

<p data-nodeid="1785">你好，我是李思嘉，前端效率工程化这门课我们会讨论一个前端项目从开发到构建和部署这一系列 workflows 的效率问题。在开发效率篇里，我们会讨论一系列影响开发效率的流程和工具。工欲善其事必先利其器，第一课时，我们首先从开发一个新项目时最基础的准备工作讲起。</p>

<p data-nodeid="1786">当你准备开发一个新项目时，在进入到实际业务编码前，通常需要做很多的基础准备工作，这里会遇到的问题有：</p>

<ol data-nodeid="1787">

<li data-nodeid="1788">

<p data-nodeid="1789">要准备好一个项目的基础开发设施，需要投入大量时间和精力，这部分的工作量是以天为单位的。</p>

<li data-nodeid="1790">

<p data-nodeid="1791">一个完备的项目基础环境就像一个精密的仪器，只有各部分都充分协调后才能运转正常。要在较短时间内配置一个技术栈完整、辅助功能丰富、兼顾不同环境下构建优化目标的项目基础代码，通常需要开发人员在工程领域长久的知识储备与实践总结，而这对于经验相对较少的开发人员而言是一个不小的挑战。</p>

<li data-nodeid="1792">

<p data-nodeid="1793">不同的项目需求和团队情况，对应我们在使用基础设施时的选择可能也各不相同，因此我们并不能依靠一套固定不变的模板，而是需要根据不同的现状来使用不同的基础设施。这又增加了整体时间成本。</p>

<p data-nodeid="1794">而<strong data-nodeid="1942">脚手架工具，正是为了解决这些问题而诞生的。</p>

<ul data-nodeid="1795">

<li data-nodeid="1796">

<p data-nodeid="1797">利用脚手架工具，我们可以经过几个简单的选项<strong data-nodeid="1948">快速生成项目的基础代码。</p>

<li data-nodeid="1798">

<p data-nodeid="1799">使用脚手架工具生成的项目模板通常是经过经验丰富的开发者提炼和检验的，很大程度上代表某一类项目开发的<strong data-nodeid="1954">最佳实践，相较于让开发者自行配置提供了更优选择。</p>

<li data-nodeid="1800">

<p data-nodeid="1801">同时，脚手架工具也支持使用<strong data-nodeid="1960">自定义模板，我们也可以根据项目中的实际经验总结、定制一个脚手架模板。</p>

<p data-nodeid="1802">因此，对于一个熟练的前端工程师来说，要掌握的基本能力之一就是通过技术选型来确定所需要使用的<strong data-nodeid="1970">技术栈，然后根据技术栈选择合适的<strong data-nodeid="1971">脚手架工具，来做项目代码的初始化。一个合适的脚手架，可以为开发人员提供反复优化后的开发流程配置，高效地解决开发中涉及的流程问题，使得工程师能够快速上手，并提升整个开发流程的效率和体验。当然，前提是建立在选择对了脚手架工具并深入掌握其工作细节的基础上。</p>

<p data-nodeid="1803">那么下面我们先来谈谈脚手架工具究竟是什么。</p>

<h3 data-nodeid="1804">什么是脚手架</h3>

<p data-nodeid="1805">说到<strong data-nodeid="1979">脚手架（Scaffold）这个词，相信你并不陌生，它原本是建筑工程术语，指为了保证施工过程顺利而搭建的工作平台，它为工人們在各层施工提供了基础的功能保障。</p>

<p data-nodeid="1806"></p>

<p data-nodeid="1807">而在<strong data-nodeid="1992">软件开发领域，脚手架是指通过各种工具来生成项目基础代码的技术。通过脚手架工具生成后的代码，通常已包含了项目开发流程中所需的<strong data-nodeid="1993">工作目录内的通用基础设施，使开发者可以方便地将注意力集中到业务开发本身。</p>

<p data-nodeid="1808">那么对于日常的前端开发流程来说，项目内究竟有哪些部分属于通用基础设施呢？让我们从项目创建的流程说起。对于一个前端项目来说，一般在进入开发之前我们需要做的准备有：</p>

<ol data-nodeid="3194">

<li data-nodeid="3195">

<p data-nodeid="3196">首先我们需要有 <strong data-nodeid="3216">package.json，它是 npm 依赖管理体系下的基础配置文件。</p>

<li data-nodeid="3197">

<p data-nodeid="3198">然后<strong data-nodeid="3226">选择使用 npm 或 Yarn 作为包管理器，这会在项目里添加上对应的<strong data-nodeid="3227">lock 文件，来确保在不同环境下部署项目时的依赖稳定性。</p>

<li data-nodeid="3199">

<p data-nodeid="3200"><strong data-nodeid="3236">确定项目技术栈，团队习惯的技术框架是哪种？使用哪一种数据流模块？是否使用 TypeScript？使用哪种 CSS 预处理器？等等。在明确选择后安装相关依赖包并在 <strong data-nodeid="3237">src 目录中建立入口源码文件。</p>

<li data-nodeid="3201">

<p data-nodeid="3202"><strong data-nodeid="3246">选择构建工具，目前来说，构建工具的主流选择还是 webpack（除非项目已先锋性地考虑尝试 nobundle 方案），对应项目里就需要增加相关的 <strong data-nodeid="3247">webpack 配置文件，可以考虑针对开发/生产环境使用不同配置文件。</p>

<li data-nodeid="3203">

打通构建流程，通过安装与配置各种 **Loader**、插件和其他配置项，来确保开发和生产环境能正常构建代码和预览效果。

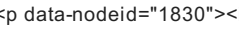
- 优化构建流程**，针对开发/生产环境的不同特点进行各自优化。例如，开发环境更关注构建效率和调试体验，而生产环境更关注访问性能等。
- 选择和调试辅助工具**，例如代码检查工具和单元测试工具，安装相应依赖并调试配置文件。
- 最后是**收尾工作**，检查各主要环节脚本是否工作正常，编写说明文档 README.md，将不需要纳入版本管理的文件目录记入 .gitignore 等。

正如下面简单的示例项目模板，经历了上面这些步骤后我们的项目目录下就新增了这些相关的文件：

```
package.json
package-lock.json
public/
src/
main.ts
router.ts
store/
webpack/
common.config.js
dev.config.js
prod.config.js
.browserslistrc
babel.config.js
tsconfig.json
postcss.config.js
.eslintrc
jest.config.js
.gitignore
README.md
```

而通过脚手架工具，我们就能免去人工处理上的环节，轻松地搭建起项目的初始环境，直接进入业务开发中。接下来我们就先来看一下前端领域的几个典型脚手架工具，了解这几个脚手架所代表的不同设计理念，接着我们会重点分析两个代表性脚手架工具包内的技术细节，以便在工作中更能得心应手地使用和优化。

三种代表性的前端脚手架工具

 <https://s0.lgstatic.com/i/image/M00/3F/B4/CgqCHI8xA46AOLMIAABL15AXwak581.png>

Yeoman

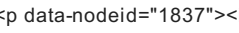
 <https://s0.lgstatic.com/i/image/M00/3F/A9/Ciqc1F8xA0KAKf0uAABJG0oh-Qs463.png>

[图：logo-yeoman]

Yeoman 是前端领域内较早出现的脚手架工具，它由 Google I/O 在 2012 年首次发布。Yeoman 提供了基于特定生成器（Generator）来创建项目基础代码的功能。时至今日，在它的网站中能找到超过 5600 个不同技术栈的代码生成器。

作为早期出现在前端领域的脚手架工具，它没有限定具体的开发技术栈，提供了足够的开放性和自由度，但也因此缺乏某一技术栈的深度集成和技术生态。随着前端技术栈的日趋复杂化，人们更倾向于选择那些以具体技术栈为根本的脚手架工具，而 Yeoman 则更多用于一些开发流程里特定片段代码的生成。

Create-React-App

 https://s0.lgstatic.com/i/image/M00/3F/B3/CgqCHI8xAqOAAmQFAAAIZny__YI029.png

[图：logo-create-react-app]

Create React App（后简称 CRA）是 Facebook 官方提供的 React 开发工具集。它包含了 create-react-app 和 react-scripts 两个基础包。其中 create-react-app 用于选择脚手架创建项目，而 react-scripts 则作为所创建项目中的运行时依赖包，提供了封装后的项目启动、编译、测试等基础工具。

正如官方网站中所说的，CRA 带来的最大的改变，是将一个项目开发运行时的各种配置细节完全封装在了一个 react-scripts 依赖包中，这大大降低了开发者，尤其是对 webpack 等构建工具不太熟悉的开发者上手开发项目的学习成本，也降低了开发者自行管理各配置依赖包的版本所需的额外测试成本。

但事情总有两面性，这种近乎黑盒的封装在初期带来便利的同时，也为后期的用户自定义优化带来了困难。虽然官方也提供了 eject 选项来将全部配置注入回项目，但大部分情况下，为了少量优化需求而放弃官方提供的各依赖包稳定升级的便利性，也仍不是一个好的选择。在这种矛盾之下，在保持原有特性的情况下提供自定义配置能力的工具 <https://github.com/timamey/react-app-rewired/>

data-nodeid="2085">react-rewired 和 customize-cra 应运而生。</p></div>

####

<h3 data-nodeid="1872">如何定制一个脚手架模板</h3>

<p data-nodeid="1873">虽然官方提供的默认脚手架模板已经代表了对应技术栈的通用最佳实践，但是在实际开发中，我们还是时常需要对通过这些脚手架创建的模板项目进行定制化，例如：</p>

<ol data-nodeid="1874">

<li data-nodeid="1875">

<p data-nodeid="1876">为项目引入新的通用特性。</p>

<li data-nodeid="1877">

<p data-nodeid="1878">针对构建环节的 webpack 配置优化，来提升开发环境的效率和生产环境的性能等。</p>

<li data-nodeid="1879">

<p data-nodeid="1880">定制符合团队内部规范的代码检测规则配置。</p>

<li data-nodeid="1881">

<p data-nodeid="1882">定制单元测试等辅助工具模块的配置项。</p>

<li data-nodeid="1883">

<p data-nodeid="1884">定制符合团队内部规范的目录结构与通用业务模块，例如业务组件库、辅助工具类、页面模板等。</p>

<p data-nodeid="1885">通过将这些实际项目开发中所需要做的定制化修改输出为标准的手脚手架模板，我们就能在团队内部孵化出更符合团队开发规范的开发流程。一方面最大程度减少大家在开发中处理重复事务的时间，另一方面也能减少因为开发风格不一导致的团队内项目维护成本的增加。接下来，我们就结合上面提到的三个脚手架工具来分别看下如何定制专属的手脚手架模板。</p>

<h4 data-nodeid="1886">使用 Yeoman 创建生成器</h4>

<p data-nodeid="21053">脚手架模板在 Yeoman 中对应的是生成器（Generator）。作为主打自由制作和分享脚手架生成器的开源工具，Yeoman 为制作生成器提供了丰富的 API 和 详细的文档。在这里，我们简单概述一下，一个基本的复制已有项目模板的生成器包含了：</p>

<ol data-nodeid="1888">

<li data-nodeid="1889">

<p data-nodeid="1890">生成器描述文件 <strong data-nodeid="2178">package.json，其中限定了 name、file、keywords 等对应字段的规范赋值。</p>

<li data-nodeid="1891">

<p data-nodeid="1892">作为主体的 <strong data-nodeid="2184">generators/app 目录，包含生成器的核心文件。该目录是执行 yo 命令时的默认查找目录，Yeoman 支持多目录的方式集成多个子生成器，篇幅原因我就不在这里展开了。</p>

<li data-nodeid="1893">

<p data-nodeid="1894"><strong data-nodeid="2189">app/index.js 是生成器的核心控制模块，其内容是导出一个继承自 yeoman-generator 的类，并由后者提供运行时上下文、用户交互、生成器组合等功能。</p>

<li data-nodeid="1895">

<p data-nodeid="1896"><strong data-nodeid="2194">app/templates 目录是我们需要复制到新项目中的脚手架模板目录。</p>

<p data-nodeid="1897">基本目录结构如下所示：</p>

```
<pre class="lang-java" data-nodeid="1898"><code data-language="java">generator-[name]/
<span class="hljs-keyword">package</span>.json
generators/
app/
templates/...
index.js
</code></pre>
```

<p data-nodeid="1899">其中 app/index.js 的核心逻辑如下：</p>

```
<pre class="lang-javascript" data-nodeid="1900"><code data-language="javascript"><span class="hljs-keyword">var</span> Generator =
<span class="hljs-built_in">require</span>(<span class="hljs-string">'yeoman-generator'</span>)
<span class="hljs-built_in">module</span>.exports = <span class="hljs-class"><span class="hljs-keyword">class</span></span> <span class="hljs-keyword">extends</span> <span class="hljs-title">Generator</span> </span>{
  writing() {
    <span class="hljs-built_in">this</span>.fs.copyTpl(
      <span class="hljs-built_in">this</span>.templatePath(<span class="hljs-string">'.'</span>),
      <span class="hljs-built_in">this</span>.destinationPath(<span class="hljs-string">'.'</span>))
  }

  install() {
    <span class="hljs-built_in">this</span>.npmInstall()
  }
}
</code></pre>
```

`writing` 和 `install` 是 Yeoman 运行时上下文两个阶段，在例子中，当我们执行下面的创建项目命令时，依次将生成器中模板目录内的所有文件复制到创建目录下，然后执行安装依赖。

在完成生成器的基本功能后，我们就可以通过在生成器目录里 `npm link`，将对应生成器包挂载到全局依赖下，然后进入待创建项目的目录中，执行 `yo` 创建命令即可。（如需远程安装，则需要先将生成器包发布到 `npm` 仓库中，支持发布到 `@scope/generator-[name]`。）



至此，制作 Yeoman 的生成器来定制项目模板的基本功能就完成了。除了基本的复制文件和安装依赖外，Yeoman 还提供了很多实用的功能，例如编写用户交互提示框或合成其他生成器等，可供开发者定制功能体验更完善的脚手架生成器。

为 create-react-app 创建自定义模板

为 `create-react-app` 准备的自定义模板在模式上较为简单。作为一个最简化的 `CRA` 模板，模板中包含如下必要文件：

- `README.md`：用于在 `npm` 仓库中显示的模板说明。
- `package.json`：用于描述模板本身的元信息（例如名称、运行脚本、依赖包名和版本等）。
- `template.json`：用于描述基于模板创建的项目中的 `package.json` 信息。
- `template` 目录：用于复制到创建后的项目中，其中 `gitignore` 在复制后重命名为 `.gitignore`，`public/index.html` 和 `src/index` 为运行 `react-scripts` 的必要文件。

具体目录结构如下所示：

```
class="lang-java" data-nodeid="1917"><code data-language="java">cra-template-[template-name]/
README.md (<span class="hljs-keyword">for</span> npm)
template.json
<span class="hljs-keyword">package</span>.json
template/
README.md (<span class="hljs-keyword">for</span> projects created from <span class="hljs-keyword">this</span> template)
gitignore
<span class="hljs-keyword">public</span>/
index.html
src/
index.js (or index.tsx)
</code></pre>

在使用时，同样还是需要将模板通过 npm link 命令映射到全局依赖中，或发布到 npm 仓库中，然后执行创建项目的命令。



```
class="lang-yaml" data-nodeid="1919"><code data-language="yaml">npx create-react-app [app-name] --template [template-name]
</code></pre>

为 Vue CLI 创建自定义模板

相比 CRA 模板而言，Vue 的模板中变化最大的当属增加了 meta.js/json 文件，用于描述创建过程中的用户交互信息以及用户选项对于模板文件的过滤等。


```
class="lang-java" data-nodeid="1922"><code data-language="java">[template-name]/
README.md (<span class="hljs-keyword">for</span> npm)
meta.js or meta.json
template/
</code></pre>

此外，Vue 的 template 目录中包含了复制到项目中的所有文件，并且在相关文件中还增加了 handlebars 条件判断的部分，根据 meta.js 中指定用户交互结果选项来将模板中带条件的文件转换为最终生成到项目中的产物。如下代码所示：



```
class="lang-dart" data-nodeid="1924"><code data-language="dart">template/package.json
...
"dependencies": {
 "vue": ^2.5.2 {{#router}},
 "vue-router": ^3.0.1 {{/router}}
},
...
meta.js
...
prompts: {
...

```


```


```


```

```
router: {
  when: <span class="hljs-string">'isNotTest'</span>,
  type: <span class="hljs-string">'confirm'</span>,
  message: <span class="hljs-string">'Install vue-router?'</span>,
},
...
}
```

</code></pre>

<p data-nodeid="1925">使用自定义模板创建项目的命令为: </p>

```
<pre class="lang-java" data-nodeid="1926"><code data-language="java">npm install -g <span class="hljs-meta">@vue</span>/cli-init
vue init [template-name] [app-name]
</code></pre>
```

<p data-nodeid="1927">这样就完成了脚手架的定制工作。有了定制化后的脚手架，我们就可以在之后的创建项目时直接进入业务逻辑的开发中，而不必重复地对官方提供的标准化模板进行二次优化。</p>

<p data-nodeid="1929">使用脚手架工具是提升开发效率的第一项内容。通过今天的学习，我们了解了脚手架的使用场景，了解了 3 个典型脚手架工具的特点，并分析了 React 和 Vue 官方提供的脚手架工具中的构建集成技术细节。最后，对于希望将业务中使用的更具定制化的基础代码转变为新的脚手架模板的同学，我们也了解了如何在不同工具环境下创建和使用自定义模板。</p> <p data-nodeid="1930"><strong data-nodeid="2226">课程最后，我想请你来回想一下：你在项目开发中使用的是哪一种脚手架工具和模板？使用的理由是？你可以将答案写在留言区与大家一起讨论。</p> <p data-nodeid="1931">下个课时我们将要学习的是一个大家一直在使用但是很少了解其中细节的技术点：热更新技术。</p>

精选评论

**英：

老师讲的很实在，很实用，希望能拿到更多offer

**荣：

我们公司部分dev机接口只能本机访问，这种情况dev-server 无法使用，前端cli有什么建议吗，旧方式是gulp watch 实时使用sftp上传到dev机进行测试。我准备做一套新的cli，放弃gulp

讲师回复：

1. 接口只能本机访问的原因是？如果因为服务启动使用的是127.0.0.1的话可以改成0.0.0.0试试，后者可以通过网络访问。
2.VS Code支持通过ssh访问远程开发机目录下的代码，代码在dev机器上启动devServer然后本地IDE通过ssh链接这样的工作流程应该也可以。

3. 构建工具的选择主要还是看开发模式是否符合以及相应工具的生态是否健全。cli是构建工具的一部分，单独开发整个工具生态的投入产出比不是很高，如果单纯是上面的网络问题建议考虑上面2点。如果是希望尝试新的构建工具建议选择其他生态比较成熟的工具。

vience:

豁然开朗呀 老师可以出本书 我预订了

编辑回复：

感谢同学支持老师

**潮：

跟着大佬学习

**靖：

老师，angular-cli不算主流吗？

讲师回复：

其实并没有一个主流非主流的绝对定义，只是`react`和`vue`的使用者人数上可能更多一些大家会更熟悉一些，所以文章篇幅限制的情况下就只介绍了这两种脚手架工具，让大家有一个对比。使用`angular`的同学也可以从相同的维度去对比`angular-cli`和其他脚手架工具的区别。

*悦：

思路很不错，一起加油 😊

*亮：

跟着大佬学习，讲得不错喔

*林：

感觉可以写一个脚手架工具而不只对已有脚手架的简单扩展

**东：

讲的真的很棒，收获很多，期待接下来的课程

*子：

课程内容很给力

**树：

超级棒的课程！！！知识点讲的详详细细，讲的通俗易懂

*红：

一起加油~