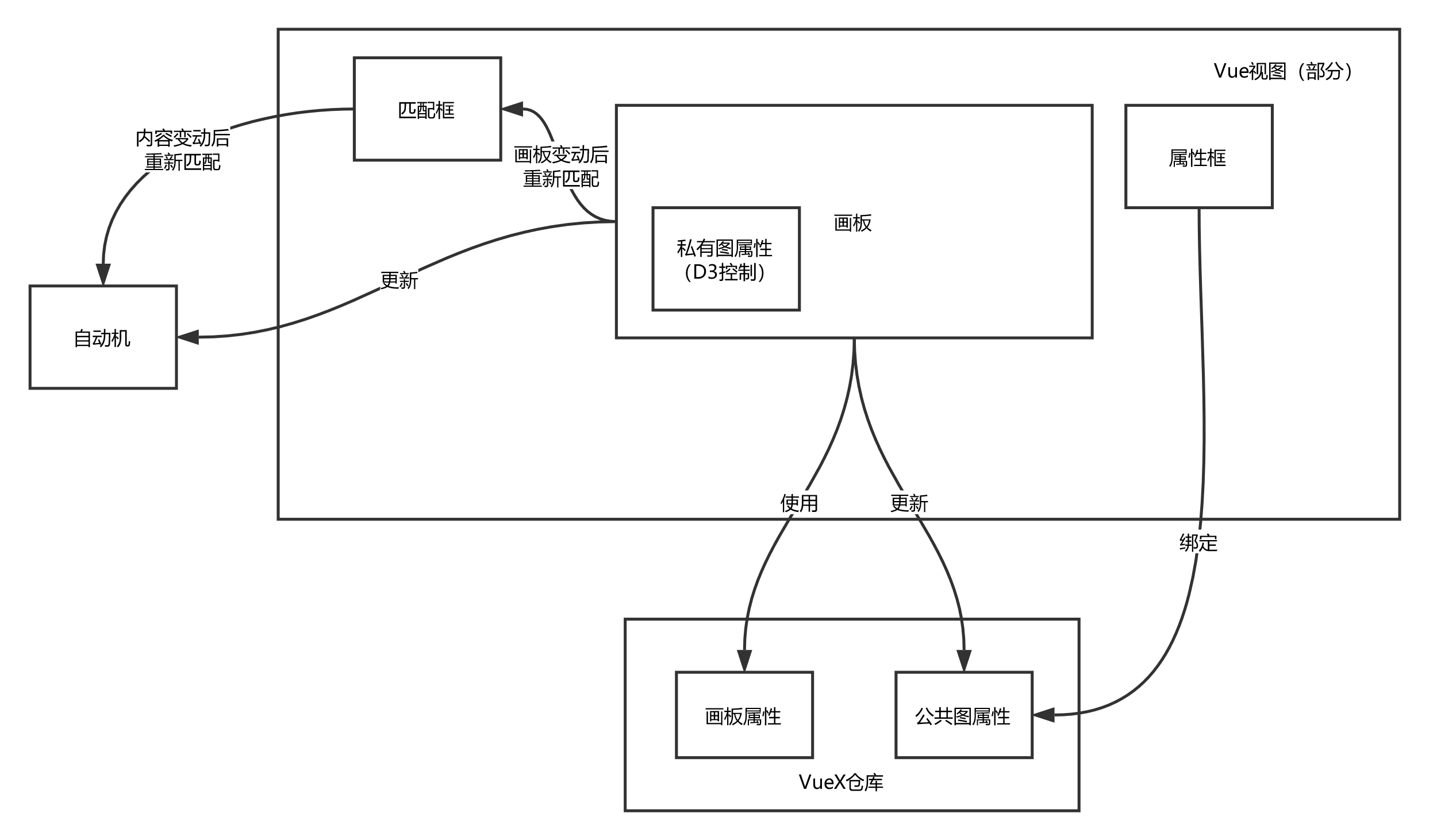
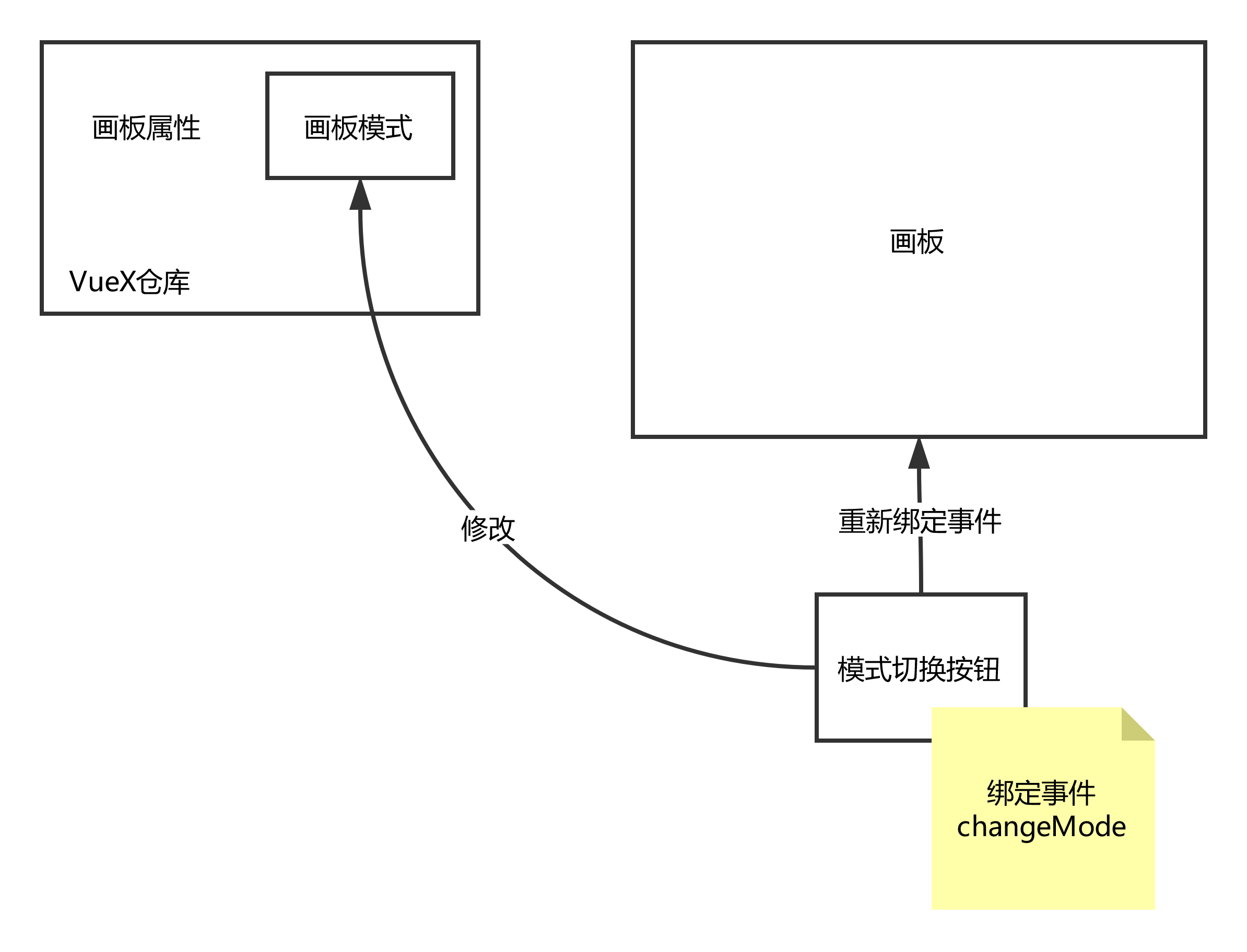
总体设计文档

# 图示



更新画板模式



# 接口声明

具体实现可以根据具体的UI、操作，具体实现可以扩充必要的类、组件和方法，但是它必须实现以下的类、组件及方法。

## 枚举类型

|  |  |
| --- | --- |
| **EditMode 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 |
| transition: String[0..\*] | 转换条件 | |

|  |  |
| --- | --- |
| 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

### 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)

#### Mutations

|  |
| --- |
| **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 |

#### Actions

|  |
| --- |
| **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，如果省略str，则用现有的字符串重新匹配结果。 |

#### Watch

|  |
| --- |
| **matchPair.str** |
| matchString中str属性的侦测器，当str属性发生变化的时候，调用Action matchString方法来更新matchPair。 |

### 公共画板属性

#### States

|  |
| --- |
| editMode: EditMode |
| 记录画板的当前模式。初始值为**EditMode.ADD** |

#### Mutations

|  |
| --- |
| **setEditMode** (state, newEditMode) |
| 更改画板模式 |

#### Actions

|  |
| --- |
| **updateEditMode** (state, newEditMode) |
| 更改画板模式，重新绑定画板中的事件。 |

## 视图组件

所有视图组件采用Vue的MVVM模式（模型-视图-视图控制器模式）。其中视图时Vue HTML Template，视图控制器是Vue实例，模型是背后的API（如自动机API）。因此，以下介绍视图采用如下的结构：

1. 视图的组成元素
2. 视图控制器属性
3. 视图控制器数据
4. 视图控制器方法
5. 视图控制器的其他元素
6. 数据的绑定关系
7. 事件的绑定关系
8. 视图控制器涉及到的模型、服务、API。

### 匹配框

匹配框的样式和matchPair.result绑定，匹配框的内容和matchPair.str绑定。当用户更改匹配框文本时会同步更新matchPair.str，其匹配结果会更新matchPair.result，最后使得匹配框样式发生变化。

#### 视图组成元素

|  |  |
| --- | --- |
| **名称** | **解释** |
| 输入框 | 提供文本输入的组件。如<input type=”text”>或者<textarea>或自定义的复合组件。 |

#### 控制器属性

|  |  |
| --- | --- |
| **名称** | **解释** |
| text: String | 匹配狂的文字 |
| state: String | 匹配框的样式，可取的值：   1. ok 2. failed 3. unknown |

#### 控制器数据

|  |  |
| --- | --- |
| **名字** | **解释/备注** |
| styleObject: Object | 这个属性的值取决于属性state |
| classObject: Object | 这个属性的值取决于属性state |

#### 数据绑定关系

|  |  |  |
| --- | --- | --- |
| **视图元素** | **视图属性** | **绑定数据** |
| 输入框 | 输入值 | 公共图属性（VueX仓库）中的matchPair.str， |
| 输入框 | 匹配框样式 | styleObject |
| 输入框 | 匹配框的类（class属性） | classObject |

#### 事件绑定关系

|  |  |  |
| --- | --- | --- |
| **视图元素** | **事件** | **侦听器** |
| 输入框 | onInput | matchString |

### 属性框

#### 视图组成元素

|  |  |
| --- | --- |
| **名称** | **解释** |
| 带标题的文本框 | 能够显示标题和内容的文本框 |

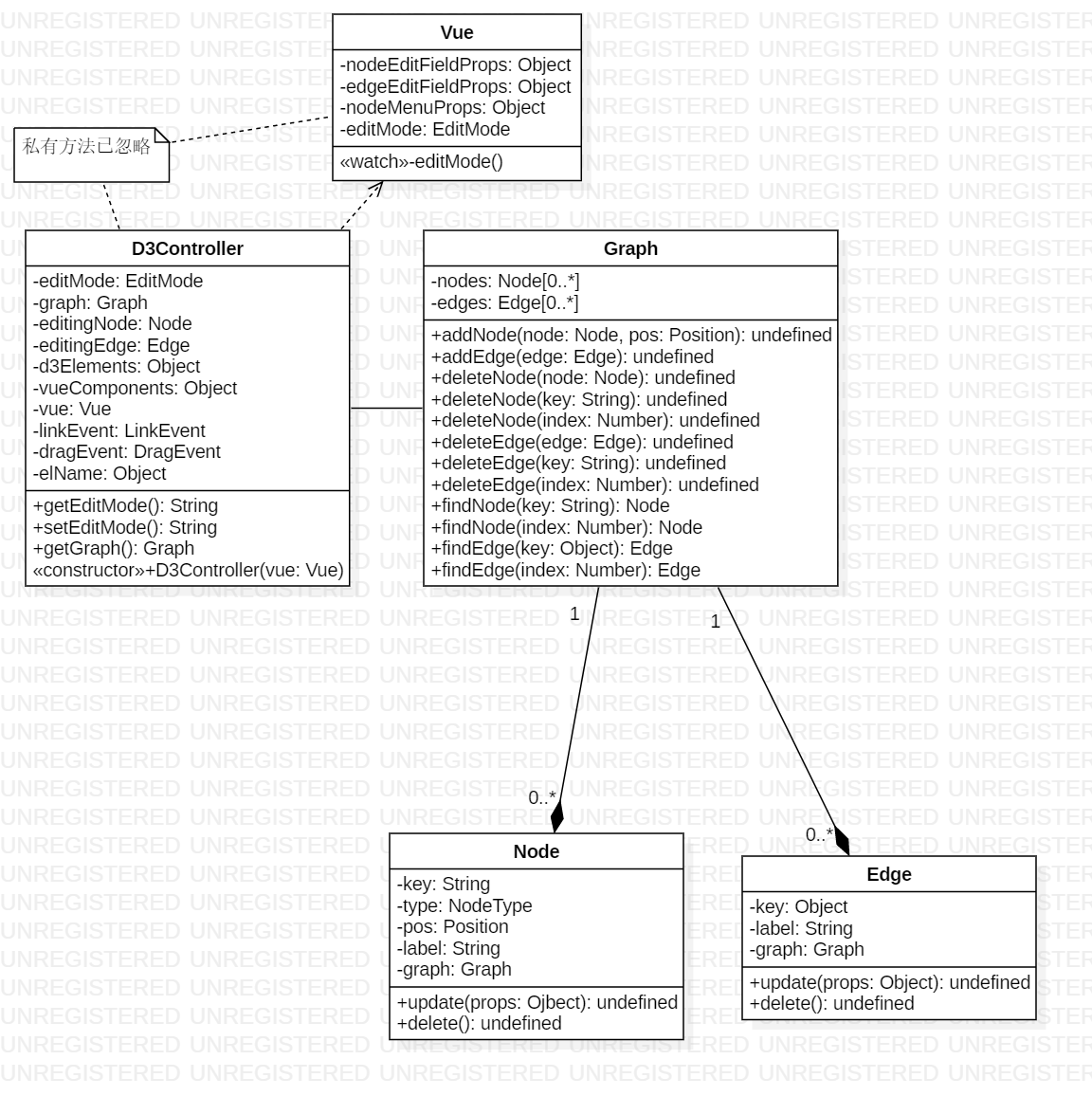
#### 控制器属性

|  |  |
| --- | --- |
| **名字** | **解释** |
| type: String | 自动机的类型 |
| nodeCount: Number | 自动机节点的数量 |
| edgeCount: Number | 自动机边的数量 |

## 画板

画板的视图控制器以及视图的表示如下：

画板视图的UML已省略



### 视图组成元素

|  |  |
| --- | --- |
| **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 |
| 节点菜单 | 用于更改节点类型 |
| 覆盖层 | 当编辑框或菜单打开时，点击覆盖层能够关闭编辑框或菜单。 |

### 画板元素的模型

涉及到的模型是画板中的节点元素和边元素。它们的类描述如下：

|  |  |
| --- | --- |
| 类Node | |
| **属性** | |
| key: String | 节点的key |
| type: NodeType | 节点的类型 |
| pos: Position | 节点的位置 |
| label | 节点的名称 |
| graph | 这个节点对应的图 |
| **方法** | |
| function **update** (props: **Object**) : undefined | |
| 更新节点  props：节点属性的更新值，合法的属性清单之前节点类的定义 | |

|  |  |  |
| --- | --- | --- |
| 类Edge | | |
| **属性** | | |
| key: Obejct | 边的key，其中： | |
| source: Node | 边的起始节点 |
| target: Node | 边的目标节点 |
| label: String | 边的名称（转换条件的字符串表示） | |
| graph | 这条边对应的图 | |
| **方法** | | |
| function **update** (props: **Object**) : undefined | | |
| props：节点属性的更新值，合法的属性清单之前节点类的定义 | | |

|  |  |
| --- | --- |
| 类Graph | |
| **属性** | |
| node: Node[0..\*] | 节点数组 |
| edge: Edge[0..\*] | 边数组 |
| **方法** | |
| function **addNode** (node: **Node**, pos: **Position**) : undefined | |
| 新建节点，设置其在画板的位置为pos，最后将其添加到节点数组的末尾 | |
| function **addEdge** (edge : **Edge**) : undefined | |
| 新建边，设置其在画板的位置为pos，最后将其添加到边数组的末尾 | |
| function **deleteNode** (node: **Node**) : undefined;  function **deleteNode** (key: **String**) : undefined;  function **deleteNode** (index: **Number**) : undefined; | |
| 删除节点，参数可以是一个节点对象、节点的键值或这个节点在数组中的下标。 | |
| function **deleteEdge** (edge : **Edge**) : undefined;  function **deleteEdge** (key : **String**) : undefined;  function **deleteEdge** (index: **Number**) : undefined; | |
| 删除边，参数可以是一个边的对象、边的键值或这个边在数组中的下标。 | |
| function **findNode** (key : **String**) : Node  function **findNode** (index: **Number**) : Node; | |
| 查找节点，参数可以是节点的键值，或节点在数组中的下标 | |
| function **findEdge** (key : **Object**) : Edge  function **findEdge** (index: **Number**) : Edge; | |
| 查找节点，参数可以是边的键值，或边在数组中的下标。  在此函数中，边的键值是包含了source和target属性的对象，这两个属性可以是两个Node类型的对象，也可以是这两个对象的键值。 | |
| function **updateEdge** (key : **Object,** props**: Object**) : Edge  function **updateEdge** (index : **Number,** props**: Object**) : Edge | |
| 查找并更新边，相当于线调用findEdge然后再调用edge的update方法 | |
| function **updateNode** (key : **String,** props**: Object**) : Edge  function **updateNode** (index : **Number,** props**: Object**) : Edge | |
| 查找并更新节点，相当于线调用findNode然后再调用node的update方法 | |

### Vue视图控制器

由于D3绑定数据的方式和Vue绑定数据的方式不一样（D3时全局更新后做数据集处理，Vue是通过观察者模式来更新元素，而且在更新数组元素时D3是直接对元素赋值，然后重新绑定而Vue是采用splice方法来替换），如果在一个数据中（特别是数组）同时绑定D3和Vue，可能会导致数据不能及时同步，所以关于画板和图的数据必须分开处理。

#### Vue事件

以下是Vue控制器涉及到的事件

|  |  |  |  |
| --- | --- | --- | --- |
| **元素** | **事件** | **侦听器描述** | **画板模式** |
| 菜单元素#set-initial | selectItem | setNodeType | edit |
| 菜单元素#set-final | selectItem | setNodeType | edit |
| 菜单元素#set-normal | selectItem | setNodeType | edit |
| 菜单元素#set-initial-final | selectItem | setNodeType | edit |

这些侦听器调用D3对外函数完成功能，然后隐藏菜单。由于这些事件和D3的其他一些事件关系紧密，所以它们都由D3视图控制器动态绑定。

#### Vue的控制器数据

|  |  |  |
| --- | --- | --- |
| **数据** | **解释** | |
| nodeEditFieldProps: Object | 节点编辑框的属性，包括pos, visible和value | |
| edgeEditFieldProps: Object | 边的编辑框的属性，包括pos, visible和value | |
| nodeMenuProps: Object | *节点菜单的属性包括：* | |
| pos | 节点菜单的x轴，y轴距离 |
|
| visible | 节点菜单是否可见 |
| editMode: EditMode | 当前画b板的状态，和Vuex仓库中的值同步 | |
| overlayProps: Object | 画板覆盖层的属性 | |

#### 视图控制器的其他元素

|  |
| --- |
| **WATCH** |
| mode |
| 当mode发生变化时，重新绑定画板事件。 |

#### 数据的绑定关系

|  |  |  |
| --- | --- | --- |
| **视图元素** | **视图属性** | **绑定数据** |
| 节点编辑框 | value | nodeEditFieldText（v-model） |
| 边的编辑框 | value | edgeEditFieldText（v-model） |

#### 事件的绑定关系

|  |  |  |
| --- | --- | --- |
| **视图元素** | **视图事件** | **侦听器** |
| 节点编辑框 | input（v-model） | （v-model） |
| 边的编辑框 | input（v-model） | （v-model） |

### Vue视图控制器中的其他属性

#### Watches

|  |  |
| --- | --- |
| 被观察的属性 | 描述 |
| editMode | 当Vue视图控制器中的editMode发生改变后，更新D3视图控制器中的editMode。 |

### D3视图控制器

D3视图控制器是一个函数，它的形式如下：

|  |
| --- |
| function **D3Controller**(vue: **Vue**): **Object** |
| 接受画板vue实例作为依赖，以及画板的模式，返回一个对象。这个对象中包含了所有控制器对外公开的函数。 |

#### 事件

这个控制器涉及到如下的事件

|  |  |  |  |
| --- | --- | --- | --- |
| **元素** | **事件** | **侦听器** | **画板模式** |
| 画板 | click | addNode | add |
| 节点 | click | deleteNode | delete |
| 节点 | .edit | openNodeEditField, closeNodeEditField | edit |
| 节点 | .link | beforeLinking, linking, afterLinking | add |
| 节点 | .drag | beforeDragging dragging, afterDragging | edit |
| 节点 | .menu | openNodeMenu, closeNodeMenu | edit |
| 边 | click | deleteEdge | delete |
| 边 | .edit | openEdgeEditField, closeEdgeEditField | edit |

另外，视图控制器包括了以下的局部变量和闭包函数：

#### *局部变量*

|  |  |
| --- | --- |
| **属性** | **解释** |
| \_editMode | 画板的属性 |
| \_graph: Graph | 图属性 |
| \_editingNode: Node | 正在编辑（节点菜单或编辑框打开之后）的节点 |
| \_editingEdge: Edge | 正在编辑（编辑框打开之后）的节点的边 |
| \_d3Elements: Object | 画板中的D3元素 |
| \_vueComponents: Object | 画板中的Vue组件，包括菜单和编辑框 |
| \_vue: Vue | 画板的Vue实例 |
| \_linkEvent | 用于连接两个节点的link事件 |
| \_dragEvent | 用于拖动两个节点的drag事件 |
| \_elName: Object | 记录画板元素的id的对象，便于在初始化时从HTML中选出画板元素。 |

#### *闭包函数1：对外公开函数*

这些函数会添加到控制器返回的对象中，供别的模块调用，以操控画板和自动机。当然，这些函数在控制器内部也应当能够使用。

|  |
| --- |
| function **getEditMode** () : **String** |
| 得到当前的画板状态 |
| function **setEditMode** (editMode: **String**) : **undefined** |
| 设置画板状态 |
| function **getGraph** () : **Graph** |
| 获得控制器中的图属性 |

#### 闭包函数2：侦听器

侦听器函数都被保存在listeners变量中。这样做是为了避免它们和对外公开函数发生名字空间的冲突。

这些侦听器都是绑定在D3选择集里的侦听器，所以它们的形式为：

|  |
| --- |
| function **listener**(datum, index) : undefined |

其中datum为选择集中的元素，index时这个元素在选择集的下标。

以下是侦听函数

|  |
| --- |
| **addNode**(datum, index) |
| 点击画板来添加节点：获取当前鼠标位置， |
| **deleteNode**(datum, index) |
| 点击并删除节点 |
| **beforeLinking**(datum, index) |
| 显示辅助边 |
| **linking**(datum, index) |
| 更新辅助边的位置 |
| **afterLinking**(datum, index) |
| 隐藏辅助边，添加边，调整边的形状。 |
| **beforeDragging**(datum, index) |
| 高亮显示这个节点 |
| **dragging**(datum, index) |
| 更新节点位置以及与之关联的边的位置 |
| **afterDragging**(datum, index) |
| 取消高亮这个节点 |
| **deleteEdge**(datum, index) |
| 点击并删除边，调整边的形状 |
| **showNodeMenu**(datum, index) |
| 打开节点菜单，设置editingNode |
| **hideNodeMenu**(datum, index) |
| 隐藏菜单，取消设置editingNode，更新节点。 |
| **showNodeEditField**(datum, index) |
| 打开节点编辑框，设置editingNode |
| **hideNodeEditField**(datum, index) |
| 隐藏节点编辑框，取消设置editingNode，更新节点。 |
| **showEdgeEditField**(datum, index) |
| 打开边的编辑框，设置editingEdge |
| **hideEdgeEditField**(datum, index) |
| 隐藏边的编辑框，取消设置editgingEdge，更新边。 |

#### 闭包函数3：内部函数

这些函数主要用于初始化控制器，进行事件的绑定和解除绑定。另外还有一些辅助函数。

|  |
| --- |
| **selectD3Elements**() |
| 从HTML中选出D3画板元素 |
| **selectVueComponents**() |
| 从HTML中选出Vue组件 |
| **enableGeneralEvents**() |
| 设置通用事件（在任何模式下都不变的事件） |
| **enableAddModeEvents**() |
| 设置添加模式下的属性 |
| **enableEditModeEvents**() |
| 设置编辑模式下的属性 |
| **enableDeleteModeEvents**() |
| 设置删除模式下的属性 |
| **enableDragging**() |
| 设置拖动事件 |
| **disableDragging**() |
| 取消设置拖动事件 |
| **enableLinking**() |
| 设置连接事件 |
| **disableLinking**() |
| 取消设置连接事件 |
| **enableNodeMenu**() |
| 设置节点菜单事件 |
| **disableNodeMenu**() |
| 取消设置节点菜单事件 |
| **enableEditNode**() |
| 设置编辑节点事件 |
| **disableEditNode**() |
| 取消设置编辑节点事件 |
| **enableAddNode**() |
| 设置添加节点事件 |
| **disableAddNode**() |
| 取消设置添加节点事件 |
| **enableDeleteNode**() |
| 设置删除节点事件 |
| **disableDeleteNode**() |
| 取消设置删除节点事件 |
| **enableDeleteEdge**() |
| 添加“删除边”事件的绑定 |
| **disabelDeleteEdge**() |
| 取消“删除边”事件的绑定 |
| **init**() |
| 选择D3和Vue元素，初始化Drag和Link事件，然后用**enableEvents**()初始化事件。 |
| **update**() |
| 更新D3数据集，删除exit元素，更新update元素，新建enter元素，然后调用**rebindEvents**()来重新绑定事件。 |
| **rebindEvents**() |
| 根据当前的editMode重新绑定事件：  先用**disableAllEvents**()解除所有事件的绑定，然后调用**enableEvents**()来绑定事件。 |
| **disableAllEvents**() |
| 取消绑定画板元素的所有事件 |
| **enableEvents**() |
| 根据当前的画板状态，设置事件 |

### D3复合事件描述

复合事件是由若干个具有先后触发次序的基本事件（DOM事件）组成。在这个项目中，存在4个复合事件，分别是：

|  |  |
| --- | --- |
| **事件名称** | **解释** |
| .drag | 拖动事件，由d3-drag完成 |
| .link | 连接事件 |

#### .drag

通过d3.drag()来绑定，通过selection.on(‘.drag’, null)来解绑。具体看D3官网。

#### .link

##### 事件涉及到的元素

|  |  |
| --- | --- |
| **元素** | **解释** |
| 节点 | 被连接的元素 |
| 容器 | 节点所在的容器，默认为节点的父级元素 |

##### 内部的DOM事件

|  |  |
| --- | --- |
| **元素** | **事件** |
| 节点 | mousedown |
| 节点 | mousemove |
| 节点 | mouseup |
| 容器 | mousemove |
| 容器 | mouseup |

**对外的子事件：**

|  |  |  |
| --- | --- | --- |
| **名称** | **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事件的属性

**.link的事件包括以下属性**

|  |  |
| --- | --- |
| 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，记录鼠标按下时鼠标在容器中的位置 |
| currentPos | 包含属性x，y的对象，记录正在连接时鼠标在容器中的位置 |

##### .link事件的事件对象

这个事件对象是指D3中的event对象。它的属性和.link的属性相同，是.link属性的拷贝。

##### .link事件的接口

###### 创建link事件

调用函数link以创建全新的link事件。

|  |  |
| --- | --- |
| 名称 | function **link**(); |
| 描述 | 创建新的link事件 |
| 返回值 | 返回一个可直接调用的link对象，调用这个对象 |

这个函数会返回一个函数，它代表了一个link事件：

|  |  |
| --- | --- |
| 名称 | function **linkEvent** (selection : **Selection**) |
| 描述 | link事件，向selection添加link事件。 |
| 返回值 | 返回一个可直接调用的link对象，调用这个对象 |

由于在JavaScript中所有类型都是对象，所以可以把link事件看作可调用的对象。这个函数除了JavaScript默认的函数属性，还添加了以下方法：

###### Link事件方法

|  |  |
| --- | --- |
| 名称 | **on**(name : **String**);  **on**(  name : **String**,  **callback** : (datum, index) => undefined,  capture: **Boolean**  ); |
| 描述 | 描述：绑定侦听器到link事件。 |
| 参数 | name的值可以为：start, link, end。  callback是指侦听器函数。  capture如果为真，事件将在捕获阶段触发，否则事件在冒泡阶段触发。这个标志相当于DOM中addEventListener中的useCapture标志。  如果调用者只提供name参数，则返回当前link事件中name子事件的侦听器。 |

|  |  |
| --- | --- |
| 名称 | **container** (container : (datum? , index?) => **DOM**);  function **container** (container : **DOM**); |
| 描述 | 设置容器的访问器，或返回当前容器的访问器 |
| 参数 | 如果调用者不提供任何参数，那么返回当前的容器的访问器。  如果给定的容器的类型是(datum, index) => DOM，那么就直接将参数设置为访问器。  如果给定的容器的类型是DOM，那么就先执行以下的包装函数，然后将返回结果设置为访问器：  function **wrapper** (container : DOM) {  return () => container  }  注意：这个函数不会检查参数的有效性。如果参数类型不是函数也不是DOM类型的对象，那么它会直接把值设置成容器；或者，虽然参数类型是函数，但是它返回类型不是DOM类型，这个函数只会简单应用函数之后，将其返回值设置成容器。 |
| 注 | 如果不设置container，那么默认的container为：  function **defaultContainer** () {  return this.parentNode  }  其中this的值为Selection中的元素。 |

### 可复用组件：编辑框

动作：提交，隐藏，显示

点击某个元素的时候显示编辑框

当编辑框显示时，点击编辑框外围或按下enter键，触发submit事件，然后隐藏菜单。

#### 控制器元素

|  |  |
| --- | --- |
| 元素 | 解释 |
| 编辑框 | 提供文本输入的组件。如<**input** type=”text”>或者<**textarea**>或自定义的复合组件。 |

#### 控制器属性

|  |  |  |
| --- | --- | --- |
| **元素** | **属性** | **解释** |
| 编辑框 | x | 编辑框的left值 |
| 编辑框 | y | 编辑框的top值 |
| 编辑框 | value | 用于v-model的属性 |

#### 控制器事件

|  |  |  |
| --- | --- | --- |
| **元素** | **事件** | **解释** |
| 编辑框 | input | 用于v-model的属性 |

### 可复用组件：菜单

当菜单打开的时候，点击其任意空白处即可关闭菜单。

#### 控制器元素

|  |  |
| --- | --- |
| **元素** | **解释** |
| 菜单 | 菜单本身 |
| 菜单选项 | 菜单选项 |

#### 控制器属性

|  |  |  |
| --- | --- | --- |
| **元素** | **属性** | **解释** |
| 菜单 | left | 编辑框的left值 |
| 菜单 | top | 编辑框的top值 |
| 菜单选项 | caption | 菜单选项的内容 |

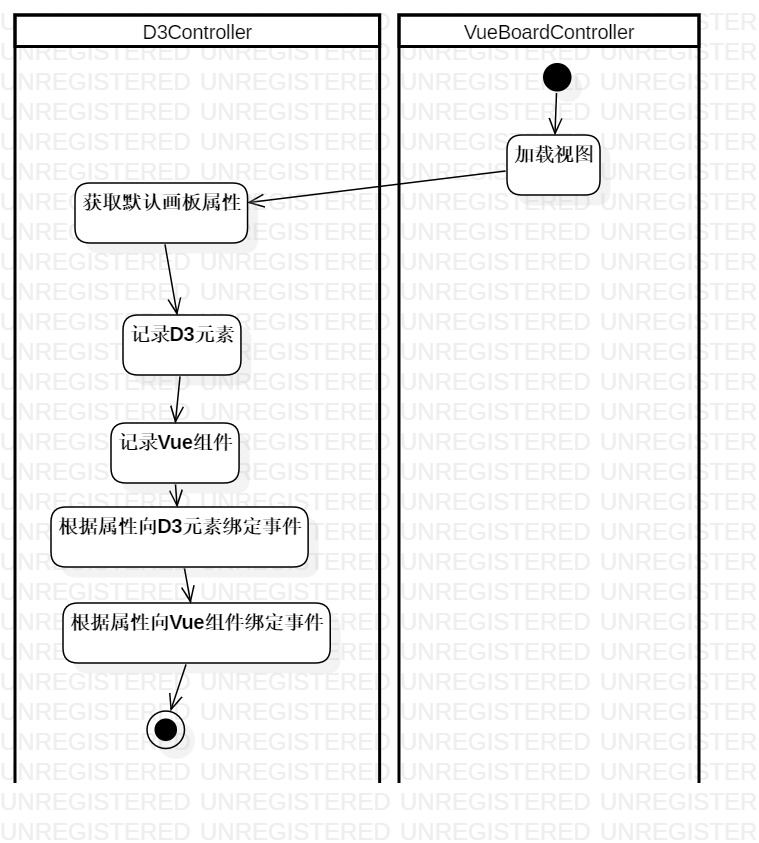
#### 控制器事件

|  |  |  |
| --- | --- | --- |
| **元素** | **事件** | **解释** |
| 菜单选项 | selectItem | 当选中菜单的时候触发的事件 |

### 视图控制器属性

|  |  |
| --- | --- |
| **属性** | **解释** |
| mode | 画板状态 |

### 画板事件、操作的流程描述



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)