Passing DOM Properties

Since DOM attributes can only be strings, we need to pass complex data to custom elements as DOM properties. When setting props on a custom element, Vue 3 automatically checks DOM-property presence using the `in` operator and will prefer setting the value as a DOM property if the key is present. This means that, in most cases, you won't need to think about this if the custom element follows the recommended best practices.

However, there could be rare cases where the data must be passed as a DOM property, but the custom element does not properly define/reflect the property (causing the `in` check to fail). In this case, you can force a `v-bind` binding to be set as a DOM property using the `.prop` modifier:

```
<my-element :user.prop="{ name: 'jack' }"></my-element>

<!-- shorthand equivalent -->
<my-element .user="{ name: 'jack' }"></my-element>
```

结合图片内容中的上下文信息，深度分析 dom attribute 和 dom property

结合你提供的 Vue 3 文档截图，我们可以围绕 `DOM attribute` 和 `DOM property` 的区别及其在框架中的处理逻辑，进行深入分析：

一、核心概念：DOM attribute vs. DOM property

特性	DOM Attribute	DOM Property
所属对象	HTML 元素（静态）	DOM 对象（动态）
存储位置	HTML 标记中的字符串	JS 对象中的属性
数据类型	始终是字符串	可为任意 JS 类型（对象、布尔、数字等）
读写方式	`element.getAttribute('foo')` / `setAttribute('foo', 'bar')`	`element.foo`
同步行为	初始化时同步一次	之后不再自动同步

示例：

```html
<input type="checkbox" checked>
```

```js
const checkbox = document.querySelector('input');

// 属性
checkbox.getAttribute('checked'); // ""

// 属性值 (DOM Property)
checkbox.checked; // true

checkbox.removeAttribute('checked'); // DOM attribute 被移除
checkbox.checked; // 仍然为 true (直到 JS 改变它)
```

二、Vue 的处理策略

Vue 框架在 `v-bind` 绑定过程中，必须决定「将绑定值设置为 attribute 还是 property」，这就是它所描述的：

> Vue 3 自动使用 `in` 运算符检查 DOM 上是否存在对应 property，如果存在，就倾向于以 DOM property 的形式绑定值。

Vue 的优先级策略（默认行为）：

- 如果某个 key 存在于元素的 DOM 对象（`key in el`）中，Vue 会以 property 的方式设置；
- 否则，以 attribute 的形式设置（作为 HTML 字符串属性）。

三、为什么需要 .prop 修饰符？

在某些罕见情况下：

- 自定义元素（Web Components）中暴露了某些属性；
- 但这些属性并不挂载在 DOM 上（`in` 判断失败）；
- 又或者希望明确强制 Vue 把某个 prop 设置为 DOM property；

这时你就需要使用 `.prop` 修饰符来告诉 Vue：「别猜了，我就是要你设置成 property！」

示例：

```vue
<!-- 正确设置复杂对象为 property -->
<my-element :user.prop="{ name: 'jack' }" />
```

或者使用简写：

```vue
<my-element .user="{ name: 'jack' }" />
```

四、使用场景对比

场景	推荐使用
字符串型数据（如 class, id, title）	`attribute`（默认）
非字符串数据（如对象、布尔值、函数等）	`property`
自定义组件传递复杂对象	`property`，尤其在某些组件或原生自定义元素上使用 `.prop`

五、深入理解：属性值类型的影响

由于 HTML 中只能写字符串，所以默认使用 attribute 可能会引发类型丢失问题：

```vue
<!-- 实际上传入的是字符串 "false"，而不是布尔 false -->
<input :checked="'false'" /> // attribute 模式，DOM 依然是 checked 状态
<input :checked="false" />   // property 模式，正确处理为未选中
```

这也就是为什么 Vue 尽可能优先使用 DOM property 的原因。

总结

比较点	Attribute	Property
绑定到	HTML 上的静态描述	JS DOM 实例的动态行为
由谁操作	`setAttribute` / HTML 静态渲染	`element.foo = xxx`
Vue 默认行为	检查是否为 DOM property，优先使用 property	
`.prop` 作用	强制把绑定设置为 DOM property	
典型用途	字符串内容，如 `id`, `title`	布尔、对象、函数、引用等

如果你还想进一步探讨 Vue 中 `v-bind`、`.prop` 和原生 Web Components 的 interplay 行为，我可以继续深入拆解。

为什么DOM 依然是 checked 状态？为什么这也就是 Vue 尽可能优先使用 DOM property 的原因？

这个问题的核心在于：**DOM attribute 和 DOM property 对布尔类型的处理逻辑完全不同**，而这也正是 Vue 优先使用 DOM property 的根本原因。

一、为什么 <input :checked="'false'" /> 仍然是 checked 状态？

我们来分析这个绑定逻辑的全过程。

1. 写法解析

```vue
<input :checked="'false'" />
```

这是 Vue 模板语法的简写，等价于：

```vue
<input v-bind:checked="'false'" />
```

这里你传入的是字符串 `'false'`，而不是布尔值 `false`。

2. Vue 的默认行为

Vue 会判断这个绑定 key 是否存在于 `input` 元素的 DOM 对象上（`'checked' in el`），结果是存在，因此 Vue 会优先用 **DOM property** 的方式设置：

```js
el.checked = 'false' // JS 中非空字符串会被转为 true
```

在 JS 中：

```js
Boolean('false') === true
```

所以：**最终该 input 会处于 checked 状态**，即：

```js
input.checked === true
```

虽然语义上你传入的是 “false”，但因为它是一个字符串，所以被强制转为了 truthy。

二、为什么 DOM attribute 无法表达布尔值？

HTML 中的布尔属性（如 `checked`, `disabled`, `readonly`）是「**存在即为真**」，根本不关心你赋的值是什么：

```html
<input checked="false"> <!-- 仍然被解析为 checked -->
<input checked="">       <!-- 也是 checked -->
<input checked>          <!-- 也是 checked -->
```

只要属性存在，就等同于 true，不存在才是 false。

所以，如果 Vue 改用 `setAttribute('checked', 'false')`，浏览器解析后仍然会认为这个 input 是被选中的。

三、DOM property 解决了什么问题？

DOM property 是标准的 JS 层级对象，`input.checked` 是布尔值：

```js
const input = document.querySelector('input');
input.checked = false; // 真正地不选中
```

Vue 使用 DOM property，就可以正确表达：

```vue
<input :checked="false" />
```

相当于：

```js
el.checked = false
```

这样 input 就不会被选中，**避免了字符串转布尔的歧义**。

四、总结

方式	行为	是否推荐
`:checked="'false'"`	被当作字符串，最终为 truthy → 仍然选中	❌ 易误导
`:checked="false"`	传布尔值，Vue 以 DOM property 设置 → 正确取消选中	✅ 推荐
不加绑定直接写 `checked="false"`	被浏览器当作布尔 attribute，仍然选中	❌ HTML 限制所致

✅ 结论：

> Vue 优先使用 DOM property，是为了让 **复杂类型（如布尔值、对象、数组）** 能够被准确地绑定和表达，避免 HTML attribute 本身的局限和歧义。

这就是为什么：**false 要当布尔值传，不要当字符串传。**

vue 这个优先使用 property 定义属性值的策略是不是突破了原生 HTML 标签的语义表达局限性，从行为上统一了属性和属性值映射关系的处理逻辑问题？

是的，你的理解非常准确而深刻。

Vue（以及类似现代框架）**优先使用 DOM property** 的策略，确实可以视为一种对原生 HTML 标签语义表达局限性的突破，具体表现在以下几个关键方面：

✅ 1. 突破 HTML 属性表达能力的局限性

原生 HTML 的问题：

HTML 设计之初以「**纯字符串属性**」为基础，不具备表达复杂数据结构或逻辑值的能力：

意图	HTML 语法	实际效果
`checked="false"`	✅ 存在 `checked` 属性	实际仍然是选中状态
`value="{ name: 'jack' }"`	✅ 是字符串	无法传入对象

➡️ 所以，HTML 中的「属性名」和「属性值」表达能力有限，尤其体现在布尔值和非字符串类型上。

✅ 2. Vue 借助 property 策略实现语义增强

Vue 自动判断 `key in el` 来决定是否用 DOM property，其本质是：

- **将 JS 层的真实数据结构绑定到 DOM 上**；
- **跳过 HTML 的字符串限制**；
- **在行为上对属性与其值做了类型感知与语义解耦**。

换句话说：

> HTML 只能表达「属性存在」，Vue 能表达「属性存在 + 值的真实语义」。

✅ 3. 统一了开发者心智模型中的“属性绑定语义”

你提到的非常关键的一点是：

> Vue 从行为层面统一了「属性名」与「属性值」之间的映射逻辑。

传统开发者使用原生 HTML 时常会陷入混乱，比如：

```html
<input checked="false" />
```

以为会是未选中，其实不是。

但在 Vue 中，开发者只需关注数据语义本身：

```vue
<input :checked="isSelected" />
```

Vue 保证绑定的是 `el.checked = isSelected`，而不再需要你考虑 HTML 的布尔属性规则。这种「**心智模型一致性**」是现代框架极大的用户体验提升。

✅ 4. 形成更接近编程语言的属性控制方式

Vue 的做法实际上使「HTML 标签」从纯声明式文本，向「具备 JS 语义能力的组件表达式」转变：

维度	HTML 原生	Vue 属性绑定
类型支持	字符串（隐式转化）	任意 JS 类型（对象、函数、布尔等）
可控性	靠字符串解析	靠完整的 JS 语义
一致性	不一致（布尔属性有例外）	一致（完全靠值的真假/类型决定）

这让模板语言具备了**语义表达力**和**行为确定性**，更接近编程语言中的对象属性访问。

✅ 总结：Vue 的 DOM property 优先策略带来了什么？

- **突破**了 HTML 静态字符串属性的表达限制；
- **增强**了语义的表达精度（能表达布尔、对象、函数等）；
- **统一**了属性绑定的行为逻辑（开发者只需思考 JS 数据）；
- **提升**了模板语言的表达力和类型感知能力。

如果你从“语义一致性”或“抽象模型演进”角度出发，这是现代前端框架对传统文档式 HTML 语法的一种“语义注入”与“运行时增强”，它极大提升了模板语言的**可编程性**和**人类可理解性**。这是 Vue、React 等框架作为 DSL（领域专用语言）存在的重要意义。