

谷歌高职华东区域联盟 Blockly 培训

Blockly 二次开发培训手册

陈锐

chenrui@niit.edu.cn

南京工业职业技术大学计算机与软件学院

2020 年 8 月 12 日

目录

第 1 节 概述	5
第 2 节 开发工具	6
第 3 节 Blockly 源代码	7
第 4 节 HTML 与 Blockly 集成	8
4.1 插入固定尺寸的 Blockly 工作区	8
4.1.1 依赖 JS 文件	8
4.1.2 插入步骤	8
4.2 插入可变尺寸的工作区	11
4.2.1 代码修改	12
4.2.2 新增代码	12
第 5 节 Blockly 工具箱 (toolbox) 的配置	16
第 6 节 Blockly 开发者工具的使用	18
6.1 配置工具箱	19
6.2 配置工作区	20
6.3 导出工作区	21
6.4 Web 前端集成	22
6.5 完整代码	23
第 7 节 Blockly 的程序执行	25
7.1 增加执行按钮	25
7.2 JS 执行函数	25
7.3 在 Web 页面中实时显示 JS 代码	26
7.4 在 Web 页面中实时显示 Python 代码	27
第 8 节 自定义 Block 与代码生成	29
8.1 Input	30
8.1.1 数值输入示例	30
8.1.2 状态声明输入	32
8.2 Block Definition	33
8.3 Generator Stub	34
第 9 节 开发实例 (Python-Turtle 集成)	35
9.1 导入 turtle 库的 block	36

9.1.1	block 设计	36
9.1.2	导出设计	36
9.1.3	代码生成	37
9.2	color() 函数的 block	38
9.2.1	block 设计	38
9.2.2	导出设计	39
9.2.3	代码生成	39
9.3	shape() 函数的 Block	40
9.3.1	block 设计	40
9.3.2	导出设计	41
9.3.3	代码生成	41
9.4	forward() 函数的 block	42
9.4.1	block 设计	42
9.4.2	导出设计	43
9.4.3	代码生成	43
9.5	right() 函数的 block	44
9.5.1	block 设计	44
9.5.2	导出设计	45
9.5.3	代码生成	45
9.6	Blockly 集成	46
9.6.1	引入 python 浏览器执行环境	46
9.6.2	在 <head></head> 标签内引入自定义的 block	47
9.6.3	在 <body></body> 标签内引入画布	47
9.7	最终效果	48
第 10 节	扩展：保存与加载	49
10.1	保存当前的 Blockly 程序	49
10.2	加载 Blockly 程序	50
第 11 节	扩展：字典数据结构	51
11.1	创建 Key/Value Block	51
11.1.1	Block 设计	51
11.1.2	导出设计	52
11.1.3	代码生成	52
11.2	创建字典 Block	53
11.2.1	Block 设计	53
11.2.2	导出设计	56

11.2.3 修改 Block	56
11.2.4 代码生成	60
11.2.5 HTML 页面集成	61
11.2.6 最终效果	61
第 12 节 扩展：开发实例 (Echart 集成)	62
12.1 Echart 实例	62
12.2 Block 设计	65
12.3 导出设计	74
12.4 代码生成	74
12.5 HTML 集成	77
12.6 最终效果	78

第 1 节 概述

本次培训的主要内容罗列如下:

- HTML 与 Blockly 集成
- Blockly 开发者工具的使用
- 开发实例

南京工业职业技术大学 - 陈锐

第 2 节 开发工具

Blockly 是基于 JS 的代码库, 任意文本编辑器都可以进行开发. 为了提升开发效率, 推荐使用 Visual Studio Code.

下载地址: <https://vscode.cdn.azure.cn/stable/91899dcef7b8110878ea59626991a18c8a6a1b3e/VSCoDeUserSetup-x64-1.47.3.exe>

VSCode 支持插件安装, 推荐安装以下插件:

- 代码美化插件: Beautify
- JS 代码自动补全: JavaScript (ES6) code snippets
- 本地 HTTP Server: Live Server
- 中文界面插件: Chinese (Simplified) Language Pack for Visual Studio Code

第 3 节 Blockly 源代码

下载地址: <https://codeload.github.com/google/blockly/zip/master>

此次培训主要用到以下几个文件:

- blockly_compressed.js
- blocks_compressed.js
- javascript_compressed.js
- python_compressed.js
- msg/js/en.js

第 4 节 HTML 与 Blockly 集成

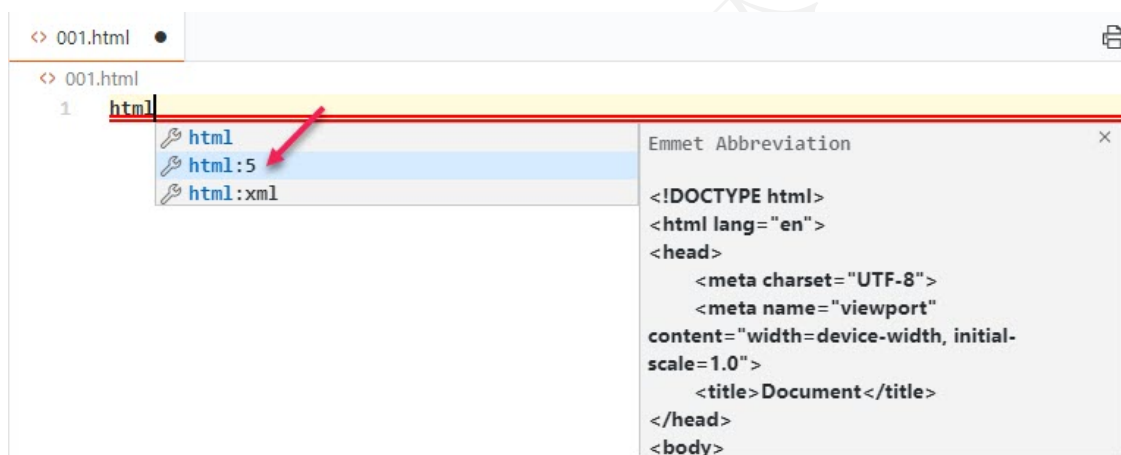
4.1 插入固定尺寸的 Blockly 工作区

4.1.1 依赖 JS 文件

- blockly_compressed.js
- blocks_compressed.js
- en.js

4.1.2 插入步骤

创建一个 HTML 文件 在 VSCode 中输入 `html`, 自动显示出如下图所示信息, 选取 `html:5` 自动生成一个 html 模板。



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
</html>
```


修改 html 文件 先将 blockly_compressed.js, blocks_compressed.js, en.js 拷贝到 html 文件的根目录

在 html 文件的 <head></head> 标签内插入以下代码, 引入上述罗列的 js 文件

```
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Document</title>

  <script src="js/blockly_compressed.js"></script>
  <script src="js/blocks_compressed.js"></script>
  <script src="js/en.js"></script>
</head>
```

在 html 文件的 <body></body> 标签内插入以下代码, 引入 blockly 工作区

- 设置 blockly 工作区尺寸

```
<div id="blocklyDiv" style="width: 500px; height: 400px; "></div>
```

- 设置 blockly 工作区内的工具箱, 以及工具箱支持的 block

```
<xml id="toolbox" style="display: none;">
  <block type="controls_if"></block>
  <block type="controls_repeat_ext"></block>
  <block type="logic_compare"></block>
  <block type="math_number"></block>
  <block type="math_arithmetic"></block>
  <block type="text"></block>
  <block type="text_print"></block>
</xml>
```

- 在 html 页面内插入 blockly 工作区

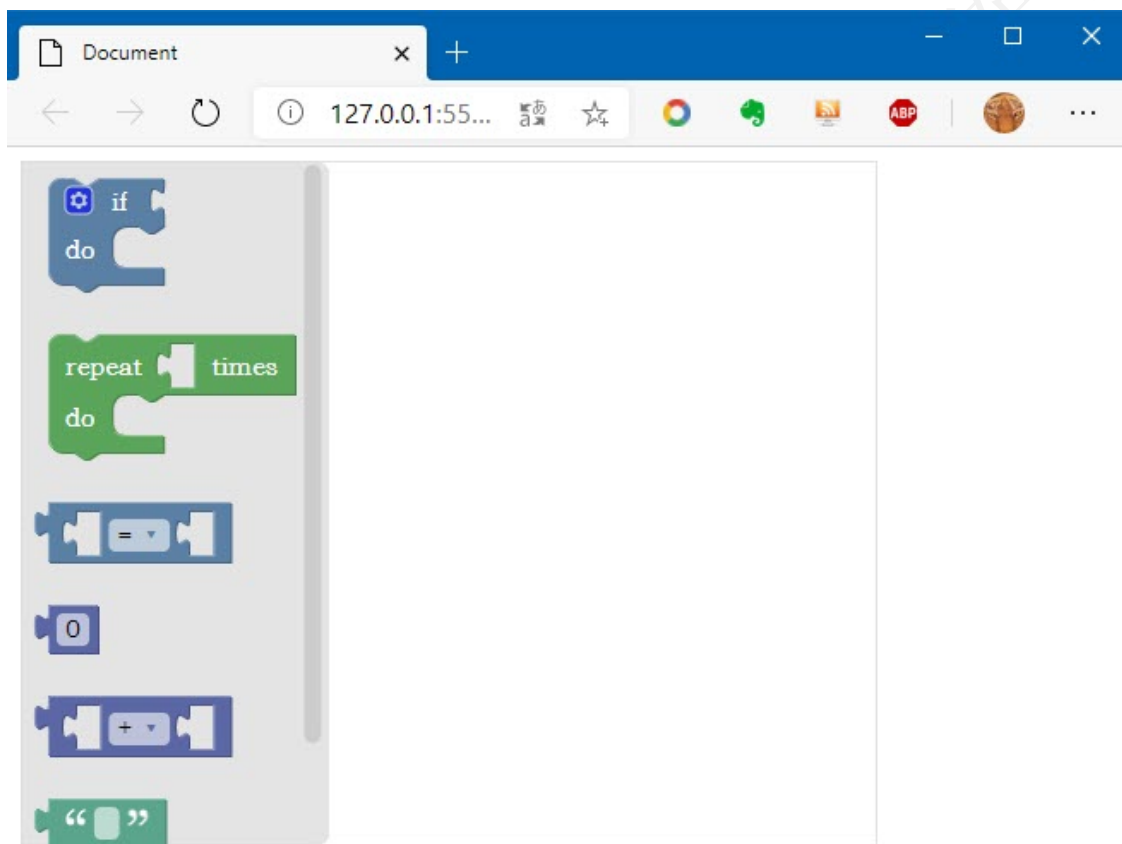
```
<script>
  var id_blockdiv = document.getElementById("blocklyDiv");
  var workspace = Blockly.inject(id_blockdiv, {
    toolbox: document.getElementById("toolbox"),
  });
</script>
```

以下两处需要与前文的保持一致:

- 标签 `xml` 的 `id` 与 `document.getElementById("toolbox")` 括号内的 `id`;
- 标签 `div` 中的 `id` 与 `Blockly.inject("blocklyDiv"` 双引号中的 `id`.

至此, 一个最简单的 Blockly 工作区搭建完毕.

浏览器打开 html 文件, 检查运行效果 浏览器打开 html 文件之后, 即可看到如下图所示的界面, 点击鼠标拖拽 block, 即可实现编程, 只不过目前支持的 block 较少.



完整代码

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>Document</title>
<script src="js/blockly_compressed.js"></script>
<script src="js/blocks_compressed.js"></script>
<script src="js/en.js"></script>
</head>

<body>
  <div id="blocklyDiv" style="height: 400px; width: 500px;"></div>

  <xml id="toolbox" style="display: none;">
    <block type="controls_if"></block>
    <block type="controls_repeat_ext"></block>
    <block type="logic_compare"></block>
    <block type="math_number"></block>
    <block type="math_arithmetic"></block>
    <block type="text"></block>
    <block type="text_print"></block>
  </xml>

  <script>
    var id_blockdiv = document.getElementById("blocklyDiv");
    var workspace = Blockly.inject(id_blockdiv, {
      toolbox: document.getElementById("toolbox"),
    });
  </script>
</body>
</html>
```

4.2 插入可变尺寸的工作区

上一节的 Blockly 工作区尺寸固定, 无法随着浏览器界面大小的调整而自动调整, 本节将在上一节代码的基础上, 增加 JS 函数以实现可变尺寸的工作区。

4.2.1 代码修改

- 修改 div 便签的样式, 并在其外再包一层 div 标签

修改之后的代码, 如下所示, 请记住两个 div 各自的 id.

```
<div id="blocklyarea">
  <div id="blocklyDiv" style="position: absolute;"></div>
</div>
```

4.2.2 新增代码

增加 CSS 样式 在 <head></head> 标签内, 增加 CSS 样式, 将 Blockly 的尺寸设置成为页面尺寸的百分比

```
<style>
  html, body {
    height: 100%;
  }

  #blockarea {
    height: 90%;
    width: 90%;
    margin: auto;
  }
</style>
```

增加缩放函数, 以计算 blockly 工作区准确尺寸

- 定义一个缩放函数 onresize

```
var id_blockarea = document.getElementById("blockarea");
var onresize = function (e) {
  // Compute the absolute coordinates and dimensions of blocklyArea.
  var element = id_blockarea;
  var x = 0;
  var y = 0;
  do {
    x += element.offsetLeft;
```

```

    y += element.offsetTop;
    element = element.offsetParent;
  } while (element);
  // Position blocklyDiv over blocklyArea.
  id_blockdiv.style.left = x + 'px';
  id_blockdiv.style.top = y + 'px';
  id_blockdiv.style.width = id_blockarea.offsetWidth + 'px';
  id_blockdiv.style.height = id_blockarea.offsetHeight + 'px';
  Blockly.svgResize(workspace);
};

```

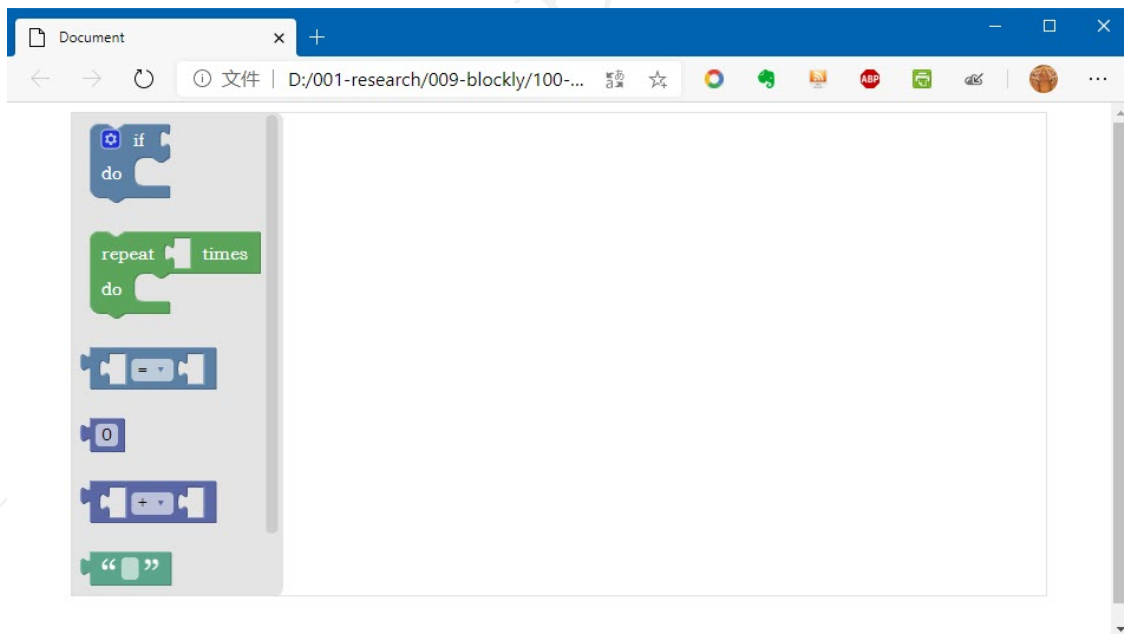
- 增加事件监听

当浏览器页面变化时, 自动运行 `onresize` 函数, 计算 blockly 工作区的尺寸, 然后重新绘制 blockly 工作区.

```

window.addEventListener('resize', onresize, false);
onresize();
Blockly.svgResize(workspace);

```



浏览器打开 html 文件, 检查运行效果

完整代码

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Document</title>
  <script src="js/blockly_compressed.js"></script>
  <script src="js/blocks_compressed.js"></script>
  <script src="js/en.js"></script>

  <style>
    html, body {
      height: 100%;
    }

    #blockarea {
      height: 90%;
      width: 90%;
      margin: auto;
    }
  </style>
</head>

<body>
  <div id="blockarea">
    <div id="blocklyDiv" style="position: absolute;"></div>
  </div>

  <xml id="toolbox" style="display: none;">
    <block type="controls_if"></block>
    <block type="controls_repeat_ext"></block>
    <block type="logic_compare"></block>
    <block type="math_number"></block>
    <block type="math_arithmetic"></block>
    <block type="text"></block>
    <block type="text_print"></block>
```

```
</xml>

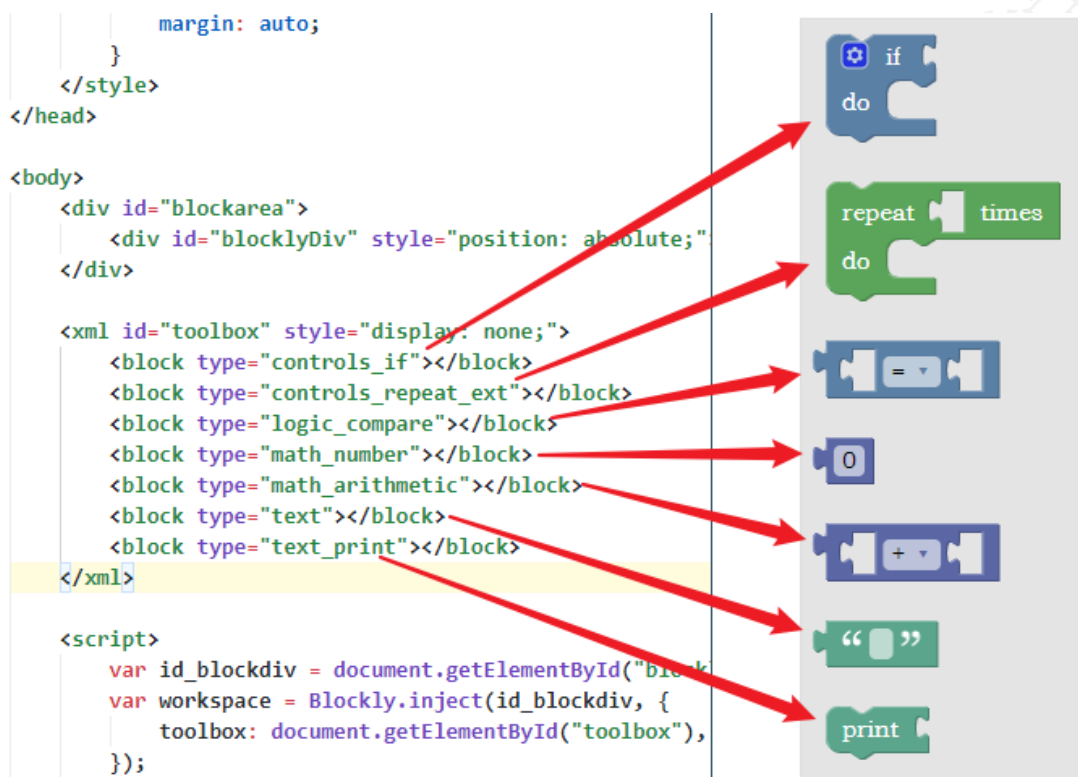
<script>
    var id_blockdiv = document.getElementById("blocklyDiv");
    var workspace = Blockly.inject(id_blockdiv, {
        toolbox: document.getElementById("toolbox"),
    });

    var id_blockarea = document.getElementById("blockarea");
    var onresize = function (e) {
        // Compute the absolute coordinates and dimensions of blocklyArea.
        var element = id_blockarea;
        var x = 0;
        var y = 0;
        do {
            x += element.offsetLeft;
            y += element.offsetTop;
            element = element.offsetParent;
        } while (element);
        // Position blocklyDiv over blocklyArea.
        id_blockdiv.style.left = x + 'px';
        id_blockdiv.style.top = y + 'px';
        id_blockdiv.style.width = id_blockarea.offsetWidth + 'px';
        id_blockdiv.style.height = id_blockarea.offsetHeight + 'px';
        Blockly.svgResize(workspace);
    };
    window.addEventListener('resize', onresize, false);
    onresize();
    Blockly.svgResize(workspace);
</script>
</body>
</html>
```

第 5 节 Blockly 工具箱 (toolbox) 的配置

前两节展示的工具箱包含的 block 很少, 并且没有按类别管理, 比较杂乱. 这一节将讲解 toolbox 的配置方式.

如下图所示, 工具箱就是在 `<xml></xml>` 标签中定义的. 每一行都有一个 `<block></block>` 的标签, 每个标签对应定义一个 block, 比如打印函数.



若想对各个 block 进行分类管理, 则可以在 `<xml></xml>` 标签中添加 `<category></category>` 标签, 每一对都可作为一类, 比如文本处理类, 循环类, 逻辑类.

如下图所示, 以 `<category></category>` 标签将 block 分为两类, 分别命名为“类别 1”和“类别 2”, 并为每种类别分配了颜色. 注意颜色需以 6 个十六进制表示, 并且属性关键词 `colour` 不能携程 `color`. 更多的颜色可以访问 <https://www.color-hex.com/popular-colors.php>



注意: 标签 `<category></category>` 是可以嵌套的. 如下图所示, 嵌套之后类别 1 中包含了一个子类别, 点击类别 1 时, 其包含的子类别才会显示出来.



第 6 节 Blockly 开发者工具的使用

手动编写工具箱费时费力。Blockly 开发者工具 (Blockly Developer Tool) 可以以图形化的方式配置工具箱。

Blockly 开发者工具位于 `demos/blockfactory` 文件夹下。可将其拷贝到另外一个文件夹下 (也可以不拷贝, 直接使用), 并将依赖文件拷贝到同一个文件夹下。依赖文件包括:

- `blockly_compressed.js`
- `blockly_compressed.js.map`
- `blocks_compressed.js`
- `blocks_compressed.js.map`
- `core`(此为文件夹)
- `en.js`(位于 `msg/js` 文件夹)
- `javascript_compressed.js`
- `javascript_compressed.js.map`
- `run_prettify.js`(需 要 从 https://cdn.rawgit.com/google/code-prettify/master/loader/run_prettify.js 下载)
- `storage.js`(位于 `appengine` 文件夹下)

将上述文件拷贝到 `js` 文件夹下, 同时将 `core/` 文件夹拷贝到 `js` 文件夹下

将文件夹 `media` 拷贝过来, 然后将下列 4 个文件中的 `../../media/` 修改为 `media/`

5 个结果 - 4 文件

`app_controller.js:`

```
691         disable: false,  
692         media: '../../media/');  
693
```

`block_exporter_tools.js:`

```
33         {collapse: false,  
34         media: '../../media/'});  
35     };
```

`factory.js:`

```
168         {rtl: rtl,  
169         media: '../../media/',
```

```

170         scrollbars: true});

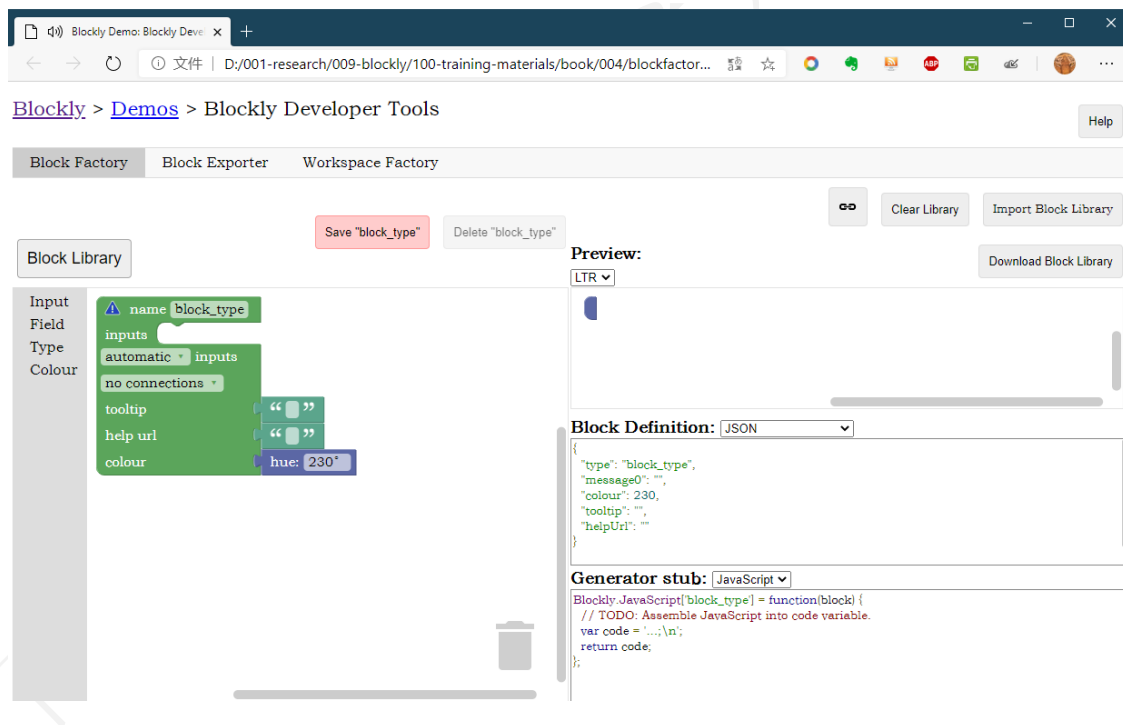
workspacefactory\wfactory_controller.js:
40         snap: true},
41         media: '../..../media/',
42         toolbox: this.toolbox

51         snap: true},
52         media: '../..../media/',
53         toolbox: '<xml xmlns="https://developers.google.com/blockly/xml"></xml>',

```

正常打开的 Blockly 开发者工具如下图所示, 其包含 3 部分:

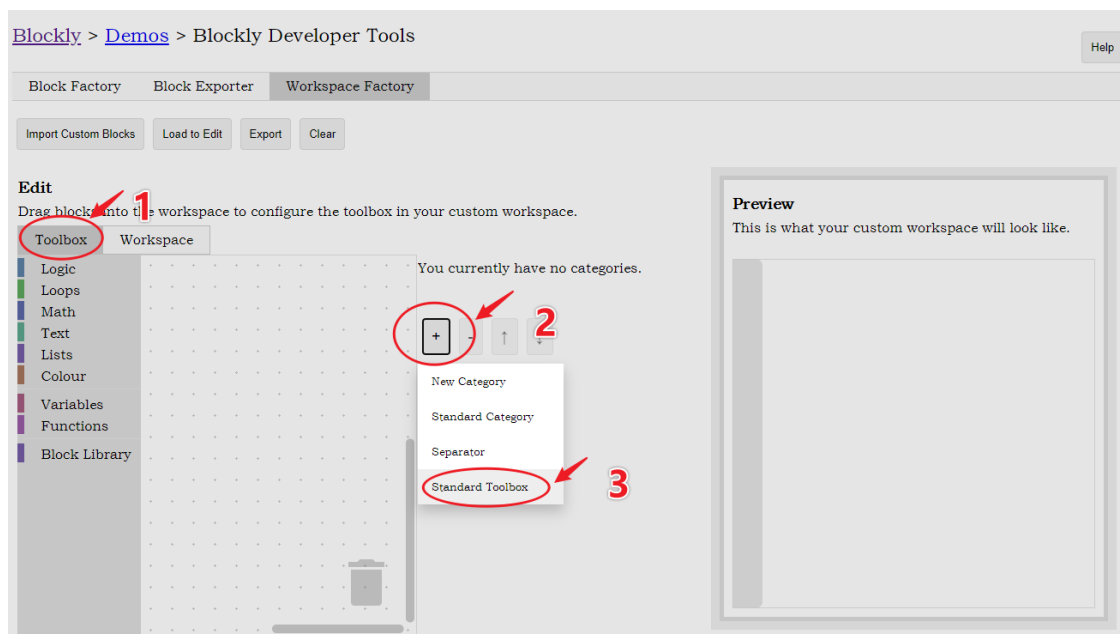
- Block Factory: 用于设计和定制 block
- Block Exporter: 用于到处定制 block
- Workspace Factory: 用于配置工作区和工具箱



本节主要使用 Workspace Factory, 配置工具箱和工作区。

6.1 配置工具箱

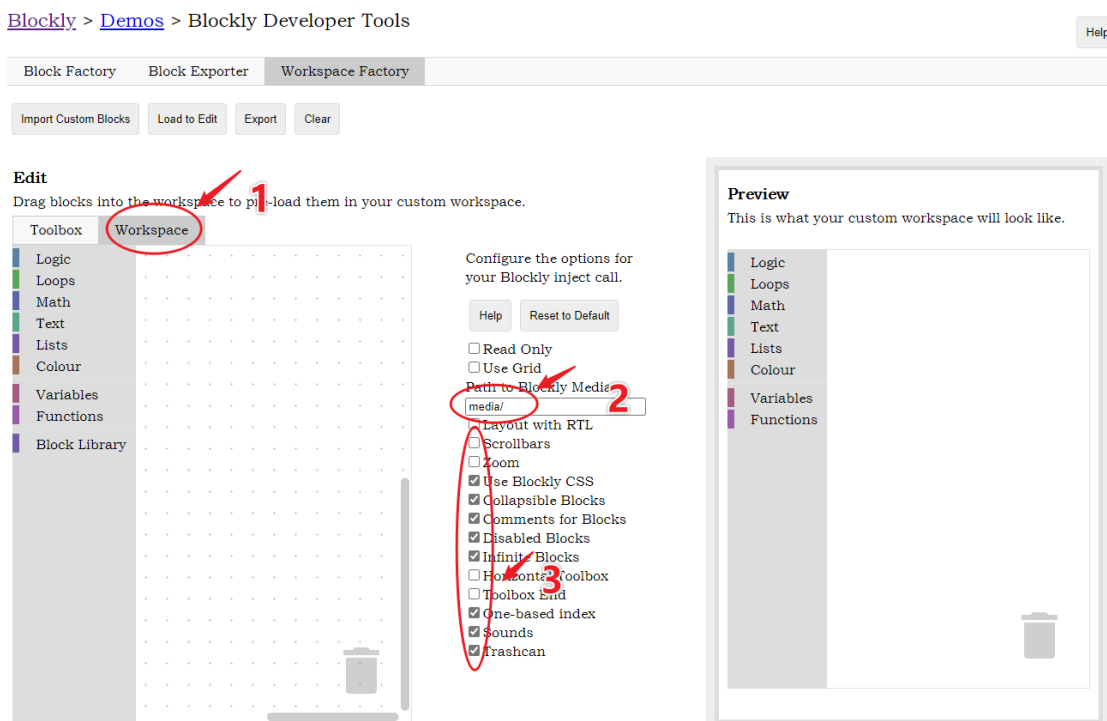
切换到 Workspace Factory, 按照下图所示顺序依次点击即可将标准的工具箱配置到工作区



6.2 配置工作区

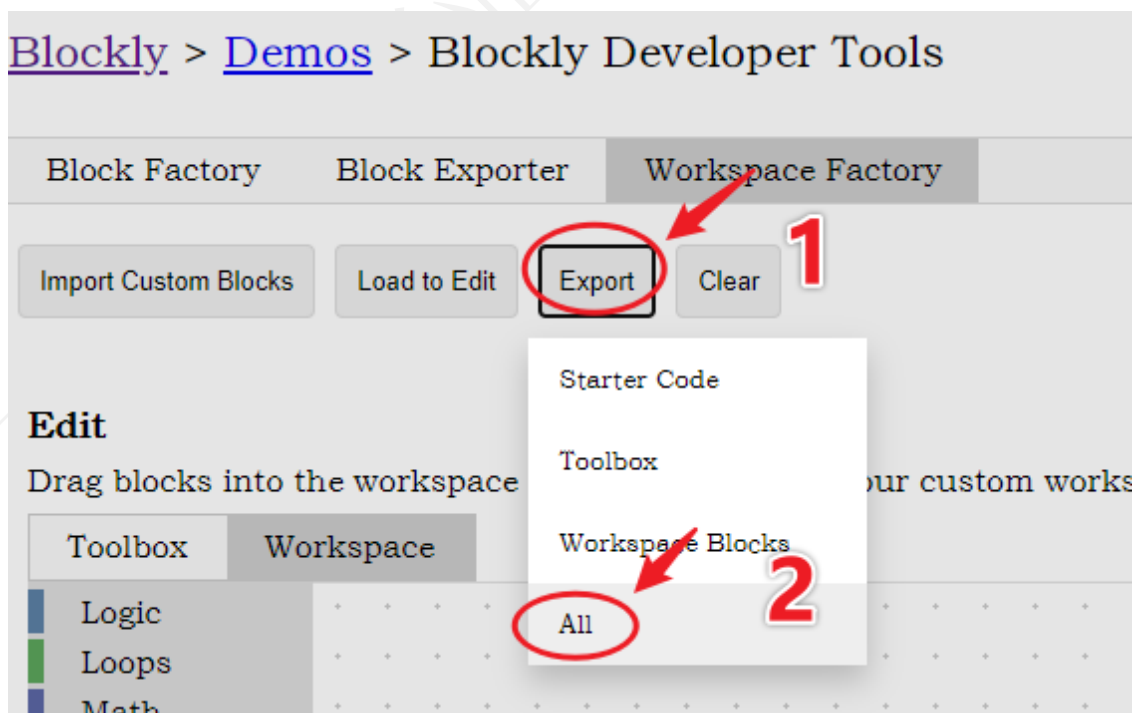
切换到 **Workspace**. 工作区的配置主要是下图所示中间勾选区域。设置 Blockly Media 的路径，该路径包含工作区所需的图标、音频文件等。

- Use Grid 设置工作区网格显示
- Zoom 设置工作区缩放按钮，建议勾选
- Transhcan 设置垃圾桶，建议勾选
- Use Blockly CSS 使用自带的 CSS 样式，建议勾选
- 其余可根据自己喜好决定是否勾选



6.3 导出工作区

待工具箱和工作区都配置完毕之后，可以如下图所示导出工具箱和工作区的配置文件。

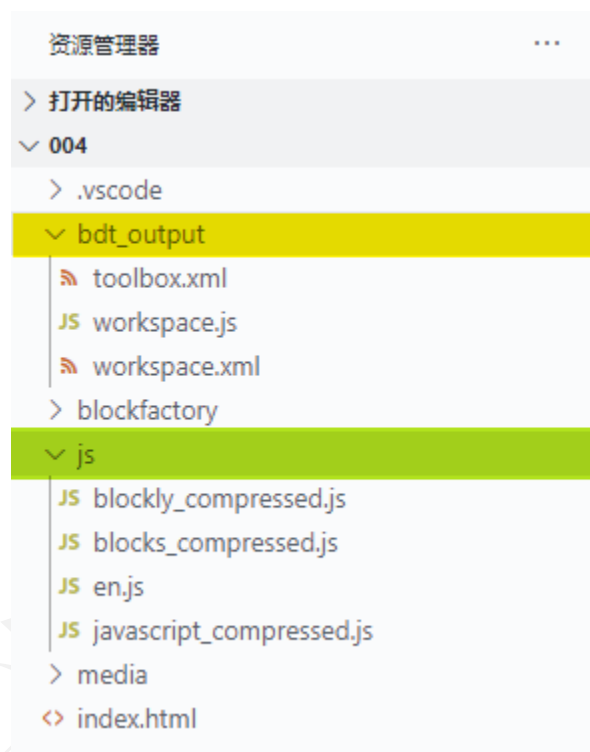


共生成 3 个文件

- workspace.js: 工作区的配置
- toolbox.xml: 工具箱的配置
- workspace.xml: 工作区加载文件

6.4 Web 前端集成

按照下图所示方式，建立文件夹目录结构，将上一节导出的文件放至 `bdt_output`，将依赖的 js 文件放至 `js` 文件夹下，将 `media` 文件夹拷贝过来。然后按照下列步骤执行 Web 集成。



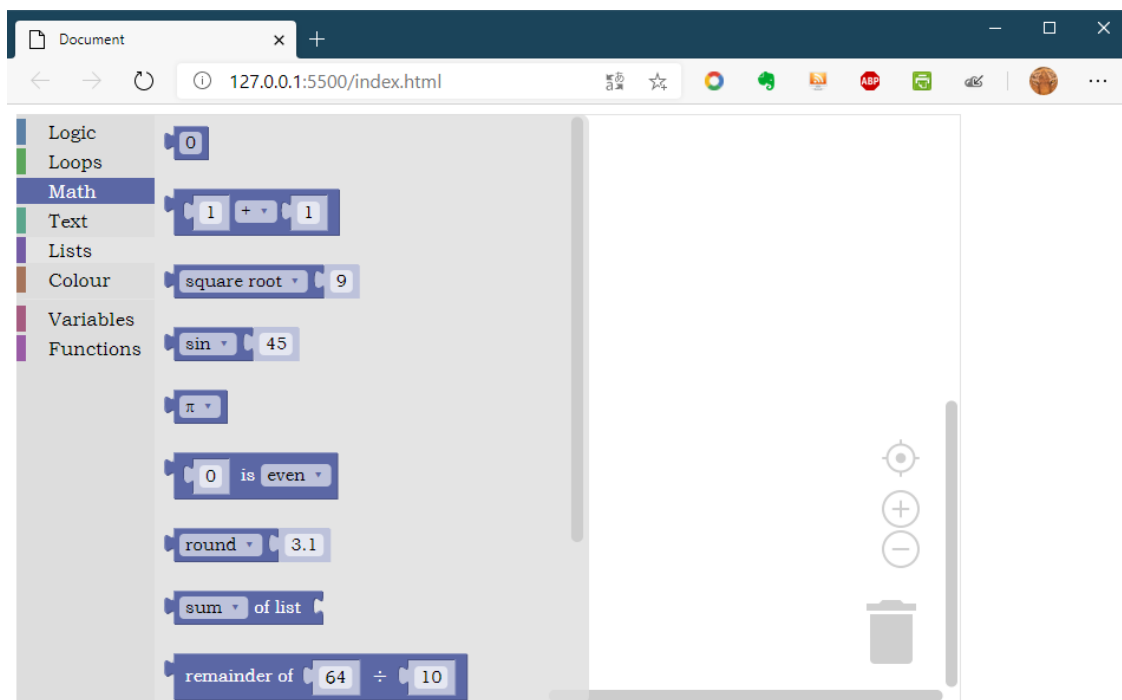
- 创建一个 html 文件，然后，将 `bdt_output/toolbox.xml` 中的内容复制到 `<body></body>` 标签内
- 在 `<head></head>` 标签内插入依赖 js 文件
- 在 `<body></body>` 插入 `<div id="blockdiv" style="width: 800px; height: 500px;"></div>`
- 复制 `workspace.xml` 中的内容，并将其插入到 `<body></body>` 最下方
- 在 `<body></body>` 最下方插入 `<script src="bdt_output/workspace.js"></script>`
- 修改 `workspace.js` 文件，具体修改方式为

```
var workspace = Blockly.inject(/* TODO: Add ID of div to inject Blockly into */, options);
```

修改为:

```
id_blockdiv = document.getElementById("blockdiv");  
var workspace = Blockly.inject(id_blockdiv, options);
```

最终效果如下图所示



6.5 完整代码

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Document</title>  
  
  <script src="js/blockly_compressed.js"></script>  
  <script src="js/blocks_compressed.js"></script>  
  <script src="js/en.js"></script>  
  <script src="js/javascript_compressed.js"></script>  
</head>
```

```
<body>
  <div id="blockdiv" style="width: 800px; height: 500px;"></div>
  <xml xmlns="https://developers.google.com/blockly/xml" id="toolbox" style="display: none">
    <category name="Logic" colour="#5b80a5">
      <block type="controls_if"></block>
      <block type="logic_compare">
        <field name="OP">EQ</field>
      </block>
      ... (略去)

      <block type="text_print">

    <sep></sep>
    <category name="Variables" colour="#a55b80" custom="VARIABLE"></category>
    <category name="Functions" colour="#995ba5" custom="PROCEDURE"></category>
  </xml>

  <xml xmlns="https://developers.google.com/blockly/xml" id="workspaceBlocks" style="display: none">

  <script src="bdt_output/workspace.js"></script>
</body>
</html>
```


第 7 节 Blockly 的程序执行

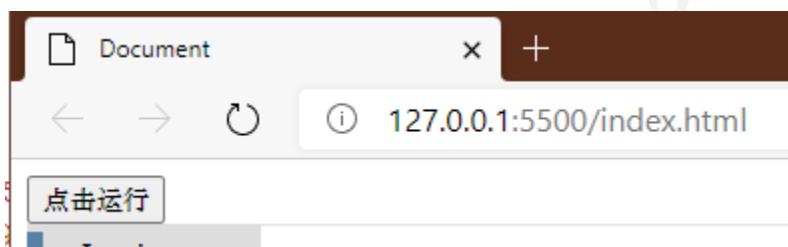
执行 Blockly 编写的程序比较简单，只需要对上一节输出 html 文件稍作修改即可。

7.1 增加执行按钮

在 html 文件的 `<body></body>` 标签内添加一个按钮，当点击此按钮，触发执行一个 JS 函数，通过此函数执行 Blockly 程序。

```
<input type="button" value=" 点击运行 " onclick="runbutton()">
```

上述代码将会在 html 页面显示出一个按钮，如下图所示。



按钮点击时，触发一个叫做 `runbutton` 的函数，该函数需要另外定义。

7.2 JS 执行函数

Blockly 编写的程序无法直接运行，需要先将其转换成 JS 代码，然后才可以在浏览器上执行。转换可以通过以下函数实现：

```
var code = Blockly.JavaScript.workspaceToCode(workspace);
```

上述代码可以将工作区内的 Blockly 程序转换成 JS 代码，需要注意的是，括号内的 `workspace` 需要与 `workspace.js` 中的 `var workspace = Blockly.inject(id_blockdiv, options);` 保持一致。

代码转换之后，可以勇敢 `eval()` 函数执行 JS 代码，完整代码如下所示

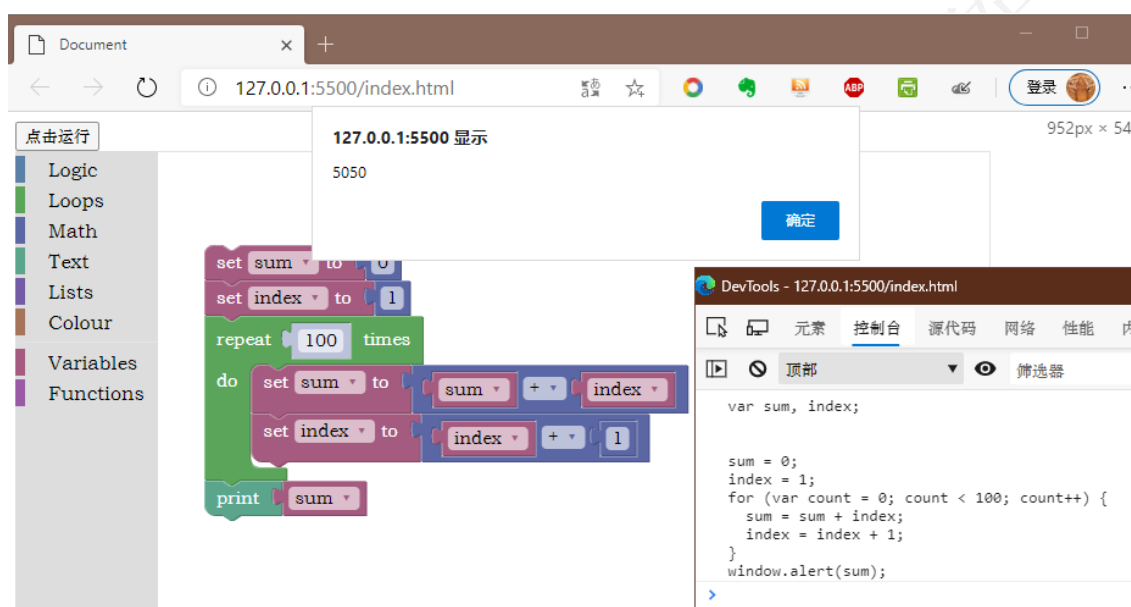
```
<script>
function runbutton (){
    var code = Blockly.JavaScript.workspaceToCode(workspace);
    try {
        console.log(code);
    }
}
```

```

        eval(code);
    } catch (e) {
        alert(e);
    }
};
</script>

```

编辑好 Blockly 之后，点击执行按钮即可运行，运行结果以弹窗的方式展现，如下图所示。右下角为浏览器 console 输出的转换之后的代码。



7.3 在 Web 页面中实时显示 JS 代码

上述转换的 JS 代码需要通过浏览器的控制台查看，如果想在 Web 页面上实时浏览，则需要插入一个事件监听，当 Blockly 发生变化时，即刻转换代码，然后将其显示在页面上。

首先，在页面上添加 `<textarea></textarea>` 标签，用于显示转换代码

```
<textarea name="code-gen" id="code-gen" cols="100" rows="20"></textarea>
```

然后，在页面上添加一个事件监听函数，用于监听 Blockly 程序的修改，代码如下：

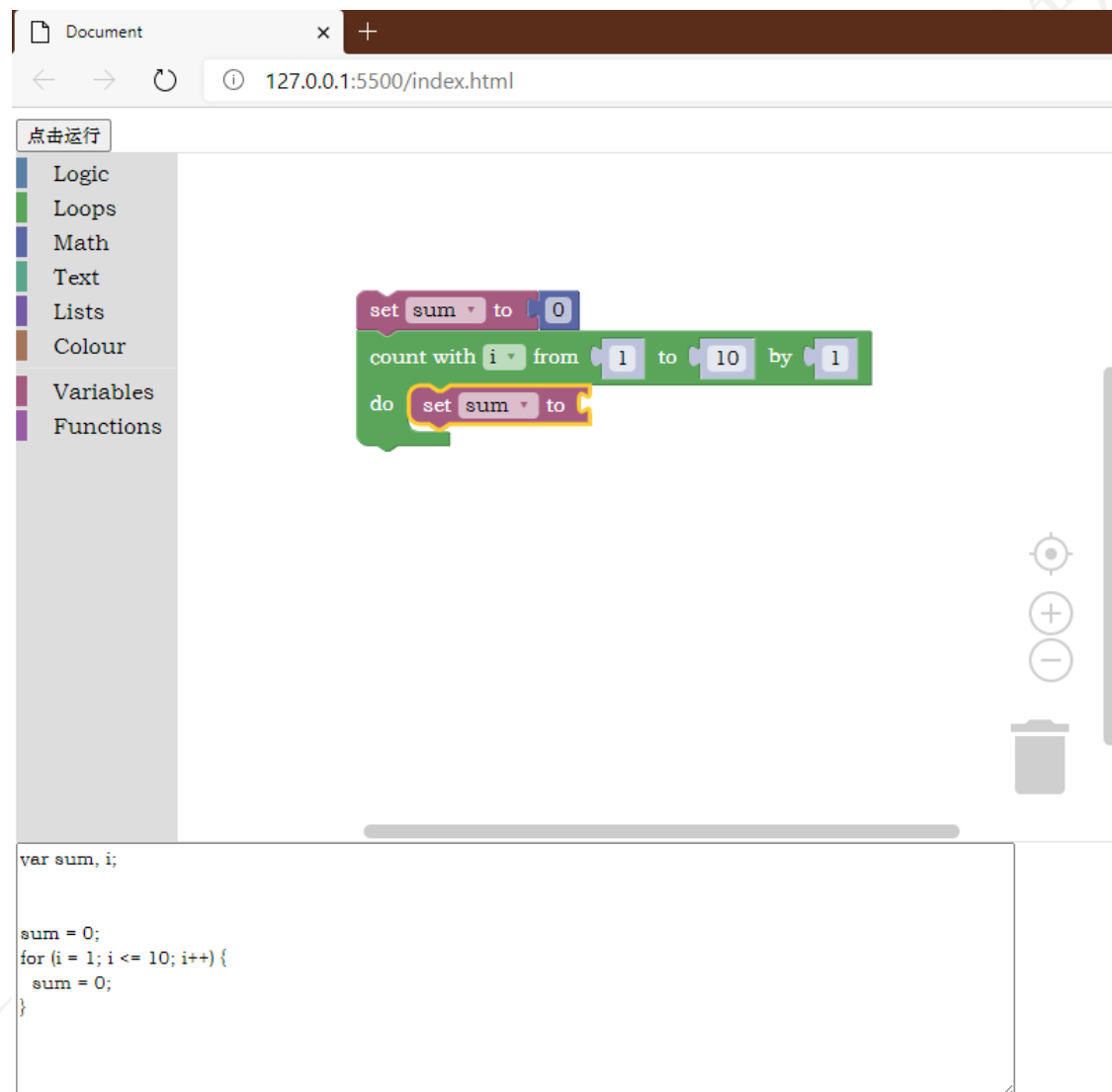
```

<script>
workspace.addChangeListener(function (event) {
    if (!(event instanceof Blockly.Events.Ui)) {
        var lastcode = Blockly.JavaScript.workspaceToCode(workspace);
    }
});

```

```
document.getElementById("code-gen").value = lastcode;
}
});
</script>
```

需要注意，`workspace.addChangeListener` 中的 `workspace` 为 `workspace.js` 中的 `var workspace = Blockly.inject(id_blockdiv, options);`。上述代码中，如果检测到事件，先将工作区内的 Blockly 程序转换成 JS，然后将 JS 代码传给 `textarea`。



7.4 在 Web 页面中实时显示 Python 代码

如果想看实时转换的 python 代码，则将 `Blockly.JavaScript.workspaceToCode` 修改为 `Blockly.Python.workspaceToCode` 即可，同时在 `<head></head>` 标签内添加 python 的代码生

成器。

```
<script src="js/python_compressed.js"></script>
```

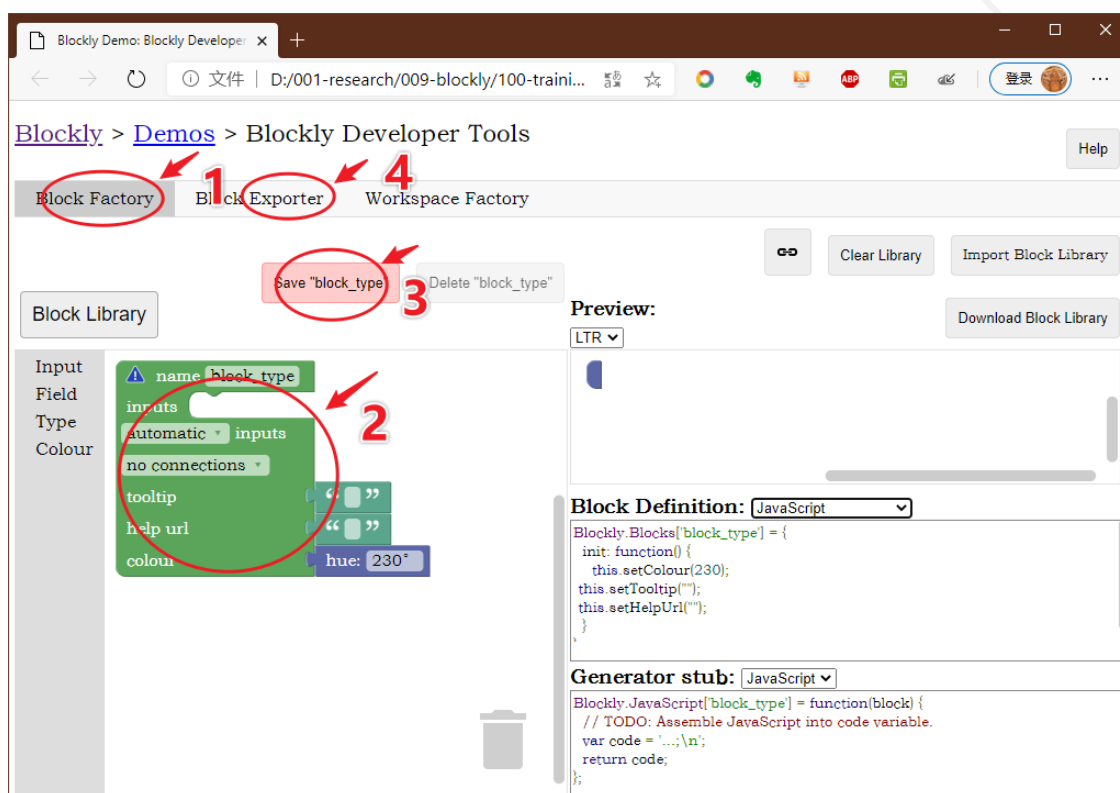
南京工业职业技术大学 - 陈锐

第 8 节 自定义 Block 与代码生成

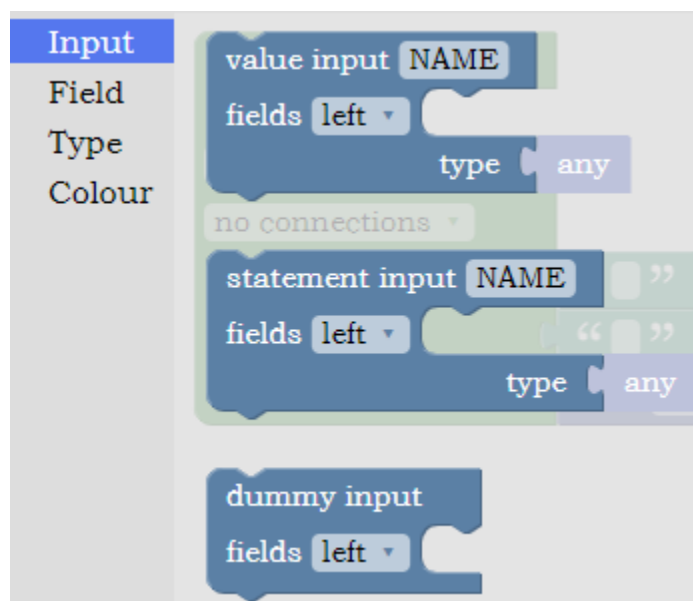
前文描述的内容已经足以将 Blockly 使用起来，但是如有其他需求，比如新增功能，则还需自定义 Block。

自定义 Block 也是通过 Blockly Developer Tool 实现，大概流程如下图所示

- 进入 Blockly Developer Tool，切换至 Block Factory；
- 在 Block Factory 设计 Block，定义其输入、输出、数据类型、颜色等；
- 保存 Block；
- 切换至 Block Factory，导出设计文件。



8.1 Input



Block Factory 提供三种输入类型：

- 数值输入：传入数值，数值类型可以在 `type` 中指定，可以是字符串、数字、Bool 值、数组等，比如，传入两个数字的加法运算结果
- 状态声明输入：声明一个语句，比如循环语句，条件控制语句等
- 虚拟输入：仅用于占位，无需实际输入，可为 Block 提供文字提示等信息

8.1.1 数值输入示例

假设，我们需要计算两个数值的和，可以按照下图所示进行设计。图中左侧的标号与右上角的预览是一一对应的。标号 5 用于设置 Block 的颜色。

The screenshot shows the Blockly IDE interface. On the left is the 'Block Library' with categories: Input, Field, Type, Colour. The main workspace contains a custom block named 'sum_of_two_var'. This block has several fields and inputs:

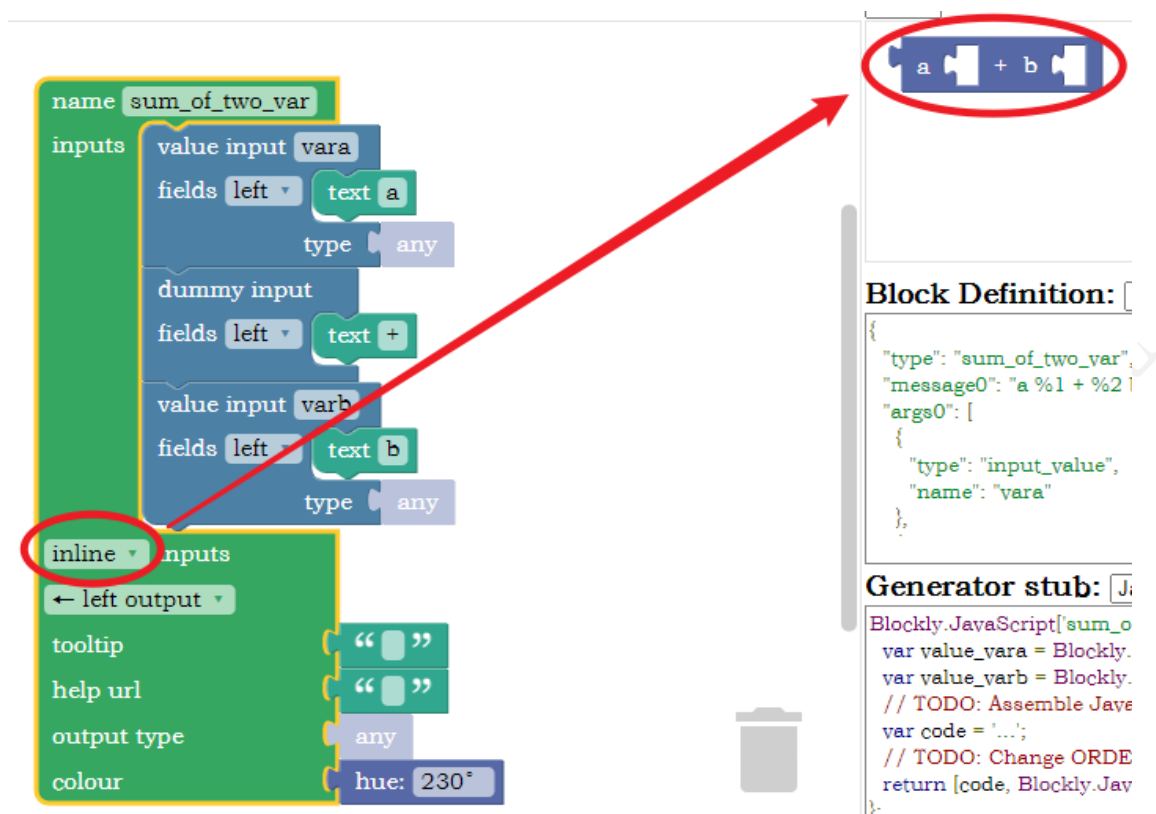
- name:** sum_of_two_var
- inputs:**
 - value input:** var a (labeled 1), type: any, field: left, text: a
 - dummy input:** type: any, field: left, text: + (labeled 2)
 - value input:** var b (labeled 3), type: any, field: left, text: b
- external inputs:**
 - left output:** (labeled 4)
- tooltip:** (empty)
- help url:** (empty)
- output type:** any
- colour:** hue: 230° (labeled 5)

On the right, the 'Preview' section shows a visual representation of the block with inputs 'a' and 'b' and a '+' sign. Below it, the 'Block Definition' section shows the JSON definition for the block. At the bottom, the 'Generator' section shows the generated JavaScript code:

```
Blockly.JavaScript.sum_of_two_var = function(block) {
  var value_a = block.getFieldValue('a');
  var value_b = block.getFieldValue('b');
  // TODO: A
  var code = `
  // TODO: C
  return [code]
`;
```

数值输入的排列方式有两种选择

- external，如上图所示的垂直罗列
- inline，如下图所示，按行罗列



8.1.2 状态声明输入

假设我们要设计一个“if ... do ...”，如果条件满足则执行操作的 Block，可以按照下图所示方式进行设计。

The screenshot shows the Blockly IDE interface. On the left, a block named 'if_do' is being edited. It has a 'value input' with 'condition' and a 'statement input' with 'action'. A red box highlights the 'statement input' section. On the right, the 'Preview' section shows a visual representation of the block, with a red box highlighting the 'do' part. Below the preview, the 'Block Definition' section shows the JSON representation of the block. At the bottom, the 'Generator' section shows the JavaScript code generated from the block.

Save "if_do" Delete "if_do" Preview: LTR ▼

name if_do

inputs

value input condition

fields right ▼ text if

type any

statement input action

fields left ▼ text do

type any

external inputs

↑ top+bottom connections ▼

tooltip

help url

top type

bottom type

colour

hue: 230°

Block Definition

```
{
  "type": "if_do",
  "message0": "if %1 do %2",
  "args0": [
    {
      "type": "input_value",
      "name": "condition",
      "align": "RIGHT"
    },
    {
      "type": "input_value",
      "name": "action",
      "align": "LEFT"
    }
  ]
}
```

Generator

```
Blockly.JavaScript
var value_con
var statement
// TODO: Ass
var code = '...';
```

8.2 Block Definition

此处为自动生成的 Block 定义代码，可选 JSON 格式或者是 JavaScript 代码

The screenshot shows the 'Block Definition' dialog box. It has a dropdown menu set to 'JSON'. Below the dropdown, there is a text area containing the JSON definition of the 'if_do' block. To the right of the text area, there are three buttons: 'JSON', 'JavaScript', and 'Manual JSON...'. The 'JSON' button is currently selected.

Block Definition: JSON ▼

```
{
  "type": "if_do",
  "message0": "if %1 do %2",
  "args0": [
    {
      "type": "input_value",
      "name": "condition",
      "align": "RIGHT"
    },
    {
      "type": "input_value",
      "name": "action",
      "align": "LEFT"
    }
  ]
}
```

JSON JavaScript Manual JSON... Manual JavaScript...

Block Definition: JavaScript ▼

```
Blockly.Blocks['if_do'] = {
  init: function() {
    this.appendValueInput("condition")
      .setCheck(null)
      .setAlign(Blockly.ALIGN_RIGHT)
      .appendField("if");
    this.appendStatementInput("action")
      .setCheck(null)
```

注意上图中的“if_do”就是自定义 Block 的名字。

这一部分生成的代码，基本上不用再修改（除了 Block with Mutator，动态生成 Block）。

8.3 Generator Stub

这一部分负责生成代码框架，具体生成什么代码，还需手动填入。

Generator stub: Python ▼

```
Blockly.Python['if_do'] = JavaScript (k) {
  var value_condition = I Python      1.valueToCode(block, 'condition', Blockly.Python.ORDER_ATO
  var statements_action = PHP          ion.statementToCode(block, 'action');
  // TODO: Assemble Py Lua           e variable.
  var code = '... \n';               Dart
  return code;
};
```

Generator stub: JavaScript ▼

```
Blockly.JavaScript['if_do'] = function(block) {
  var value_condition = Blockly.JavaScript.valueToCode(block, 'condition', Blockly.JavaScript.ORDER_
  var statements_action = Blockly.JavaScript.statementToCode(block, 'action');
  // TODO: Assemble JavaScript into code variable.
  var code = '... \n';
  return code;
};
```

举个例子，假设要生成 JS 代码，那么就将 `var code= '...; \n'` 修改成为

```
var code = 'if( ' + value_condition + ') { \n' + statements_action + '};\n';
```

简单而言，生成代码就是将 Block 的输入以字符串的方式拼接起来。

第 9 节 开发实例 (Python-Turtle 集成)

初学 Python 时，必然会接触到一个叫做 Turtle 的库，利用该库可以画出一些漂亮的几何图形。第一个实例，就是将 Turtle 集成到 Blockly 中，使得通过 Blockly 也可以调用 Turtle 库，画出漂亮的几何图形。以下是一段 Python 代码示例，利用 turtle 画出了一个五角星。这个实例是希望 Blockly 能够自动生成与下列所示完全一样的代码，然后在浏览器中运行 Python，并在浏览器中展示出最终的画图。

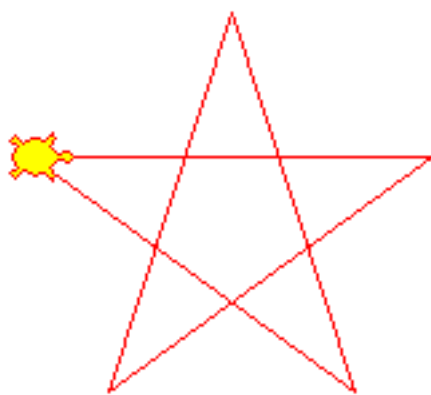
```
[1]: from turtle import * # 导入 turtle 库

color('red') # 设置小乌龟填充色为红色
shape('turtle') # 画出小乌龟

def draw_a_star(): # 小乌龟干的活，画星星的边
    forward(150)
    right(144)

for i in range(5): # 需要画 5 次
    draw_a_star()

done() # 浏览器执行时可不加
```



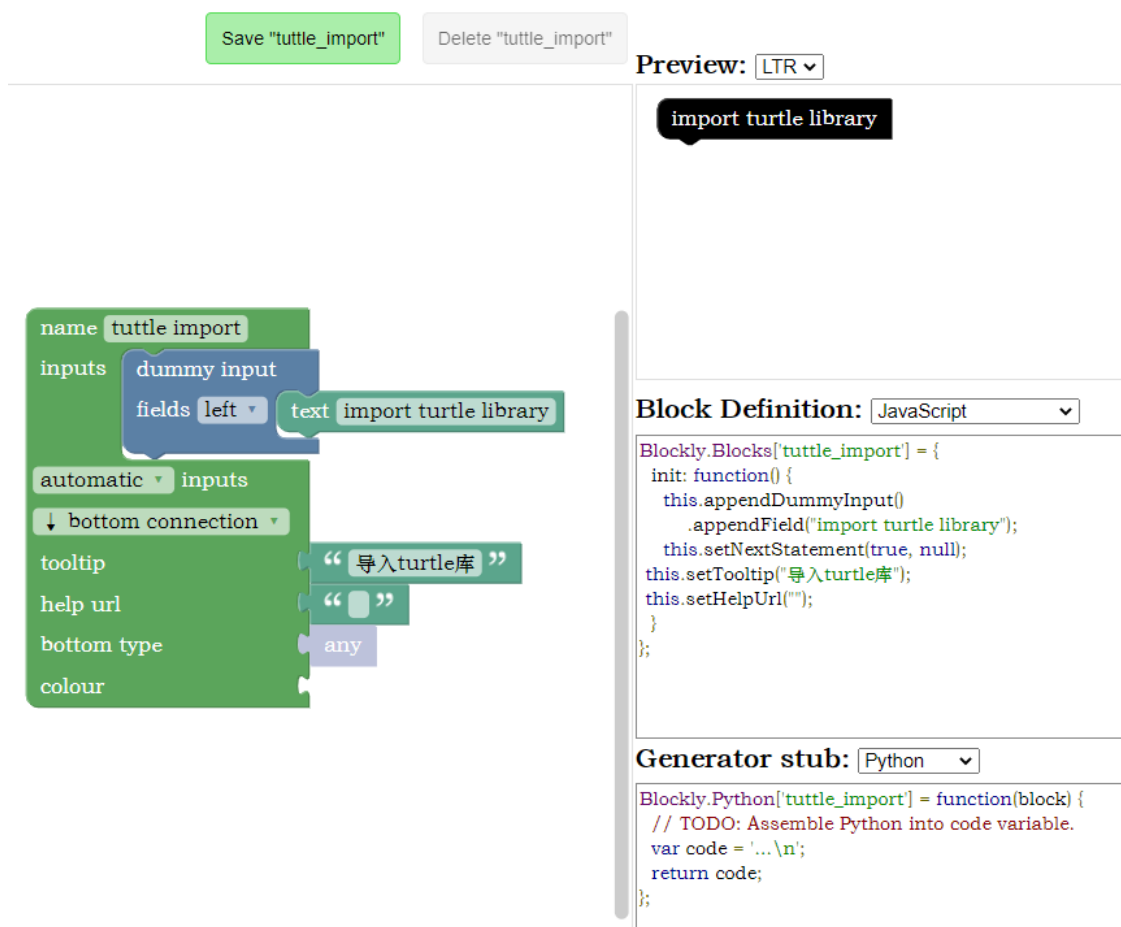
首先，分析上述的代码，通过 Blockly 自动生成上述代码，至少需要以下 5 个自定义的 Block

- 导入 turtle 库的 block
- color() 函数需要一个数值输入的 block
- shape() 函数需要一个数值输入的 block

- forward() 函数需要一个数值输入的 block
- right() 函数需要一个数值输入的 block

9.1 导入 turtle 库的 block

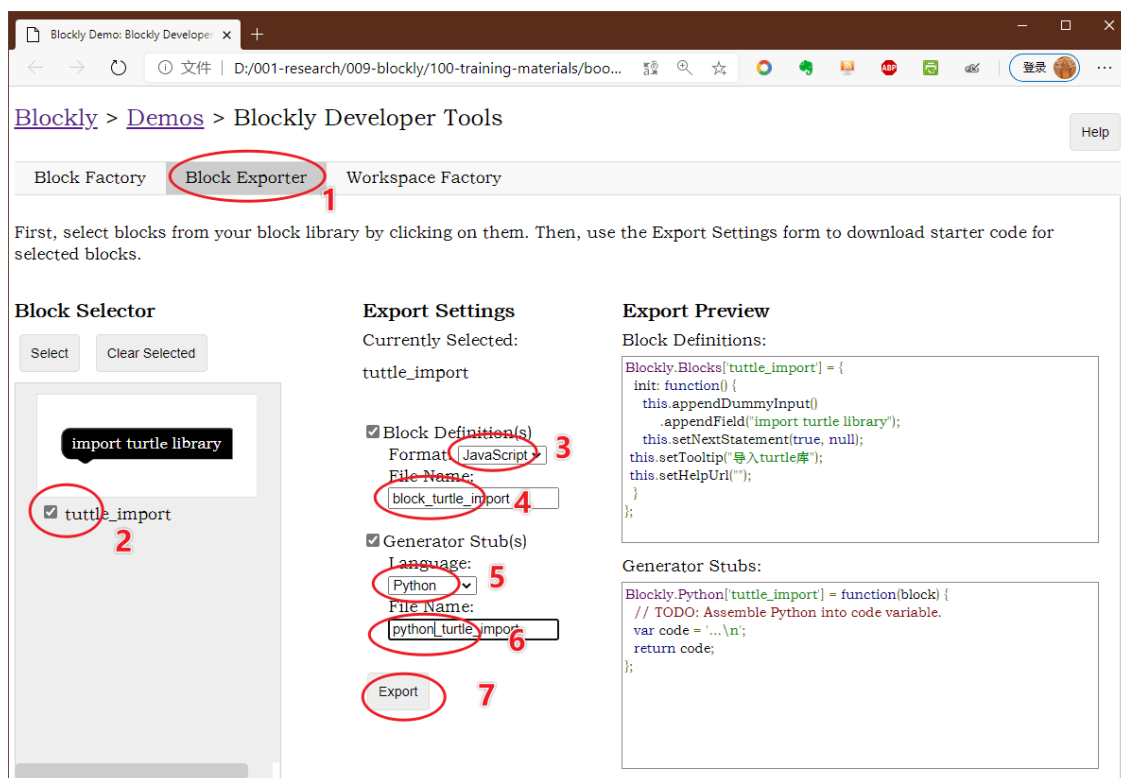
9.1.1 block 设计



第一个 block 不需要任何输入，因此采用 dummy import，并将右下角的 Generator stub 设置为 Python。

9.1.2 导出设计

切换至 Block Exportor，按照下图所示顺序导出设计文件



导出以下两个文件：

- block_turtle_import.js
- python_turtle_import.js

9.1.3 代码生成

修改 python_turtle_import.js 文件，使这个 block 能够生成符合预期的 python 代码

- 原始代码

```
Blockly.Python['tuttle_import'] = function(block) {
  // TODO: Assemble Python into code variable.
  var code = '...\n';
  return code;
};
```

- 修改之后的代码

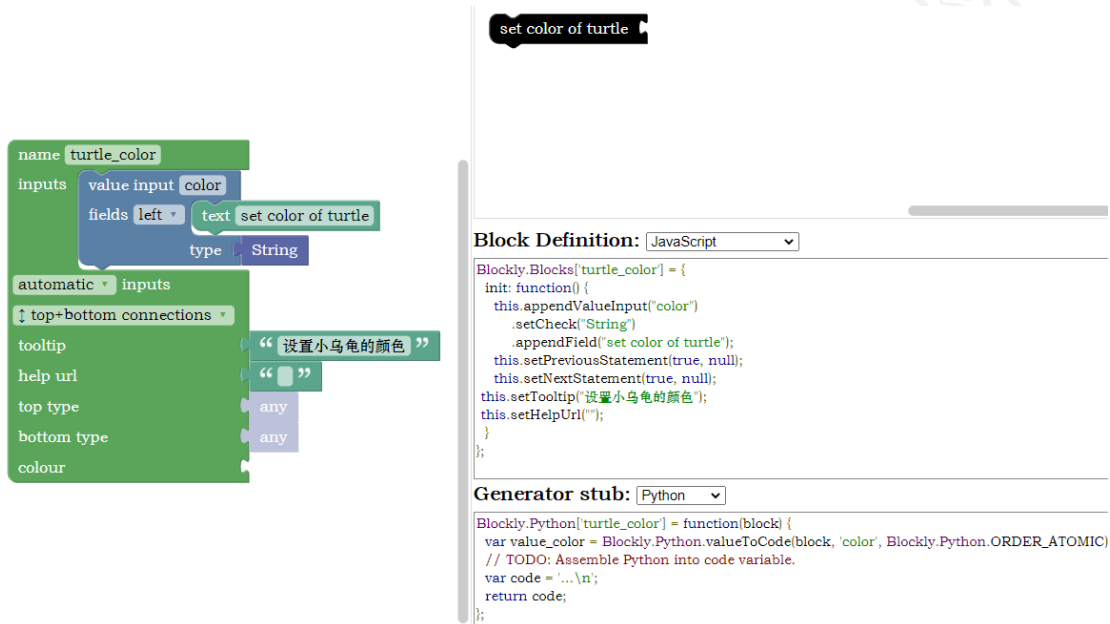
```
Blockly.Python['tuttle_import'] = function(block) {
  // TODO: Assemble Python into code variable.
```

```
var code = 'from turtle import *\n';  
return code;  
};
```

可以看出, 修改内容只不过是将 Python 的代码以字符串的方式插入到 JS 变量 `code` 中。

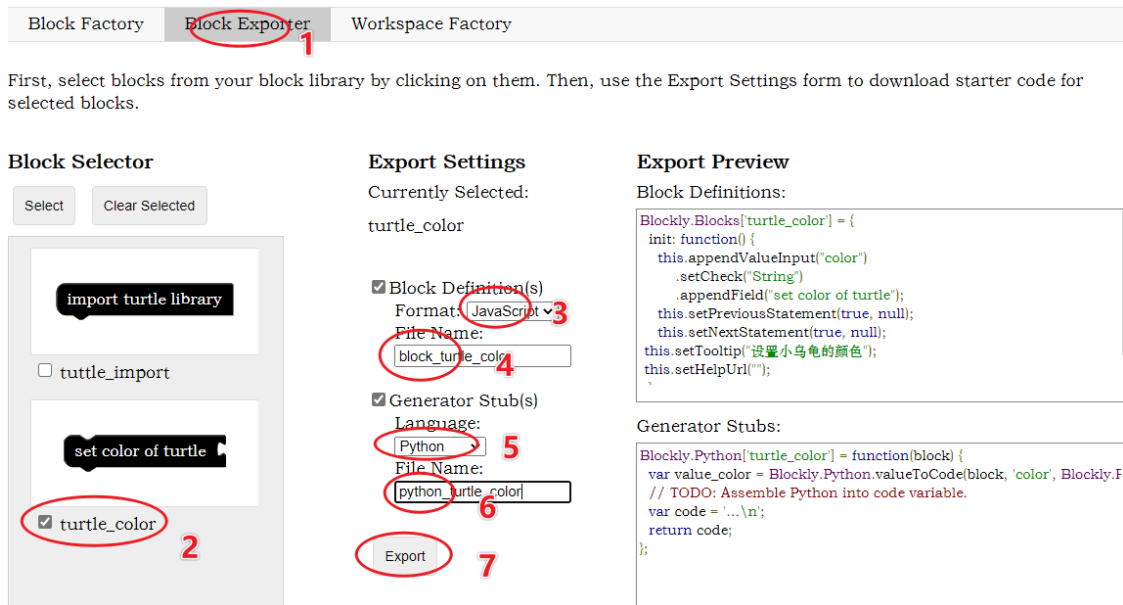
9.2 color() 函数的 block

9.2.1 block 设计



这个 block 需要用户一个表示颜色的字符串, 比如 'red', 'blue', 'green', 'pink' 等。

9.2.2 导出设计



导出两个 js 文件

- block_turtle_color.js
- python_turtle_color.js

9.2.3 代码生成

修改 python_turtle_color.js 文件，使这个 block 能够生成符合预期的 python 代码

- 原始代码

```
Blockly.Python['turtle_color'] = function(block) {
  var value_color = Blockly.Python.valueToCode(block, 'color', Blockly.Python.ORDER_AFTER);
  // TODO: Assemble Python into code variable.
  var code = '...\n';
  return code;
};
```

- 修改后的代码

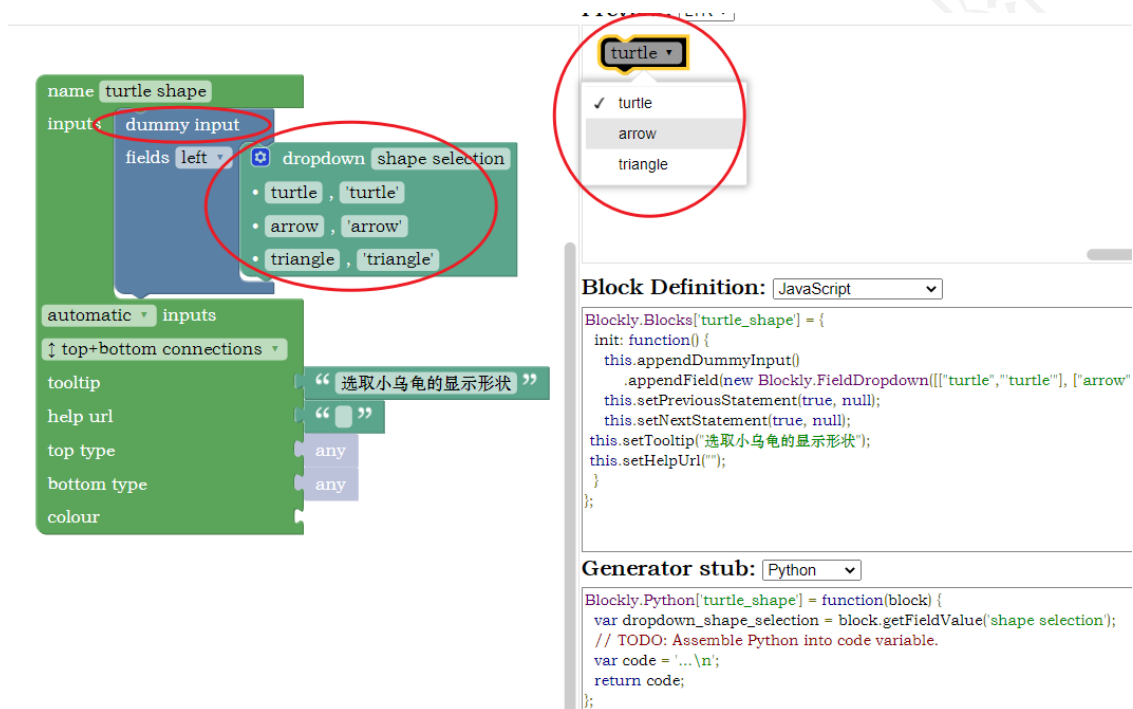
```
Blockly.Python['turtle_color'] = function(block) {
  var value_color = Blockly.Python.valueToCode(block, 'color', Blockly.Python.ORDER_AFTER);
  // TODO: Assemble Python into code variable.
```

```
var code = 'color('+ value_color + ')\n';  
return code;  
};
```

上述代码将 color 与用户的输入进行字符串拼接，然后输出。

9.3 shape() 函数的 Block

9.3.1 block 设计



The image shows the Blockly IDE interface for designing a 'turtle shape' block. The block is named 'turtle shape' and has a 'dummy input' field. A 'dropdown shape selection' field is also present, with a list of options: 'turtle', 'arrow', and 'triangle'. The 'Block Definition' section shows the JavaScript code for the block, and the 'Generator stub' section shows the Python code stub.

Block Definition: JavaScript

```
Blockly.Blocks['turtle_shape'] = {  
  init: function() {  
    this.appendDummyInput()  
    .appendField(new Blockly.FieldDropdown([["turtle", "turtle"], ["arrow", "arrow"], ["triangle", "triangle"]]), "shape selection");  
    this.setPreviousStatement(true, null);  
    this.setNextStatement(true, null);  
    this.setTooltip("选取小乌龟的显示形状");  
    this.setHelpUrl("");  
  }  
};
```

Generator stub: Python

```
Blockly.Python['turtle_shape'] = function(block) {  
  var dropdown_shape_selection = block.getFieldValue("shape selection");  
  // TODO: Assemble Python into code variable.  
  var code = '... \n';  
  return code;  
};
```

这个 block 与前面 2 个不同，没有直接的数值输入，而是采用下拉菜单的方式选取。

9.3.2 导出设计

The screenshot shows the Blockly Block Exporter interface with three main panels: Block Selector, Export Settings, and Export Preview.

- Block Selector:** The 'Block Exporter' tab is selected. In the 'Block Selector' panel, the 'turtle_shape' block is selected (indicated by a red circle and the number 2).
- Export Settings:** The 'Currently Selected:' block is 'turtle_shape'. The 'Block Definition(s)' checkbox is checked. The 'Format' is set to 'JavaScript'. The 'File Name' is 'block_turtle_shape' (indicated by a red circle and the number 3). The 'Generator Stub(s)' checkbox is checked. The 'Language' is set to 'Python' (indicated by a red circle and the number 4). The 'File Name' is 'python_turtle_shape' (indicated by a red circle and the number 5). The 'Export' button is highlighted (indicated by a red circle and the number 6).
- Export Preview:** The 'Block Definitions:' section shows the JavaScript code for the 'turtle_shape' block. The 'Generator Stubs:' section shows the Python code for the 'turtle_shape' block.

导出两个文件：

- python_turtle_shape.js
- block_turtle_shape.js

9.3.3 代码生成

修改 python_turtle_shape.js 文件，使这个 block 能够生成符合预期的 python 代码

- 原始代码

```
Blockly.Python['turtle_shape'] = function(block) {
  var dropdown_shape_selection = block.getFieldValue('shape selection');
  // TODO: Assemble Python into code variable.
  var code = '...\n';
  return code;
};
```

- 修改之后的代码

```
Blockly.Python['turtle_shape'] = function(block) {  
  var dropdown_shape_selection = block.getFieldValue('shape selection');  
  // TODO: Assemble Python into code variable.  
  var code = 'shape(' + dropdown_shape_selection + ')\n';  
  return code;  
};
```

9.4 forward() 函数的 block

9.4.1 block 设计

The image shows the Blockly IDE interface for designing a 'turtle forward' block. The block is green and has the following properties:

- name:** turtle forward
- inputs:** value input NAME
- fields:** left (dropdown), text forward distance (text input)
- type:** Number
- automatic:** inputs
- top+bottom connections:** top+bottom connections (dropdown)
- tooltip:** “设置小乌龟前进的距离，以像素为单位”
- help url:** “ ”
- top type:** any
- bottom type:** any
- colour:** (empty)

The **Block Definition:** is set to JavaScript and shows the following code:

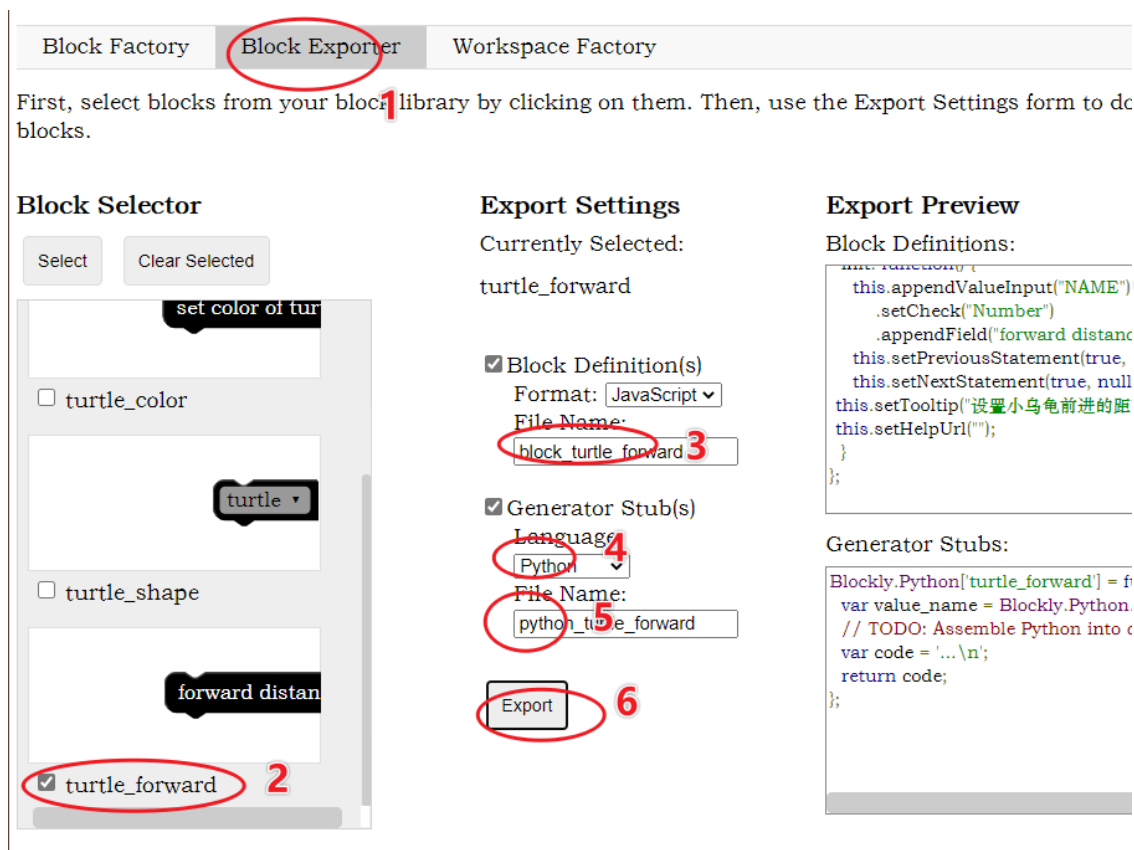
```
Blockly.Blocks['turtle_forward'] = {  
  init: function() {  
    this.appendValueInput("NAME")  
      .setCheck("Number")  
      .appendField("forward distance");  
    this.setPreviousStatement(true, null);  
    this.setNextStatement(true, null);  
    this.setTooltip("设置小乌龟前进的距离，以像素为单位");  
    this.setHelpUrl("");  
  }  
};
```

The **Generator stub:** is set to Python and shows the following code:

```
Blockly.Python['turtle_forward'] = function(block) {  
  var value_name = Blockly.Python.valueToCode(bl  
  // TODO: Assemble Python into code variable.  
  var code = '...\n';  
  return code;  
};
```

这个 block 需要用户输入一个具体的数值，以表示小乌龟爬行的距离。

9.4.2 导出设计



导出以下 2 个文件

- python_turtle_forward.js
- block_turtle_forward.js

9.4.3 代码生成

- 原始代码

```

Blockly.Python['turtle_forward'] = function(block) {
  var value_name = Blockly.Python.valueToCode(block, 'NAME', Blockly.Python.ORDER_ATOM
  // TODO: Assemble Python into code variable.
  var code = '...\n';
  return code;
};

```

- 修改之后的代码

```

Blockly.Python['turtle_forward'] = function(block) {
  var value_name = Blockly.Python.valueToCode(block, 'NAME', Blockly.Python.ORDER_ATOM);
  // TODO: Assemble Python into code variable.
  var code = 'forward(' + value_name + ')\n';
  return code;
};

```

9.5 right() 函数的 block

9.5.1 block 设计

Save "turtle_right"
Delete "turtle_right"

name turtle right

inputs

value input NAME

fields left

text set angle of turn right

type Number

automatic inputs

↑ top+bottom connections

tooltip “ 设置向右转的角度 ”

help url “ ”

top type any

bottom type any

colour

Preview: LTR

set angle of turn right

Block Definition: JavaScript

```

Blockly.Blocks['turtle_right'] = {
  init: function() {
    this.appendValueInput("NAME")
      .setCheck("Number")
      .appendField("set angle of turn right");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setTooltip("设置向右转的角度");
    this.setHelpUrl("");
  }
};

```

Generator stub: Python

```

Blockly.Python['turtle_right'] = function(block) {
  var value_name = Blockly.Python.valueToCode(block, 'NAME', Blockly.Python.ORDER_ATOM);
  // TODO: Assemble Python into code variable.
  var code = '... \n';
  return code;
};

```

9.5.2 导出设计

The screenshot shows the 'Block Exporter' interface with three main panels:

- Block Selector:** Contains a list of blocks. The 'turtle_right' block is selected and circled in red. Other visible blocks include 'turtle_shape', 'forward distance', 'turtle_forward', and 'set angle of turn'.
- Export Settings:** Shows 'Currently Selected: turtle_right'. It has two sections:
 - Block Definition(s):** 'Format' is set to 'JavaScript'. 'File Name' is 'block_turtle_right' (circled in red).
 - Generator Stub(s):** 'Language' is 'Python' (circled in red). 'File Name' is 'python_turtle_right' (circled in red).
 An 'Export' button is at the bottom.
- Export Preview:** Displays the generated code for the selected block.


```
Blockly.Blocks['turtle_right'] = {
  init: function() {
    this.appendValueInput("NAME")
      .setCheck("Number")
      .appendField("set angle of turn right");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setTooltip("设置向右转的角度");
    this.setHelpUrl("");
  }
};

Blockly.Python['turtle_right'] = function(block) {
  var value_name = Blockly.Python.valueToCode(block, 'NAME', Blockly.Python.ORDER_ATOM);
  // TODO: Assemble Python into code variable.
  var code = '...\n';
  return code;
};
```

导出 2 个文件

- python_turtle_right.js
- block_turtle_right.js

9.5.3 代码生成

- 原始代码

```
Blockly.Python['turtle_right'] = function(block) {
  var value_name = Blockly.Python.valueToCode(block, 'NAME', Blockly.Python.ORDER_ATOM);
  // TODO: Assemble Python into code variable.
  var code = '...\n';
  return code;
};
```

- 修改之后的代码

```
Blockly.Python['turtle_right'] = function(block) {
    var value_name = Blockly.Python.valueToCode(block, 'NAME', Blockly.Python.ORDER_ATOM);
    // TODO: Assemble Python into code variable.
    var code = 'right(' + value_name + ')\n';
    return code;
};
```

9.6 Blockly 集成

浏览器中执行 Python 需要第三方库的支持，本文采用 Skulpt 作为 Python 支持库。使用 Skulpt 非常简单，只需要在 HTML 文件的 `<head></head>` 标签内插入 Skulpt 的 JS 文件即可，以及少许 JS 代码即可。

9.6.1 引入 python 浏览器执行环境

- 在 `<head></head>` 标签内插入以下 js 文件

```
<script src="js/skulpt.min.js" type="text/javascript"></script>
<script src="js/skulpt-stdlib.js" type="text/javascript"></script>
```

- 在 `<head></head>` 标签内 (页面最底部) 插入以下 JS 代码

```
<script type="text/javascript">
    function builtinRead(x) {
        if (
            Sk.builtinFiles === undefined ||
            Sk.builtinFiles["files"][x] === undefined
        )
            throw "File not found: '" + x + "'";
        return Sk.builtinFiles["files"][x];
    }

    function runbutton(){
        var python_code = Blockly.Python.workspaceToCode(workspace);
        console.log(python_code);
        Sk.configure({
            read: builtinRead
        });
        (Sk.TurtleGraphics || (Sk.TurtleGraphics = {})).target = "mycanvas";
```

```
var myPromise = Sk.misceval.asyncToPromise(function () {  
    return Sk.importMainWithBody("<stdin>", false, python_code, true);  
});  
}  
</script>
```

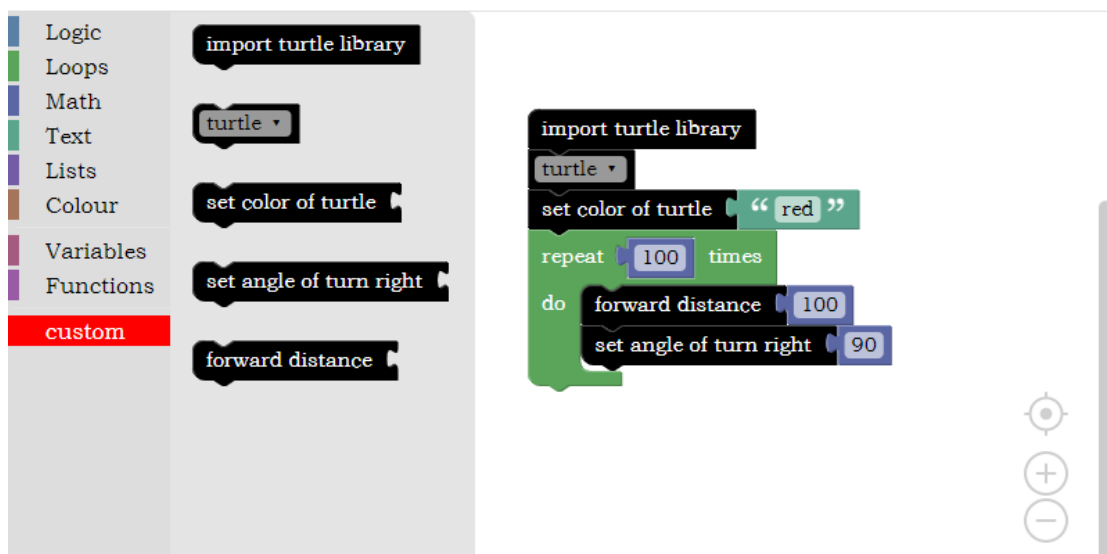
9.6.2 在 <head></head> 标签内引入自定义的 block

```
<script src = "custom/block_turtle_forward.js"></script>  
<script src = "custom/block_turtle_import.js"></script>  
<script src = "custom/block_turtle_right.js"></script>  
<script src = "custom/block_turtle_shape.js"></script>  
<script src = "custom/block_turtle_color.js"></script>  
<script src = "custom/python_turtle_color.js"></script>  
<script src = "custom/python_turtle_forward.js"></script>  
<script src = "custom/python_turtle_import.js"></script>  
<script src = "custom/python_turtle_right.js"></script>  
<script src = "custom/python_turtle_shape.js"></script>
```

9.6.3 在 <body></body> 标签内引入画布

```
<div id="mycanvas"></div>
```

9.7 最终效果



第 10 节 扩展：保存与加载

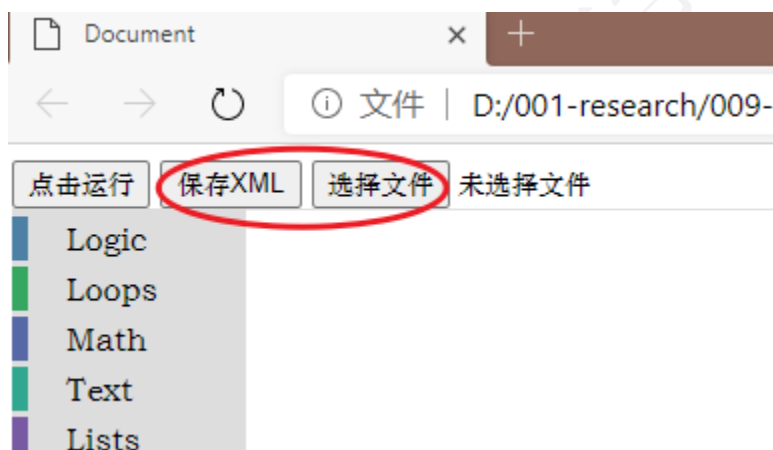
随着自定义 Block 的增多，或者编写程序的复杂度增加，可能有加载和保存当前程序的需求。本节就是为了解决这个问题。

在上一节代码的基础上，增加两个按钮，一个负责加载之前保存的 blockly 程序，一个负责保存当前的程序。注意：blockly 程序将以 xml 文件的方式保存。

首先，在 html 页面的 `<body></body>` 上添加这按钮

```
<input type="button" value=" 保存 XML" onclick="storexml()">
<input id="loadxml" type="file" value=" 加载 XML">
```

第一行，点击按钮时调用一个 JS 函数将当前的 blockly 程序保存下来。第二行，弹出一个界面选取一个 xml 文件，然后自动加载到 Blockly 的工作区中。添加之后的效果如下图所示。



10.1 保存当前的 Blockly 程序

首先，在 html 页面的 `<head></head>` 标签内插入保存程序所依赖的 JS 文件，该文件可以从 Github 上下载，下载地址为：

<https://github.com/eligrey/FileSaver.js/>

```
<script src = "js/FileSaver.js"></script>
```

然后，在 html 页面的 `<body></body>` 上添加此按钮

```
<input type="button" value=" 保存 XML" onclick="storexml()">
```

将下列 JS 程序插入到 `<body></body>` 标签内。当点击这个按钮时，调用下列的函数，自动将当前工作区内的 Blockly 程序保存至 `block.xml` 文件中

```
<script>
    function storexml() {
        var xmlDom = Blockly.Xml.workspaceToDom(workspace);
        var xmlText = Blockly.Xml.domToPrettyText(xmlDom);

        var blob = new Blob([xmlText]);
        window.saveAs(blob, "blockly.xml")
    }
</script>
```

10.2 加载 Blockly 程序

首先，在 html 页面的 `<body></body>` 标签内添加一个输入按钮，注意要给它分配一个 id。

```
<input id="loadxml" type="file" value="加载 XML">
```

将下列 JS 程序插入到 `<body></body>` 标签内。点击这个输入按钮时，触发一个 JS 函数，该函数监听对应的 id 的输入按钮，发现其被点击时，执行加载程序。

```
<script>
    document.getElementById("loadxml").addEventListener("change", function () {
        var file = this.files[0];
        var reader = new FileReader(),
            rawlog = "empty";
        reader.readAsText(file);

        reader.onload = function (e) {
            rawlog = reader.result;
            workspace.clear();
            var xml = Blockly.Xml.textToDom(rawlog);
            Blockly.Xml.domToWorkspace(xml, workspace);
        };
    });
</script>
```

第 11 节 扩展：字典数据结构

字典是一种常用的数据结构，但是 Blockly 本身并不支持该结构。本节将创建能够动态添加不定数量 Key/Value 对的 Block。

11.1 创建 Key/Value Block

该 Block 能够生成 KEY : VALUE 这样的代码，作为字典的元素。

11.1.1 Block 设计

The image shows the Blockly IDE interface for creating a 'key value pair' block. The block is designed with two inputs: 'KEY' and 'VALUE'. Each input has a text field (containing 'key' and 'value' respectively) and a type dropdown menu set to 'any'. The block is named 'key value pair'. The 'inline' checkbox is checked, and the 'left output' is selected. The tooltip is '创建一对键值' (Create a key-value pair). The help url is '“ ”' (Double quotes). The output type is 'any'. The colour is green. On the right, the 'Block Definition' and 'Generator stub' are shown in JavaScript.

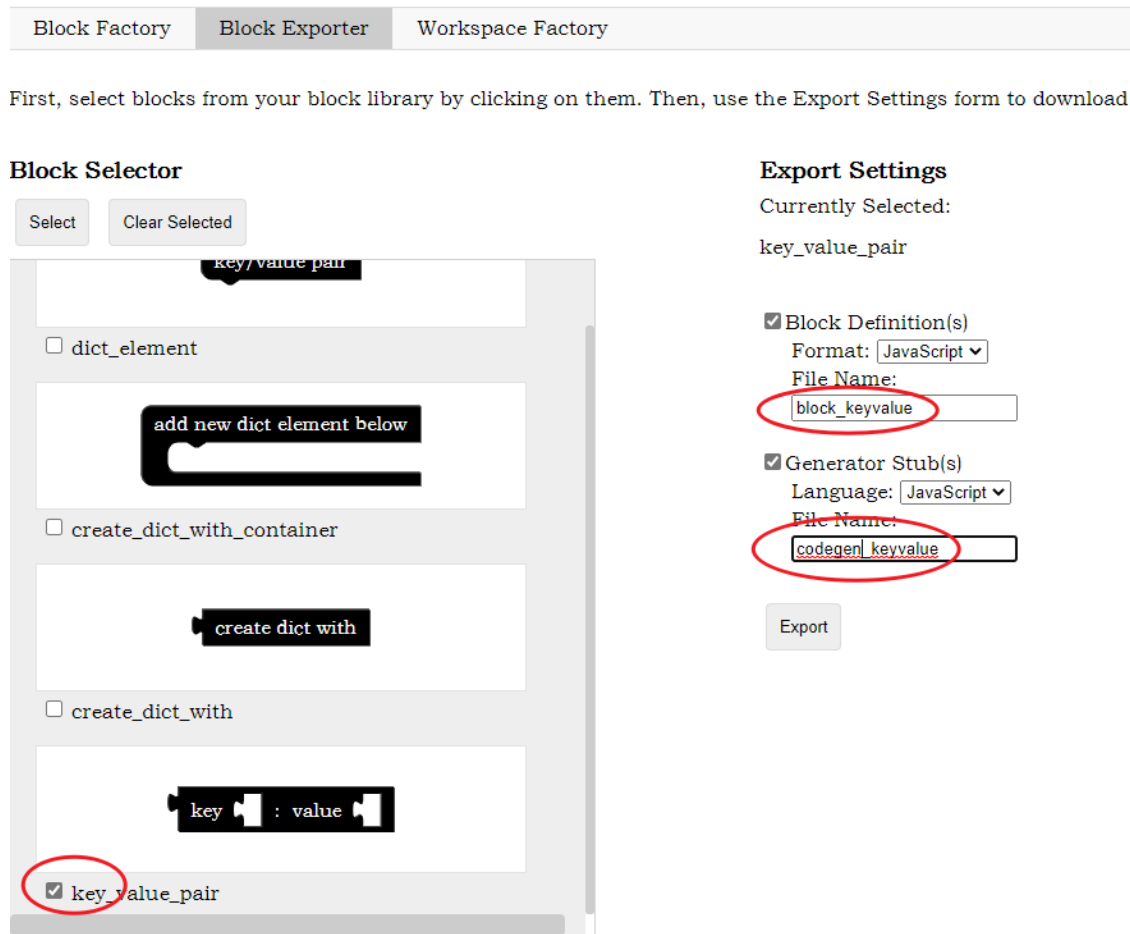
Block Definition: JavaScript

```
appendDummyInput()
.appendField(":");
this.appendValueInput("VALUE")
.setCheck(null)
.appendField("value");
this.setInputsInline(true);
this.setOutput(true, null);
this.setTooltip("创建一对键值");
this.setHelpUrl("");
};
```

Generator stub: JavaScript

```
Blockly.JavaScript['key_value_pair'] = function(block)
```

11.1.2 导出设计



导出 2 个 JS 文件：

- block_keyvalue.js
- codegen_keyvalue.js

11.1.3 代码生成

- 原始代码

```
Blockly.JavaScript['key_value_pair'] = function(block) {
  var value_key = Blockly.JavaScript.valueToCode(block, 'KEY', Blockly.JavaScript.ORDER_ATOMIC);
  var value_value = Blockly.JavaScript.valueToCode(block, 'VALUE', Blockly.JavaScript.ORDER_ATOMIC);
  // TODO: Assemble JavaScript into code variable.
  var code = '...';
}
```

```
// TODO: Change ORDER_NONE to the correct strength.  
return [code, Blockly.JavaScript.ORDER_NONE];  
};
```

- 修改之后的代码

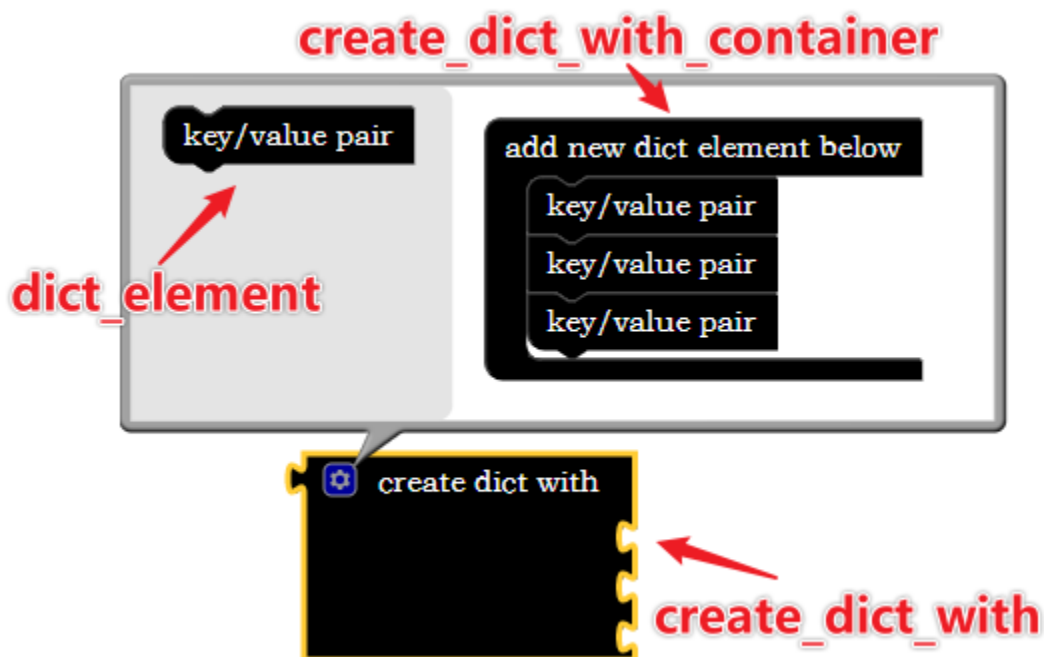
```
Blockly.JavaScript['key_value_pair'] = function(block) {  
  var value_key = Blockly.JavaScript.valueToCode(block, 'KEY', Blockly.JavaScript.ORDER_ATOMIC);  
  var value_value = Blockly.JavaScript.valueToCode(block, 'VALUE', Blockly.JavaScript.ORDER_ATOMIC);  
  // TODO: Assemble JavaScript into code variable.  
  var code = value_key + ' : ' + value_value;  
  // TODO: Change ORDER_NONE to the correct strength.  
  return [code, Blockly.JavaScript.ORDER_ATOMIC];  
};
```

11.2 创建字典 Block

由于需要字典中的 key/value 的数量不定，需要设计能够动态调整输入数量的 Block。这一部分稍微复杂一些，具体实现参考 Blockly 中的创建列表。

11.2.1 Block 设计

我们要创建的 Block 能够通过一个弹窗调整输入的数量，具体如下图所示。其中 dict_element 和 create_dict_with_container 并不需要生成代码，这两个 Block 仅用于控制输入数量。



Preview: LTR ▼

key/value pair

Block Definition: JavaScript ▼

```
Blockly.Blocks['dict_element'] = {
  init: function() {
    this.appendDummyInput()
      .appendField("key/value pair");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setTooltip("");
    this.setHelpUrl("");
  }
};
```

Generator stub: JavaScript ▼

```
Blockly.JavaScript['dict_element'] = function(block) {
  // TODO: Assemble JavaScript into code variable.
  var code = '...;\n';
  return code;
};
```

name dict element

inputs

dummy input

fields left text key/value pair

automatic inputs

top+bottom connections

tooltip

help url

top type any

bottom type any

colour

The screenshot shows the Blockly IDE interface. On the left, a block named 'create_dict_with_container' is visible. It has a 'dummy input' field with a 'left' dropdown and a 'text' field containing 'add new dict element below'. Below this is a 'statement input' field with a 'left' dropdown and a 'type' dropdown set to 'any'. The 'statement input' field is highlighted with a red circle and a red arrow pointing to it. The block also has an 'automatic' dropdown, an 'inputs' dropdown, a 'no connections' dropdown, and fields for 'tooltip', 'help url', and 'colour'.

On the right, the 'Block Definition' section shows the JavaScript code for the block:

```
Blockly.Blocks['create_dict_with_container'] = {
  init: function() {
    this.appendDummyInput()
      .appendField("add new dict element below");
    this.appendStatementInput("STACK")
      .setCheck(null);
    this.setTooltip("");
    this.setHelpUrl("");
  }
};
```

Below the definition, the 'Generator stub' section shows the JavaScript code for the block:

```
Blockly.JavaScript['create_dict_with_container'] = function(block) {
  var statements_stack = Blockly.JavaScript.statementToCode(block, 'STACK');
  // TODO: Assemble JavaScript into code variable.
  var code = '...';
  return code;
};
```

创建弹出窗口内的 Block

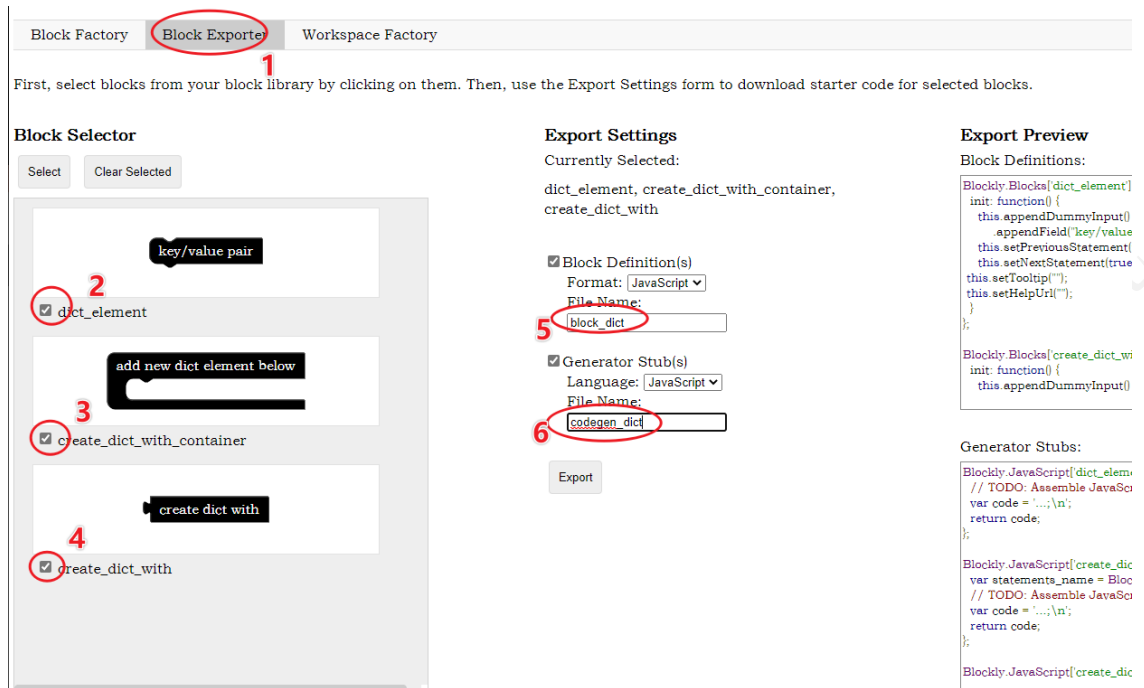
The screenshot shows the Blockly IDE interface. On the left, a block named 'create_dict_with' is visible. It has a 'dummy input' field with a 'left' dropdown and a 'text' field containing 'create dict with'. Below this is an 'automatic' dropdown, an 'inputs' dropdown, a 'left output' dropdown, and fields for 'tooltip', 'help url', 'output type', and 'colour'.

On the right, the 'Block Definition' section shows the JavaScript code for the block:

```
Blockly.Blocks['create_dict_with'] = {
  init: function() {
    this.appendDummyInput()
      .appendField("create dict with");
    this.setOutput(true, null);
    this.setTooltip("");
    this.setHelpUrl("");
  }
};
```

创建字典 Block

11.2.2 导出设计



11.2.3 修改 Block

- 原始 Block 代码

```
Blockly.Blocks['create_dict_with'] = {
  init: function() {
    this.appendDummyInput()
      .appendField("create dict with");
    this.setOutput(true, null);
    this.setTooltip("");
    this.setHelpUrl("");
  }
};
```

- 修改之后的 Block 代码：需要检测用户输入的数量，然后动态调整 Block，这段代码参考 Blockly 源代码中的创建列表。

```
Blockly.Blocks["create_dict_with"] = {
  init: function () {
    this.appendDummyInput()
      .appendField("create dict with");
```



```

    this.setOutput(true, null);
    this.itemCount_ = 3;
    this.updateShape_();
    // this.setColour(0);
    this.setMutator(new Blockly.Mutator(["dict_element"]));
    this.setTooltip("");
    this.setHelpUrl("");
  },

  /**
   * Create XML to represent list inputs.
   * @return {!Element} XML storage element.
   * @this {Blockly.Block}
   */
  mutationToDom: function () {
    var container = Blockly.utils.xml.createElement("mutation");
    container.setAttribute("items", this.itemCount_);
    return container;
  },

  /**
   * Parse XML to restore the list inputs.
   * @param {!Element} xmlElement XML storage element.
   * @this {Blockly.Block}
   */
  domToMutation: function (xmlElement) {
    this.itemCount_ = parseInt(xmlElement.getAttribute("items"), 10);
    this.updateShape_();
  },

  /**
   * Populate the mutator's dialog with this block's components.
   * @param {!Blockly.Workspace} workspace Mutator's workspace.
   * @return {!Blockly.Block} Root block in mutator.
   * @this {Blockly.Block}
   */
  decompose: function (workspace) {
    var containerBlock = workspace.newBlock("create_dict_with_container");
    containerBlock.initSvg();

```

```

    var connection = containerBlock.getInput("STACK").connection;
    for (var i = 0; i < this.itemCount_; i++) {
        var itemBlock = workspace.newBlock("dict_element");
        itemBlock.initSvg();
        connection.connect(itemBlock.previousConnection);
        connection = itemBlock.nextConnection;
    }
    return containerBlock;
},
/**
 * Reconfigure this block based on the mutator dialog's components.
 * @param {!Blockly.Block} containerBlock Root block in mutator.
 * @this {Blockly.Block}
 */
compose: function (containerBlock) {
    var itemBlock = containerBlock.getInputTargetBlock("STACK");
    // Count number of inputs.
    var connections = [];
    while (itemBlock) {
        connections.push(itemBlock.valueConnection_);
        itemBlock =
            itemBlock.nextConnection && itemBlock.nextConnection.targetBlock();
    }
    // Disconnect any children that don't belong.
    for (var i = 0; i < this.itemCount_; i++) {
        var connection = this.getInput("ADD" + i).connection.targetConnection;
        if (connection && connections.indexOf(connection) == -1) {
            connection.disconnect();
        }
    }
    this.itemCount_ = connections.length;
    this.updateShape_();
    // Reconnect any child blocks.
    for (var i = 0; i < this.itemCount_; i++) {
        Blockly.Mutator.reconnect(connections[i], this, "ADD" + i);
    }
},

```

```

/**
 * Store pointers to any connected child blocks.
 * @param {!Blockly.Block} containerBlock Root block in mutator.
 * @this {Blockly.Block}
 */
saveConnections: function (containerBlock) {
  var itemBlock = containerBlock.getInputTargetBlock("STACK");
  var i = 0;
  while (itemBlock) {
    var input = this.getInput("ADD" + i);
    itemBlock.valueConnection_ = input && input.connection.targetConnection;
    i++;
    itemBlock =
      itemBlock.nextConnection && itemBlock.nextConnection.targetBlock();
  }
},
/**
 * Modify this block to have the correct number of inputs.
 * @private
 * @this {Blockly.Block}
 */
updateShape_: function () {
  if (this.itemCount_ && this.getInput("EMPTY")) {
    this.removeInput("EMPTY");
  } else if (!this.itemCount_ && !this.getInput("EMPTY")) {
    this.appendDummyInput("EMPTY").appendField(
      // Blockly.Msg["LISTS_CREATE_EMPTY_TITLE"]
      " please insert key/value pair"
    );
  }
  // Add new inputs.
  for (var i = 0; i < this.itemCount_; i++) {
    if (!this.getInput("ADD" + i)) {
      var input = this.appendValueInput("ADD" + i).setAlign(
        Blockly.ALIGN_RIGHT
      );
      // if (i == 0) {

```

```

        // input.appendField( " key/value pair");
        // }
    }
}
// Remove deleted inputs.
while (this.getInput("ADD" + i)) {
    this.removeInput("ADD" + i);
    i++;
}
},
};

```

11.2.4 代码生成

- 原始代码

```

Blockly.JavaScript["create_dict_with"] = function (block) {
    // TODO: Assemble JavaScript into code variable.
    var code = "...";
    // TODO: Change ORDER_NONE to the correct strength.
    return [code, Blockly.JavaScript.ORDER_NONE];
};

```

- 修改之后的代码：需要检测用户调整的输入数量，然后依据输入数量生成代码。

```

Blockly.JavaScript['create_dict_with'] = function(block) {
    // Create a list with any number of elements of any type.
    var elements = new Array(block.itemCount_);
    for (var i = 0; i < block.itemCount_; i++) {
        elements[i] = Blockly.JavaScript.valueToCode(block, 'ADD' + i,
            // Blockly.JavaScript.ORDER_COMMA) || 'null';
            Blockly.JavaScript.ORDER_NONE) || 'null';
    }
    var code = '{' + elements.join(', ') + '}';
    return [code, Blockly.JavaScript.ORDER_ATOMIC];
};

```

11.2.5 HTML 页面集成

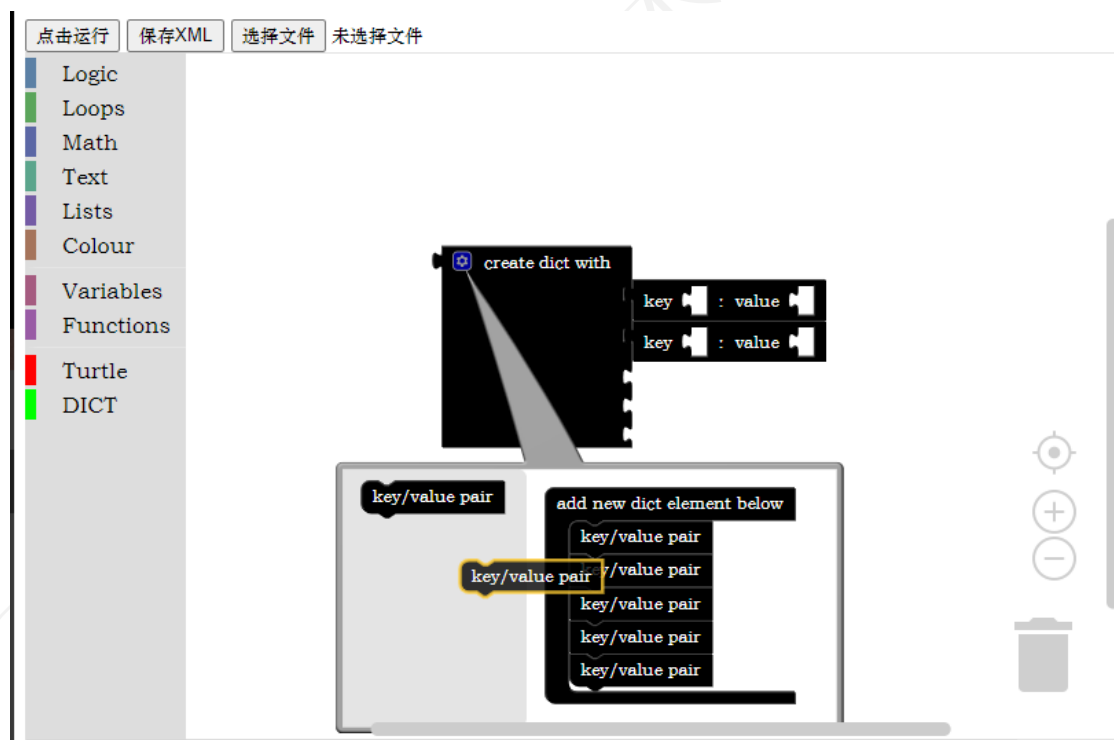
- 添加 JS 文件至 <head></head> 标签

```
<script src = "custom/block_dict.js"></script>
<script src = "custom/codegen_dict.js"></script>
<script src = "custom/block_keyvalue.js"></script>
<script src = "custom/codegen_keyvalue.js"></script>
```

- 修改 Blockly 工具箱的 XML 文件，插入以下代码

```
<category name="DICT" colour="#00FF00">
  <block type="create_dict_with"> </block>
  <block type="key_value_pair"> </block>
</category>
```

11.2.6 最终效果



第 12 节 扩展：开发实例 (Echart 集成)

Echart 是一个 JS 数据可视化库，通过此库可以在网页上实现数据可视化。本节内容是将 Echart 集成到 Blockly 中，使得 Blockly 也可以是实现数据可视化的功能。

12.1 Echart 实例

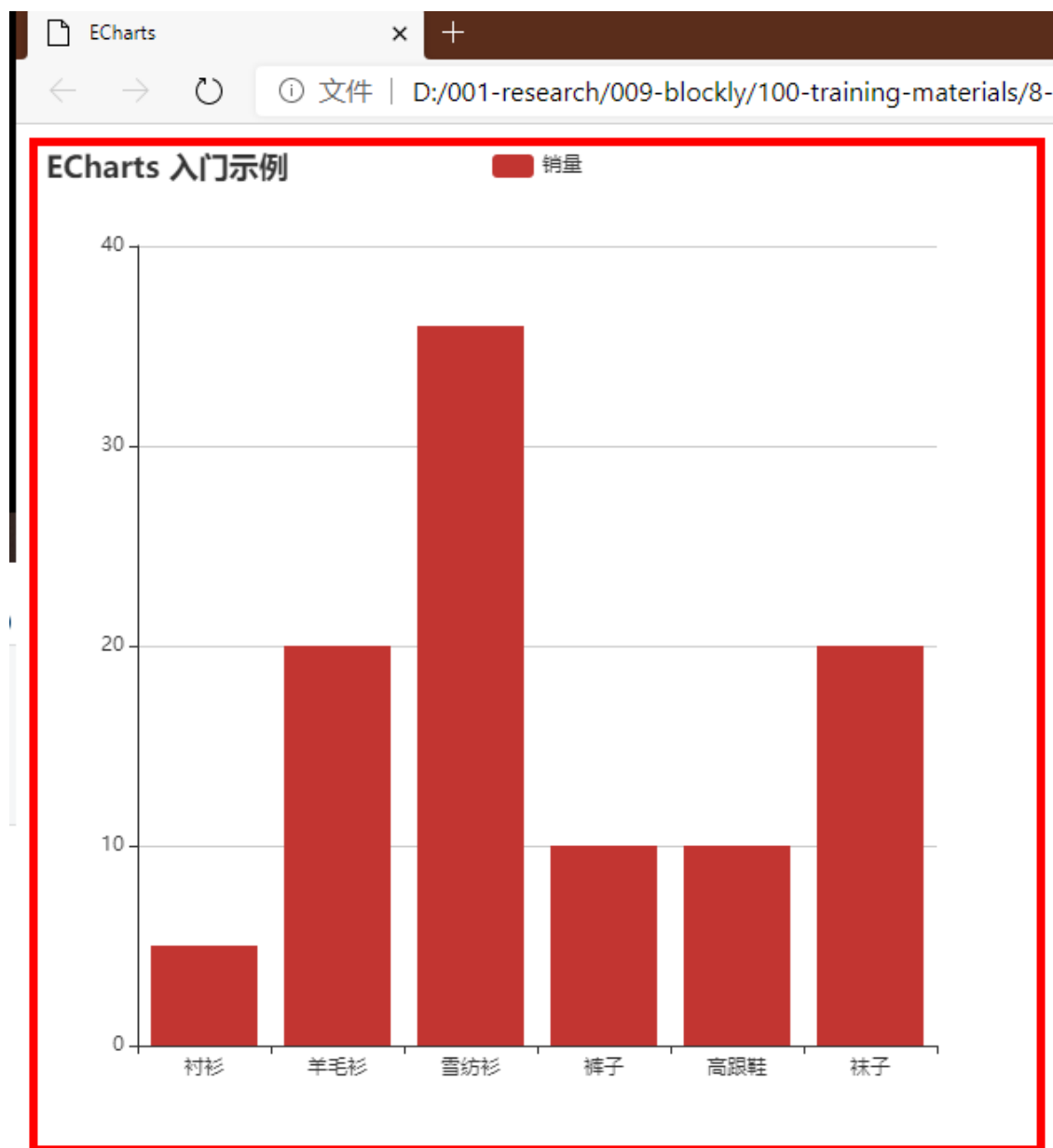
首先来看一个简单的 Echart 实例，通过网页内嵌 JS 代码，实现柱状图的绘制。

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>ECharts</title>
  <!-- 引入 echarts.js -->
  <script src="https://cdn.jsdelivr.net/npm/echarts@4.8.0/dist/echarts.min.js"></script>
</head>
<body>
  <!-- 为 ECharts 准备一个具备大小（宽高）的 Dom -->
  <div id="main" style="border:solid red 5px; width: 600px;height:600px;"></div>
  <script type="text/javascript">
    // 基于准备好的 dom，初始化 echarts 实例
    var myChart = echarts.init(document.getElementById('main'));

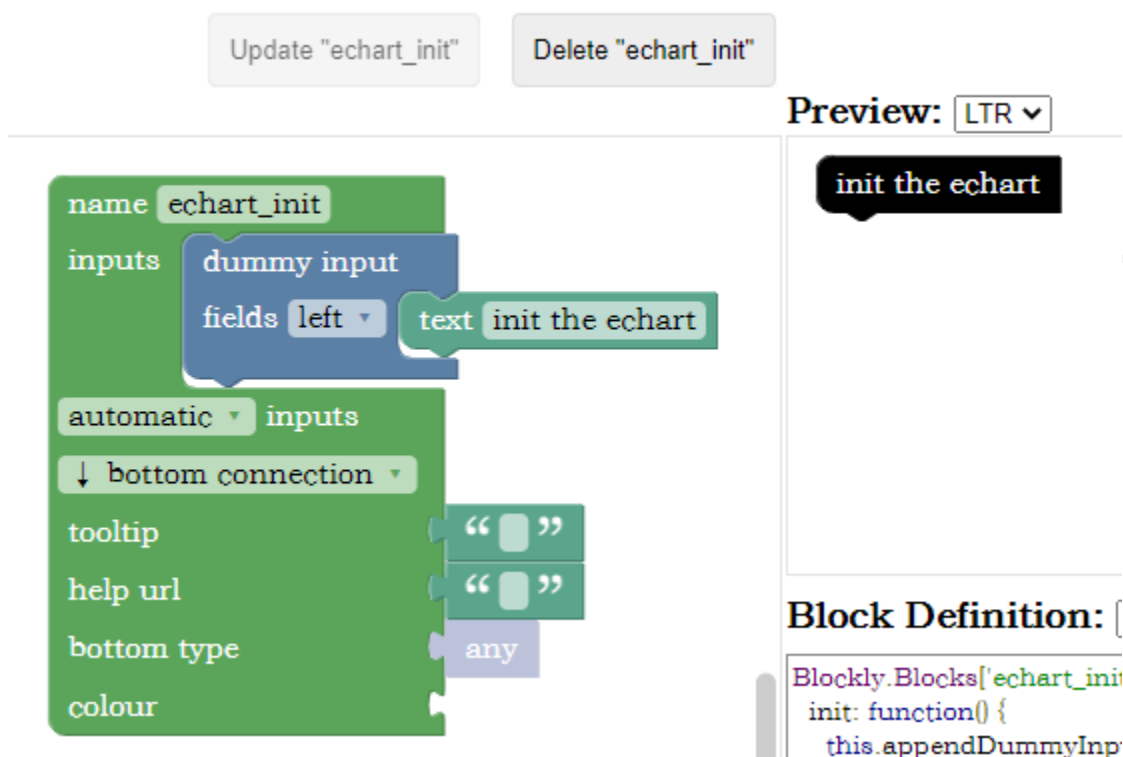
    // 指定图表的配置项和数据
    var option = {
      title: {
        'text': 'ECharts 入门示例'
      },
      legend: {
        'data': ['销量']
      },
      tooltip: {},
      xAxis: {
        'data': [" 衬衫"," 羊毛衫"," 雪纺衫"," 裤子"," 高跟鞋"," 袜子"]
      },
      yAxis: {},
    };
```

```
        series: [{  
            'name': '销量',  
            'type': 'bar',  
            'data': [5, 20, 36, 10, 10, 20]  
        }]  
    };  
  
    // 使用刚指定的配置项和数据显示图表。  
    myChart.setOption(option);  
</script>  
</body>  
</html>
```

上述代码能够实现下图所示的效果



12.2 Block 设计



上述 Block 用于初始化 echart，对应的代码为

```
var myChart = echarts.init(document.getElementById('main'));
```

Update "echart_title"
Delete "echart_title"

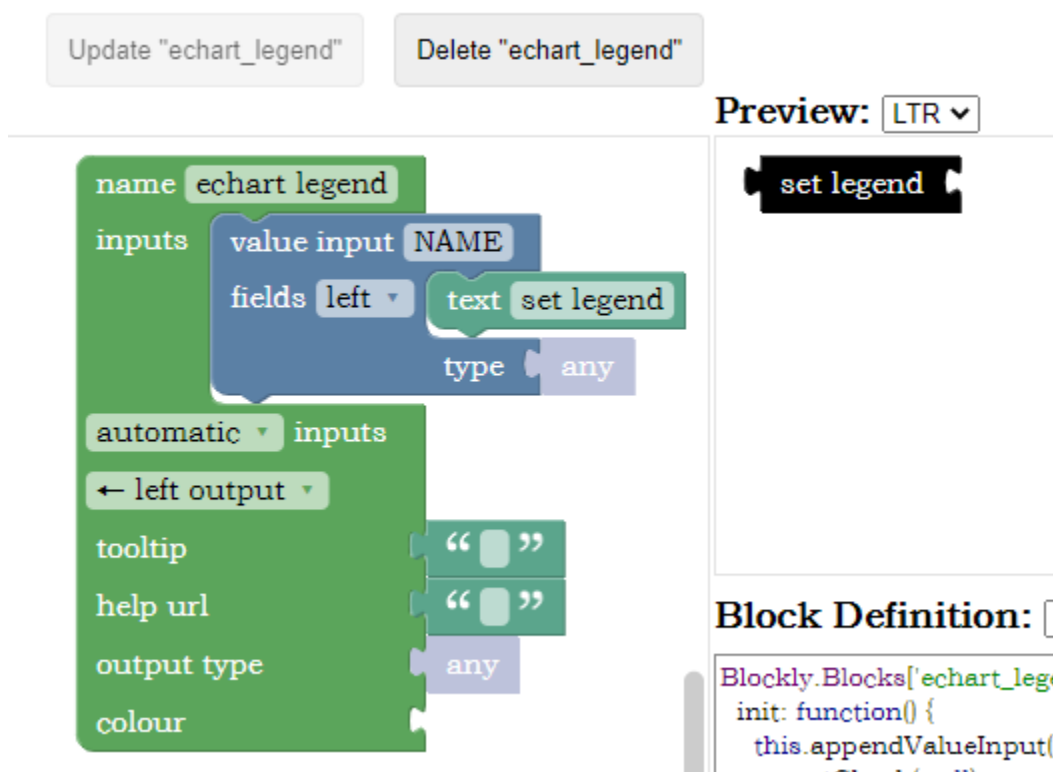
Preview: LTR ▾

Block Definition: JavaS

```
Blockly.Blocks['echart_title'] = {
  init: function() {
    this.appendValueInput("NAME")
    this.appendText("set title")
  }
}
```

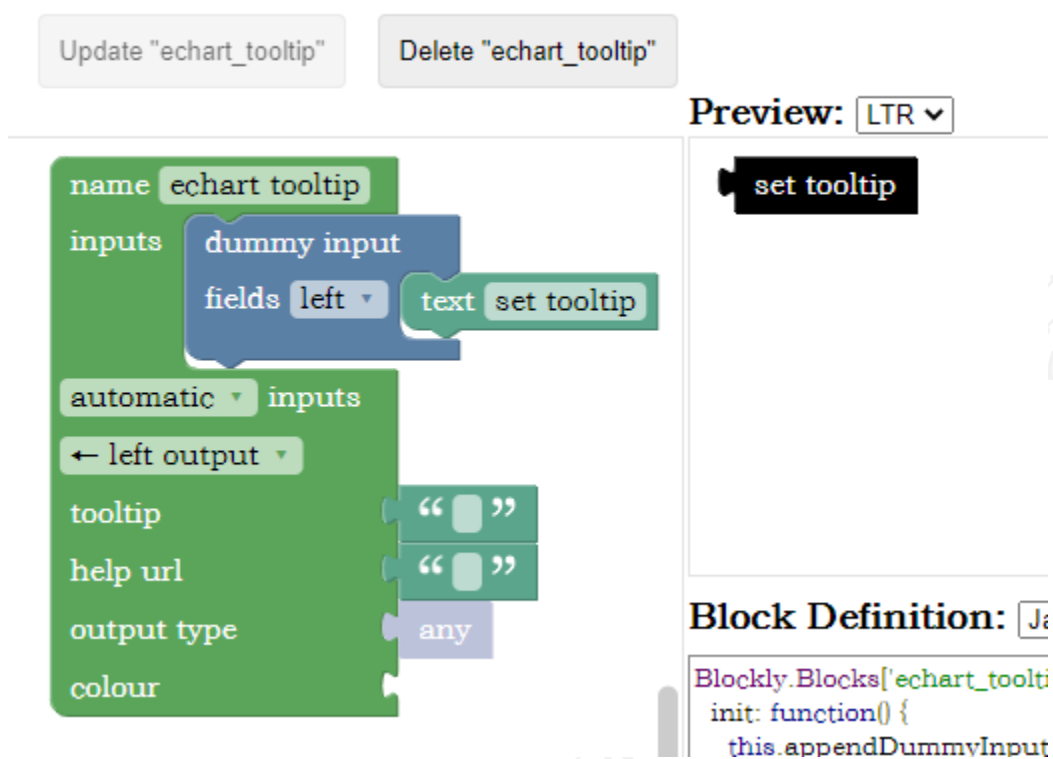
上述代码用于实现图标标题的设计，对应的代码

```
title: {
  'text': 'ECharts 入门示例'
},
```



上述 block 用于实现图例的设置，对应的代码为

```
legend: {  
  'data': ['销量']  
},
```



The screenshot displays the Blockly IDE interface for defining a custom block named 'echart tooltip'. At the top, there are two buttons: 'Update "echart_tooltip"' and 'Delete "echart_tooltip"'. The main workspace shows the block definition with the following fields and inputs:

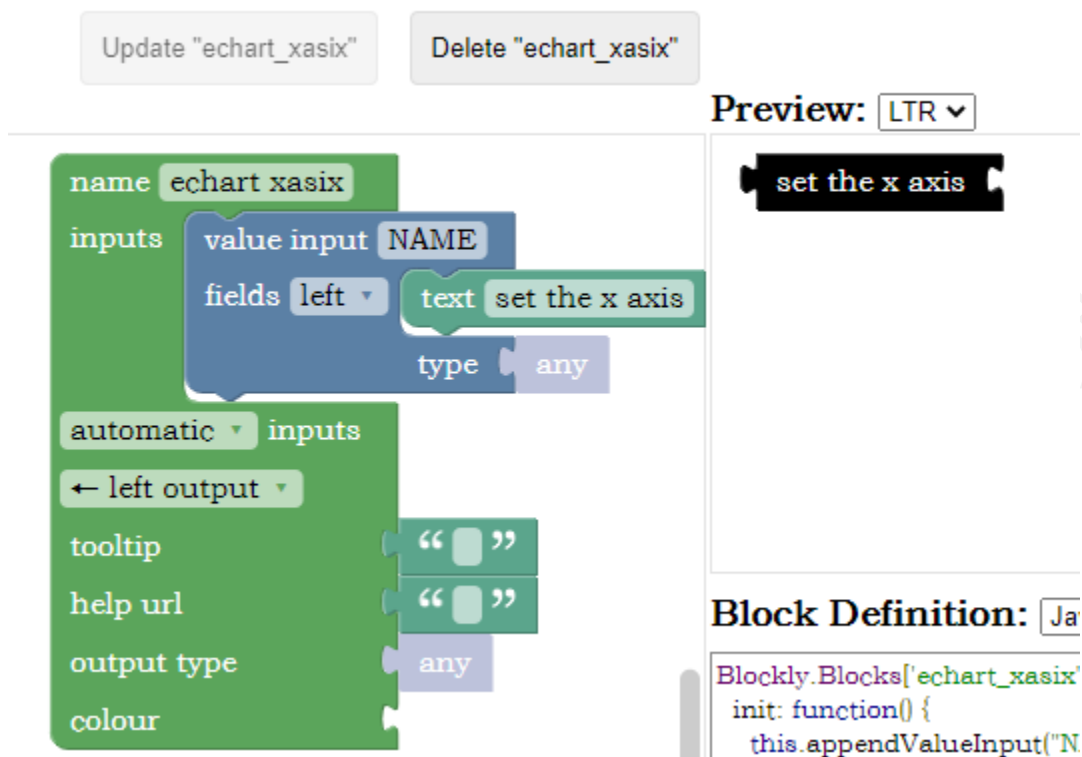
- name:** echart tooltip
- inputs:** dummy input (with a 'fields' dropdown set to 'left' and a 'text' input set to 'set tooltip')
- automatic inputs:** left output
- tooltip:** (with a quote icon)
- help url:** (with a quote icon)
- output type:** any
- colour:** (with a color picker icon)

On the right, the 'Preview' section shows a 'set tooltip' block. Below it, the 'Block Definition' section shows the corresponding JavaScript code:

```
Blockly.Blocks['echart_tooltip'] = {
  init: function() {
    this.appendDummyInput
  }
}
```

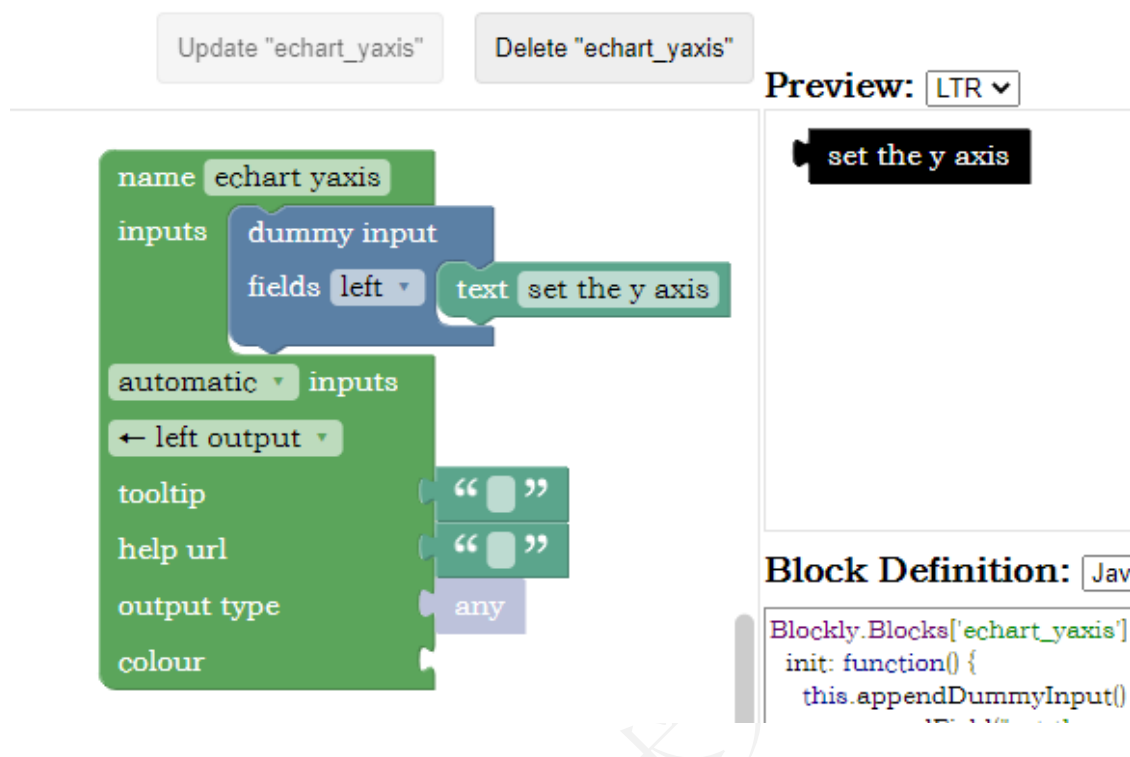
上述代码用于实现动态数据的显示，对应的代码为

```
tooltip: {},
```



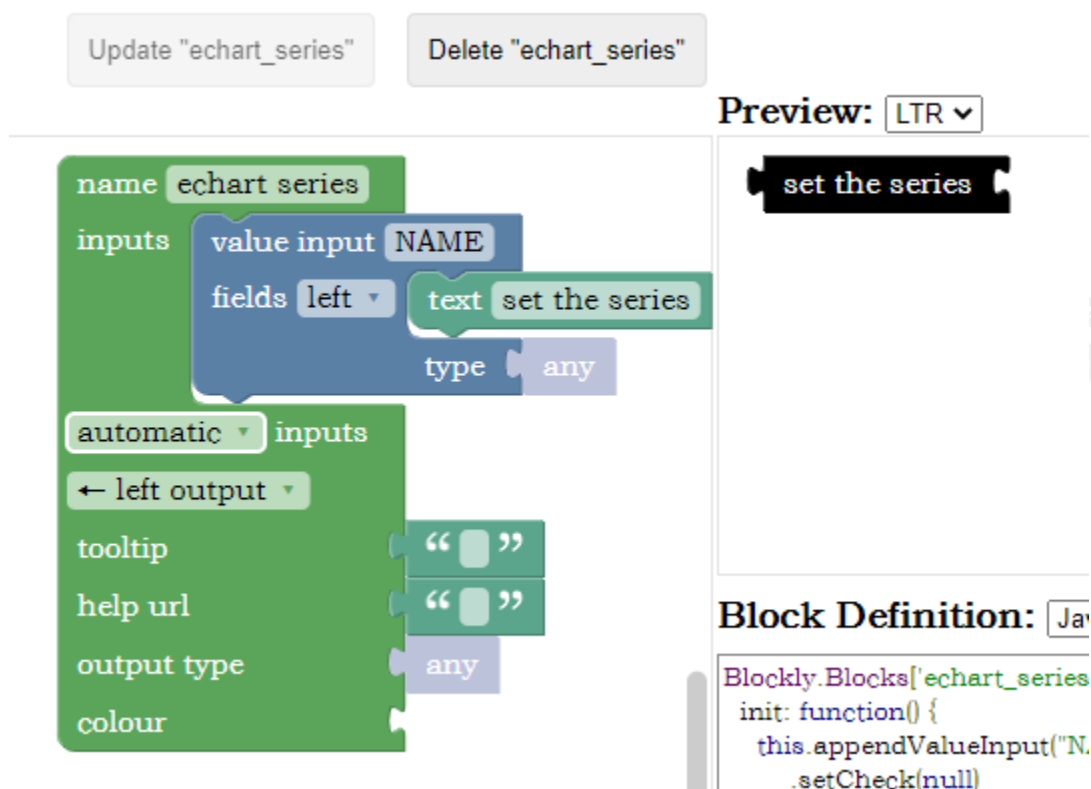
上述代码用于设置图表的 X 轴，对应的代码为

```
xAxis: {
  'data': [" 衬衫", " 羊毛衫", " 雪纺衫", " 裤子", " 高跟鞋", " 袜子"]
},
```



上述代码用于设置图表的 y 轴，对应的代码为

```
yAxis: {},
```



上述 block 是为了设置图表中的数据，对应的代码为

```
series: [{
  'name': '销量',
  'type': 'bar',
  'data': [5, 20, 36, 10, 10, 20]
}]
```

Update "echart_options"
Delete "echart_options"

name echart options

inputs

- dummy input
 - fields left text set options of echart
- value input title
 - fields right text title
 - type any
- value input legend
 - fields right text legend
 - type any
- value input tooltip
 - fields right text tooltip
 - type any
- value input xaxis
 - fields right text xaxis
 - type any
- value input yaxis
 - fields right text yaxis
 - type any
- value input series
 - fields right text series
 - type any

automatic inputs

← left output

Preview: LTR ▼

set options of echart
 title
 legend
 tooltip
 xaxis
 yaxis
 series

Block Definition: JavaScript

```
Blockly.Blocks['echart_options'] = {
  init: function() {
    this.appendDummyInput()
      .appendField("set options of e
    this.appendValueInput("title")
      .setCheck(null)
      .setAlign(Blockly.ALIGN_RIGHT)
      .appendField("title");
    this.appendValueInput("legend")
      .setCheck(null)
      .setAlign(Blockly.ALIGN_RIGHT)
      .appendField("legend");
  }
};
```

Generator stub: JavaScript

```
Blockly.JavaScript['echart_options'] =
function(block) {
  var value_title = Blockly.JavaScript.valueToCode(block, 'title',
    Blockly.JavaScript.ORDER_NONE);
  var value_legend = Blockly.JavaScript.valueToCode(block, 'legend',
    Blockly.JavaScript.ORDER_NONE);
  var value_tooltip = Blockly.JavaScript.valueToCode(block, 'tooltip',
    Blockly.JavaScript.ORDER_NONE);
  var value_xaxis = Blockly.JavaScript.valueToCode(block, 'xaxis',
    Blockly.JavaScript.ORDER_NONE);
  var value_yaxis = Blockly.JavaScript.valueToCode(block, 'yaxis',
    Blockly.JavaScript.ORDER_NONE);
  var value_series = Blockly.JavaScript.valueToCode(block, 'series',
    Blockly.JavaScript.ORDER_NONE);
  // TODO: Assemble JavaScript into code
  var code = '';
  // TODO: Change ORDER_NONE to the correct value
  return [code, Blockly.JavaScript.ORDER_NONE];
};
```

上述 block 将所有的设置收集起来，对应的代码为

```
// 指定图表的配置项和数据
var option = {
  title: {
    'text': 'ECharts 入门示例'
```



```

    },
    legend: {
        'data': ['销量']
    },
    tooltip: {},
    xAxis: {
        'data': [" 衬衫", " 羊毛衫", " 雪纺衫", " 裤子", " 高跟鞋", " 袜子"]
    },
    yAxis: {},
    series: [{
        'name': '销量',
        'type': 'bar',
        'data': [5, 20, 36, 10, 10, 20]
    }]
};

```

The image shows the Blockly IDE interface. On the left, a green block named 'echart_setoptions' is shown with its inputs and outputs. It has two 'dummy input' fields: one on the left labeled 'final step steoptions' and one on the right labeled 'options'. The right field has a 'type' dropdown set to 'any'. Below these are 'automatic' inputs for 'tooltip', 'help url', 'top type', and 'colour'. On the right, a 'Block Definition' panel shows the JavaScript code for this block:

```

Blockly.Blocks['echart_setoptions']
init: function() {
    this.appendDummyInput()
        .appendField("final step steop
    this.appendValueInput("NAME"
        .setCheck(null)
        .setAlign(Blockly.ALIGN_RIGI
        .appendField("options");

```

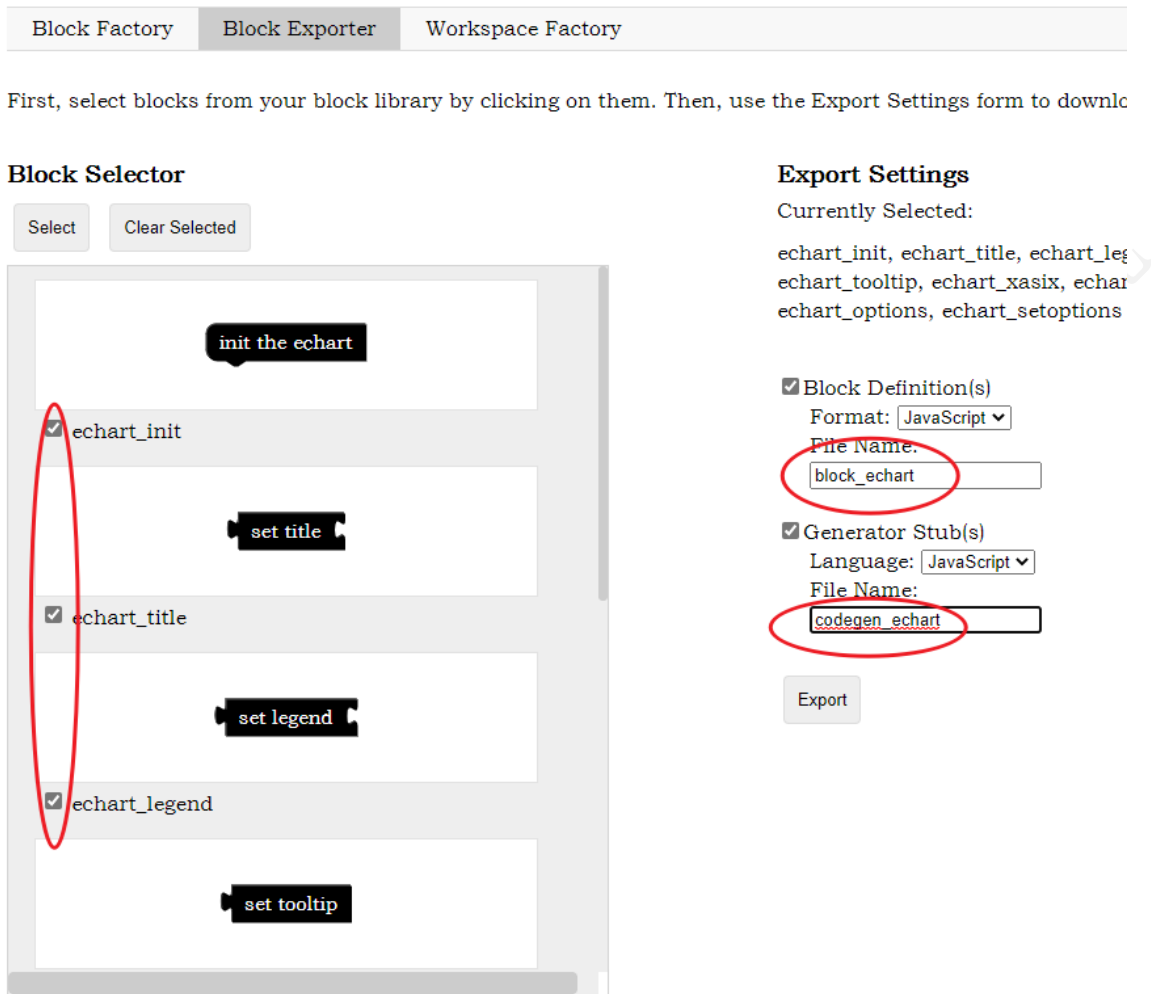
上述 block 是为了调用 `setOption` 函数，依据用户配置的选项开始画图，对应的代码为

```

// 使用刚指定的配置项和数据显示图表。
myChart.setOption(option);

```

12.3 导出设计



12.4 代码生成

代码生成其实就是字符串拼接，与前文描述的例子没什么差别，在此不再赘述。

```
Blockly.JavaScript['echart_init'] = function(block) {
  // TODO: Assemble JavaScript into code variable.
  var code = 'myechart = echarts.init(document.getElementById(\'myechart\'));\n';
  // TODO: Change ORDER_NONE to the correct strength.
  return code;
};

Blockly.JavaScript['echart_title'] = function(block) {
  var value_name = Blockly.JavaScript.valueToCode(block, 'NAME', Blockly.JavaScript.ORDER_
```

```

    // TODO: Assemble JavaScript into code variable.
    var code = value_name;
    // TODO: Change ORDER_NONE to the correct strength.
    return [code, Blockly.JavaScript.ORDER_ATOMIC];
};

```

```

Blockly.JavaScript['echart_legend'] = function(block) {
    var value_name = Blockly.JavaScript.valueToCode(block, 'NAME', Blockly.JavaScript.ORDER_
    // TODO: Assemble JavaScript into code variable.
    var code = value_name;
    // TODO: Change ORDER_NONE to the correct strength.
    return [code, Blockly.JavaScript.ORDER_ATOMIC];
};

```

```

Blockly.JavaScript['echart_tooltip'] = function(block) {
    // TODO: Assemble JavaScript into code variable.
    var code = 'tooltip: {}';
    // TODO: Change ORDER_NONE to the correct strength.
    return [code, Blockly.JavaScript.ORDER_ATOMIC];
};

```

```

Blockly.JavaScript['echart_series'] = function(block) {
    var value_name = Blockly.JavaScript.valueToCode(block, 'NAME', Blockly.JavaScript.ORDER_
    // TODO: Assemble JavaScript into code variable.
    var code = value_name;
    // TODO: Change ORDER_NONE to the correct strength.
    return [code, Blockly.JavaScript.ORDER_ATOMIC];
};

```

```

Blockly.JavaScript['echart_options'] = function(block) {
    var value_title = Blockly.JavaScript.valueToCode(block, 'title', Blockly.JavaScript.ORDER_
    var value_legend = Blockly.JavaScript.valueToCode(block, 'legend', Blockly.JavaScript.ORI
    var value_tooltip = Blockly.JavaScript.valueToCode(block, 'tooltip', Blockly.JavaScript.O
    var value_xaxis = Blockly.JavaScript.valueToCode(block, 'xaxis', Blockly.JavaScript.ORDER_
    var value_yaxis = Blockly.JavaScript.valueToCode(block, 'yaxis', Blockly.JavaScript.ORDER_
    var value_series = Blockly.JavaScript.valueToCode(block, 'series', Blockly.JavaScript.ORI
    // TODO: Assemble JavaScript into code variable.

```

```

var code = '{ ' ;
code += value_title + ',\n ' ;
code += value_legend + ',\n ' ;
code += value_tooltip + ',\n ' ;
code += value_xaxis + ',\n ' ;
code += value_yaxis + ',\n ' ;
code += value_series + '\n';
code += '}' ;

// TODO: Change ORDER_NONE to the correct strength.
return [code, Blockly.JavaScript.ORDER_ATOMIC];
};

Blockly.JavaScript['echart_setoptions'] = function(block) {
  var value_name = Blockly.JavaScript.valueToCode(block, 'NAME', Blockly.JavaScript.ORDER_
  // TODO: Assemble JavaScript into code variable.
  var code = 'myechart.setOption(' + value_name + ')\n';
  return code;
};

Blockly.JavaScript['echart_xaxis'] = function(block) {
  var value_name = Blockly.JavaScript.valueToCode(block, 'NAME', Blockly.JavaScript.ORDER_
  // TODO: Assemble JavaScript into code variable.
  var code = value_name;
  // TODO: Change ORDER_NONE to the correct strength.
  return [code, Blockly.JavaScript.ORDER_ATOMIC];
};

Blockly.JavaScript['echart_yaxis'] = function(block) {
  // TODO: Assemble JavaScript into code variable.
  var code = 'yAxis: {}';
  // TODO: Change ORDER_NONE to the correct strength.
  return [code, Blockly.JavaScript.ORDER_ATOMIC];
};

```

12.5 HTML 集成

- 在 `<head></head>` 标签内插入 echart 依赖库

```
<script src="https://cdn.jsdelivr.net/npm/echarts@4.8.0/dist/echarts.min.js"></script>
```

- 在 `<head></head>` 标签内插入 echart 的 block 和代码生成器

```
<script src = "custom/block_echart.js"></script>
<script src = "custom/codegen_echart.js"></script>
```

- 在 `<body></body>` 标签内插入一个画布，用于放置 echart 输出的图表，注意：此处设置的 id 必须与代码生成器初始化模块中的 id 保持一致

```
<div id="myechart" style="width: 600px; height: 200px;"></div>
```

- 配置 Blockly 的工具箱

```
<category name="ECHART" colour="#0000FF">
  <block type="echart_init"> </block>
  <block type="echart_title"> </block>
  <block type="echart_legend"> </block>
  <block type="echart_tooltip"> </block>
  <block type="echart_xaxis"> </block>
  <block type="echart_yaxis"> </block>
  <block type="echart_series"> </block>
  <block type="echart_options"> </block>
  <block type="echart_setoptions"> </block>
  <block type="echart_title"> </block>
</category>
```

12.6 最终效果

