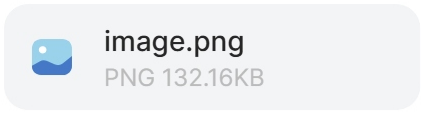

这张图片说明了什么内容?

这张图片展示了一个关于字符串解析的**概念示意图**，具体内容如下：

核心概念

字符串分割：

将一个字符串分成两部分：**左子串（Left substring）**和**右子串（Right substring）**。

竖线 "|" 表示分割点，左侧是已解析的部分，右侧是待解析的部分。

组成部分：

左子串：包含已识别的**终结符（Terminals）**和**非终结符（Non-terminals）**（通常用于语法分析，如编译器设计）。

右子串：尚未被解析的原始字符串内容。

目的：

这种表示法常用于描述**语法解析过程**（如LL/LR分析），通过逐步移动分割点 "|" 来模拟解析的进度。

示例解释

假设字符串是 `A + B * C`，解析过程中的某一步可能是：

左子串：`A + |`（已解析出 `A` 和 `+`）

右子串：`B * C`（待解析部分）

应用场景

编译器设计：用于语法分析阶段，标记已匹配的规则和剩余输入。

形式语言理论：描述推导步骤（如上下文无关文法）。

图片中的数字 "15" 可能是页码或示例编号，无直接关联。

你能跟我说说什么是自底向上的解析吗?

自底向上解析（Bottom-Up Parsing）

自底向上解析是语法分析的一种策略，从输入字符串的**具体符号（终结符）**开始，逐步组合成**更抽象的结构（非终结符）**，最终归约为文法的起始符号。其核心思想是“从叶子到根”构建语法树。

关键特点

归约（Reduce）驱动：

解析器不断将输入的子串与文法规则匹配，替换为对应的非终结符（即“归约”），直到得到起始符号。

例如：将 `a + b` 归约为 `表达式`。

与自顶向下的对比：

自顶向下：从起始符号展开，尝试匹配输入（如递归下降法）。

自底向上：从输入符号反向推导，验证是否符合文法。

常用算法：

LR 解析器（如 LR(0)、SLR(1)、LALR(1)、LR(1)）

优先分析法（如算符优先分析法）。

工作流程

移进（Shift）：将输入符号压入栈中。

归约（Reduce）：当栈顶符号匹配某条文法规则时（如 `A → a b`），将栈顶的 `a b` 弹出，压入 `A`。

接受（Accept）：栈中只剩起始符号，且输入为空时，解析成功。

错误处理：若无法移进或归约，则报错。

示例

文法规则：

`E → E + T | T`

`T → T * F | F`

`F → (E) | id`

解析输入 `id + id`：

移进 `id` → 归约为 `F` → 归约为 `T` → 归约为 `E`。

移进 `+`，移进 `id` → 归约为 `F` → 归约为 `T`。

栈顶 `E + T` 归约为 `E`（起始符号），完成解析。

优缺点

优点：

能处理更复杂的文法（如左递归，自顶向下无法直接处理）。

效率高（如 LR 解析器是线性时间复杂度）。

缺点：

算法复杂（需预先生成解析表）。

调试困难（错误恢复不如自顶向下直观）。

典型应用

编译器（如 Yacc/Bison 生成的解析器）。

解释器（处理表达式、语句结构）。

数据查询语言（如 SQL 解析）。

如果需要更具体的例子或算法细节，可以进一步探讨！