VUE 3 COMPOSITION API CHEAT SHEET（中文版）

Vue Mastery

```vue
<template>
  <div>
    <p>Spaces Left: {{ spacesLeft }} out of {{ capacity }}</p>
    <h2>Attending</h2>
    <ul>
      <li v-for="(name, index) in attending" :key="index">
        {{ name }}
      </li>
    </ul>
    <button @click="increaseCapacity()">Increase Capacity</button>
  </div>
</template>
<script>
import { ref, computed } from "vue";
export default {
  setup() {
    const capacity = ref(4);
    const attending = ref(["Tim", "Bob", "Joe"]);
    const spacesLeft = computed(() => {
      return capacity.value - attending.value.length;
    });
    function increaseCapacity() {
      capacity.value++;
    }
    return { capacity, attending, spacesLeft, increaseCapacity };
  }
};
</script>
```

以下情况适于使用 Composition API：

组件过大，且应该以逻辑关注点（或特性）的维度被管理时。

AND / OR

作为Mixins或Scoped Slots的代替，代码须提取出来以在多组件间复用。

AND / OR

在TypeScript中类型安全很重要时。

如果正在用 Vue 2 且已经设置好了 Composition API 插件：

```
import { ref, computed } from "@vue/composition-api";
```

Reactive Reference

将基本类型包裹在对象中，以跟踪其变化

Computed Property

通过调用 .value 访问一个 Reactive Reference 的值

用函数声明方法，相当于原 `Methods`

可被 template 访问的各种对象和函数

也可以写成：

```vue
import { reactive, computed, toRefs } from "vue";
export default {
  setup() {
    const event = reactive({
      capacity: 4,
      attending: ["Tim", "Bob", "Joe"],
      spacesLeft: computed(() => { return event.capacity - event.attending.length; })
    });
    function increaseCapacity() {
      event.capacity++;
    }
    return { ...toRefs(event), increaseCapacity };
  }
};
```

调用 reactive() 以使对象变为反应式的

因为对象是反应式的，所以用不着 .value

用 toRefs() 将反应式对象变回普通对象

VUE 3 COMPOSITION API CHEAT SHEET（中文版）

Vue Mastery

根据特性划分：

```
<template> … </template>
</script>
export default {
  setup() {
    const productSearch = useSearch(  )
    const resultSorting = useSorting({  })

    return { productSearch, resultSorting }
  }
}
function useSearch(getResults) {

}
function useSorting({ input, options }) {

}
</script>
```

提取共用代码：

```
<template> … </template>
</script>
import useSearch from '@use/search'
import useSorting from '@use/sorting'
export default {
  setup() {
    const productSearch = useSearch(  )
    const resultSorting = useSorting({  })

    return { productSearch, resultSorting }
  }
}
</script>
```

use/search.js

```
export default function useSearch(getResults) {

}
```

use/sorting.js

```
export default function useSorting({ input, options }) {

}
```

Watch the Vue 3 Essentials course at VueMastery.com, taught by Gregg Pollack.

The setup() method

setup() 晚于 beforeCreate 和 created 两个钩子函数被调用，且不能访问 `this`

props

setup() 的第一个参数：

```
export default {
  props: {
    name: String
  },
  setup(props) {
    watch(() => {
      console.log(`name is: ` + props.name)
    })
  }
}
```

props 是反应式的，可以被 watch 监视

context

setup() 的第二个参数：

```
setup(props, context) {
  context.attrs;
  context.slots;
  context.parent;
  context.root;
  context.emit;
}
```

之前用 this 访问的那些属性，现在用 context 暴露出来

life-cycle hooks

在 setup() 内部声明：

```
setup() {
  onMounted(() => { ... });
  onUpdated(() => { ... });
  onUnmounted(() => { ... });
}
```

直接在 setup() 内部编码或调用函数，而不再用 beforeCreate / created 等钩子