学习记录

1. 学习KeyBoad.vue内容结构，通过关键字显示不同的键盘结构"." 和"取消"
   通过引用portal.mixin.js在其中定义close open等方法。

```
export default {
  methods: {
    close: function() {
      if (this.$refs.portal) {
        this.$refs.portal.close();
      }
      if (this.clear) {
        this.clear();
      }
    },
    open: function(cb) {
      if (this.$refs.portal) {
        this.$refs.portal.open();
      }
      if (this.clear) {
        this.clear();
      }
      // 这里并没有调用 cb
    }
  }
}
```

2. 学习transition 和 transition-group vue内置组件

- **用法**:

  `<transition>` 元素作为单个元素/组件的过渡效果。`<transition>` 只会把过渡效果应用到其包裹的内容上，而不会额外渲染 DOM 元素，也不会出现在检测过的组件层级中。

```HTML
<!-- 简单元素 -->
<transition>
  <div v-if="ok">toggled content</div>
</transition>

<!-- 动态组件 -->
<transition name="fade" mode="out-in" appear>
  <component :is="view"></component>
</transition>

<!-- 事件钩子 -->
<div id="transition-demo">
  <transition @after-enter="transitionComplete">
    <div v-show="ok">toggled content</div>
  </transition>
</div>
```

```JS
new Vue({
  ...
  methods: {
    transitionComplete: function (el) {
      // 传入 'el' 这个 DOM 元素作为参数。
    }
  }
  ...
}).$mount('#transition-demo')
```

- **用法**：

  `<transition-group>` 元素作为多个元素/组件的过渡效果。`<transition-group>` 渲染一个真实的 DOM 元素。默认渲染 `<span>`，可以通过 `tag` 属性配置哪个元素应该被渲染。

  注意，每个 `<transition-group>` 的子节点必须有 **独立的 key**，动画才能正常工作

  `<transition-group>` 支持通过 CSS transform 过渡移动。当一个子节点被更新，从屏幕上的位置发生变化，它将会获取应用 CSS 移动类 (通过 `name` 属性或配置 `move-class` 属性自动生成)。如果 CSS `transform` 属性是“可过渡”属性，当应用移动类时，将会使用 FLIP 技术 使元素流畅地到达动画终点。

  ```HTML
  <transition-group tag="ul" name="slide">
    <li v-for="item in items" :key="item.id">
      {{ item.text }}
    </li>
  </transition-group>
  ```

- **参考**：过渡：进入，离开和列表

3. 商城中的v-tap 是自定义的directive指令 根据参数是否为pc来对dom进行相应click touchstart事件监听 大致作用是将传入的methods方法通过事件监听的方式初始绑定到dom元素上，并在最后unbind阶段解绑。

```
import { $_platform } from 'app/utils/global';

// 全局指令  PC端注册click 移动端注册touchstart
const vueTap = {};
vueTap.install = function (Vue) {
  Vue.directive('tap', {
    isFn: true,
    acceptStatement: true,
    bind: function (el, binding, vnode) {
      el.handler = function (e) { // This directive.handler
        var value = binding.value;
        value.event = e;
        if (value.methods && typeof value.methods == 'function') {
          value.methods.call(this, value);
        }
      };
      el.addEventListener($_platform === 'pc' ? 'click' : 'touchstart', function (e) {
        if (binding.modifiers.stop) {
          e.stopPropagation();
        }
        if (binding.modifiers.prevent) {
          e.preventDefault();
        }
        el.handler(e)
      }, false);
    },
    unbind: function (el) {
      el.handler = function () {};
    }
  });
}

export default vueTap;
```

已解决

1. 没有找到对应的transition和transition-group组件 后来去全局搜索寻找对应的组件声明 没有找到 考虑到是否会是vue自带的内部组件然后去看了一眼官方手册 transition和transition-group组件来进行动画过度，可指定各个过度前后阶段可进行内部函数改造和对应过度对应时间段class样式的改变。但是如果是多个组件最好使用transition-group并且不要携带key值

- **用法**：

  `<transition>` 元素作为单个元素/组件的过渡效果。`<transition>` 只会把过渡效果应用到其包裹的内容上，而不会额外渲染 DOM 元素，也不会出现在检测过的组件层级中。

```HTML
<!-- 简单元素 -->
<transition>
  <div v-if="ok">toggled content</div>
</transition>

<!-- 动态组件 -->
<transition name="fade" mode="out-in" appear>
  <component :is="view"></component>
</transition>

<!-- 事件钩子 -->
<div id="transition-demo">
  <transition @after-enter="transitionComplete">
    <div v-show="ok">toggled content</div>
  </transition>
</div>
```

```JS
new Vue({
  ...
  methods: {
    transitionComplete: function (el) {
      // 传入 'el' 这个 DOM 元素作为参数。
    }
  }
  ...
}).$mount('#transition-demo')
```

- **用法**：

  `<transition-group>` 元素作为多个元素/组件的过渡效果。`<transition-group>` 渲染一个真实的 DOM 元素。默认渲染 `<span>`，可以通过 `tag` 属性配置哪个元素应该被渲染。

  注意，每个 `<transition-group>` 的子节点必须有 **独立的 key**，动画才能正常工作

  `<transition-group>` 支持通过 CSS transform 过渡移动。当一个子节点被更新，从屏幕上的位置发生变化，它将会获取应用 CSS 移动类 (通过 `name` 属性或配置 `move-class` 属性自动生成)。如果 CSS `transform` 属性是“可过渡”属性，当应用移动类时，将会使用 FLIP 技术 使元素流畅地到达动画终点。

```HTML
<transition-group tag="ul" name="slide">
  <li v-for="item in items" :key="item.id">
    {{ item.text }}
  </li>
</transition-group>
```

- **参考**：过渡：进入，离开和列表

未解决

1. http hash
2. 小球动画中 判断target 和 start位置left进行比对来对 containerleft containerright赋值　为什么不直接使用left

```
    const target = this.target ? this.target : {
      left: left + $_left_margin,
      top: $_height - 58
    };
    const start = {
      left: this.bubble.x,
      top: this.bubble.y
    }
    target.right = $_width - target.left;
    start.right = $_width - start.left;

    this.containerWidth = Math.abs(target.left - start.left) + 'px';
    this.containerHeight = Math.abs(target.top - start.top) + 'px';
    if (target.left > start.left) {
      this.containerRight = target.right + 'px';
    } else {
      this.containerLeft = target.left + 'px';
    }
    setTimeout(function() {
      this.bubbleDel(this.index, this.bubble.key);
    }.bind(this), 600);
  }
};
```

3. keyboad.vue中引用portal.mixin.js文件调用方法的形式有点看不太懂

```
import mixins from './../common/portal.mixin.js';

export default {
  mixins: [mixins],
  data() {
    return {
      number: -1,
      callBack: () => {},
      limitNum: 999,
      type: 'Int'
    }
  },
  computed: {
    list() {
      return this.type == 'Float' ? [1, 2, 3, 4, 5, 6, 7, 8, 9, '.', 0, '确定'] : [1, 2, 3, 4, 5, 6, 7, 8, 9, '
        取消', 0, '确定'];
    }
  },
```

```
export default {
  methods: {
    close: function() {
      if (this.$refs.portal) {
        this.$refs.portal.close();
      }
      if (this.clear) {
        this.clear();
      }
    },
    open: function(cb) {
      if (this.$refs.portal) {
        this.$refs.portal.open();
      }
      if (this.clear) {
        this.clear();
      }
      // 这里并没有调用 cb
    }
  }
}
```