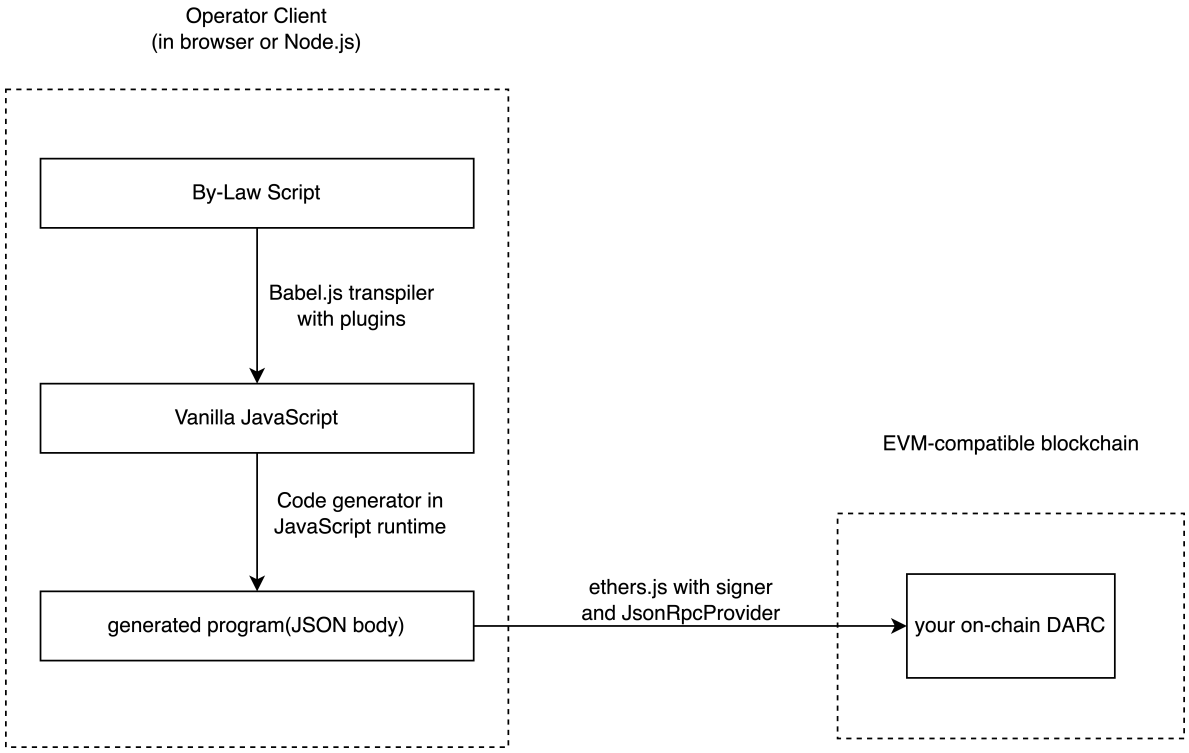

图 1: 编译和执行法规脚本的过程

1 法规脚本

法规脚本是一种专为 DARC 协议设计的、具有类 JavaScript 语法的编程语言。用户可以使用法规脚本执行多种任务，包括：

- 操作：用户可以使用法规脚本发起一系列操作。这些操作包括铸造和销毁新代币、代币转移、代币购买、启用或禁用插件以及资金提取等行动。
- 插件设计：法规脚本允许用户使用二元表达式树设计插件，结合逻辑运算符和带参数的 DARC 判断表达式。这些插件可以定义一个返回类型，并在必要时指定一个投票规则。
- 投票：法规脚本简化了当前操作的投票过程。

法规脚本的语法与原生 JavaScript 语法非常相似，包括变量、常量、赋值、基本数据类型、函数、类和其他各种特性。除了基本的 JavaScript 语法外，法规脚本还支持操作符重载，这对于描述插件的条件节点特别有用。

图 **??** 展示了法规脚本在 DARC 协议中的编译和执行过程。在操作者客户端，法规脚本的程序需要被转译为原生 JavaScript。随后，它在本地 JavaScript 运行时使用代码生成器执行。在整个程序以 JSON 体的形式生成后，它将成为一个包含操作码和参数列表的操作列表。此时，程序已准备好从客户端作为函数调用提交给 EVM 兼容区块链上的 DARC 应用二进制接口（ABI）。

要在 DARC 协议中执行程序，DARC 软件开发包（SDK）访问 DARC 智能合约的地址，并通过调用入口函数并附上钱包签名启动执行过程。这一通信过程通过 EVM 兼容区块链的 JSON RPC 服务器进行。

1.1 转译器

用户运行法规脚本后，转译过程的第一步是将法规脚本转换为原生 JavaScript。这一初步转换使用 Babel.js [**?**] 进行前端语法分析，并利用操作符重载插件生成结果原生 JavaScript 代码。

使用操作符重载插件的主要意义在于它能够转译用户设计的条件节点中逻辑运算符的语法。用户创建插件来逻辑连接并描述各种条件表达式使用的逻辑运算符。当这些插件使用操作符重载功能进行转译时，插件中的每个条件都被转换为一个对象构造的树状结构，使用纯函数和参数表示。

此外，转译器还支持其他高级 ECMAScript 语法和语法糖，使用户使用起来更为方便。

1.2 代码生成器

一旦用户成功从法规脚本的转译生成了原生 JavaScript，它就可以在 JavaScript 运行时环境中执行，无论是在 Node.js 还是 Web 浏览器中。

用户生成的转译原生 JavaScript 代码包括一个或多个操作命令函数，每个函数都包含操作码及其对应的参数。执行时，每个这样的操作段都存储在一个数组中，最终形成一个程序对象。随后，代码生成器利用这个程序对象创建一个完整的程序 JSON 体，符合 DARC ABI 入口规范。

一旦程序 JSON 体成功生成，用户可以使用 ethers.js 根据 ABI 将此程序体发送至 EVM 兼容区块链上的 DARC 协议。这个过程确保了程序在 DARC 内完整地执行。

图 **??** 展示了一个完整的编译过程。

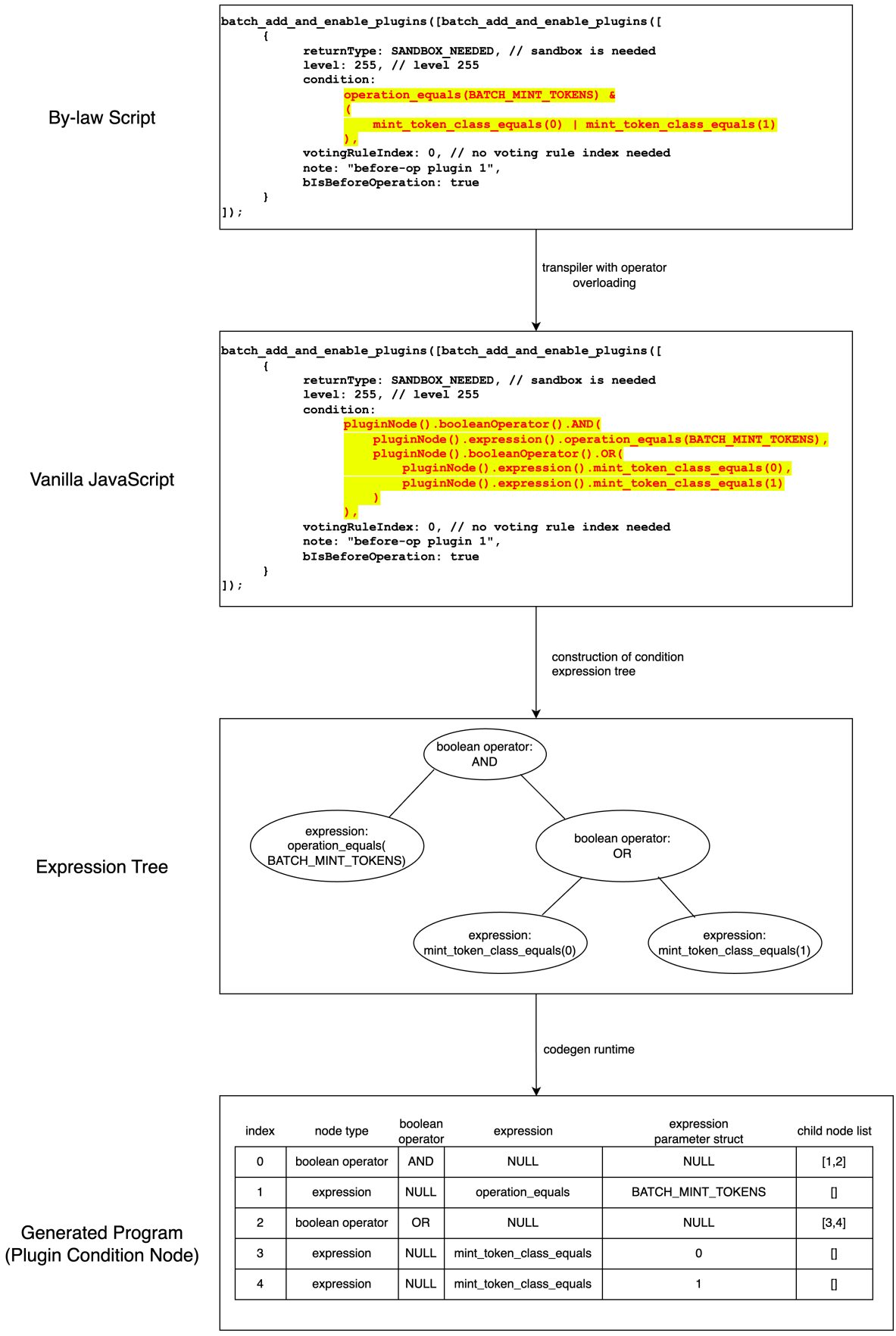

图 2: 插件的转译和代码生成过程