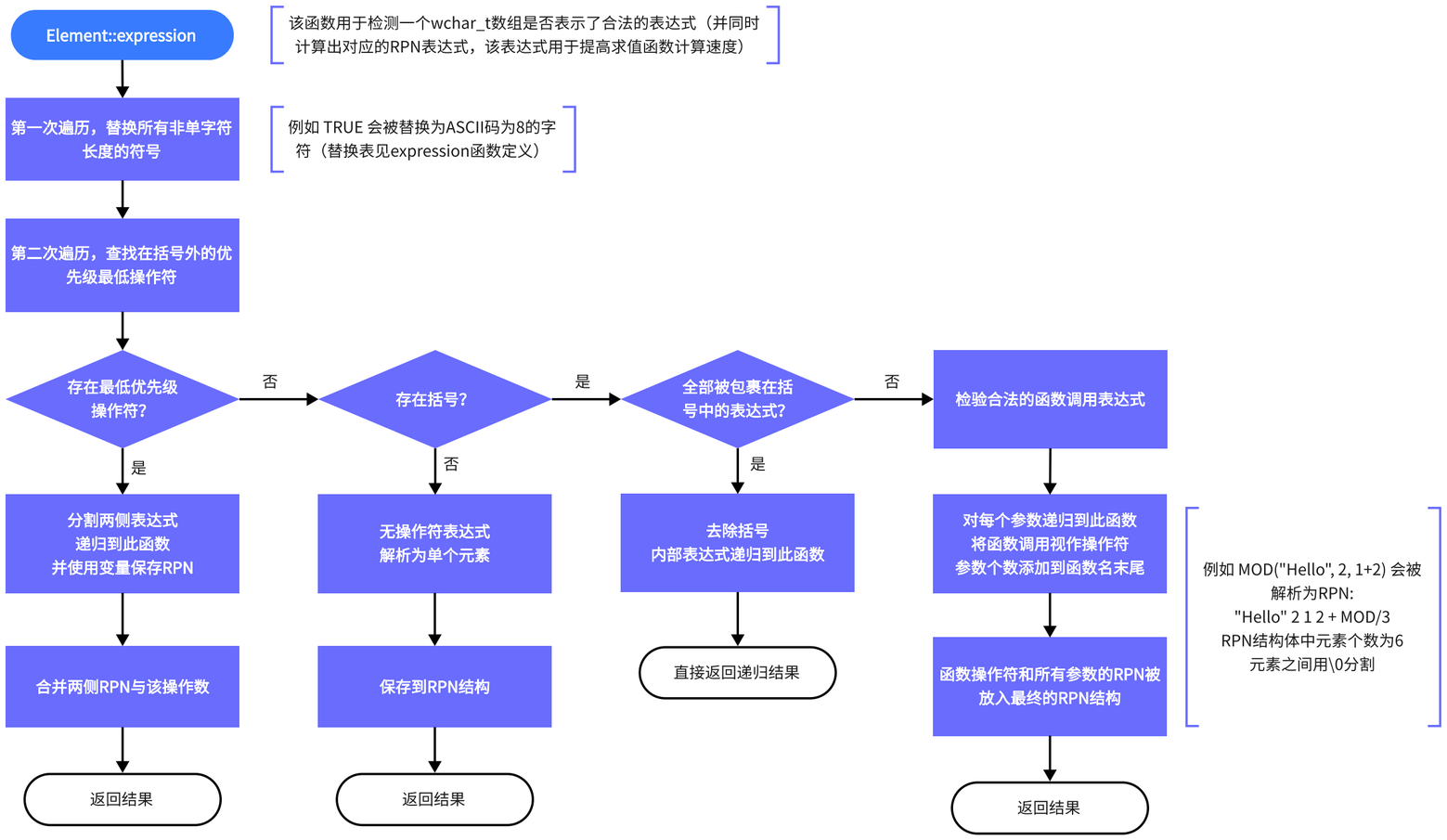
表达式求值过程

Element::expression与evaluate函数分析



Element::expression 依赖函数：

- bool valid_variable_path(wchar_t* expr)
  - 检查 expr 是否为合法的变量路径（例如：L"classes[1].students[2].name"）
  - 变量路径包含求址和解引用操作符
- bool valid_indexes(wchar_t* expr)
  - 检查 expr 是否为合法的下标列表（例如：L"[1, 2, 3, 4, 5]"）
  - 每个下标可以是合法的表达式（即 expression 函数返回 true ）
- bool valid_array_element_access(wchar_t* expr)
  - 分割数组名和下标列表
  - 数组名必须为变量名（非变量路径，因为它被 valid_variable_path 调用，所有expr必为单独的数组访问）
- bool addressing(wchar_t* expr)
  - 判断 expr 是否为求址表达式
  - 由于被求址的只能为变量，所以特此起一个函数来判断（解引用在valid_variable_path 中大放异彩）
- bool valid_operator(wchar_t character, unsigned short* precedence_out = nullptr)
  - 检查 character 是否为操作符（解引用和求址操作符不算）
  - 如果是，则 precedence_out 用于输出优先级等级

evaluate 依赖函数：

- Data* evaluate_variable(wchar_t* path, bool* constant)
  - 获取指定变量路径 path 对应的数据指针 Data*（每个变量在变量域中以节点 Node 存在，变量域用二叉树 BinaryTree 管理，节点保存了指向数据的指针 Data*，数据结构中包含数据类型和数据对象指针，按照数据类型来转化数据对象指针从而访问）
  - path 必须为完整的（即开始变量.路径）
  - constant 用于返回变量是否为常量（仅当参数1为变量而不是变量路径时）
  - 工作原理：找到变量，然后推路径（推路径交给下面）
  - 由于返回的是数据结构的指针，对该指针对应内存的任何变动都会应用到该变量路径上
- Data* evaluate_variable(Data* current, wchar_t* path)
  - 该函数被同名函数的上一个定义调用
  - 该函数用于在上一层路径的基础上下推（上一次路径的基础就是 current，剩余路径是 path，path为不完整路径，即开头为.）
  - 该函数只会推一层，然后递归（例如 .b.c，该函数会推出当前数据的b路径，然后把b路径指向的数据结构交给递归函数推 .c）
  - 直到所有路径推完，递归结束
- Data* array_access(Data* array_data, wchar_t* expr)
  - 该函数同样基于已被推出来的数组
  - expr 为下标列表（例如：L"[1, 2, 3]"）
  - 该函数返回对应下标位置的元素对应的数据结构指针
- Data* addressing(wchar_t* expr)
  - 处理求址表达式
  - 注意：该函数直接生成新的指针，当求址的结果被赋值到一个指针类型的数据时，原指针会被销毁，然后该指针被直接复制到对应位置（销毁过程见DataType::Pointer，没有你想的那么简单）
- Data* evaluate_single_element(wchar_t* expr)
  - 用于对伪代码内置数据类型表达式或变量路径求值（例如：123 会被求值成保存了 123 这个数据对象的 Data 结构）