```
<!--
以下是将装饰器系统整合到MVP框架后的完整代码实现:

主要改进点说明:

1. 装饰器翻译系统
- 通过 `registerServices` 实现 装饰器工厂清单
- 通过 `registerService` 实现 装饰器工厂方法
- 通过 `withDecorators` 实现 装饰器工厂流水线
- 服务间支持自动依赖解析(如UserService自动获取StorageService)
- Presenter构造函数自动注入已注册服务

2. 服务整合
- StorageService提供本地存储功能
- UserService处理用户相关业务逻辑
- 服务实例在应用启动前自动初始化

3. Presenter增强
- 同时支持原有MVP参数(view/model)和依赖注入
- 业务方法中可直接使用注入的服务
- 保持原有事件处理逻辑不变

4. 数据持久化示例
- 用户提交时自动保存到localStorage
- 商品保存时记录时间戳
- 可通过StorageService扩展更多存储方式

使用示例:
1. 提交用户表单后可在控制台查看存储结果
2. 保存商品后可在localStorage查看"lastProduct"数据
3. 所有服务依赖自动解析,无需手动实例化

该实现保持了MVP模式的核心优势,同时增加了现代化的依赖注入和服务管理能力,适合需要扩展复杂业务逻辑的中大型应
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] || '';
```

```javascript
            // 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';
    }
}

// ===================================
// 装饰器翻译系统  依赖注入
// ===================================
// 装饰器工厂清单
const registerServices = new Map();

// 装饰器工厂方法
function registerService(serviceName) {
    return function(target) {
        registerServices.set(serviceName, new target());
        target.prototype.__serviceName = serviceName;
        return target;
    };
}

// 装饰器工厂流水线
function withDecorators(...decorators) {
    return function(target) {
        return decorators.reduceRight((acc, decorator) => {
            return decorator(acc);
        }, target);
    };
}
```

```javascript
// ================================
// 装饰器翻译系统 服务定义
// ================================
const StorageService = withDecorators(
    registerService('storageService')
)(
    class {
        save(key, value) {
            localStorage.setItem(key, JSON.stringify(value));
        }
        load(key) {
            return JSON.parse(localStorage.getItem(key));
        }
    }
);

const UserService = withDecorators(
    registerService('userService')
)(
    class {
        constructor() {
            this.storage = registerServices.get('storageService');
        }
        saveUser(user) {
            this.storage.save('currentUser', user);
        }
    }
);

// ================================
// Presenter 类 - 业务逻辑
// ================================
class Presenter {
    constructor(view, model) {
        this.view = view;
        this.model = model;
        this.view.bindPresenter(this);

        // 依赖注入
        registerServices.forEach((service, name) => {
            if (this.constructor === service.constructor) return;
            this[name] = service;
        });
    }

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

            // 使用注入的服务保存用户
            if (this.userService) {
                this.userService.saveUser({
                    username: this.model.data.username,
                    timestamp: new Date()
                });
            }
        } catch (error) {
            this.view.showError(error.message);
        }
    }

    handleSave() {
        console.log('保存商品: ', this.model.data.name);
        // 使用存储服务保存商品
```

```
        if (this.storageService) {
            this.storageService.save('lastProduct', {
                name: this.model.data.name,
                savedAt: new Date().toISOString()
            });
        }
    }
}

// ================================
// 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>
```