生成并运行JavaScript

块状应用程序通常生成JavaScript作为其输出语言，通常在网页（可能是同一网页或嵌入式WebView）中运行。 像任何生成器一样，第一步是导入javascript生成器。

对于Web Blockly，请在blockly_compressed.js之后添加javascript_compressed.js：

```
<script src="blockly_compressed.js"></script>
<script src="javascript_compressed.js"></script>
```

要从工作空间生成JavaScript代码，请调用：

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

默认情况下，Blockly for Android会生成JavaScript，并且不需要配置。 有关更多详细信息，请参见使用自定义JavaScript生成器。

生成的代码可以直接在目标网页上执行：

```
try {
  eval(code);
} catch (e) {
  alert(e);
}
```

基本上，上面的代码片段只是生成代码并对其进行评估。 但是，有一些改进。 一种改进是，将评估结果包装在try / catch中，以便可以看到所有运行时错误，而不是悄无声息地失败。 另一个改进是，将代码添加到保留字列表中，这样，如果用户的代码包含该名称的变量，它将被自动重命名而不是冲突。 任何局部变量都应以这种方式保留。

突出显示块（仅Web）highlightBlock

在同一页面中运行代码的Web应用程序通常会在代码运行时突出显示当前正在执行的块。 可以在生成JavaScript代码之前通过设置STATEMENT_PREFIX在逐条语句级别上完成此操作：

```
Blockly.JavaScript.STATEMENT_PREFIX = 'highlightBlock(%1);\n';
Blockly.JavaScript.addReservedWords('highlightBlock');
```

定义highlightBlock以标记工作空间上的块。

```
function highlightBlock(id) {
  workspace.highlightBlock(id);
}
```

这将导致语句 HighlightBlock（'123'）; 被添加到每个语句之前，其中123是要突出显示的块的序列号。

无限循环

尽管可以保证生成的代码在语法上始终正确，但是它可能包含无限循环。 由于解决暂停问题超出了Blockly的范围（！），处理这些情况的最佳方法是维护一个计数器，并在每次执行迭代时将其递减。 为此，只需将Blockly.JavaScript.INFINITE_LOOP_TRAP设置为代码片段，即可将其插入每个循环和每个函数中。 这是一个例子：

```
window.LoopTrap = 1000;
Blockly.JavaScript.INFINITE_LOOP_TRAP = 'if(--window.LoopTrap == 0) throw
"Infinite loop.";\n';
var code = Blockly.JavaScript.workspaceToCode(workspace);
```

JS转换

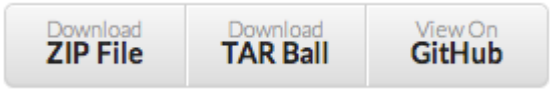
如果您认真考虑正确运行用户块，那么可以使用JS Interpreter项目。 该项目与Blockly分开，但是专门为Blockly编写。

- 以任何速度执行代码。
- 暂停/继续/逐步执行。
- 在执行时突出显示块。
- 与浏览器的JS完全隔离。

这是使用Blockly和JS Interpreter生成和执行JavaScript的现场演示:https://blockly-demo.appspot.com/static/demos/generator/index.html。

运行

首先,从GitHub上下载 JS Interpreter:



然后添加到你的页面:

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

调用它的最简单方法是生成JavaScript，创建解释器并运行代码：

```
var code = Blockly.JavaScript.workspaceToCode(workspace);
var myInterpreter = new Interpreter(code);
myInterpreter.run();
```

转换步骤

为了更慢地执行代码或以更可控的方式执行代码，请使用逐步循环（在这种情况下，每10毫秒执行一次）的循环替换调用以运行：

```
function nextStep() {
  if (myInterpreter.step()) {
    window.setTimeout(nextStep, 10);
  }
}
nextStep();
```

请注意，每个步骤都不是一行或一个块，而是JavaScript中的一个语义单元，它的粒度可能非常细。

添加一个API

JS Interpreter是一个沙箱，与浏览器完全隔离。 任何对外界执行动作的块都需要将API添加到解释器中。 有关完整说明，请参见JS-Interpreter文档。 但是首先，这是支持警报和提示块所需的API：

```
function initApi(interpreter, scope) {
  // Add an API function for the alert() block.
  var wrapper = function(text) {
    return alert(arguments.length ? text : '');
  };
  interpreter.setProperty(scope, 'alert',
      interpreter.createNativeFunction(wrapper));

  // Add an API function for the prompt() block.
  wrapper = function(text) {
    return prompt(text);
  };
  interpreter.setProperty(scope, 'prompt',
      interpreter.createNativeFunction(wrapper));
}
```

然后修改您的解释器初始化以传递initApi函数：

```
var myInterpreter = new Interpreter(code, initApi);
```

警报和提示块是默认块集中仅有两个块，它们需要解释器的自定义API。

连接

在JS Interpretter中运行时，当用户逐步执行程序时，highlightBlock（）应该立即在沙箱外部执行。 为此，创建一个包装器函数highlightBlock（）来捕获函数参数，并将其注册为本地函数。

```
function initApi(interpreter, scope) {
  // Add an API function for highlighting blocks.
  var wrapper = function(id) {
    return workspace.highlightBlock(id);
  };
  interpreter.setProperty(scope, 'highlightBlock',
      interpreter.createNativeFunction(wrapper));
}
```

更复杂的应用程序可能希望不停地重复执行步骤，直到到达高亮命令，然后再暂停。 此策略模拟逐行执行。 下面的示例使用这种方法。

JS Interpreter例子

这是逐步解释JavaScript的现场演示。 这个演示包括一个等待块，这是一个用于其他异步行为（例如，语音或音频，用户输入）的好例子。