网站设计文档

# 用例描述

## 状态显示

|  |  |
| --- | --- |
| 用例名称 | 显示边的数目 |
| 概述 | 在面板中显示自动机边的数目 |
| 参与者 | 读者 |
| 前置条件 | 无 |
| 后置条件 | 无 |
| 相关回调事件 | 无 |
| 基本事件流 | 1. （如果有）点击对应的属性面板 2. 在面板中查看当前自动机边的数目 |

|  |  |
| --- | --- |
| 用例名称 | 显示节点的数目 |
| 概述 | 在面板中显示自动机节点的数目 |
| 参与者 | 用户 |
| 前置条件 | 无 |
| 后置条件 | 无 |
| 相关回调事件 | 无 |
| 基本事件流 | 1. （如果有）切换到对应的属性面板 2. 在面板上查看当前自动机节点的数目 |

|  |  |
| --- | --- |
| 用例名称 | 查看自动机的类型 |
| 概述 | 查看自动机的类型 |
| 参与者 | 用户 |
| 前置条件 | 无 |
| 后置条件 | 无 |
| 相关回调事件 | 无 |
| 基本事件流 | 1. （如果有）切换到对应的属性面板 2. 在面板上查看当前自动机是否合法 |
| 约束 | 自动机的类型 = {不合法，NFA，DFA} |
| 链接 |  |

## 画板操作

|  |  |
| --- | --- |
| 用例名称 | 选择画板元素 |
| 概述 | 选择画板中的一条边或者一个节点 |
| 主要参与者 | 用户 |
| 前置条件 |  |
| 基本事件流 | 在画板的一个元素上方按下鼠标左键就可以将其选中。 |

节点操作

|  |  |
| --- | --- |
| 用例名称 | 增加节点 |
| 概述 | 用户点击画板，添加空节点（没有其他边与之相连） |
| 参与者 | 用户 |
| 前置条件 | 画板处于添加模式 |
| 后置条件 | 添加成功后，更新自动机的属性   1. 把节点的信息添加到有向图中 2. 更新边的数目 3. 更新节点的数目 4. 重新判断自动机是否合法 |
| 相关回调事件 |  |
| 基本事件流 | 1. 点击画板空白处 2. 输入节点名称 3. 确认   如果2.中输入的名称为空，节点名称为默认名称  如果2.中输入的名称不为空，节点名称为2.中所输入的名称 |

|  |  |
| --- | --- |
| 用例名称 | 删除节点 |
| 概述 | 删除画板中已有的节点 |
| 参与者 | 用户 |
| 前置条件 | 画板不处于添加模式 |
| 后置条件 | 删除之后立即更新自动机的属性  添加成功后，更新自动机的属性   1. 把节点的信息从有向图中移除 2. 更新边的数目 3. 更新节点的数目   重新判断自动机是否合法 |
| 基本事件流 | 如果画板处于删除模式：   1. 用户点击对应节点，将其删除   或者如果画板处于选择模式：   1. 用户点击对应节点，将其选中 2. 用户按下键盘“delete”键或者右键打开菜单，选择“删除”选项   删除该节点后，与其相连的边（入边，出边）都应删除 |

|  |  |
| --- | --- |
| 用例名称 | 修改节点名称 |
| 概述 | 修改节点的名称 |
| 主要参与者 | 用户 |
| 前置条件 | 画板处于选择模式 |
| 后置条件 | 更新对应节点的信息 |
| 基本事件流 | 1. 双击节点 2. 编辑节点名称 3. 确认   或者   1. 右键打开菜单，选择“重命名选项” |

|  |  |
| --- | --- |
| 用例名称 | 拖动节点 |
| 概述 | 拖动画板上的节点 |
| 参与者 | 用户 |
| 前置条件 | 画板处于选择状态 |
| 后置条件 | 无 |
| 基本事件流 | 1. 将鼠标移动到节点上，按住鼠标，拖动节点   调节与之相连接的边的位置  当鼠标按住节点，但是已经移出画板区域的时候：  节点与画板边缘相切，X方向或Y方向随着鼠标移动而移动。 |

|  |  |
| --- | --- |
| 用例名称 | 设置节点为初始节点类型 |
| 概述 | 把画板上的一个节点设置为初始状态 |
| 主要参与者 | 用户 |
| 前置条件 | 画板处于选择状态 |
| 后置条件 | 操作成功后，更新有向图中相应节点的类型 |
| 相关回调事件 |  |
| 基本事件流 | 1. 右键打开菜单，选择节点类型 2. 按下Enter键或画板空白处确认 |

|  |  |
| --- | --- |
| 用例名称 | 设置节点为结束节点类型 |
| 概述 | 把画板上的一个节点设置为结束状态 |
| 主要参与者 | 用户 |
| 前置条件 | 画板处于选择状态 |
| 后置条件 | 操作成功后，更新有向图中相应节点的类型 |
| 基本事件流 | 1. 右键打开菜单，选择节点类型 2. 按下Enter键或画板空白处确认 |

|  |  |
| --- | --- |
| 用例名称 | 设置节点为中间状态 |
| 概述 | 把画板上的一个节点设置为结束状态 |
| 主要参与者 | 用户 |
| 前置条件 | 画板处于选择状态 |
| 后置条件 | 操作成功后，更新有向图中相应节点的类型 |
| 基本事件流 | 1. 右键打开菜单，选择节点类型   按下Enter键或画板空白处确认 |

## 边操作

|  |  |
| --- | --- |
| 用例名称 | 连接节点 |
| 概述 | 连接画板上任意两个节点 |
| 参与者 | 用户 |
| 前置条件 | 画板处于添加模式 |
| 后置条件 | 更新有向图对应边的属性 |
| 相关回调事件 |  |
| 基本事件流 | 1. 鼠标按住，从一个节点移动到另外一个节点 2. 松开鼠标以确认添加。 3. 输入转换条件 4. 按Enter或者画板空白处确认 |
| 约束 | 连接两个节点的不同情况：  横向轴表示两个节点之间边的情况  纵向轴表示新建边源/目标节点的情况   |  |  |  |  | | --- | --- | --- | --- | |  | 没有边 | 与已有边同向 | 与已有边反向 | | 自我连接 | 创建一个边，其形状**近似一个圆** | 修改转换条件 | 修改转换条件 | | 连接两个不同的边 | 创建一个边，其形状为一条**直线** | 修改转换条件 | 创建一条边，并且使两条边**向外弯曲。** |  1. 如果两个节点没有边，新建一条直边。 2. 如果原来的边和Edge方向相反，那么将两条边调整成凸起方向相对的弧线。 3. 如果原来的边与Edge的方向相同，那么编辑转换条件。 4. 如果边的起始节点和末尾节点相同， |

|  |  |
| --- | --- |
| 用例名称 | 删除边 |
| 概述 | 删除画板中已有的边 |
| 参与者 | 用户 |
| 前置条件 | 画板不处于添加模式 |
| 后置条件 | 删除之后立即更新自动机的属性  添加成功后，更新自动机的属性   1. 把边的信息从有向图中移除 2. 更新边的数目 3. 更新节点的数目   重新判断自动机是否合法 |
| 基本事件流 | 如果画板处于删除模式：   1. 用户点击对应边，将其删除   或者如果画板处于选择模式   1. 用户点击对应边，将其选中 2. 用户按下键盘“delete”键或者在右键菜单选择“删除”选项确认 |

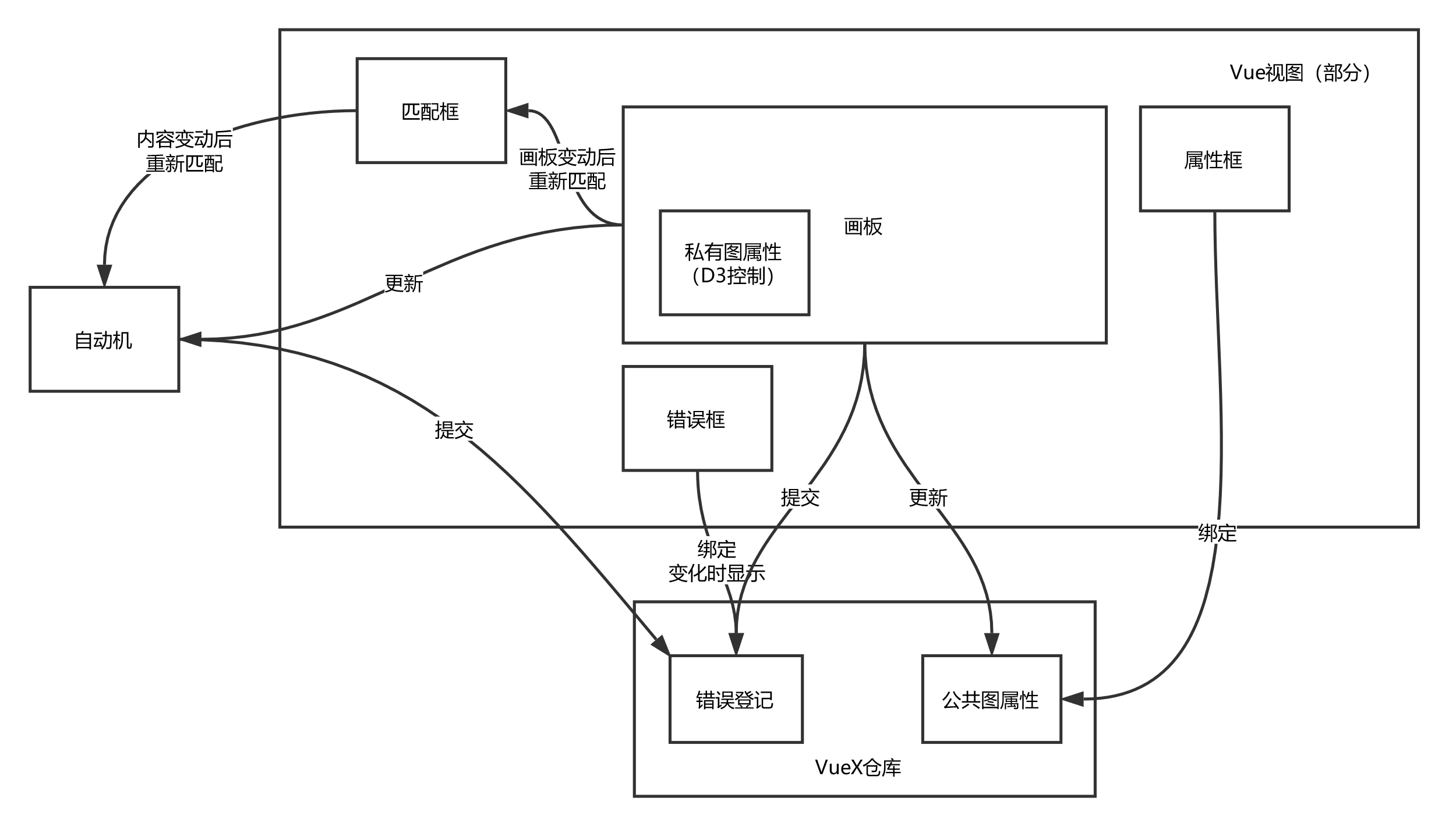
|  |  |
| --- | --- |
| 用例名称 | 修改转换条件 |
| 概述 | 修改边的转换条件 |
| 主要参与者 | 用户 |
| 前置条件 | 画板处于选择状态 |
| 后置条件 | 更行有向图中对应边的属性 |
| 基本事件流 | 1. 在边的上方双击左键 2. 输入转换条件 |
| 约束 | 如果转换条件包含多个转换字符，那么每个字符用空格隔开，空格本身使用\s来转义 |

## 匹配

|  |  |
| --- | --- |
| 用例名称 | 匹配字符串 |
| 概述 | 根据当前的自动机匹配字符串 |
| 参与者 | 用户 |
| 前置条件 | 无 |
| 后置条件 | 在显示匹配结果 |
| 相关回调事件 | 当输入框的字符串发生变动：  系统重新匹配字符串并输出结果 |
| 基本事件流 | 1. 输入或修改输入框的字符串   系统触发匹配行为  如果自动机是DFA，将结果显示到面板上。  如果自动机不合法，显示匹配结果未知。 |

|  |  |
| --- | --- |
| 用例名称 | 查看匹配结果 |
| 概述 | 查看字符串的匹配结果 |
| 参与者 | 用户 |
| 前置条件 | 无 |
| 后置条件 | 无 |
| 基本事件流 | 1. 切换到对应的匹配结果面板（如果有） 2. 查看面板中的匹配结果 |
| 约束 | 匹配结果 = {未定义，成功，失败} |

# 总体结构



## 枚举类型

|  |  |
| --- | --- |
| **EditType 画板模式** | |
| ADD | 添加模式 |
| EDIT | 编辑模式 |
| DELETE | 删除模式 |

|  |  |
| --- | --- |
| **AutometaType 自动机类型** | |
| DFA | 当前自动机为DFA |
| NFA | 当前自动机为NFA |
| INVALID | 当前自动机不合法 |

|  |  |
| --- | --- |
| **MatchResult 匹配结果** | |
| OK | 成功匹配 |
| FAILED | 匹配失败 |
| UNKNOWN | 匹配结果未知（当图不合法的时候） |

|  |  |
| --- | --- |
| **NodeType 节点类型** | |
| NORMAL | 普通节点 |
| INITIAL | 起始节点 |
| FINAL | 终止节点 |
| INITIAL\_FINAL | 起始-终止节点 |

注：如果一个节点为起始-终止节点，则该节点同时为起始节点和终止节点。

## 自动机[[1]](#footnote-1)

功能：匹配字符串，支持DFA和NFA匹配

#### 类接口

|  |  |
| --- | --- |
| 类Automata | |
| **属性** | |
| \_nodeRecords: NodeRecord[] | 记录了当前节点及其所有出边信息的对象，初始值为空数组 |
| \_edgeCount: Number | 记录当前边的数目，初始值为0 |
| \_nodeCount: Number | 记录当前节点的数目，初始值为0 |
| **方法** | |
| hasEdge(key: String): Boolean | 判断图中是否存在key为key的边 |
| hasNode(key: String): Boolean | 判断图中是否存在key为key的节点 |
| addEdge(node: Node): undefined | 增加边 |
| addNode(edge: Edge): undefined | 增加节点 |
| removeNode(key: String): undefined | 删除key为key的节点 |
| removeEdge(key: String): undefined | 删除key为key的边 |
| updateNode(key: String, propObj: Object): undefined | 修改key为key的节点，其中propObj是要修改的属性 |
| updateEdge(key: String, propObj: Object): undefined | 修改key为key的边，其中propObj是要修改的属性 |
| findNode(key: String): Node | 寻找key为key的节点 |
| findEdge(key: String): Edge | 寻找key为key的边 |
| matchWholeString(str: String): MatchResult | 匹配整个字符串 |
| nodeCount(): Number | 返回当前节点数目 |
| edgeCount(): Number | 返回当前边数目 |
| type(): AutometaType | 返回当前自动机类型 |

|  |  |
| --- | --- |
| 类Node | |
| **属性** | |
| key: String | 节点的key |
| type: NodeType | 节点的类型 |

|  |  |  |
| --- | --- | --- |
| 类Edge | | |
| **属性** | | |
| key: {  source: NodeRecord,  target: NodeRecord  } | 边的key，其中： | |
| source | 边的起始节点 |
| target | 边的目标节点 |
| type: NodeType | 节点的类型 | |

|  |  |
| --- | --- |
| 类 NodeRecord | |
| **属性** | |
| node: Node | 该NodeRecord对应的节点 |
| outEdges: Edge[] | 该NodeRecord对应的外向边 |
| **方法** | |
| static fromNode(node, [outEdges]) : NodeRecord | |
| 类方法，用于将一个节点转换成一个节点记录。  如果调用时省略outEdges参数，则生成的节点记录的outEdges属性为空数组。 | |
| toNode() : Node | |
| 用于将一个节点记录转换成一个节点。 | |

#### 自动机异常类

|  |  |
| --- | --- |
| 类AutometaError | |
| **属性** | |
| code: AutometaError | 错误代码 |
| reason: String | 描述错误原因的字符串 |

#### 错误代码

|  |  |
| --- | --- |
| 代码 | 解释 |
| NODE\_EXISTS | 节点已存在 |
| NODE\_NOT\_EXISTS | 节点不存在 |
| SOURCE\_NOT\_EXISTS | 边的源节点不存在 |
| TARGET\_NOT\_EXISTS | 边的目标节点不存在 |
| EDGE\_EXISTS | 边已经存在 |
| EDGE\_NOT\_EXISTS | 边不存在 |
| UNKNOWN\_EDGE\_PROPERTY | 未知边属性 |
| UNKNOWN\_NODE\_PROPERTY | 未知节点属性 |

#### 方法异常说明

|  |
| --- |
| 方法 |
| hasEdge(key: String): Boolean |
| 1. 不会抛出异常 |
| hasNode(key: String): Boolean |
| 1. 不会抛出异常 |
| insertEdge(node: Node): undefined |
| 1.当图中存在该边时（拥有相同key的边）时，抛出异常AutomataError，错误码为NODE\_EXISTS |
| insertNode(edge: Edge): undefined |
| 1. 当图中不存在边的起始节点的时候，抛出异常AutomataError(SOURCE\_NOT\_EXISTS) 2. 当图中不存在边的目标节点的时候，抛出异常AutomataError(TARGET\_NOT\_EXISTS) 3. 当该边在自动机中已经存在，抛出异常AutomataError(EDGE\_EXISTS) |
| deleteNode(key: String): undefined |
| 1. 当自动机不存在key为key的节点的时候，抛出异常AutomataError(NODE\_NOT\_EXISTS) |
| removeEdge(key: String): undefined |
| 1. 当自动机不存在key为key的边的时候，抛出异常AutomataError(EDGE\_NOT\_EXISTS) |
| updateNode(key: String, propObj: Object): undefined |
| 1. 当自动机不存在key为key的节点的时候，抛出异常AutomataError(NODE\_NOT\_EXISTS) 2. 当propObj中存在未知的（不合法的）属性的时候，抛出异常AutometaError(UNKNOWN\_NODE\_PROPERTY) |
| updateEdge(key: String, propObj: Object): undefined |
| 1. 当自动机不存在key为key的边的时候，抛出异常AutomataError(EDGE\_NOT\_EXISTS) 2. 当propObj中存在未知的（不合法的）属性的时候，抛出异常AutometaError(UNKNOWN\_EDGE\_PROPERTY) |
| findNode(key: String): Node |
| 不会抛出异常 |
| findEdge(key: String): Edge |
| 不会抛出异常 |
| matchWholeString(str: String): MatchResult |
| 不会抛出异常 |
| nodeCount(): Number |
| 不会抛出异常 |
| edgeCount(): Number |
| 不会抛出异常 |
| type(): AutometaType |
| 不会抛出异常 |

## VueX仓库

### 错误登记

#### States

|  |  |
| --- | --- |
| message: String | 错误信息 1. 如果为undefined，则当前没有错误，否则有错误 |

#### Mutations

|  |
| --- |
| **updateErrorMsg** (state, msg) |
| 设置错误信息。如果原来的message不为undefined，则更新原来的message。 |

#### Actions

|  |
| --- |
| **popUpError** ({ commit, clearError }, message, duration) |
| 设置错误信息，并且通知错误框显示错误，然后在duration秒内清空错误 |

#### Getters

|  |
| --- |
| **isError**() |
| 用来判断当前是否有错误 |

### 公共图属性

#### States

|  |  |
| --- | --- |
| nodeCount: Number | 当前图的节点数目 |
| edgeCount: Number | 当前图的边数目 |
| autometaType: AutometaType | 当前自动机的类型 |
| matchPair: {  str: String,  result: MatchResult  } | 当前的匹配对（匹配字符串+匹配结果） |

[[2]](#footnote-2)

#### Actions

|  |
| --- |
| **updateNodeCount** (state, count) |
| 更新节点的数目 |
| **updateEdgeCount** (state, count) |
| 更新边的数目 |
| **updateAutometaType** (state, type) |
| 更新自动机类型 |
| **updateMatchString** (state, str) |
| 更新匹配对中的匹配字符串 |
| **updateMatchResult** (state, result) |
| 更新匹配对中的匹配结果 |

#### Mutations

|  |
| --- |
| **matchString** ({ commit, state }, [str]) |
| 更新匹配字符串，匹配该字符串，然后更新匹配结果。  注意：str是可选的参数，如果省略str，则用现有的字符串重新匹配结果。 |

### 公共画板属性

#### States

|  |
| --- |
| editMode: EditMode |
| 记录画板的当前状态。初始状态为**EditMode.ADD** |

#### Mutations

#### Actions

## 视图组件

### 匹配框

#### Properties

#### Data

#### Methods

### 错误框

#### Properties

#### Data

#### Methods

### 属性框

#### Properties

#### Data

#### Methods

### 画板

因为画板涉及到D3和Vue的混合操作，而且交互比较复杂，所以单独在下一节展开描述。

# 可重用组件

## 编辑框 EditField

#### Properties

#### Data

#### Methods

## 带标题标签 TitleLabel

#### Properties

#### Data

#### Methods

## 选择菜单 Menu

#### Properties

#### Data

#### Methods

# 画板

## 画板元素

|  |  |
| --- | --- |
| **元素** | **解释** |
| 节点 | 图节点 |
| 边 | 图的边 |
| 节点编辑框 | 用于更改节点的名字 |
| 边的编辑框 | 用于更改边的名字 |
| 节点右键菜单 | 用于更改节点类型 |

## 画板模式及其操作

画板有三种模式：

1. **添加模式**

在添加模式下，用户可以在画板中添加新的节点，或者连接两个节点来新建边。

1. **选择模式**

在选择模式下，用户可以选中画板中已有的节点，然后可以将其拖动，也可以通过右键菜单来编辑、删除节点。

1. **删除模式**

在删除模式下，用户可以点击节点并将其删除。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 添加模式 | 选择模式 | 删除模式 |
| 增加节点 | √ |  |  |
| 重命名节点 |  | √ |  |
| 拖动节点 |  | √ |  |
| 设置节点类型 |  | √ |  |
| 删除节点 |  | √ | √ |
| 连接节点 | √ |  |  |
| 删除已有的边 |  | √ | √ |
| 修改转换条件 |  | √ |  |
| 打开右键菜单 |  | √ |  |

注意：设置节点类型包括设置成初始节点，终止节点和中间状态节点三种类型类型。

## 画板的状态

画板中大多数事件监听函数需要先判断当前画板的状态，然后根据不同的状态执行不同的代码。在画板中，状态用一个属性来表示，他可以为以下的值。

|  |  |
| --- | --- |
| 状态名称（英文） | 解释 |
| Normal | 画板不做任何操作时的状态 |
| NodeMenuOpened | 节点菜单已经打开的情况 |
| NodeEditFieldOpened | 节点编辑框已经打开 |
| DraggingNode | 节点正在被拖动 |
| EdgeEditFieldOpened | 边的编辑框已经打开 |
| LinkingTwoNodes | 正在连接两条边 |

### 属性

*Vue属性*contextMenuProps:

|  |  |
| --- | --- |
| menuNode:  Object | 用于记录左键菜单正在处理的节点。当左键菜单隐藏时，该属性应该为null。在菜单被显示之前我们将其设置为被右键点击的节点。 |
| editingEdge:  Object | 用于记录编辑框打开时，正在编辑的边，当边的编辑框被隐藏时，该属性应该设置为null. |
| editingNode:  Object | 用于记录编辑框打开时，正在编辑的节点。当节点的编辑框隐藏时，该属性应该设置为null |

### 编辑框和菜单选项的回调函数

*以下代码省略了包装函数function () {}*

|  |
| --- |
| **节点的编辑框** |
| saveNodeInfo(editingNode); |

|  |
| --- |
| **边的编辑框** |
| saveEdgeInfo(editingEdge) |

|  |  |
| --- | --- |
| **菜单选项的回调函数** | |
| 设置节点为初始类型 | setNodeType(‘initial’, editingNode) |
| 设置节点为结束类型 | setNodeType(‘final’, editingNode) |
| 设置节点为普通类型 | setNodeType(‘normal’, editingNode) |
| 设置节点为初始-结束类型 | setNodeType(‘initFinal’, editingNode) |

## 事件处理

### 归类

触发事件的对象：

|  |  |
| --- | --- |
| 画板 | click |
| 节点 | .link, dblclick, .drag, click |
| 边 | click, dblclick |

按照不同画板模式来分类：

|  |
| --- |
| 添加模式（mode Add） |
| **Board**.click,  **Board**.link {  **Node**.mousemove,  **Node**.mouseup,  **Node**.mousedown,  **Board**.mousemove,  **Board**.mouseup  } |
| 编辑模式（mode Edit） |
| **Node**.dblclick,  **Node**.drag {  Node.mousedown,  **Node.mouseup,**  **Node.mousemove,**  },  **Node**.contextmenu,  **Edge**.dblclick |
| 删除模式（mode Delete） |
| **Node**.click,  **Edge**.click |

## 回调函数的伪代码

|  |
| --- |
| 添加节点 |
| **board**.on(‘click.addNode’) {  if (boardState == ‘normal’) {  let position = getNodePosition();  addNode(getNodeInitialProperties(position));  setBoardState('nodeEditFieldOpened);  }  } |

|  |
| --- |
| 编辑节点 |
| **board**.on(‘dblclick.editNode’) {  if (boardState == ‘normal’) {  /\* D3可以使用datum直接得到绑定的数据 \*/  setBoardState(nodeEditFieldOpened, /\* current dblclicked node’s data \*/)  }  } |

|  |
| --- |
| 打开右键菜单 |
| **node**.on(‘contextmenu.openMenu’) {  if (boardState == ‘normal’) {  /\* D3可以使用datum直接得到绑定的数据 \*/  setBoardState(‘nodeMenuOpened’, /\* current dblclicked edge’s data \*/);  }  } |

|  |
| --- |
| 编辑边 |
| **edge**.on(‘dblclick.editEdit’) {  if (boardState == ‘normal’) {  /\* D3可以使用datum直接得到绑定的数据 \*/  setBoardState(‘edgeMenuOpened’, editingEdge);  }  } |

|  |
| --- |
| 删除边 |
| edge.on(‘click.deleteEdge’) {  if (boardState == ‘normal’) {  /\* D3可以使用datum直接得到绑定的数据 \*/  deleteEdge(/\* current clicked edge \*/)  }  } |

|  |
| --- |
| 删除节点 |
| **edge**.on(‘click.deleteNode’) {  if (boardState == ‘normal’) {  /\* D3可以使用datum直接得到绑定的数据 \*/  deleteNode(/\* current clicked node \*/)  }  } |

|  |
| --- |
| 单击面板以关闭节点编辑框 |
| board.on(‘click.editNode’) {  if (boardState == ‘nodeEditFieldOpened’) {  try {  /\* Old transition can access through datum arg \*/  let transition = getTransition(/\* text in edit field \*/);  if (transition equals old transition) {  updateNode(transition)  }  setNodeState(‘normal’)  } catch (InvalidInput e) {  /\* infrom error vuex store to broadcast the error. \*/  }  }  } |

### .Link事件

**对外的子事件：**

|  |  |  |
| --- | --- | --- |
| 子事件名字 | 对应DOM事件 | 解释 |
| start | node.mousedown | 当鼠标按下节点的时候触发的事件 |
| link | node.mousemove  board.mousemove | 当按着鼠标移动的时候触发的事件 |
| end | node.mouseup  board.mouseup | 当松开鼠标的时候触发的事件 |

**.link的事件对象：**

|  |  |
| --- | --- |
| target | 事件对应的link事件 |
| type | 字符串，表示当前的子事件的名字，可以为start，link或者end。 |
| source | 表示鼠标按下的时候对应的节点元素 |
| target | 表示在连接中鼠标指向的节点元素，如果鼠标不指向任何元素，则该属性为null。 |
| sourcePos | 包含属性x，y的对象，记录source的位置 |
| targetPos | 包含属性x，y的对象，记录target的位置 |

事件描述

|  |  |
| --- | --- |
| node.mousemove | 执行start回调函数，设置节点和容器的mousemove和mouseup事件 |
| node.mousemove | 执行link回调函数 |
| board.mousemove | 执行link回调函数 |
| node.mouseup | 执行end回调函数，清空mousemove和mouseup事件 |
| board.mouseup | 执行end回调函数，清空mousemove和mouseup事件 |

内部的属性

|  |  |
| --- | --- |
| target | 事件对应的link事件 |
| type | 字符串，表示当前的子事件的名字，可以为start，link或者end。 |
| source | 表示鼠标按下的时候对应的节点元素 |
| target | 表示在连接中鼠标指向的节点元素，如果鼠标不指向任何元素，则该属性为null。 |
| sourcePos | 包含属性x，y的对象，记录source的位置 |
| targetPos | 包含属性x，y的对象，记录target的位置 |

注：内部属性和事件属性相同，当触发事件时，创建一个新的事件对象，然后将内部属性逐一赋值给事件对象的属性。

## 限制

|  |  |
| --- | --- |
| 转换条件的字符串 | 包含字符（也就是长度等于1的字符串）[ε, a-z, A-Z]以逗号分隔开的字符序列，字符串不能为空。 |
| 转换条件 | 包含字符[a-Z, A-Z]（也就是长度等于1的字符串）或空字符串（代表ε转换）的数组。 |
| 节点名字的长度 | 大于等于1且小于等于10 |

## 错误

### 错误对象的结构

### 错误类

|  |  |
| --- | --- |
| InvalidInupt | 当用户输入不符合限制条件的时候，抛出此错误 |
|  |  |

### 错误码

|  |  |
| --- | --- |
| INVALID\_TRANSITION | 当修改的边的转换条件不符合限制（1，2）的时候，抛出此错误。 |
| INVALID\_NODE\_NAME | 当修改的节点名字不符合限制（3）时候，抛出此错误 |

## 节点的类型

节点的类型包括：

1. 初始节点
2. 终止节点
3. 中间状态节点

## 自动机类型及其操作

|  |  |  |  |
| --- | --- | --- | --- |
|  | DFA | NFA | 不合法 |
| 画板操作 | √ | √ | √ |
| 匹配字符串 | √ |  |  |

### 自动机类型的判别规定

错误状态

1. 自动机默认的字符集合ASCII集合，暂时不支持非英语单词。
2. 如果将要匹配的字符不符合某个状态的所有转换条件，那么自动机就会转换成错误状态，在此状态下，输入任何字符之后，自动机仍然处于错误状态。
3. 如果自动机处于错误状态，字符串匹配应当失败。
4. 错误状态不显示在画板上。也就是说，在画板中没有相应的节点对应这个状态。

不合法

画板中的自动机只要符合以其中一个有条件，那么它不合法

1. 只存在错误状态
2. 没有终止状态或者初始状态

NFA

如果画板中的自动机符合以下所有条件，那么他就可以被认为是合法的NFA。

1. 除了错误状态，它至少有一个状态
2. 它至少拥有一个终止状态和初始状态
3. 对于它的每个节点，给定一个输入，能找到一个以上的转换条件，从而转换到其中任意一个符合转换条件的状态。

DFA

如果画板中的自动机符合以下的所有条件，那么他就可以被认定为合法的DFA。

1. 它是合法的NFA
2. 对于它的每个节点，给定一个输入，都能找到唯一的转换条件，从而转移到确定的一个状态B。即存在函数关系：

1. 该自动机主要用于字符串匹配，属于外部API，它的属性没有被Vue或D3绑定，所以当该自动机的属性变动之后，视图中有关自动机属性的元素不会随之变动，反之亦然。因此我们需要单独一个VueX仓库来记录这些属性。 [↑](#footnote-ref-1)
2. 匹配对：用于记录当前匹配的字符串及其匹配结果的对象。 [↑](#footnote-ref-2)