调研选型

开发需求	项目开发需要基于一些现有的技术以节省开发时间	代码风格需要统一，以便于减少维护成本	将代码编辑区文本转化为 blockly 块需要分析、解释代码	项目开发成功后需要打包上线
选型问题	开发的起点	代码风格检查工具	代码编译工具	前端打包工具

选型问题：开发的起点

可行方案：

一、`blockly`

- 开源协议：`Apache License 2.0`
- 创建时间：2013年10月
- `Blockly` 编辑器使用卡合的图形块来表示代码概念，如变量，逻辑表达式，循环等。它使得用户可以不必关注语法细节就能直接按照编程原则进行编程，可以支持 `python` 语言。

二、`blockpy`

- 开源协议：`Apache-2.0 license`
- 创建时间：2015年1月
- 基于 `blockly` 进行的二次开发项目，在可视化图形界面编程的基础上可以实现 `block` 块转为代码的逆向操作即代码转化为 `block` 方块的功能，同时可以进行数据分析等操作，但不支持 `python3` 以上的版本并且主要靠大学老师工作之余进行维护，因此在下载、使用方面不是很周到。

三、`scratch3.0`

- 开源协议：`Creative Commons Attribution-ShareAlike 2.0`
- 麻省理工学院的"终身幼儿园团队"开发的图形化编程工具，`Scratch 3.0` 放弃了 `Flash`，采用了 `HTML5` 来编写，同样通过积木块来进行程序编写，但是不支持 `python` 语言。

选择与理由：选择`blockly`。用于用户要求支持`python`语言，因此`scratch3.0`无法选择，而`blockpy`不支持`python3`以上的版本，同时他本身也是对`blockly`的二次开发，创作团队要对他做出修改成本过大，因此选择最基础的`blockly`作为可视化编程界面的基础，在此之上进行二次开发。

选型问题：代码风格审查工具

`HTML`风格审查工具:

可行方案:

一、`HTMLlint`

- 开源协议：`MIT License`
- 创建时间：2014年
- 它将自己定位为一个"*HTML5 linter and validator*"，它提供了较全面的 rule，曾有不少的人参与了开发，文档齐全，支持作为 `Grunt` 任务或 `Node.js` 模块的使用形式，可传入规则配置。不支持配置文件，支持行内注释配置。较好地同时支持了格式规则及语义规则，同时也支持自定义规则。使用 `HTMLparser2` 解析代码，性能较好。

二、`HTMLlcs`

- 开源协议：`MIT License`
- 创建时间：2014年
- `HTMLcs` 是百度 EFE 出品的的 `HTML` 代码风格检查工具。支持配置文件。另外也支持代码行内配置。提供了较丰富的规则实现，包括格式规则及语义规则，也支持添加自定义规则。支持传入额外 `linter` 对 `HTML` 中内嵌的 `JS` 及 `CSS` 内容进行检查。对于可自动修正的规则，`HTMLcs` 提供了 format 方法。自定义规则同样可以自定义对应的 format 行为。页面内嵌的 `JS` 及 `CSS` 内容也可以通过传入对应的 format 方法进行格式化。

三、`Bootlint`

- 开源协议：`The MIT license`
- 创建时间：2014年
- `Bootlint` 由 `Bootstrap` 团队开发。项目的完善度较高，文档齐全，使用方式包括在浏览器中引入，作为 `Grunt` 任务、`Nodejs` 模块及命令行工具。`Bootlint` 支持规则粒度的配置，但不支持配置文件或行内配置。

选择与理由：选择`HTMLlcs`，因为`Bootlint`不支持配置文件或行内配置，且提供的大多数规则都明显只适用于 `Bootstrap` 项目。而且，`Bootlint` 不支持添加自定义规则。`HTMLlint` 在大部分情况下能满足基本需求，但优点不明显。而 `HTMLcs` 在扩展性、自定义能力上表现突出，覆盖的规则也是这几个里面最全的，从我们团队需求角度看是最推荐的选择。

`CSS`风格审查工具

一、`CSSlint`:

- 开源协议：`The MIT license`
- `stylelint`拥有超过150条的规则，包括捕捉错误、最佳实践、控制可以使用的语言特性和强制代码风格规范。它支持最新的`CSS`语法，并且灵活可配置。

二、`stylelint`:

- 开源协议：`The MIT license`
- 在社区活跃度上，有非常多的第三方插件。
- 在Facebook，GitHub，WordPress 等公司得到实践，能够覆盖很多场景

选择与理由：选择`stylelint`，因为`CSSlint`社区活跃度没有`stylelint`高，且`stylelint`第三方插件更丰富，更符合团队需求。

选型问题：代码编译工具

> 主要是一些能将`Python`转为`JavaScript`的工具库，通过它们可以直接在浏览器上实时编译运行`Python`语法的代码。虽然本项目未要求实现代码在网页上运行出结果，但在开发过程中可以参考这些工具关于词法分析、抽象语法树生成等部分的内容，因此在浏览器中提供Python运行时环境的工具如`Batavia`、`PyPy.js`、`Pyodide`等不在考虑范围内。

可行方案：

一、`Skulpt`

- 开源协议：`The MIT license`
- 创建时间：2012年5月
- 编译时机：实时编译（just-in-time）
- 性能与规范是编译工具所追求的两个互斥的指标，若想要得到更加规范的代码，则需要通过增加代码量来达到目的，这样一来性能势必会有所下降，反之亦然。Skulpt在这一处权衡上倾向于合规性，因此在性能上也许不如其他编译工具。
- 开发者社区成熟稳定，而且有良好的用户体验。如`Skulpt`在编译后的`JavaScript`中维护源代码位置，而其他实现可能不会或可能使用外部源映射，并以不同的方式评估代码。`Skulpt`朝着友好的方向优化，有利于教学和教育。
- 支持`python`版本达到3.7.3

二、`Brython`

- 开源协议：`BSD-3-Clause license`
- 创建时间：2014年9月
- 编译时机：页面加载时（on page load）
- 编译速度快。最好将其视为“浏览器的`Python`实现”，因为它不会生成独立的`JavaScript`文件。
- 包含大量`Python`标准库，其中有大量用于`Brython`自己处理Web相关的DOM事件。
- `Brython`通过使用HXR请求获取单独的`.js`文件来处理`python`标准库导入的问题，其中的部分库已经编译在`brython.js`中，若用户对标准库的额外需求增多，则意味着运行速度会降低。
- 更接近真正的`Python`，而且很容易上手并测试代码。
- 目前对`Python`的版本支持达到3.10.6（但是这个版本标签似乎只是一个噱头，实际上它并没有实现`python3.10`及之前的所有特性）

三、`Transcrypt`

- 开源协议：`Apache-2.0 license`
- 创建时间：2015年12月

- 编译时机：提前编译（ahead-of-time）
- `Transcrypt`更像是一个传统的编译器，你可以自主选择编译输出的标准（如ES5、ES6），或者要求它输出`sourcemaps`，等等。
- `Transcrypt`得到的`JavaScript`输出非常简洁，相比于`Brython`，其编译输出得到的js文件更加轻量。
- 在一些细节上的处理并不完备，如不支持对字典的单变量迭代（`for i in dict`），不支持将JSON格式的文件直接赋值给字典类型。等等。
- 支持`python`版本到3.9 (?)

选择与理由：选择`Skulpt`，因为上述三个编译器都未支持到最新`python`版本，而`Skulpt`内置友好的错误消息、调试模式，并专注于可视化编程模块的Web兼容性，与我们要开发的项目存在一定程度的重叠。

选型问题：前端打包工具

可行方案：

一、`Vite`：

- 开源协议：`MIT License`
- 创建时间：2020年4月
- `Vite` 通过在一开始将应用中的模块区分为**依赖**和**源码**两类，改进了开发服务器启动时间。其中：
  - 依赖的部分使用`esbuild`预构建，而`esbuild`是用`Go`编写的，比`JS`编写的打包器快很多；
  - 源码的部分以 原生ESM方式提供。这让浏览器接管了打包程序的部分工作：`Vite` 只需要在浏览器请求源码时进行转换并按需提供源码。根据情景动态导入代码，即只在当前屏幕上实际使用时才会被处理。
- 打包快。只启动一台静态页面的服务器，不会打包全部项目文件代码，服务器根据客户端的请求加载不同的模块处理，实现按需加载。
- 可进行热模块更新（HMR）。对于热更新问题，`Vite`采用**立即编译当前修改文件**的办法，同时结合`Vite`的缓存机制，加载更新后的文件内容。

二、`Webpack`：

- 开源协议：`MIT License`
- 创建时间：2012年3月
- 模块化。把项目当做一个整体，通过一个给定的主文件（如：`index.js`），`Webpack`将从这个文件开始找到项目的所有依赖文件，使用`loaders`处理它们，最后打包为一个浏览器可识别的`JavaScript`文件。
- 需要一個配置文件来指定入口、输出、加载器、插件、转换等，没有用于`import/export`的节点，且配置中不支持相对路径，这样可能会使得配置变得复杂，但它为第三方导入、图像、`CSS`预处理器等提供了广泛的支持，适用于构建一个复杂的前端应用程序。
- 通过各种`loader`插件实现不同类型非`JavaScript`模块资源的引入与打包，功能十分丰富。
- 在代码拆分（通过延迟加载以提高性能）方面表现优秀，提供三种实现方式：手动实现通过添加`entry`配置信息设置；使用`CommonsChunkPlugin`插件删除冗余并分割代码；在模块中使用内联函数实现动态导入

- 可以使用`webpack-dev-server`这个工具来实现对`webpack`自己的服务器的操作，通过它来支持HMR，可进行热模块更新，使用时稳定、安全。
- 技术成熟，随着前端的进步发展推陈出新，现在公认较为优秀的就是`webpack`。

三、`rollup`：

- 开源协议：`MIT License`
- 创建时间：2015年5月
- 在浏览器中支持使用`node_modules`。
- 压缩文件代码使文件大小尽可能最小化，静态分析代码中的`import`，并将排除任何未实际使用的代码（Tree-shaking）。
- 主要用于打包ES模块，对于`CommonJS`模块的打包不推荐使用此工具。
- ES 模块打包，不必配置，直接使用，操作简单。

选择与理由：选择`webpack`。随着版本的更新，`webpack`已经拥有了与`Vite`、`rollup`类似的热替换（HMR）与Tree-sharking等功能，且`webpack`已经开发了较长的时间，技术相对成熟，选择他不但能为我们提供较为全面的功能支持，且当我们遇到问题时也能找到更多的解决方案。

参考资料

1. Comparing the speed of CPython, Brython, Skulpt and pypy.js
2. Python in Browser: How to choose between Brython, PyPy.js, Skulpt and Transcrypt?
3. Join efforts with brython and pyjs #246 (Skulpt issue)
4. Blockly官方文档
5. BlockPy官方网站
6. Is Vite Better than Webpack?
7. webpack官方文档
8. Rollup vs. Parcel vs. webpack: Which Is the Best Bundler?