南京大学

课程作业

标题： Software　Architecture Assignment 1

姓名：	曹信
学号：	201250066
教师：	张贺
学院：	软件学院
年级：	2020
班级：	2班
日期：	2023.05.12

1 Architecture Pattern Analysis (8 points)

1.1 Broker patterns

1.1.1 availability

- possible impacts

Broker pattern 可以提高系统的可用性。由于 Broker 模式的中间件负责转发消息，所以可以在组件间建立多条通信路径，从而提高了冗余度和可用性。

- illustrative example

在一个分布式系统中，如果一个组件发生故障，中间件可以将请求转发到其他健康的组件，从而保证服务的可用性。

- candidate tactics
- working condition

1.1.2 interoperability

- possible impacts

Broker pattern 可以提高系统的可用性。由于 Broker 模式的中间件负责转发消息，所以可以在组件间建立多条通信路径，从而提高了冗余度和可用性。

- illustrative example

Broker pattern 是为了提高系统的互操作性而设计的。通过中间件实现了不同组件之间的协同工作，提高了组件之间的兼容性和交互性，从而提高了系统的互操作性。

1.1.3 modifiability

- possible impacts

Broker pattern 可以提高系统的可修改性，因为 Broker 的存在使得系统中的组件能够独立分布式地变更，并且 Broker 能够处理变更所带来的后果。

- illustrative example

一个在线购物网站可以使用 Broker pattern 模式来优化订单处理流程，增加新的订单支付方式，并且通过更改中间件配置来实现对支付方式的支持。

1.1.4 performance

- possible impacts

Broker pattern 会对系统的性能产生一定的负面影响。

- illustrative example

在处理大量并发请求时，由于中间件的添加会增加通信延迟和消息处理的开销，可能会成为系统的瓶颈。

- candidate tactics

将 Broker 模式与缓存模式结合使用，从而减少通信和消息序列化的开销，并提高系统的性能。

在具体实现时，可以使用一个缓存模块来存储经常使用的数据，并通过中间件来检查数据是否存在于缓存中。如果数据在缓存中，则可以直接返回响应，否则将请求转发到后端服务进行处理，并将结果写入缓存中。

- working condition

在系统中有大量重复的请求，需要对它们进行快速响应，并且需要保持缓存和数据一致性。此外，还需要考虑缓存的大小和命中率等因素，以保证系统的性能和可用性。

1.1.5 security

- possible impacts

Broker 作为系统的中央控制器，集中承载了系统的核心功能，一旦 Broker 被攻击或崩溃，整个系统都会受到影响。同时，由于 Broker 需要管理和协调不同的服务，如果开发人员没有充分考虑和保护 Broker 的安全性，可能会导致 Broker 本身成为安全威胁的来源，如恶意代码注入、拒绝服务攻击等。

- illustrative example

一个在线金融交易平台的设计中，可以使用 Broker Pattern 来管理各种服务（如用户认证、订单管理、资金结算等），并在 Broker 中实施强制访问控制和加密保护等机制，以确保整个系统的安全性和可靠性。这将有助于提高系统的安全性，避免被恶意攻击者利用其中的漏洞进行攻击。

- candidate tactics

将 Broker 进行安全性认证，并实施强制访问控制。

具体来说，可以在 Broker 的访问控制策略中定义各种类型的用户、角色和权限，以确保只有授权的用户可以使用 Broker 提供的服务。同时，还可以使用加密技术和数字签名来保护 Broker 的通信和交换数据，防止被篡改或拦截。

- working condition

对于需要使用 broker pattern 的系统，需要确保其核心功能受到人为干扰风险较高，如金融交易系统等。

1.1.6 testability

- possible impacts

Broker pattern 可以提高系统的可测试性。中间件的添加可以使得测试系统更加容易，减少了测试工作的时间和代价。

- illustrative example

在软件开发过程中，可以使用 Broker pattern 模式来模拟系统中的组件和数据流，从而进行单元测试和集成测试。

1.1.7 usability

- possible impacts

Broker pattern 对于系统的易用性没有直接影响。

1.1.8 reliability

- possible impacts

Broker pattern 可以提高系统的可靠性。通过增加冗余路径和利用中间件进行消息处理等技术手段，可以提高系统的容错能力和可靠性。

- illustrative example

在物流系统中，使用 Broker pattern 模式可以保证货物运输的可靠性，当一个货运节点故障时，中间件可以将货物转移到其他节点进行运输。

1.1.9 recoverability

- possible impacts

Broker pattern 可以提高系统的可恢复性。由于中间件的存在，当一个组件不能正常工作时， Broker pattern 可以自动重启或者迁移 某些模块，从而保证系统的可恢复性。

- illustrative example

在电商平台上，使用 Broker pattern 模式可以帮助系统从故障中自动恢复，将请求转移到其他健康的组件或服务上，确保系统的业务连续性和服务可用性。

1.1.10 reusability

- possible impacts

Broker pattern 可以提高系统的可重用性。由于各个组件是独立分布式的，它们可以在不同的系统中相互通信和协同工作，从而增加了系统的可重用性和扩展性。

- illustrative example

在一个企业内部应用系统中，使用 Broker pattern 模式可以实现不同业务线之间的协作和信息交流，从而促进了组件和代码的重用。

1.1.11 modularity

- possible impacts

Broker pattern 可以提高系统的模块化和松散耦合性。由于 Broker pattern 通过中间件实现组件之间的通信，因此使得组件之间的耦合程度更低，系统具有更高的松耦合性和模块化程度。

- illustrative example

在基础设施系统中，使用 Broker pattern 模式可以将不同组件分离和解耦，实现松耦合架构，提高了系统的可扩展性和灵活性。

1.2 Peer-to-peer patterns

1.2.1 availability

- possible impacts

P2P 模式可以提高系统的可用性。每个节点都是独立的，如果一个节点宕机，其他节点仍可以继续工作，从而保证了服务的可用性。

- illustrative example

BitTorrent 是一个流行的 P2P 文件共享协议，它可以通过多个节点共享和下载文件，使得文件的有效可用性更高。

1.2.2 interoperability

- possible impacts

P2P 模式可以增加系统的互操作性。各个节点之间可以直接通信，通过共享协议和格式来交换信息，从而提高了系统的互操作性。

- illustrative example

当不同的 P2P 文件共享应用程序使用不同的协议时，它们之间可能无法交换文件、搜索内容或浏览其他节点等。

1.2.3 modifiability

- possible impacts

P2P 模式对于系统的可维护性和可修改性有正面和负面的影响。

正面：由于 P2P 系统中各个节点之间相互独立，没有中心化的控制器，因此可以轻松地增加或移除节点，而不需要考虑其他节点的影响。这使得 P2P 系统更容易扩展和升级，具备更好的可修改性。

负面：在 P2P 系统中，需要设计良好的协议来管理节点之间的通信和数据交换。这需要在协议上和代码实现上进行更加细致的设计，以避免出现节点之间的不匹配和通信故障。这可能会增加开发和测试的工作量，从而降低了系统的可修改性。

- illustrative example

正面影响示例：假设我们正在开发一个在线游戏服务，在传统的中心化系统中，如果需要增加服务器或改善性能，则需要对整个系统进行修改。然而，在 P2P 系统中，我们可以轻松地添加更多的节点来支持更多的玩家，而不需要对整个系统进行重新设计。这使得 P2P 系统更加具有可维护性和可扩展性，并提高了其可修改性。例如，如果我们决定支持更多的游戏玩法，我们可以通过添加新的节点来实现，而不需要对现有服务器进行重大修改。

负面影响示例：假设我们正在开发一个 P2P 文件共享应用程序，每个节点都可以发布和下载文件。在这种情况下，协议的设计和实现变得至关重要。如果我们不小心设计了一个恶意的协议，可能会出现以下情况：某些节点将无法与其他节点进行通信，甚至会导致节点之间的信息泄露和文件损坏等问题。这会降低系统的可修改性，因为我们需要在代码中修改或替换协议来修复这些问题。如果我们早期考虑和避免这些问题，通过优化协议设计和引入错误处理机制，我们可以提高系统的可修改性，并减少未来的开发负担。

- candidate tactics

采用适当的错误处理机制。在一个 P2P 系统中，节点之间的通信可能会出现故障或异常，例如消息丢失、超时等。为了保证系统的可修改性，开发人员需要在代码中嵌入适当的错误处理机制，以捕获并解决这些问题。具体地，开发人员可以在代码中引入异常处理机制，记录日志，并进行适当的重试。同时，为了保证效率和可扩展性，错误处理机制需要基于异步通信模式，以避免阻塞和降低系统性能。

- working condition

1. 当节点之间的通信频繁、通信质量较差或网络环境不稳定时，错误处理机制会更加重要和必要。
2. 当开发人员面对复杂的协议设计和代码实现时，错误处理机制可以帮助他们识别和纠正潜在的错误和漏洞，从而提高系统的可修改性。
3. 当系统需要支持弹性伸缩和容错能力时，错误处理机制可以帮助系统自动化地检测和解决故障，并保证系统的连续性和可用性。

1.2.4 performance

- possible impacts

P2P 模式对于系统的性能具有一定的优势和劣势。它可以使系统具有更好的伸缩性和容错能力，但是，由于节点之间需要相互通信和协作，因此会增加网络延迟和消息传输的开销，可能会影响系统的性能。

- illustrative example

对于某些应用程序（如大型文件共享或分布式计算），P2P 系统可以通过并行计算和数据分发来提高系统的性能。但是，如果在设计和实现 P2P 系统时没有考虑到通信和数据交换的成本，可能会降低系统的性能。例如，使用 P2P 系统进行在线游戏可能会增加延迟和通信故障。

- candidate tactics

使用缓存技术来减轻节点之间的通信和数据交换压力。如果在 P2P 系统中，存在某些经常被访问的数据或资源，我们可以使用缓存技术将这些数据或资源缓存到某些节点上，而不是每次都从其他节点获取。这样做可以降低节点之间的通信和数据交换量，从而减少网络拥塞和延迟。

- working condition

数据或资源具有重复访问的特征，且可缓存的数据或资源不会频繁地发生变化。

1.2.5 security

- possible impacts

P2P 模式对于系统的安全性可能产生负面影响。由于节点之间相互通信，并且没有中心化的控制器，所以可能存在一定的安全风险，例如数据泄露、拒绝服务等。

- illustrative example

拦截通信、篡改数据、放大攻击等，如果开发人员没有充分考虑和保护系统的安全性，可能会导致节点之间的信息泄露和文件损坏。

- candidate tactics

使用加密技术来保护节点之间的通信和数据交换。如果在 P2P 系统中，节点之间需要进行加密通信和身份验证，才能确保只有授权的节点可以访问和共享数据。加密技术可以保护节点之间的通信和数据传输，从而避免被拦截或篡改。此外，还可以使用数字签名和证书来验证节点的身份，以确保只有真实的节点才能参与系统的数据交换。

- working condition

P2P 系统中存在敏感的数据或资源，需要进行安全保护，并且注意到在使用加密技术时需要充分考虑其带来的性能和效率问题。

1.2.6 testability

- possible impacts

P2P 模式对于系统的可测试性没有直接影响。

1.2.7 usability

- possible impacts

P2P 模式对于系统的易用性没有直接影响。

1.2.8 reliability

- possible impacts

P2P 模式可以提高系统的可靠性。每个节点都是独立的，如果一个节点宕机，其他节点仍可以继续工作，从而保证了系统的可靠性。

- illustrative example

使用 P2P 系统进行在线视频流媒体可能会减少视频断流和卡顿等问题。

1.2.9 recoverability

- possible impacts

P2P 模式可以提高系统的可恢复性。当一个节点宕机时，其他节点可以继续工作，从而保持系统的业务连续性和服务可用性。

- illustrative example

在使用 P2P 网络进行数据备份和存储时，即使其中某些节点出现故障，也可以通过其他可用节点进行恢复。

1.2.10 reusability

- possible impacts

P2P 模式可以提高系统的可重用性。由于各个节点之间通过共享协议和格式来交换信息，所以可以重复利用设计好的通信协议和数据格式，从而促进了组件和代码的重用。

- illustrative example

可以使用相同的 P2P 文件传输协议来实现不同的文件共享应用程序。

1.2.11 modularity

- possible impacts

P2P 模式可以提高系统的模块化和松散耦合性。各个节点之间相互独立，通过共享协议和格式来交换信息，实现了松耦合架构，从而提高了系统的可扩展性和灵活性。

- illustrative example

在使用 P2P 网络进行在线视频流媒体时，可以将不同的视频片段分配到不同的节点中以提高服务的效率和质量。

2 Architecture Patterns vs. Quality Attributes & Tactics (12 points)

	Tactics for Improving Availability								
Architectural Pattern	**Monitor**	**Timestamp**	**Exception Detection**	**Active Redundancy**	**Rollback**	**Software Upgrade**	**Transactions**	**Predictive Model**	**Exception Prevention**
Mediator Pattern	x	x	x	x	x		x	x	x
Pipe-Filter	x	x	x	x	x		x	x	x
Client-Server	x	x	x	x	x	x	x	x	x
Module View Controller	x	x	x	x	x	x	x	x	x

	Tactics for Improving Interoperability		
Architectural Pattern	Discover Service	Orchestrate	Tailor Interface
Mediator Pattern	x	x	x
Pipe-Filter			x
Client-Server	x	x	x
Module View Controller			x

	Tactics for Improving Performance								
Architectural Pattern	Limit Event Response	Prioritize Events	Bound Execution Times	Increase Resource Efficiency	Increase Resources	Introduce Concurrency	Maintain Multiple Copies of Computations	Bound Queue Sizes	Schedule Resources
Mediator Pattern	x	x	x			x		x	x
Pipe-Filter			x	x			x		x
Client-Server	x	x	x	x	x	x		x	x
Module View Controller	x	x	x		x	x		x	x

	Tactics for Improving Security								
Architectural Pattern	Verify Message Integrity	Detect Message Delay	Authenticate Actors	Limit Access	Encrypt Data	Revoke Access	Inform Actors	Maintain Audit Trail	See Availability
Mediator Pattern	x		x	x	x	x	x	x	
Pipe-Filter		x		x	x			x	
Client-Server	x		x	x	x	x	x	x	x
Module View Controller	x		x	x	x	x	x	x	x

Architectural Pattern	Tactics for Improving Testability						
	Specializes Interfaces	Record/Playback	Localize State Storage	Abstract Data Sources	Executeable Assertions	Limit Structural Complexity	Limit Non-Determinism
Mediator Pattern		x	x	x	x	x	
Pipe-Filter		x	x	x	x	x	x
Client-Server				x	x		
Module View Controller	x		x	x		x	x

Architectural Pattern	Tactics for Improving Usability						
	Cancel	Undo	Pause/Resume	Aggregate	Maintain Task Model	Maintain User Model	Maintain System Model
Mediator Pattern	x	x	x		x	x	
Pipe-Filter	x	x	x		x	x	x
Client-Server	x	x			x	x	x
Module View Controller	x	x	x	x	x	x	x

Architectural Pattern	Tactics for Improving Portability								
	User Cross-Platform API	User Internal API	Auto-Deploy	Auto-Testing	Design by Users	Consider by Users	Consider More Before Coding	Community Version	Embrace Contribution
Mediator Pattern			x	x			x		
Pipe-Filter		x		x			x		
Client-Server	x		x	x	x	x	x	x	
Module View Controller	x	x	x	x			x	x	

Architectural Pattern	Tactics for Improving Flexibility								
	User Cross-Platform API	User Internal API	Auto-Deploy	Auto-Testing	Design by Users	Consider by Users	Consider More Before Coding	Community Version	Embrace Contribution
Mediator Pattern				x	x	x	x	x	
Pipe-Filter		x		x	x	x	x	x	
Client-Server	x		x	x	x	x	x	x	
Module View Controller	x	x	x	x			x	x	

Architectural Pattern	Tactics for Improving Reliability						
	Load Balancing	Fault Tolerance	Monitoring and Recovery	Redundancy	Error Handling	Consistent Data Management	Automated Scaling
Mediator Pattern		x	x		x	x	x
Pipe-Filter	x		x	x	x		
Client-Server	x	x	x	x			x
Module View Controller		x	x	x		x	x

Architectural Pattern	Tactics for Improving Recoverability						
	Incremental Backup	Rollback and Recovery	Checkpointing	Redundancy	Error Handling	Fault Isolation	Automated Recovery
Mediator Pattern		x	x		x	x	x
Pipe-Filter	x		x	x	x		
Client-Server	x	x	x	x			x
Module View Controller		x	x	x		x	x

Architectural Pattern	Tactics for Improving Reusability						
	Abstraction	Encapsulation	Inheritance	Polymorphism	Composition	Dependency Injection	Separation of Concerns
Mediator Pattern	x	x			x	x	x
Pipe-Filter	x	x			x	x	x
Client-Server		x			x	x	x
Module View Controller	x	x	x	x	x	x	x

Architectural Pattern	Tactics for Improving Modularity						
	Information Hiding	Encapsulation	Abstraction	Separation of Concerns	Loose Coupling	High Cohesion	Single Responsibility Principle
Mediator Pattern		x	x	x	x	x	x
Pipe-Filter		x	x	x	x	x	x
Client-Server	x	x	x	x	x		x
Module View Controller	x	x	x	x	x	x	x