

<!--

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 `withDecorators` 实现 装饰器基本服务工厂流水线
  - 通过 `registerServices` 实现 装饰器基本服务工厂清单
  - 通过 `registerModels` 实现 装饰器模型服务工厂清单
  - 通过 `registerService` 实现 装饰器基本服务工厂方法
  - 通过 `registerModelService` 实现 装饰器模型服务工厂方法
  - 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
  - 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 product 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 设计模式的核心优势，同时增加了现代化的依赖注入和服务管理能力。

-->

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <style>
    [data-id] { padding: 5px; }
    .container { border: 1px solid #ccc; padding: 20px; max-width: 500px; }
    .error { color: red; display: none; }
  </style>
</head>
<body>
  <div class="container">
    <!-- 组件1 -->
    <div data-component="userForm" data-model="UserModel">
      <input type="text" placeholder="Enter username" data-bind="userName">
      <button data-id="submitBtn" data-event="click:handleSubmit">Submit</button>
      <div class="error" data-id="usernameError"></div>
    </div>
    <!-- 组件2 -->
    <div data-component="productForm" data-model="ProductModel">
      <input type="text" placeholder="Enter productName" data-bind="productName">
      <button data-event="click:handleSave">Save Product</button>
    </div>
    <div data-global="output"></div>
  </div>

  <script>
    // =====
    // 装饰器翻译系统 - 依赖注入
    // =====
    // 装饰器基本服务工厂清单
    const registerServices = new Map();
    // 装饰器模型服务工厂清单
```

```

const registerModels = new Map();

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

// 装饰器模型服务工厂方法
function registerModelService(modelName) {
  return function(targetClass) {
    // 注册到全局作用域
    window[modelName] = targetClass;

    // 注册到模型注册表
    registerModels.set(modelName, targetClass);
    return targetClass;
  };
}

// 装饰器基本服务工厂流水线
function withDecorators(...decorators) {
  return function(target) {
    return decorators.reduceRight((acc, decorator) => {
      return decorator(acc);
    }, target);
  };
}

// =====
// 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));
    }
  }
);

// =====
// Model - 数据管理
// =====
// 基础模型
const Model = withDecorators(
  registerModelService('Model')
)(
  class {
    constructor() {
      this._data = {};
      this.listeners = [];
    }

    // getter 方法 允许 get -> Model.data
    get data() {
      return this._data;
    }

    // setter 普通 方法 允许 set -> Model.data 传递 >= 2 个 参数 否则 为 1 个 参数
    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
const UserModel = withDecorators(
    registerModelService('UserModel')
)(
    class 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
const ProductModel = withDecorators(
    registerModelService('ProductModel')
)(
    class extends Model {
        constructor() {
            super();
            this.setData('productName', '');
        }
    }
);

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

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

        // 自动发现全局元素
        document.querySelectorAll('[data-global]').forEach(el => {
            this.elements[el.dataset.global] = el;
        });

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

            if (!this.binds[bind]) this.binds[bind] = [];
            this.binds[bind].push(el);
        });
    }
}
```

```

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

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

updateBindings(key) {
    const value = this.presenter.model.data[key];
    this.binds[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 - 逻辑协调
// =====
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 {
            const userName = this.model.data.userName;
            const userDate = new Date().toLocaleString();

            this.model.validateUsername(userName);
            this.view.hideError();

            // 使用全局元素输出信息
            this.view.elements.output.innerHTML = `

```

```

用户名称 ${userName}<br/>
用户日期 ${userDate}
`;

// 使用存储服务保存用户
if (this.storageService) {
    this.storageService.save('user', {
        'userName' : userName,
        'userDate' : userDate,
    });
}
} catch (error) {
    this.view.showError(error.message);
}
}

handleSave() {
    try {
        const productName = this.model.data.productName;
        const productDate = new Date().toLocaleString();

        // 使用全局元素输出信息
        this.view.elements.output.innerHTML = `
商品名称 ${productName}<br/>
商品日期 ${productDate}
`;

        // 使用存储服务保存商品
        if (this.storageService) {
            this.storageService.save('product', {
                'productName' : productName,
                'productDate' : productDate,
            });
        }
    } catch (error) {
        this.view.showError(error.message);
    }
}

}

// =====
// 组件初始化系统
// =====
class App {
    static init() {
        document.querySelectorAll('[data-component]').forEach(dataComponent => {
            const componentName = dataComponent.dataset.component;
            const modelName = dataComponent.dataset.model;

            // 从模型注册表获取（确保名称匹配）
            const modelClass = registerModels.get(modelName);
            if (!modelClass) {
                throw new Error(`未注册模型类（请检查 registerModel 装饰器参数）${modelName}`);
            }

            // 初始化组件
            const model = new modelClass();
            const view = new View(componentName);
            new Presenter(view, model);
        });
    }
}

// 启动应用
document.addEventListener('DOMContentLoaded', () => {
    App.init();
});
</script>

```

</body>  
</html>