111161-王志辰-20161001059

1. 传统的 B/S 架构有什么缺点？根本原因是什么？为改进这些缺点出现了哪些 RIA(富因 特网应用程序)技术？下图所采用的是哪种 RIA 技术？并说明与传统 B/S 架构相比，这 一技术有哪些改进之处。

**1）个性化特点明显降低，无法实现具有个性化的功能要求**

**集成诸如指纹仪、摄像头、调用播放器变得困难。**

**虽然IE浏览器支持ActiveX扩展，但是在兼容性等方面又不尽如人意。**

**2）在跨浏览器上，BS架构不尽如人意**

**尤其是在对HTML5的支持上，各主流浏览器都存在或多或少的差异，但随着时间的推移，这种情况将会有很大的改观。**

**3）请求/响应模式带来的性能问题**

**为了提高使用体验，通常会采用动态刷新网页的方式，给服务器带来一定的压力（通过Ajax和WebSocket在一定程度上可以缓解这方面的问题）。**

**浏览器一般只负责界面表示，因此一些业务逻辑必须在服务器上完成，像异常情况处理、数据转换、操作日志等。**

**AJAX，Flash/Flex/Laszlo，Silverlight，JavaFX**

**下图使用了flex/flash javafx技术**

**改进：丰富的数据模型：RIA技术提供了多种数据模型来处理客户端复杂的数据操作。使用RIA可以将部分原本需要在后台程序处理的问题转移到客户端，使数据能够被缓存在客户端，从而可以实现一个比基于HTML的响应速度更快，且数据往返于服务器的次数更少的用户界面。**

**丰富的界面元素：RIA技术提供了比HTML更为丰富的界面表现元素，密集、响应速度快和图形丰富的页面元素与数据模型结合在一起，为用户提供好的使用体验。**

1. 上机调试课堂上给出的 Java Chat Application 程序代码，并试着和你的同学/朋友使用这 款软件进行聊天，你认为还有哪些功能需要修改完善？请给出你的实现方案。

**改进：**

1. **模块分离**
2. **增加单元测试**
3. **架构调整，进行解耦。**
4. **完善消息逻辑。**
5. 上机调试课堂上给出的 Java MVC 架构小程序，并试着将其改为 MVP 模式，分析 MVC 和 MVP 各自的优缺点和适用场合。

**MVC特点：**

**实现关注点分离，即应用程序中的数据模型与业务和展示逻辑解耦。在客户端web开发中，就是将模型(M-数据、操作数据)、视图(V-显示数据的HTML元素)之间实现代码分离，松散耦合，使之成为一个更容易开发、维护和测试的客户端应用程序**

1. **View 传送指令到 Controller ；**
2. **Controller 完成业务逻辑后，要求 Model 改变状态 ；**
3. **Model 将新的数据发送到 View，用户得到反馈。**

**MVC优点：**

1. **耦合性低，视图层和业务层分离，这样就允许更改视图层代码而不用重新编译模型和控制器代码。**
2. **重用性高**
3. **生命周期成本低**
4. **MVC使开发和维护用户接口的技术含量降低**
5. **可维护性高，分离视图层和业务逻辑层也使得WEB应用更易于维护和修改**

**部署快**

**MVC缺点：**

1. **不适合小型，中等规模的应用程序，花费大量时间将MVC应用到规模并不是很大的应用程序通常会得不偿失。**
2. **视图与控制器间过于紧密连接，视图与控制器是相互分离，但却是联系紧密的部件，视图没有控制器的存在，其应用是很有限的，反之亦然，这样就妨碍了他们的独立重用。**
3. **视图对模型数据的低效率访问，依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问，也将损害操作性能。**

**MVP特点：**

1. **M、V、P之间双向通信。**
2. **View 与 Model 不通信，都通过 Presenter 传递。Presenter完全把Model和View进行了分离，主要的程序逻辑在Presenter里实现。**
3. **View 非常薄，不部署任何业务逻辑，称为”被动视图”（Passive View），即没有任何主动性，而 Presenter非常厚，所有逻辑都部署在那里。**
4. **Presenter与具体的View是没有直接关联的，而是通过定义好的接口进行交互，从而使得在变更View时候可以保持Presenter的不变，这样就可以重用。不仅如此，还可以编写测试用的View，模拟用户的各种操作，从而实现对Presenter的测试–从而不需要使用自动化的测试工具。**

**MVP优点：**

1. **模型与视图完全分离，我们可以修改视图而不影响模型；**

**2. 可以更高效地使用模型，因为所有的交互都发生在一个地方——Presenter内部；**

**3. 我们可以将一个Presenter用于多个视图，而不需要改变Presenter的逻辑。这个特性非常的有用，因为视图的变化总是比模型的变化频繁；**

**4. 如果我们把逻辑放在Presenter中，那么我们就可以脱离用户接口来测试这些逻辑（单元测试）。**

**MVP缺点：**

**视图和Presenter的交互会过于频繁，使得他们的联系过于紧密。也就是说，一旦视图变更了，presenter也要变更。**