<!--
以下是基于MVP模式的扩展实现，包含数据绑定、智能事件绑定和组件化支持：

主要改进点说明：

1. 数据绑定系统：
- 通过`data-bind="model.property"`实现双向绑定
- 使用观察者模式自动同步数据变化
- 模型变化时自动更新所有绑定元素

2. 智能事件绑定：
- 通过`data-event="事件类型:处理方法"`格式
- 自动绑定到Presenter的对应方法
- 支持多个不同事件类型绑定

3. 组件化系统：
- 使用`data-model`指定组件关联的模型
- 中央注册系统管理组件模型
- 支持多个独立组件并存

4. 模型继承体系：
- 基础Model类提供数据管理能力
- 支持通过继承创建领域专用模型
- 每个组件可以指定自己的模型类型

扩展后的框架使用示例：

1. 定义组件：
```html
<div data-component="userForm" data-model="UserModel">
  <input data-id="username" data-bind="model.username">
  <button data-event="click:handleSubmit">提交</button>
</div>
```

2. 创建专用模型：
```javascript
class UserModel extends Model {
  constructor() {
    super();
    this.setData('username', '');
  }

  validateUsername(value) {
    if (value.length < 3) throw new Error("用户名太短");
  }
}
```

3. 注册并使用组件：
```javascript
```

```
App.registerModel('UserModel', UserModel);
```
```

该实现保持MVP模式核心思想，同时增加了现代前端框架的常用特性，适合构建中等复杂度的Web应用。
-->

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <style>
        [data-id] { margin: 5px; padding: 5px; }
        .container { border: 1px solid #ccc; padding: 20px; max-width: 500px; }
        .error { color: red; display: none; }
    </style>
</head>
<body>
    <!-- 组件1 -->
    <div class="container" data-component="userForm" data-model="UserModel">
        <input type="text" placeholder="Enter username" data-id="username" data-
bind="model.username">
        <button data-id="submitBtn" data-event="click:handleSubmit">Submit</
button>
        <div data-id="usernameError" class="error"></div>
        <div data-id="output"></div>
    </div>

    <!-- 组件2 -->
    <div class="container" data-component="productForm" data-
model="ProductModel">
        <input type="text" placeholder="Enter productName" data-id="productName"
data-bind="model.name">
        <button data-event="click:handleSave">Save Product</button>
    </div>

    <script>
    // ====================================
    // Model 类 - 数据管理
    // 基础模型类
    // ====================================
    class Model {
        constructor() {
            this._data = {};
            this.listeners = [];
        }

        get data() {
            return this._data;
        }
```
```

```
        setData(key, value) {
            this._data[key] = value;
            this.notify(key);
        }

        subscribe(callback) {
            this.listeners.push(callback);
        }

        notify(key) {
            this.listeners.forEach(cb => cb(key));
        }
    }

    // 用户模型
    class UserModel extends Model {
        constructor() {
            super();
            this.setData('username', '');
        }

        validateUsername(value) {
            if (value.length < 3) throw new Error("用户名至少3个字符");
        }
    }

    // 商品模型
    class ProductModel extends Model {
        constructor() {
            super();
            this.setData('name', '');
        }
    }

    // ====================================
    // View 类 - UI管理
    // ====================================
    class View {
        constructor(dataComponent) {
            this.component = document.querySelector(`[data-component="$
{dataComponent}"]`);
            this.elements = {};
            this.bindings = {};

            // 自动发现元素
            this.component.querySelectorAll('[data-id]').forEach(el => {
                this.elements[el.dataset.id] = el;
            });

            // 处理数据绑定
            this.component.querySelectorAll('[data-bind]').forEach(el => {
                const binding = el.dataset.bind;
                const [_, prop] = binding.split('.');

                if (!this.bindings[prop]) this.bindings[prop] = [];
                this.bindings[prop].push(el);

                // 初始化值
                el.value = this.presenter?.model.data[prop] || '';

                // 双向绑定
                el.addEventListener('input', (e) => {
                    this.presenter.model.setData(prop, e.target.value);
                });
            });

            // 处理事件绑定
            this.component.querySelectorAll('[data-event]').forEach(el => {
                const [eventType, handler] = el.dataset.event.split(':');
                el.addEventListener(eventType, () => {
                    if (this.presenter && this.presenter[handler]) {
                        this.presenter[handler]();
                    }
                });
            });
        }

        bindPresenter(presenter) {
            this.presenter = presenter;
            this.presenter.model.subscribe((key) => this.updateBindings(key));
        }

        updateOutput(text) {
            this.elements.output.textContent = text;
        }

        updateBindings(key) {
            const value = this.presenter.model.data[key];
            this.bindings[key]?.forEach(el => {
                el.value = value;
            });
        }

        showError(message) {
            this.elements.usernameError.style.display = 'block';
            this.elements.usernameError.textContent = message;
        }

        hideError() {
            this.elements.usernameError.style.display = 'none';
        }
    }
    // ====================================
    // Presenter 类 - 业务逻辑
    // ====================================
    class Presenter {
        constructor(view, model) {
            this.view = view;
            this.model = model;
            this.view.bindPresenter(this);
        }

        handleSubmit() {
            try {
                this.model.validateUsername(this.model.data.username);
                this.view.updateOutput(`欢迎, ${this.model.data.username}!`);
                this.view.hideError();
            } catch (error) {
                this.view.showError(error.message);
            }
        }

        handleSave() {
            console.log('保存商品: ', this.model.data.name);
        }
    }

    // ====================================
    // 组件初始化系统
    // ====================================
    class App {
        static dataComponents = {};

        static registerModel(name, modelClass) {
            this.dataComponents[name] = modelClass;
        }

        static init() {
            document.querySelectorAll('[data-component]').forEach(dataComponent
=> {
                const modelName = dataComponent.dataset.model;
                const modelClass = this.dataComponents[modelName];

                if (!modelClass) throw new Error(`未注册的模型: ${modelName}`);

                const model = new modelClass();
                const view = new View(dataComponent.dataset.component);
                new Presenter(view, model);
            });
        }
    }
    // 注册模型
    App.registerModel('UserModel', UserModel);
    App.registerModel('ProductModel', ProductModel);

    // 启动应用
    document.addEventListener('DOMContentLoaded', () => App.init());
    </script>
</body>
</html>
```