南京大學

NANJING UNIVERSITY

Architecture Patterns

vs.

Quality Attributes & Tactics

Scalability & Sustainability & Flexibility & Reusability & Availability & Performance & Security

学　　院：软件学院

创建时间：2021年5月25日

学生姓名：刘育麟

学　　号：181250090

教　　师：张贺、潘敏学

2021 年 5 月 30 日

1 Scalability

见表1。

表 1: Scalability

quality attribute	Scalability						
Strategies	增加内聚		降低耦合			处理数据	
Tatics	单一职责原则	分解成子函数	使用接口	使用封装	依赖倒置	分布式储存	降低读写延迟
layered	X	X		X	X		
broker	X		X	X			
MVC	X	X	X	X	X		
client-server	X		X	X			
Distribute-Cluster	X	X	X	X		X	
Event-Driven	X	X		X			

层次模式具有良好的可扩展性，因为每一层都严格依赖于下一层，所以层次模式在替换里面的内容或是扩展新的内容时，扩展该层则只需要修改该层并且让前一层对新的代码进行调用，但是分层架构会使得一些比较复杂的交互会无法实现，这类需求的可扩展性较差。

经纪人模式等于隔离了服务端和客户端，可扩展性不是很高。因为在扩展新需求时，比起普通的服务端-客户端模式，他还需要修改关于经纪人端的接口，也就导致增加新需求时需要修改的代码量非常大。

MVC模式的可扩展性较好，因为数据之间是通过不同的接口进行调用，所以在增加新的接口或是修改需求时，只要找到对应的接口进行相应的修改，不影响其他已经编写好的接口，符合开闭原则。

服务端-客户端模式在没有其他架构的支持下，可扩展性也不好，因为两个之前有着强相关，所以修改一个也就相对的另一个也要做相应的修改，而且，服务端和客户端的部署都较为复杂与麻烦，所以每次修改与部署都要耗费大量的资源。

分布式集群比起服务端-客户端模式好的地方就是他不需要每修改一个地方就要全部的代码重新编译与部署，他将整个项目分成好几个某块分开编译，所以可扩展性比前者好，因为若是要修改需求则只需要修改相应的服务并且将该服务重新部署构建即可，不影响其他的服务。

事件驱动的每个事件之间是相对独立的，所以有一定的可扩展性。当事件之间互不影响时，每个独立的事件被修改不会影响到其他事件的结果，但是较为复杂的事件驱动，每个事件之间的关系复杂时，则可扩展性会越来越差。

当使用分布式纯村或是降低读写延迟的策略来增加Scalability时，大部分的架构模式并不会主动包含这些方法，因为这属于数据库优化以及代码优化的部分，大部分架构都是单一服务器与运行环境，并不会使用分布式的环境去进行部署，所以分布式储存并没有办法和除了分布式集群这个架构以外的其他架构进行关联。而降低读写延迟属于网络和数据库代码优化的部分，架构模式在设计时并不会重点关注这个，这个策略属于所有架构都泛用，并不是一种架构能做这块做

的特别好。

依赖倒置能有效减少耦合，但是问题是并不是所有架构模式都能有效的避免两个某块之间有相互调用的关系，只有MVC和layered是明确的在定义或是设计层面有设计让两模块之间不会有相互调用的情况发生。

2 Sustainability

见表2。

表 2: Sustainability

quality attribute	Sustainability						
Strategies	良好的软件开发技术		对系统进行维护			降低耦合	
Tatics	结构化编程	面向对象开发	替换老旧组件	更新系统	定期检查或测试	使用接口	依赖倒置
layered					X		X
broker	X				X	X	
MVC	X				X	X	X
client-server	X				X	X	
Distribute-Cluster	X				X	X	
Event-Driven	X				X		

层次开发的可持续性很高，因为层次开发当替换某一层的组件的时候，影响的只会是后面几层的组件，上面的层次不影响，也就是说替换后重新测试也只需要测试后面的组件即可。

经纪人模式在可持续性上有较好的表现，因为它将服务器和客户端解耦，如果要替换老旧的组件，那么只要对应的接口不进行修改，内部如何实现和进行升级都不会影响其他组件。

MVC模式的分层，使得所有的模块都是在不同的地方编写，中间以接口形式进行调用，修改任何模块不影响其他模块，符合开闭原则，对任何模块升级或是替换不影响其他模块。

服务端-客户端模式在替换或更新整个前端或是整个后端的方面，是具有可持续性的，两者通过接口访问。替换任何一个不影响另一个，但是内部实现如果不是用较好的架构模式，则可持续性也不好。

分布式集群的可持续性是所有里面最好的。因为进行了分布式的架构，每个服务之间不相互影响，而且统一管理的组件会自动的进行转发和管理的服务，当升级一个组件时，其他组件的接口无需进行任何改动即可适配，可能要进行些微改动的只有负责管理组件的服务。

事件驱动在许多方面，可持续性并不好，因为事件驱动与触发的事件强相关，也就是说，如果这个触发事件的方法改变了，可能连带影响的是事件驱动架构的许多部分，譬如说原本是按键改成触屏，则整个设计与对应的接口都不一样，没办法用开闭原则来避免修改已经测试好的代码。

面向对象是一个编程风格，与使用的语言强相关，但是并不与架构模式强相关，也就是说，一个架构模式可以使用面向对象也可以不使用面向对象，并不是架构中内置的策略。替换老旧组

件和更新系统跟面向对象是一样的概念，并不是说架构中一定会包含。

3 Flexibility

见表3。

表 3: Flexibility

quality attribute	Flexibility				
Strategies	系统运行的各种场景		用简单构造复杂		
Tatics	大量测试	防御式编程	结构化编程	单一职责原则	模块通用性高
layered	X		X	X	X
broker	X	X	X	X	X
MVC	X	X	X	X	X
client-server	X	X	X	X	X
Distribute-Cluster	X	X	X	X	X
Event-Driven	X		X	X	X

层次模式因为将所有的组件分层，所以当更换环境或是改变需求时，只需要修改相应的层就可以达到效果。

经纪人模式的灵活性主要源于经纪人这个中间组件，他大大增加了服务端-客户端架构在调用时的灵活性，服务端与客户端并不需要知道自己具体要怎么获得数据，可以全部交由中间件去做，灵活性大大增加。

MVC模式将前后端之间的所有调用通过接口调用，并且对后端分层使得后端只需要在对应的层是想对应的逻辑并提供接口给别的层去调用，架构中每个组件的灵活性能大大提升，任何层都可以调用不同层的接口，而这个层只要实现对应的接口就可以了。

服务端-客户端模式没有经纪人模式那么好的灵活性，但是他的灵活性在于将前后端进行拆分，也就是两者之间不需要知道对方的具体实现，只需要通过接口进行调用，两边数据交互的灵活性增加。

分布式集群可以允许每一个服务部署在不同的环境上面，当一个服务出现问题或是要改需求，只需要修改对应的服务就行。

事件驱动对应的事件比较特定，所以当环境发生改变或改变需求时，会因为事件的改动使得整个代码的逻辑需要进行相应的修改，并不能保证灵活性。

防御式编程只有在部分已经有特别进行设计的架构中有，前后端交互常常可能出问题所以有部分防御式编程的思想在里面，但是其他的就没有。

4 Reusability

见表4。

表 4: Reusability

quality attribute	Reusability					
Strategies	使用较好的开发方法			用简单构造复杂		
Tatics	结构化编程	面向对象	面向切面	开闭原则	单一职责原则	切割成子模块
layered	X			X	X	X
broker	X			X	X	X
MVC	X		X	X	X	X
client-server	X		X	X	X	X
Distribute-Cluster	X			X	X	X
Event-Driven	X			X	X	X

分层开发在可复用性方面可以说是非常好，层次设计的复用性很好，分层使得上层可以随意地调用下层的组件而不会有循环依赖的问题，可以多个上层组件公用一个下层组件来进行计算，每个组件的可复用性可以在层次架构中得到加强。

经纪人架构是将客户端与服务端进行代理，也就是说客户端可以随意的调用任何服务端的接口，复用性很高，但是主要是体现在客户端-服务端架构上。

MVC模式将controller提供的数据可以被view随意的复用，controller也可以调用任意model进行操作，view只要调用相应的接口就可以获得数据，同一个接口可以不断地被复用。

客户端-服务端架构将前后端分离，也就是说并不需要指定接口才能获得数据和视图，将数据和视图分开可以让数据被很多个不同的视图调用，增加可复用性。

分布式集群可以说是比经纪人模式灵活性更高的架构模式，他将每一个模块拆成不同的服务，每一个服务之间相互调用，服务与服务之间可以互相使用对方的接口来实现自己的逻辑。

事件驱动架构的灵活性并没有特别高，他主要取决于每一个事件过于单一与特定化，没办法像其他的模式一样去复用同一个事件。

面向对象和面向切面都是不一定会应用到给定的架构上面，也就是每一个架构可以使用不同的代码编写方式进行设计。

5 Availability

见表5。

由于层次模式中的每一层只能依赖严格的下一层，因此当错误发生时，可以对定位和排除错误产生帮助，也可以应用degrade使得服务正常运行，这有利于层次模式进行错误的定位和修复。但是因为分层，如果错误发生在最上层，则无法进行处理与修复。

表 5: Availability

quality attribute	Availability												
Strategies	检测错误					错误复原					预防错误		
Tatics	Ping值	异常检测	时间戳	自我测试	健全性检查	条件监听	异常捕获	回退	软件升级	重试	预测模型	异常预防	增加功能
layered		X											X
broker	X					X				X		X	
MVC	X	X				X	X				X	X	
client-server						X	X			X		X	
Distribute-Cluster	X			X	X	X	X						
Event-Driven		X				X	X	X		X	X	X	

经纪人模式对错误的发现和预防很容易，因为服务端和客户端是经由中间的组件转发，当发现错误时，中间的组件可以选择拦截不发回前端造成更多的错误。但是他不利于错误的定位，因为可能是前端发送错误信息，也可能是后端有逻辑异常。

MVC模式与分层模式类似，将每个层次进行划分并进行不同的逻辑，所以发生错误时很容易定位到错误产生的原因。但是结构的复杂使得有时候会获得错误的异常信息导致无法精准的定位错误并修复。

客户端-服务端模式将前后端分离，可以比较好的进行错误查找，能精准的的定位错误的信息并进行预防，但是如果后端的信息错误，前端没办法有较好的手段进行异常的捕获与处理。

分布式集群的可用性很好，因为每一个模块彼此间互不影响，使得每一个错误都能精准的被定位到固定的位置以及进行相应的预防措施，但是因为分布式的关系，如果某个模块因为某种原因响应时间较长，则会影响整个系统的速度。

事件驱动的可用性只能说普通，因为如果有不好的代码编写，可能每个事件的响应时间会很长影响操作，但是捕获异常方面比较不需要考虑，因为事件驱动通常会有代理模块来负责处理事件，有健全的错误预防。

软件升级和时间戳并不是这些架构模式必有的功能，有了能提高可用性，但是不是架构本身包含的。

6 Performance

见表6。

表 6: Performance

quality attribute	Performance										
Strategies	控制资源需求						资源管理				
Tatics	限制回复	事件优先级	减少管理经费	响应时间限制	增加资源效率	抽样计算	使用并发	数据多备份	设定队列边界	调节资源	
layered	X		X								
broker	X			X			X				
MVC	X			X			X				
client-server	X			X		X	X				
Distribute-Cluster	X	X		X	X	X	X	X		X	
Event-Driven	X	X		X			X			X	

分层模式可以对数据进行处理，在每一层限制相应次数和进行事件优先级的排列。但是这个的问题在于每一个模块之间的交互需要时间，所以可能会导致响应时间延长。

经纪人模式用一个组件来分开客户端与服务端的交互，也就是说可以用这个组件进行流量控制与分配资源，但是问题就是增加一个组件可能导致响应时间增加。

MVC模式是一个将每一个模块进行分层的设计，也就是说每个模块可以进行自己的资源调节和控制，controller更能主动的控制与前端的交互和事件相应次数，但是一样的问题，分层的问题会使得相应事件增加，而且不同层的优先级不同可能产生冲突。

客户端-服务端模式将前后端分离后，数据不再需要一定跟着视图一起返回，增加了后端与前端在资源使用上的一些弹性，可以减少获得视图所需要的响应时间，但是前后端分离需要更多的资源进行部署与配置，增加管理经费。

分布式集群是将每一个服务进行拆分部署，可以对每个服务进行事件的控管与资源分配，能大大的增加并发性以减少阻塞可能，而且每一个服务都有数据备份，但是问题是这样子需要增加很多的经费进行每一个服务的服务的管理。

事件驱动能很方便的设定事件的响应速度与允许某个时间片内发生几次或是最多响应几次，但是，单次事件响应过长可能影响后续的事件响应。

数据多备份和设定队列边界并不是每一个架构一定会有的模式，这属于要一定的资源才能达到的策略，与架构无关。

7 Security

见表7。

表 7: Security

quality attribute	Security										
Strategies	拒绝攻击					侦测攻击			回应攻击		
Tatics	身份认证	用户授权	身份识别	限制权限	数据加密	侦测侵入	侦测服务拒绝	验证数据完整性	废除访问	设备锁定	提示用户
layered				X			X	X			
broker	X			X			X	X	X	X	X
MVC	X			X			X	X			X
client-server	X			X		X	X	X	X	X	
Distribute-Cluster	X			X	X	X	X	X	X	X	X
Event-Driven				X		X	X	X		X	

层次模式的安全性不怎么样，因为每一层之间是通过接口联通，并没有什么额外的操作进行识别，只要最顶层被攻破，很容易从上而下的入侵。

经纪人模式的好处在于中间的组件可以识别是否是自己的客户端或自己的服务端发出的数据，要是不是则不允许转发。

MVC模式controller需要验证头文件与指定的URL才能允许调用方法，在安全性上非常严格，而且数据库并不直接与整个后端代码联通，而是使用接口进行相应的调用。

客户端-服务端模式一样可以通过头文件进行判断来进行访问与获取数据，也可以通过一些证书来进行解密，允许访问或是拒绝不良访问接入。

分布式集群都是通过注册服务的组件进行转发，每个服务之间相互独立，攻破一个服务并不能影响到别的服务，最多导致该服务失效。

事件驱动安全性比较差，因为没有进行很有效的验证，在获取到一个响应之后很难判断这个响应的安全性，必须增加其他的额外保护措施。

用户授权与身份识别并不属于每个架构必须有的，属于额外添加的功能。