所以要绑定一个特殊的值 `bind:group={variable}`



©稀土掘金技术社区

```
<script>
  let selected = '2'
</script>

<input type="radio" bind:group={selected} value="1" />
<input type="radio" bind:group={selected} value="2" />
<input type="radio" bind:group={selected} value="3" />
<p>{selected}</p>
```

## checkbox 复选框



© 稀土掘金技术社区

```
<script>
  let roles = []
</script>

<input type="checkbox" bind:group={roles} value="雷猴" />
<input type="checkbox" bind:group={roles} value="鲨鱼辣椒" />
<input type="checkbox" bind:group={roles} value="蟑螂恶霸" />
<input type="checkbox" bind:group={roles} value="蝎子莱莱" />

<p>{roles}</p>
```

## select 选择器

a ▼ a



html 复制代码

```
<script>
  let selected = 'a'
</script>

<select bind:value={selected}>
  <option value='a'>a</option>
  <option value='b'>b</option>
  <option value='c'>c</option>
</select>

<span>{selected}</span>
```

## select multiple 选择器

multiple 和 checkbox 有点像。



html 复制代码

```
<script>
  let selected = []
```



```
</script>

<select multiple bind:value={selected}>
  <option value="雷猴">雷猴</option>
  <option value="鲨鱼辣椒">鲨鱼辣椒</option>
  <option value="蟑螂恶霸">蟑螂恶霸</option>
  <option value="蝎子莱莱">蝎子莱莱</option>
</select>

<span>{selected}</span>
```

## | 简写形式

如果 `bind` 绑定的属性和在 `JS` 里声明的变量名相同，那可以直接绑定

hello

hello



©稀土掘金技术社区

html 复制代码

```
<script>
  let value = 'hello'
</script>

<input type="text" bind:value />

<p>{value}</p>
```

这个例子中，`bind:value` 绑定的属性是 `value`，而在 `JS` 中声明的变量名也叫 `value`，此时就可以使用简写的方式。

## \$: 声明反应性

通过使用 `$:` [JS label 语法](#) 作为前缀。可以让任何位于 top-level 的语句（即不在块或函数内部）具有反应性。每当它们依赖的值发生更改时，它们都会在 component 更新之前立即运行。

上面这段解释是官方文档的解释。

`$:` 在文档中称为 `Reactivity`，中文文档成它为 `反应性能力`。

但我使用 `$:` 时，觉得这个功能有点像 Vue 的 `computed`。

`$:` 可以监听表达式内部的变化从而做出响应。

点击加1

0 翻倍后 0



html 复制代码

```
<script>
  let count = 0;
  $: doubled = count * 2;

  function handleClick() {
    count += 1;
  }
</script>

<button on:click={handleClick}>
  点击加1
</button>

<p>{count} 翻倍后 {doubled}</p>
```

使用 `$:` 声明的 `double` 会自动根据 `count` 的值改变而改变。

如果将以上代码中 `$:` 改成 `let` 或者 `var` 声明 `count` , 那么 `count` 将失去响应性。

这样看来, 真的和 Vue 的 `computed` 的作用有那么一点像。

## 异步渲染 `#await`

Svelte 提供异步渲染标签, 可以提升用户体验。

语法:

```
{#await expression}
...
{:then name}
...
{:catch name}
...
{/await}
```

[csharp 复制代码](#)

以 `#await` 开始, 以 `/await` 结束。

`:then` 代表成功结果, `:catch` 代表失败结果。

`expression` 是判断体, 要求返回一个 `Promise` 。

其实用法和 `#if ... :else if ... /if` 有那么一丢丢像。

举个例子

Loading...



html 复制代码

```
<script>
  const api = new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('请求成功, 数据是xxxxx')
    }, 1000)
  })
</script>

{#await api}
  <span>Loading...</span>
{:then response}
  <span>{response}</span>
{:catch error}
  <span>{error}</span>
{/await}
```

如果将上面的 `resolve` 改成 `reject` 就会走 `:catch` 分支。

## 基础组件

在 Svelte 中, 创建组件只需要创建一个 `.svelte` 为后缀的文件即可。

通过 `import` 引入子组件。

比如, 在 `src` 目录下有 `App.svelte` 和 `Phone.svelte` 两个组件。

App.svelte 是父级，想要引入 Phone.svelte 并在 HTML 中使用。

子组件 Phone 的内容：  
电话：13266668888

@稀土掘金技术社区

## App.svelte

html 复制代码

```
<script>  
  import Phone from './Phone.svelte'  
</script>  
  
<div>子组件 Phone 的内容:</div>  
<Phone />
```

## Phone.svelte

html 复制代码

```
<div>电话：13266668888</div>
```

# 组件通讯

组件通讯主要是 父子组件 之间的数据来往。

## | 父传子

比如上面的例子，手机号希望从 App.svelte 组件往 Phone.svelte 里传。

可以在 Phone.svelte 中声明一个变量，并公开该变量。

`App.svelte` 就可以使用对应的属性把值传入。

子组件 Phone 的内容：  
电话：88888888

© 稀土掘金技术社区

## App.svelte

html 复制代码

```
<script>
  import Phone from './Phone.svelte'
</script>

<div>子组件 Phone 的内容:</div>
<Phone number="88888888" />
```

## Phone.svelte

html 复制代码

```
<script>
  export let number = '13266668888'
</script>

<div>电话: {number}</div>
```

如果此时 `App.svelte` 组件没有传值进来，`Phone.svelte` 就会使用默认值。

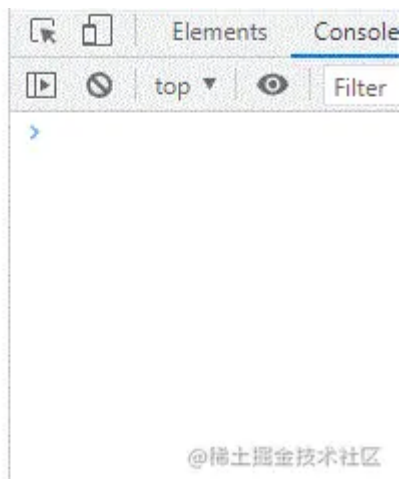
## 子传父

如果想在子组件中修改父组件的内容，需要把修改的方法定义在父组件中，并把该方法传给子组件调用。

同时需要在子组件中引入 `createEventDispatcher` 方法。

子组件 Phone 的内容:

输出手机号



## App.svelte

html 复制代码

```
<script>
  import Phone from './Phone.svelte'
  function print(data) {
    console.log(`手机号: ${data.detail}`)
  }
</script>

<div>子组件 Phone 的内容: </div>
<Phone on:printPhone={print} />
```

## Phone.svelte

html 复制代码

```
<script>
  import { createEventDispatcher } from 'svelte'
  const dispatch = createEventDispatcher()

  function printPhone() {
    dispatch('printPhone', '13288888888')
  }
</script>

<button on:click={printPhone}>输出手机号</button>
```

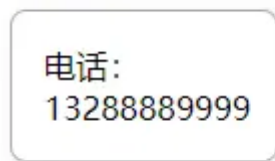
父组件接受参数是一个对象，子组件传过来的值都会放在 `detail` 属性里。

# 插槽 slot

和 Vue 一样，Svelte 也有组件插槽。

在子组件中使用 `<slot>` 标签，可以接收父组件传进来的 HTML 内容。

子组件 Phone 的内容：



@稀土掘金技术社区

## App.svelte

html 复制代码

```
<script>
  import Phone from './Phone.svelte'
</script>

<div>子组件 Phone 的内容:</div>
<Phone>
  <div>电话:</div>
  <div>13288889999</div>
</Phone>
```

## Phone.svelte

html 复制代码

```
<style>
  .box {
    width: 100px;
    border: 1px solid #aaa;
    border-radius: 8px;
    box-shadow: 2px 2px 8px rgba(0,0,0,0.1);
    padding: 1em;
    margin: 1em 0;
  }
</style>
```



```
<div class="box">
  <slot>默认值</slot>
</div>
```

## 生命周期

生命周期是指项目运行时，指定时期会自动执行的方法。

Svelte 中主要有以下几个生命周期：

- `onMount`：组件挂载时调用。
- `onDestroy`：组件销毁时执行。
- `beforeUpdate`：在数据更新前执行。
- `afterUpdate`：在数据更新完成后执行。
- `tick`：DOM元素更新完成后执行。

以上生命周期都是需要从 `svelte` 里引入的。

用 `onMount` 举个例子

# Hello world

@稀土掘金技术社区

html 复制代码

```
<script>
  import { onMount } from 'svelte'
```

```
let title = 'Hello world'

onMount(() => {
  console.log('onMount')
  setTimeout(() => title = '雷猴', 1000)
})
</script>

<h1>{title}</h1>
```

在组件加载完1秒后，改变 `title` 的值。

`onDestroy`、`beforeUpdate` 和 `afterUpdate` 都和 `onMount` 的用法差不多，只是执行的时间条件不同。你可以自己创建个项目试试看。

`tick` 是比较特殊的，`tick` 和 Vue 的 `nextTick` 差不多。

在 Svelte 中，`tick` 的使用语法如下：

js 复制代码

```
import { tick } from 'svelte'

await tick()
// 其他操作
```

## 总结

本文主要讲解了 Svelte 的基础用法，但 Svelte 的内容和 API 远不止此。它还有很多高级的用法以及提供了过渡动画功能等，这些都会放在高级篇讲解。

Svelte 是一个 Web 应用的构建工具，它打包出来的项目体积比较小，性能强，不使用虚拟DOM。

但 Svelte 的兼容性和周边生态相比起 Vue 和 React 会差一点。

所以日常项目中需要根据 Svelte 的优缺点进行取舍。