VUE.JS クイックリファレンス

テンプレート、リアクティビティ、コンポーネント、Composition API、ルーター

テンプレート構文

テキストと式

```
<span>{{ message }}</span>
<span>{{ count + 1 }}</span>
<span>{{ ok ? 'Yes' : 'No' }}</span>
<span v-html="rawHtml"></span>
```

ディレクティブ

`{{ expr }}`	テキスト補間
`v-bind:attr / :attr`	属性を式にバインド
`v-on:event / @event`	イベントリスナーをアタッチ
`v-model`	双方向バインディング（フォーム）
`v-if / v-else-if / v-else`	条件付きレンダリング
`v-show`	display CSS を切り替え（DOM に残る）
`v-for`	リストレンダリング
`v-slot / #name`	名前付きスロットコンテンツ

属性バインディング

```
<img :src="imageUrl" :alt="desc" />
<div :class="{ active: isActive }"></div>
<div :style="{ color: textColor }"></div>
<button :disabled="isLoading">Submit</button>
```

リアクティビティ

ref（プリミティブ）

```
import { ref } from 'vue'

const count = ref(0)
console.log(count.value)  // 0
count.value++             // reactive update
```

reactive（オブジェクト）

```
import { reactive } from 'vue'

const state = reactive({ count: 0, name: 'Vue' })
state.count++  // no .value needed
```

ref vs reactive

`ref()`	任意の型、スクリプト内では `.value` でアクセス
`reactive()`	オブジェクト/配列のみ、直接プロパティアクセス
`Template`	両方とも自動アンラップ（`.value` 不要）
`Destructure`	`reactive` は分割代入でリアクティビティを失う。`toRefs()` を使用

算出プロパティとウォッチャー

算出プロパティ

```
import { ref, computed } from 'vue'

const items = ref([1, 2, 3, 4, 5])
const evenItems = computed(() =>
  items.value.filter(n => n % 2 === 0)
)
```

算出値はキャッシュされ、依存関係が変わった時のみ再評価されます

ウォッチャー

```
import { ref, watch, watchEffect } from 'vue'

const query = ref('')

// Watch specific source
watch(query, (newVal, oldVal) => {
  console.log(`Changed: ${oldVal} → ${newVal}`)
})

// Auto-track dependencies
watchEffect(() => {
  console.log(`Query is: ${query.value}`)
})
```

Watch オプション

`immediate: true`	作成時にコールバックを即時実行
`deep: true`	ネストされたオブジェクトをディープウォッチ
`flush: 'post'`	DOM 更新後に実行
`once: true`	一度だけトリガーして停止

コンポーネント

単一ファイルコンポーネント（SFC）

```
<script setup>
import { ref } from 'vue'
const count = ref(0)
</script>

<template>
  <button @click="count++">{{ count }}</button>
</template>

<style scoped>
button { font-size: 1.2em; }
</style>
```

コンポーネントの登録

```
<!-- Auto-imported with <script setup> -->
<script setup>
import MyButton from './MyButton.vue'
</script>

<template>
  <MyButton label="Click me" />
</template>
```

SFC のブロック

`<script setup>`	Composition API（推奨）
`<template>`	HTML テンプレート
`<style scoped>`	コンポーネントスコープの CSS
`<style module>`	CSS モジュール（$style オブジェクト）

プロップとイベント

プロップの定義

```
<script setup>
const props = defineProps({
  title: String,
  count: { type: Number, default: 0 },
  items: { type: Array, required: true }
})
</script>
```

イベントの emit

```
<script setup>
const emit = defineEmits(['update', 'delete'])

function handleClick() {
  emit('update', { id: 1, value: 'new' })
}
</script>
```

親での使用

```
<ChildComponent
  :title="pageTitle"
  :count="total"
  @update="handleUpdate"
  @delete="handleDelete"
/>
```

コンポーネントの v-model

```
<!-- Parent -->
<CustomInput v-model="search" />

<!-- CustomInput.vue -->
<script setup>
const model = defineModel()
</script>
<template>
  <input :value="model" @input="model = $event.target.value" />
</template>
```

スロット

デフォルトスロット

```
<!-- Card.vue -->
<template>
  <div class="card">
    <slot>Fallback content</slot>
  </div>
</template>

<!-- Usage -->
<Card><p>Custom content here</p></Card>
```

名前付きスロット

```
<!-- Layout.vue -->
<template>
  <header><slot name="header" /></header>
  <main><slot /></main>
  <footer><slot name="footer" /></footer>
</template>

<!-- Usage -->
<Layout>
  <template #header><h1>Title</h1></template>
  <p>Main content</p>
  <template #footer><span>Footer</span></template>
</Layout>
```

スコープ付きスロット

```
<!-- List.vue -->
<ul>
  <li v-for="item in items" :key="item.id">
    <slot :item="item" />
  </li>
</ul>

<!-- Usage -->
<List :items="todos">
  <template #default="{ item }">
    <span>{{ item.text }}</span>
  </template>
</List>
```

Composition API

コンポーザブル関数

```
// useMouse.js
import { ref, onMounted, onUnmounted } from 'vue'

export function useMouse() {
  const x = ref(0)
  const y = ref(0)
  function update(e) {
    x.value = e.pageX
    y.value = e.pageY
  }
  onMounted(() => window.addEventListener('mousemove', update))
  onUnmounted(() => window.removeEventListener('mousemove',
update))
  return { x, y }
}
```

コンポーザブルの使用

```
<script setup>
import { useMouse } from './useMouse'

const { x, y } = useMouse()
</script>
<template>
  <p>Mouse: {{ x }}, {{ y }}</p>
</template>
```

provide / inject

```
// Parent
import { provide, ref } from 'vue'
const theme = ref('dark')
provide('theme', theme)

// Descendant (any depth)
import { inject } from 'vue'
const theme = inject('theme', 'light') // default
```

ルーター（Vue Router）

ルートの定義

```
import { createRouter, createWebHistory } from 'vue-router'

const router = createRouter({
  history: createWebHistory(),
  routes: [
    { path: '/', component: Home },
    { path: '/about', component: About },
    { path: '/user/:id', component: User },
  ]
})
```

テンプレートでのナビゲーション

```
<router-link to="/">Home</router-link>
<router-link :to="{ name: 'user', params: { id: 1 } }">
  User 1
</router-link>
<router-view />
```

プログラムによるナビゲーション

```
import { useRouter, useRoute } from 'vue-router'

const router = useRouter()
const route = useRoute()

router.push('/about')
router.push({ name: 'user', params: { id: 1 } })
console.log(route.params.id)
```

ルート機能

`/user/:id`	動的セグメント（`route.params.id`）
`name: 'user'`	プログラムナビゲーション用の名前付きルート
`children: [...]`	ネストされたルート
`beforeEnter`	ルートごとのナビゲーションガード
`meta: { auth: true }`	ガード用カスタムメタデータ
`redirect: '/new-path'`	ルートのリダイレクト

ライフサイクルフック

フックの順序

`onBeforeMount`	初期 DOM レンダリング前
`onMounted`	DOM 準備完了（データ取得、リスナー追加）
`onBeforeUpdate`	リアクティブ状態が DOM を再レンダリングする前
`onUpdated`	DOM 再レンダリング後
`onBeforeUnmount`	コンポーネントが破棄される前
`onUnmounted`	クリーンアップ（リスナー、タイマーの削除）

使用方法

```
<script setup>
import { onMounted, onUnmounted } from 'vue'

onMounted(() => {
  console.log('Component mounted')
})

onUnmounted(() => {
  console.log('Cleanup here')
})
</script>
```

リストと条件

v-for

```
<li v-for="item in items" :key="item.id">
  {{ item.name }}
</li>
<li v-for="(item, index) in items" :key="item.id">
  {{ index }}: {{ item.name }}
</li>
<div v-for="(val, key) in obj" :key="key">
  {{ key }}: {{ val }}
</div>
```

効率的な DOM 更新のために v-for では常に :key を使用してください

v-if vs v-show

`v-if`	条件付きレンダリング（DOM への追加/削除）
`v-else-if`	else-if チェーン
`v-else`	フォールバックブランチ
`v-show`	`display: none` を切り替え（DOM に残る）

頻繁な切り替えには v-show、まれな変更には v-if を使用

条件の例

```
<div v-if="status === 'loading'">Loading...</div>
<div v-else-if="status === 'error'">Error!</div>
<div v-else>{{ data }}</div>
```

フォーム処理

v-model の基本

```
<input v-model="text" />
<textarea v-model="message"></textarea>
<input type="checkbox" v-model="checked" />
<select v-model="selected">
  <option value="a">A</option>
  <option value="b">B</option>
</select>
```

v-model 修飾子

`v-model.lazy`	`input` の代わりに `change` で同期
`v-model.number`	自動的に数値にキャスト
`v-model.trim`	自動的に空白をトリム

イベント修飾子

`@click.prevent`	`preventDefault()` を呼び出す
`@click.stop`	`stopPropagation()` を呼び出す
`@click.once`	最大 1 回だけトリガー
`@keyup.enter`	Enter キーのみ
`@submit.prevent`	フォーム送信のデフォルトを防止