**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** | **编译技术** |
| **学生姓名：** | **葛子午** |
| **学生学号：** | **202330550381** |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2024-2025第二学期** |

**软件学院**

**2025年05月**

# 正则表达式的实现（RE to NFA）

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **地 点：** | | | B7楼 | | 133房； | | **实验台号：** | |  | |
| **实验日期与时间：** | | | 2025.5.10 | | | | **评 分：** | |  | |
| **预习检查纪录：** | | |  | | | | **实验教师：** | |  | |
| **电子文档存放位置：** | | |  | | | | | | | |
| **电子文档文件名：** | | |  | | | | | | | |
| **批改意见：** | | |  | | | | | | | |
| 出勤情况（20%） | 课堂态度（20%） | | 实验报告质量（60%） | | | | | |
| 任务完成情况 | | 源代码注释 | | 报告内容 | |
| □ 出勤  □ 缺勤  □ 早退  □ 请假 | □ 认真  □ 不认真 | | □ 完成  □ 未完成 | | □ 优秀  □ 良好  □ 通过  □ 未通过 | | □ 优秀  □ 良好  □ 通过  □ 未通过 | |

# 报告内容

1、实验目标

1. 实现一个将正则表达式转换为非确定性有限自动机(NFA)的软件工具
2. 掌握Thompson构造法的基本原理和实现方法
3. 巩固正则文法、有穷自动机相关理论知识
4. 理解正则表达式操作符的优先级和结合性
5. 学习构建正则表达式树并递归转换为NFA的过程

2、实验环境

**操作系统：Windows 11**

**开发工具：**

* **Java开发环境：JDK 17**
* **集成开发环境：IntelliJ IDEA**

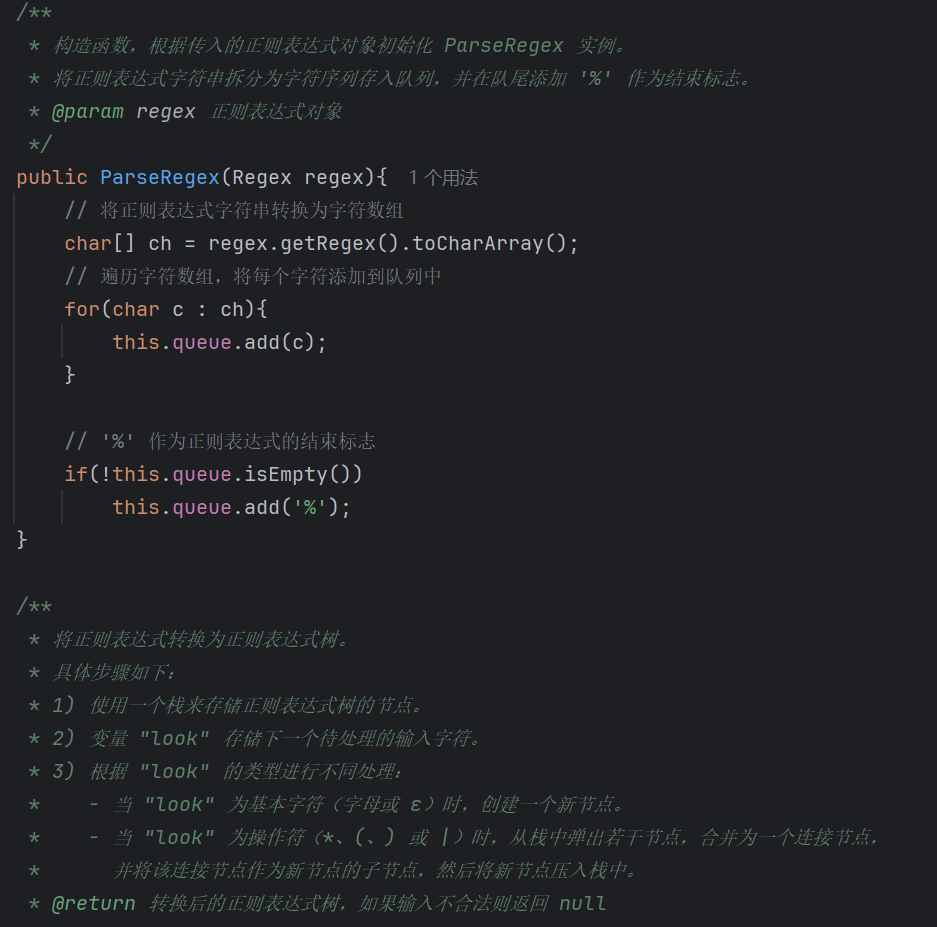
3、关键数据结构和核心算法

3.1 数据结构

3.1.1 正则表达式树

RE经过ParseRegex的parse函数后，返回一棵正则表达式树，将这棵树的根节点传给ThompsonConstruction 的translate函数后，得到NFA

3.1.1.1 ParseRegex类



3.1.1.2 正则表达式树结点（RegexTreeNode）

① 作为正则表达式语法树的基础单元，支持Thompson构造法所需的树形结构表示

② 封装正则表达式元素的类型和值信息，实现以下操作：

基础字符（字母/ε）

连接（Concatenation）

并集（Union/|）

Kleene闭包（\*）

括号优先级处理

**核心数据结构​​**

| 属性 | 类型 | 说明 |
| --- | --- | --- |
| value | Character | 存储节点对应的字符（运算符或操作数） |
| type | int | 节点类型标识： 0-字符, 1-连接, 2-并集, 3-Kleene闭包, 4-左括号, 5-右括号 |
| firstChild | RegexTreeNode | 长子节点指针（树形结构基础） |
| nextSibling | RegexTreeNode | 右兄弟节点指针（实现多子节点平级存储） |

**关键方法**

| 方法 | 功能 | 算法复杂度 |
| --- | --- | --- |
| getLastChild() | 获取当前节点的最后一个子节点 | O(n)（需遍历兄弟链） |
| toString() | 生成"值:类型"格式的节点描述 | O(1) |
| 构造方法 | 支持三种初始化方式： - 基础值+类型 - 带子节点和兄弟节点 - 序列化兼容 | O(1) |

3.1.1.3 正则树（RegexTree）

① 正则表达式语法树的容器类，基于树形结构存储解析后的正则表达式

② 提供树遍历和字符串表示功能，支持后续的NFA转换算法

**核心数据结构​​**

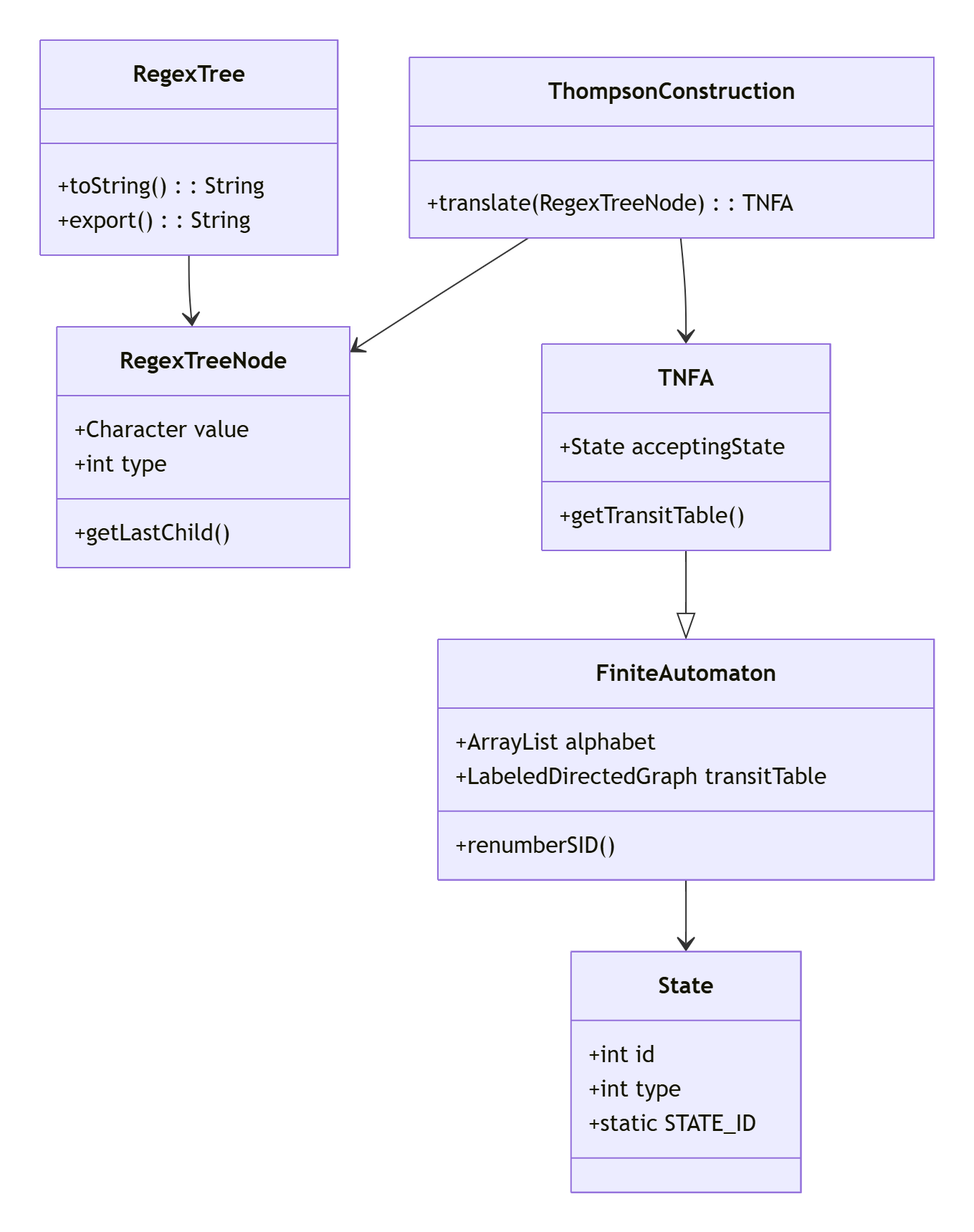
|  |  |  |
| --- | --- | --- |
| root | RegexTreeNode | 继承自DefaultTree的根节点 |
| ArrayDeque<RegexTreeNode> | 队列 | 用于层序遍历的临时存储 |

**关键方法**

|  |  |  |  |
| --- | --- | --- | --- |
| toString() | 生成树的结构化字符串表示 | ​​层序遍历（BFS）​​ 1. 使用队列逐层处理节点 2. 按父子关系缩进格式化 | O(n) |

3.1.2 NFA状态机

3.1.2.1 核心架构



3.1.2.2 数据结构一览

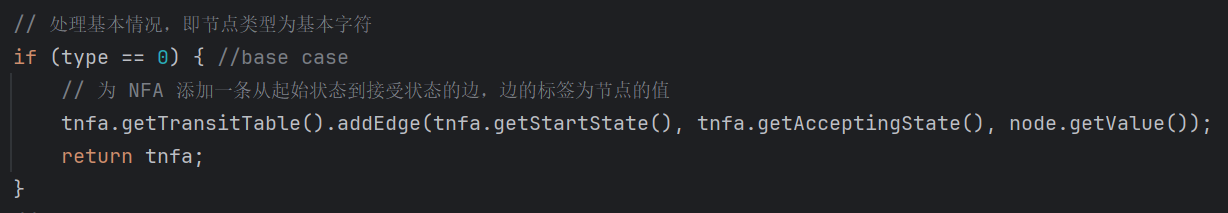
| **类名** | **父类/接口** | **核心字段** | **重要方法** | **设计特点** |
| --- | --- | --- | --- | --- |
| ​****​State​****​ | Serializable | final int id String sid int type | setType() equals() | 1. 全局自增ID 2. 分离逻辑ID与显示ID 3. 状态类型常量 |
| ​****​FiniteAutomaton​****​ | Serializable | ArrayList<Character> alphabet LabeledDirectedGraph<State> transitTable State startState | renumberSID() getTransitTable() | 1. 基于有向图的转移表 2. 广度优先重编号 |
| ​****​TNFA​****​ | FiniteAutomaton | State acceptingState | getAcceptingState() setAcceptingState() | 1. 强制单接受状态 2. 合并转移表方法 |

3. 2 核心算法

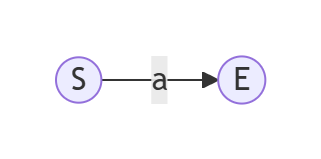
**Thompson构造算法**​

1. 基础字符处理

功能：将单个字符转换为最简单的NFA结构  
代码段：



状态机结构：



关键点：

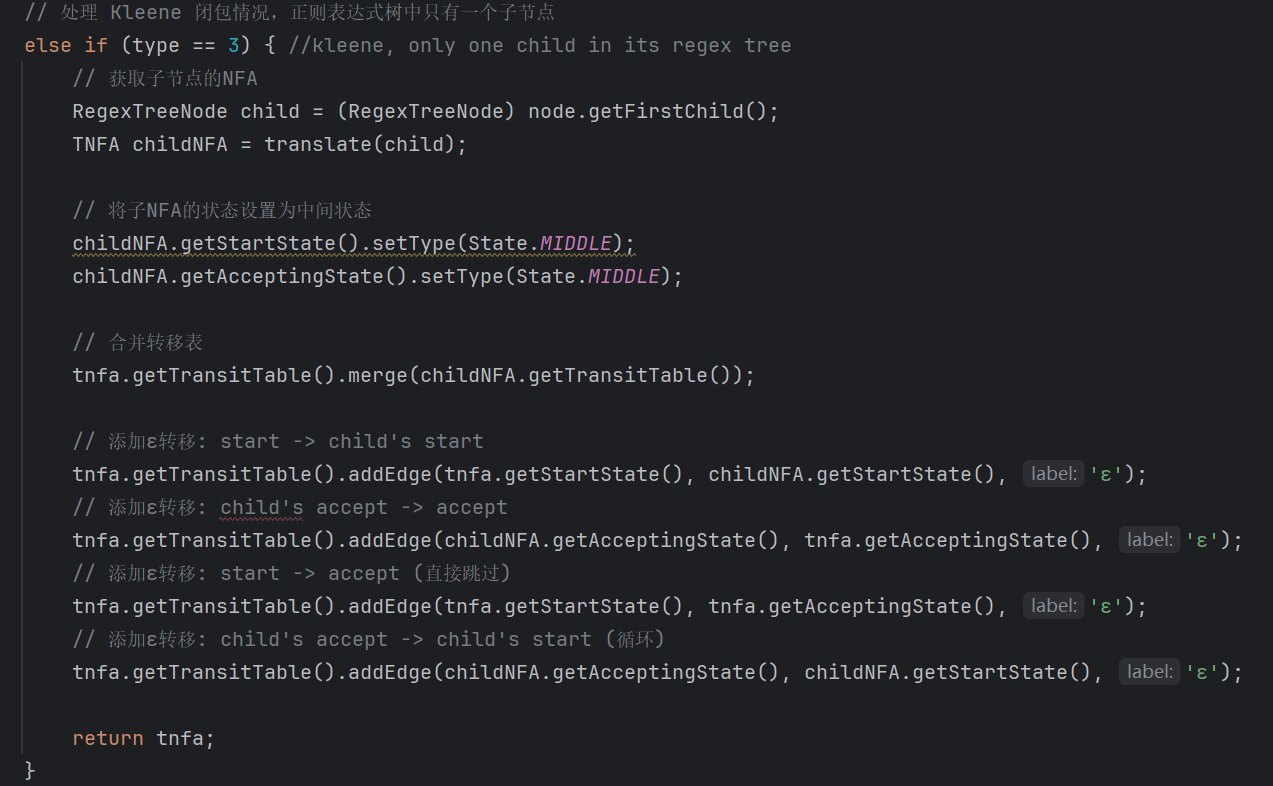
直接创建两状态NFA（起始+接受）

转移边标签为字符本身（如'a'）

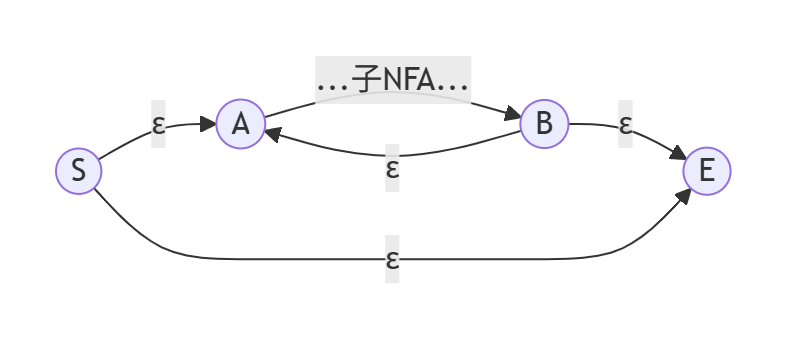
时间复杂度：O(1)

1. Kleene闭包处理

功能：处理R\*形式的正则表达式  
代码段：



状态机结构：



关键点：

ε环：允许跳过子表达式（S→E）

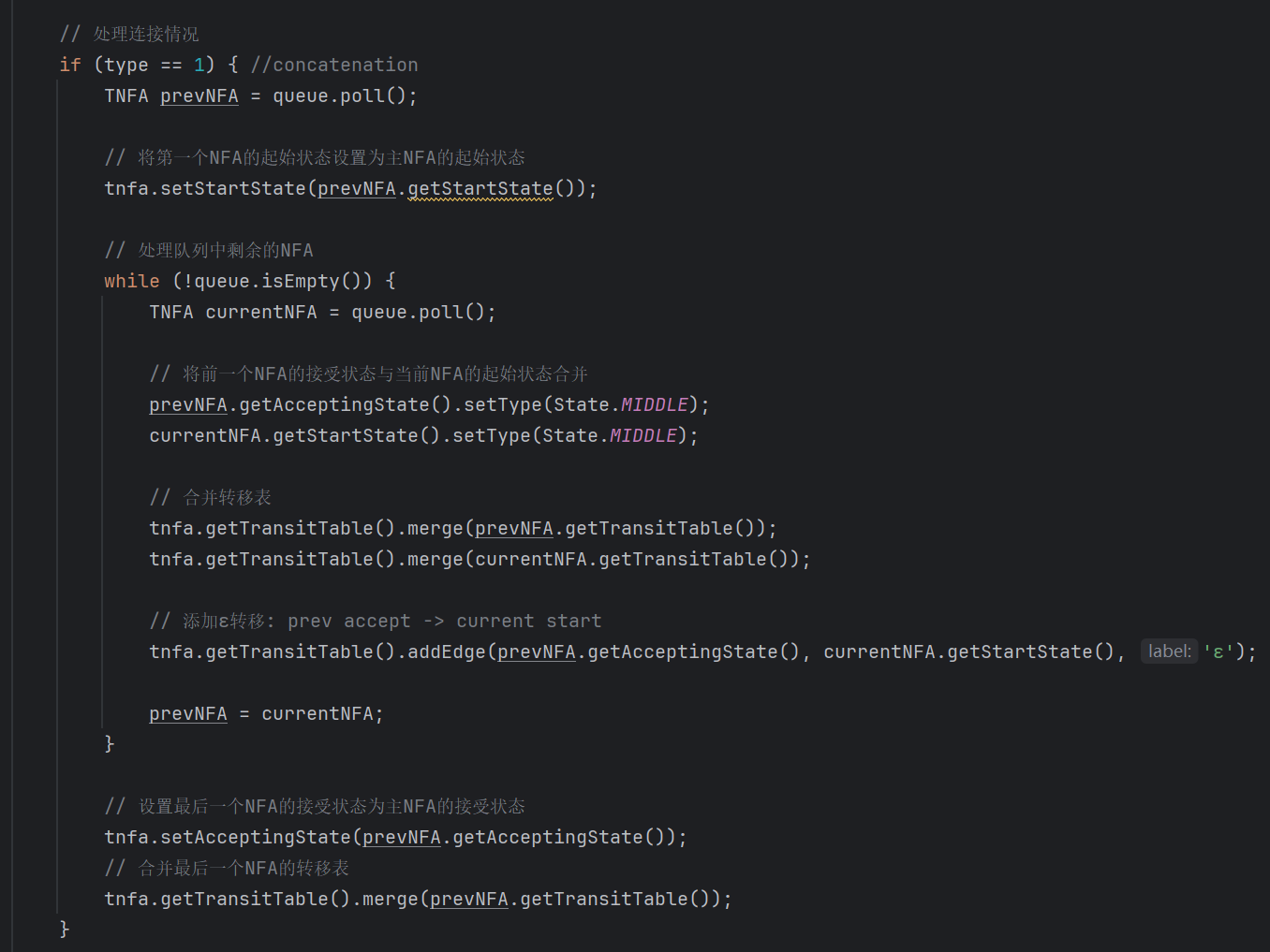
循环路径：通过B→A实现重复匹配

子NFA状态必须降级为MIDDLE（避免多接受状态）

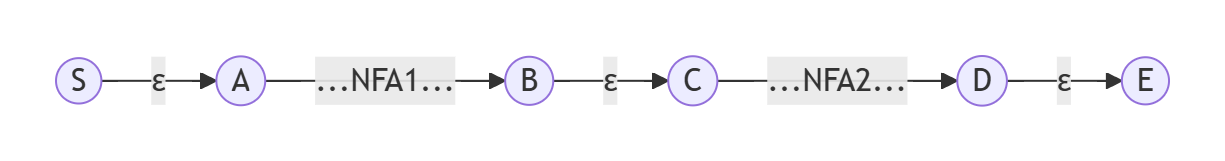
1. 连接运算处理

功能：处理连续的表达式连接  
代码段：





状态机结构：



关键点：

队列管理：按顺序连接所有子NFA

状态合并：前一个NFA的接受状态连接后一个NFA的起始状态

时间复杂度：O(n)（n为连接的操作数）

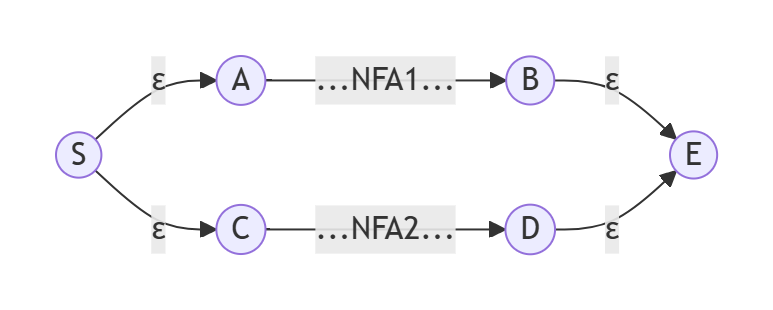
1. 并集运算处理

功能：处理选择型表达式  
代码段：





状态机结构：



关键点：

并行路径：通过S→A和S→B实现选择

最终汇聚：所有子NFA接受状态连接到同一终点

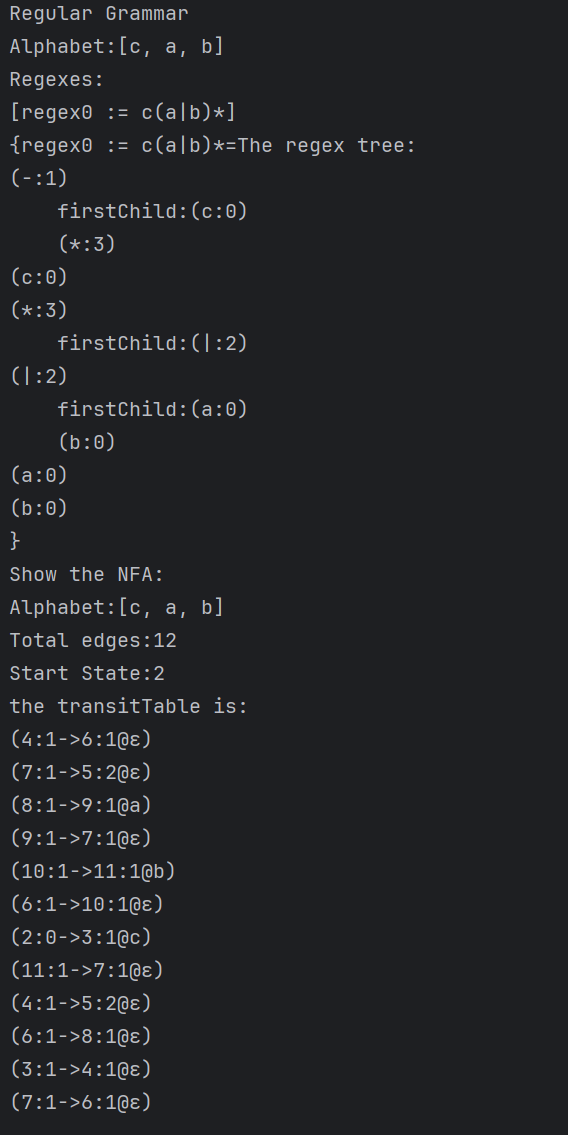
需要保证子NFA无状态冲突（通过全局ID管理）

3. 3 错误处理

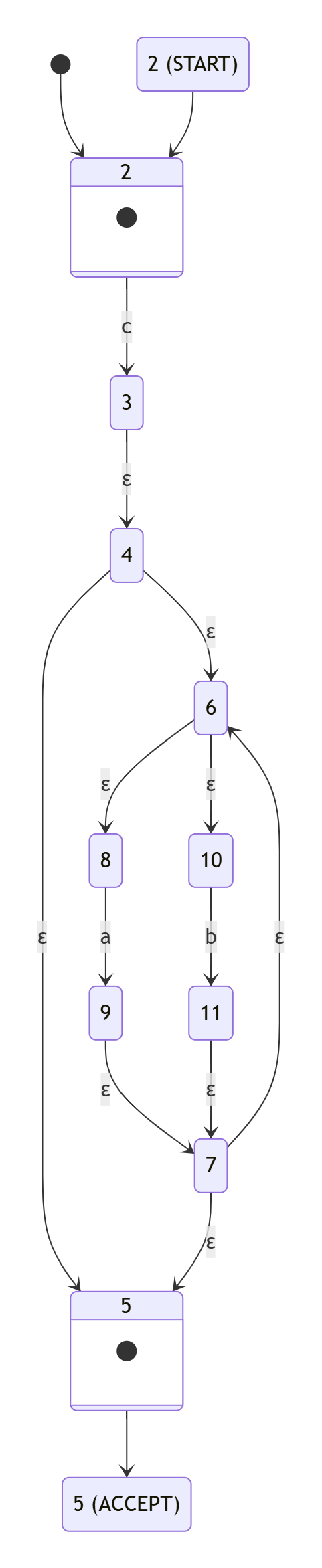
| 错误类型 | 检测方法 | 处理策略 |
| --- | --- | --- |
| 非法运算符 | 词法分析阶段 | 抛出RegexSyntaxException |
| 括号不匹配 | 栈深度检查 | 提示错误位置 |
| 空输入 | length()检测 | 返回INVALID状态 |
| \*前置错误 | 语法树验证 | 标记错误节点 |

3.4 运行结果

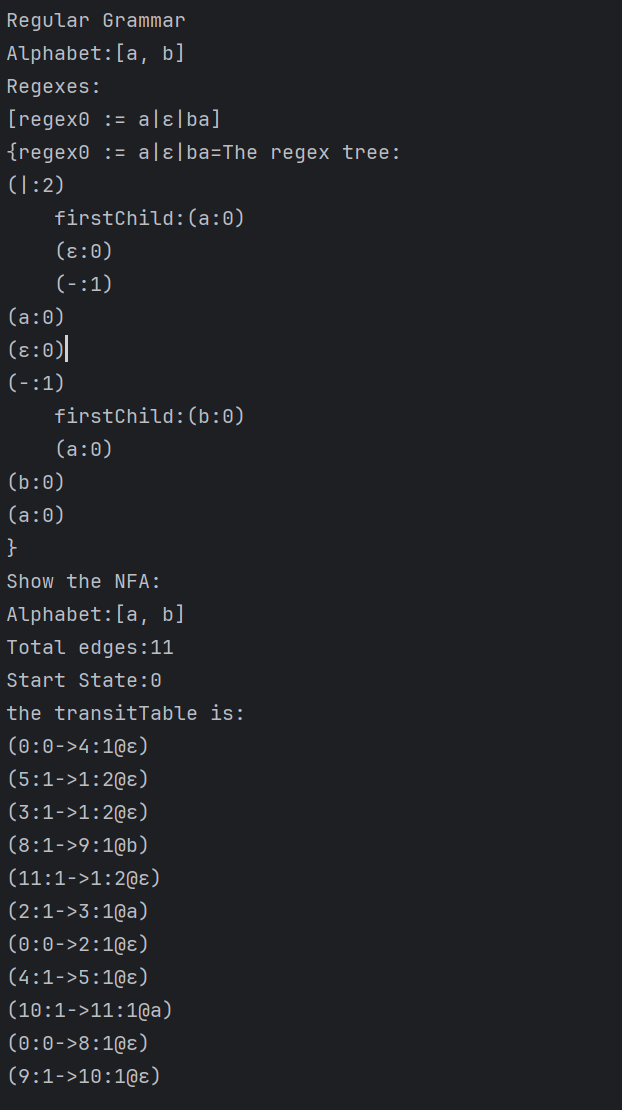
案例1：c(a|b)\*

​

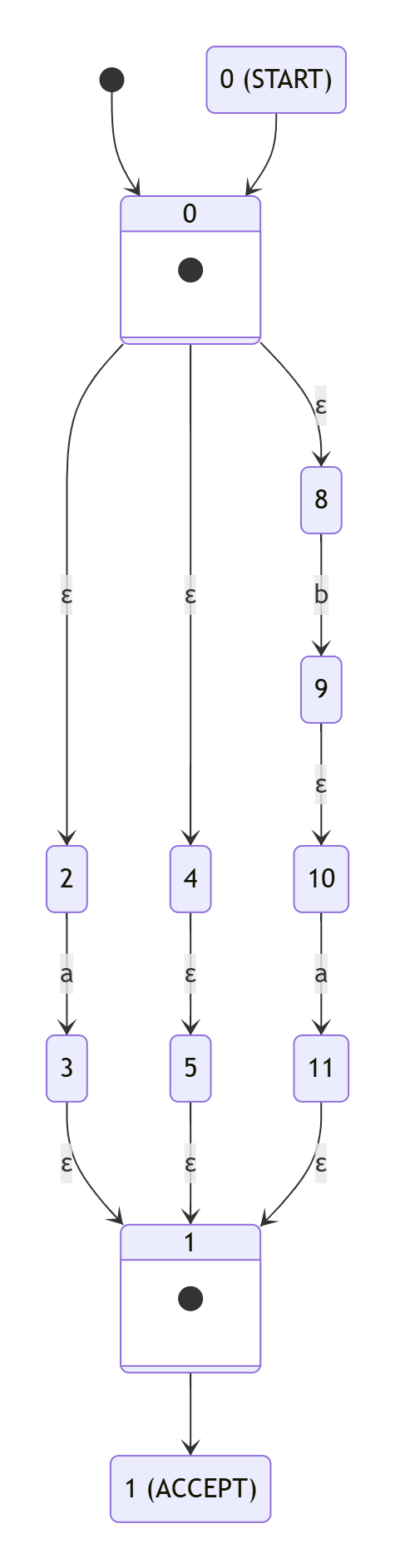
状态转移图分析



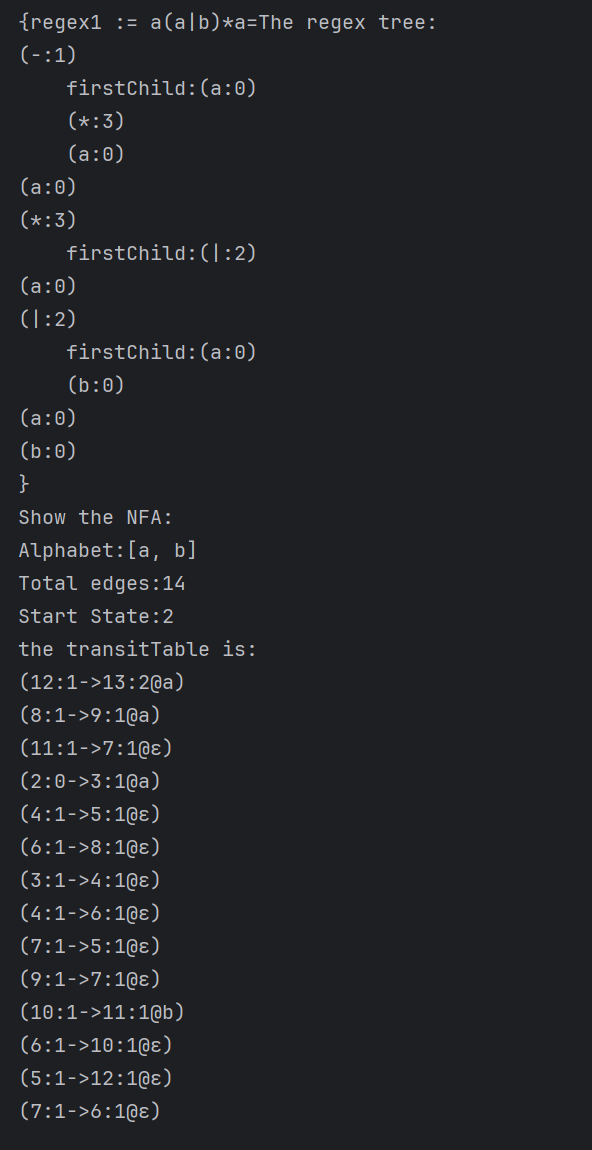
案例2：a|ε|ba



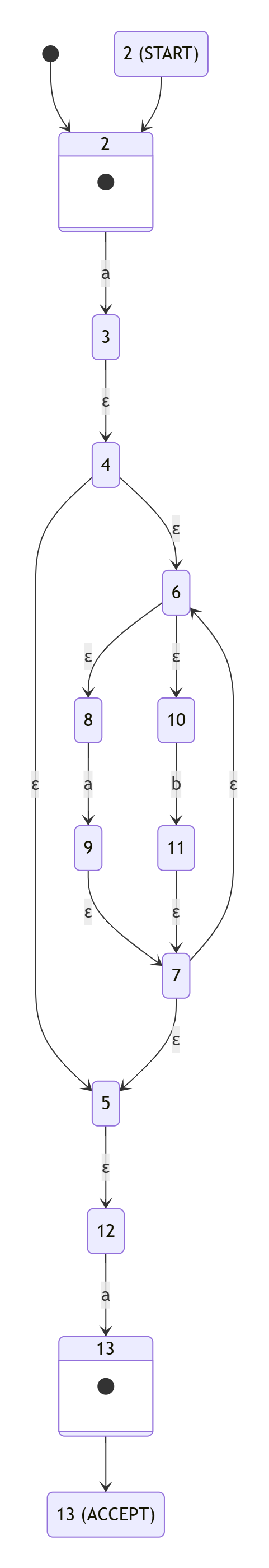
状态转移图



案例3：a(a|b)\*a



状态转移图



1. 总结

通过这次实验，我真正理解了正则表达式和NFA之间的转换原理。Thompson构造法确实很巧妙，把每个正则操作符都对应到NFA的特定结构上，比如看到a(a|b)\*a的NFA时，那个首尾必须匹配a、中间可以循环a或b的设计，跟理论完全吻合。递归处理语法树的方法也很实用，让我对编译器如何处理复杂表达式有了直观认识。

不过实际动手时也遇到不少问题。最头疼的就是状态数增长太快，稍微复杂点的正则表达式就会产生大量状态和转移，看着都晕。ε转移虽然理论简单，但调试起来特别麻烦，经常要反复检查状态间的连接是否正确。还有一个潜在风险是递归深度问题，如果正则表达式嵌套太深，可能会引发栈溢出。

下一步打算尝试把NFA转成DFA来简化状态，同时研究下怎么优化递归算法。如果能实现自动可视化可能更好，毕竟看文字描述的状态转移远没有图形直观。总的来说，这次实验收获很大，验证了理论知识