编号____

南京大学

课程作业

题　　目　　**Software Architecture Assignment 3**

学生姓名　　201250070 郁博文 201250071 邢佳勇
201250072 陈子凡 201250229 郑启睿

学　　院　　软件学院

专　　业　　软件工程

2023 年 6 月

一、架构驱动设计

1、重要的非功能性需求和约束

1.1 非功能性需求的场景分析

1.1.1 可用性分析

场景的组成部分	可能的值
刺激源	C4 系统与外部交互过程
刺激	用户数据的完整性以及最终请求配置的完整性可能会遭到破坏；客户在使用系统中可能会发生冲突事件
制品	与数据库的连接器，与 NOSS 模块的连接器
环境	正常过程：通过发布/订阅组件发布请求，下游系统对此作出反应
响应	如果用户发起重处理，将通知发送给用户，等待用户处理；如果用户无法解决发生的问题，系统尝试从备份中恢复；面对发生的冲突，可以对冲突进行优先级分类，优先支持高优先级
响应量度	解决冲突所用时间；备份恢复时间：恢复用户数据，并可以开始提供服务的时间

质量需求场景 1：可用性

1.1.2 互交互性分析

场景的组成部分	可能的值
刺激源	C4 系统与内部元素交互过程
刺激	需要进行数据的储存、提取或是变更；需要进行网络资源的请求和处理；需要处理用户信息；需要处理服务配置请求
制品	C4 系统可以与 NOSS、下流系统、DB 和 Bill 系统进行互相交互
环境	C4 系统、NOSS、下流系统、DB 和 Bill 系统在运行前互相发现，或是运行前就相互已知
响应	刺激中所描述的请求在失败或是被拒绝时有相应的反馈； 刺激中所描述的请求在接受时，相应的信息被成功的修改； 刺激中所描述的请求被至少一个系统所记录成日志文件
响应量度	正确处理的信息交换的概率；正确拒绝的信息交换的概率；

质量需求场景 2：互交互性

1.1.3 性能分析

场景的组成部分	可能的值
刺激源	系统用户

刺激	用户发起了长时间的会话；用户在同一通电话中使用不同服务
制品	-
环境	用户正常发起会话
响应	支持大量的并发任务；进行容量规划，根据预定时间预留充足资源
响应量度	支持的并发数量数；最长可提供服务时间

质量需求场景 3：性能

1.1.4 可恢复性分析

场景的组成部分	**可能的值**
刺激源	网络波动、网络中断
刺激	用户发起通话请求时，出现网络网络波动或是网络终端
制品	支持中断的对话
环境	用户正常发起会话
响应	在对话发生中断时，可以在网络恢复正常之后迅速恢复通话
响应量度	从中断中恢复所需的时间

质量需求场景 4：可恢复性

1.2 约束

CON1: 不应该需要高额的初始设备投资

CON2: 应该允许“更精简”的增长（相对于成本/容量函数）

CON3: 应该能够以快速的速度增长

CON4: 需要在 7x24 小时内提供应用程序可用性

2、 迭代一

2.1 元素选择

我们第一次迭代的元素选择主要是对整个 C4 系统提出的必须满足的需求和约束进行分析。

2.2 架构关键需求（ASRs）

#	**架构驱动**	**重要性**	**难度**
1	场景 1 系统会实时备份用户的信息，在遇到完整性配置错误时及时恢复可用	高	高
2	场景 2 在请求交由 NOSS 前，解决请求的冲突	高	高

3	场景 3 可以与 NOSS、下流系统、DB 和 Bill 系统完成可靠交互	高	中
4	场景 4 支持通话中的并发任务	中	高
5	场景 5 用户发起长时间会话	中	中
6	场景 6 通话中断后可以恢复	中	低
7	约束 1 不应该需要高额的初始设备投资	高	中
8	约束 2 应该允许“更精简”的增长（相对于成本/容量函数）	中	低
9	约束 3 应该能够以快速的速度增长	高	中
10	约束 4 需要在 7x24 小时内提供应用程序可用性	高	中

2.3 选择满足 ASRs 的设计构思

2.3.1 确定设计关注点

设计关注点	从属关注点
高可用性保持	保证客户数据完整性
	冲突解决
互交互性保证	记录交互日志
	提供完整交互反馈
高性能	管理资源利用
	控制资源需求
高可恢复性	中断恢复机制

2.3.2 为从属设计关注点列举可选战略并选择合适的战略

1. 保证客户数据完整性

#	模式名称	模式介绍

1	定期的数据备份	将客户数据定期备份到可靠的存储系统中。在数据完整性遭到破坏时，可以通过从备份中恢复数据来确保数据的完整性
2	实时的数据复制机制	将客户数据实时复制到多个独立的存储系统中

选择：定期的数据备份

理由：考虑到 C4 系统有不应该需要高额的初始设备投资的约束，因此应当选取对于设备要求更低的定期的数据备份，并且定期备份足够确保系统拥有可靠的数据副本，可以保证系统在遇到用户数据完整性乃至数据缺失时，较快地恢复，使得系统可用。

2. 冲突解决

#	**模式名称**	**模式介绍**
1	自动冲突解决算法	检测和解决简单的冲突的情况
2	优先级排队与回拨机制	将请求根据优先级分类，先处理优先级最高的，然后在回拨其他请求

选择：优先级排队与回拨机制

理由：通过优先级排队机制，可以根据不同的冲突情况合理分配资源，高优先级的来电得到优先处理。回拨机制则为客户提供了灵活性，让他们选择在适当的时间重新与客服人员通话，从而保证服务的质量和数据的完整性。而自动冲突解决算法只能处理较为简单的冲突情况，这难以应付多冲突的情况，不能较好地提高系统的可用性。

3. 记录交互日志

#	**模式名称**	**模式介绍**
1	实时日志记录和存储	在系统中实时记录和存储 C4 与其它组件之间的交互日志
2	批量日志记录和存储	将交互日志在批量方式下进行记录和存储

选择：实时日志记录和存储

理由：实时日志记录还为故障排查和安全审计提供了更好的支持。当出现问题或安全事件时，可以迅速检查相关的交互日志，了解事件的详细情况，帮助进行故障排除和安全审计。实时日志还能做到几乎实时地监控和分析 C4 系统交互情况，帮助更快速发现和解决潜在的问题，为用户的可用性体验做出提升。同时这也是约束 4 的要求。

4. 提供完整交互反馈

#	**模式名称**	**模式介绍**
1	即时反馈	进行交互的同时，即时地提供反馈给用户
2	持续反馈和状态指示	进行交互的过程中持续地提供反馈和状态指示

选择：即时反馈

理由：即时反馈可以立即向用户传达操作结果或状态更新，使用户能够几乎实时地了解他们的操作是否成功。可以帮助他们快速纠正错误或采取进一步的行动。此外，通过可视化的方式呈现反馈信息，如界面提示、弹出窗口或状态图标，可以使反馈更加明确和易于理解。即时反馈还有助于用户导向和错误预防。通过即时告知用户其操作的结果或可能存在的问题，可以帮助用户避免犯错或做出不符合预期的操作。这提高了系统的可操作性。

5. 管理资源利用

#	**模式名称**	**模式介绍**

1	资源水平调整	根据通话的实际情况动态的调整资源分配
2	资源优先级	通过智能调度算法，确保资源的最优利用

选择：资源水平调整

理由：根据约束 3，系统需要实行快速的增长，而系统能够实现资源水平调整是有利于系统应对用户的增长的，为每个客户的实际需求分配所需的资源，既能做到资源的高效利用，又能合理分配出长时间会话用户所需的资源，最大化系统的性能。

6. 控制资源需求

#	模式名称	模式介绍
1	控制并发数	通过压缩数据传输来减少通话过程中的网络带宽和资源需求
2	压缩并发任务数量	限制一次通话中同时进行的并发任务数量

选择：压缩数据传输

理由：压缩并发任务数量无法较好地适配高并发需求。而压缩数据传输可以减少通话过程中的网络带宽和资源需求。通过使用有效的压缩算法，可以大幅降低数据传输量，减少对网络带宽和系统资源的占用。这样可以实现资源的节约和优化，提高系统的效率和性能。同时压缩数据传输可以提高系统的响应速度。较小的数据传输量意味着更快的传输速度和响应时间。

7. 中断恢复

#	模式名称	模式介绍
1	自动重连机制	在通话中断时系统能够自动尝试重新建立连接
2	保留通话状态并提供恢复选项	保留通话状态并提供恢复选项给用户

选择：自动重连机制

理由：无需用户干预：自动重连机制可以在通话中断时自动触发重连过程，无需用户的干预。这可以减少用户的不便和干扰，提高通话的连续性和可靠性。用户可以更轻松地恢复到之前的通话状态，而无需手动重新连接。提供快速恢复：自动重连机制可以快速恢复通话连接。系统可以立即尝试重新建立连接，减少通话中断的时间和影响。这可以提供更好的用户体验，保持通话的连续性和流畅性。

2.3.3 决定所选模式/战略和 ASRs 的关系

#	模式类型	模式选择	架构驱动
1	保证客户数据完整性	定期的数据备份	保证用户的系统可用性（场景 1） 同时满足较低的额外设备投资（设计约束 1）
2	冲突解决	优先级排队与回拨机制	保证用户在遇到冲突时系统也有较高的可用性（场景 2）
3	记录交互日志	实时日志记录和存储	保证系统不同组件之间的交互的可记录性（场景 3）
4	提供完整交互反馈	即时反馈	保证系统交互的及时反馈（场景 4）
5	管理资源利用	资源水平调整	保证系统的高性能 同时满足系统能够以快速的速度增长（设计约束 3）
6	控制资源需求	压缩数据传输	保证系统的高性能

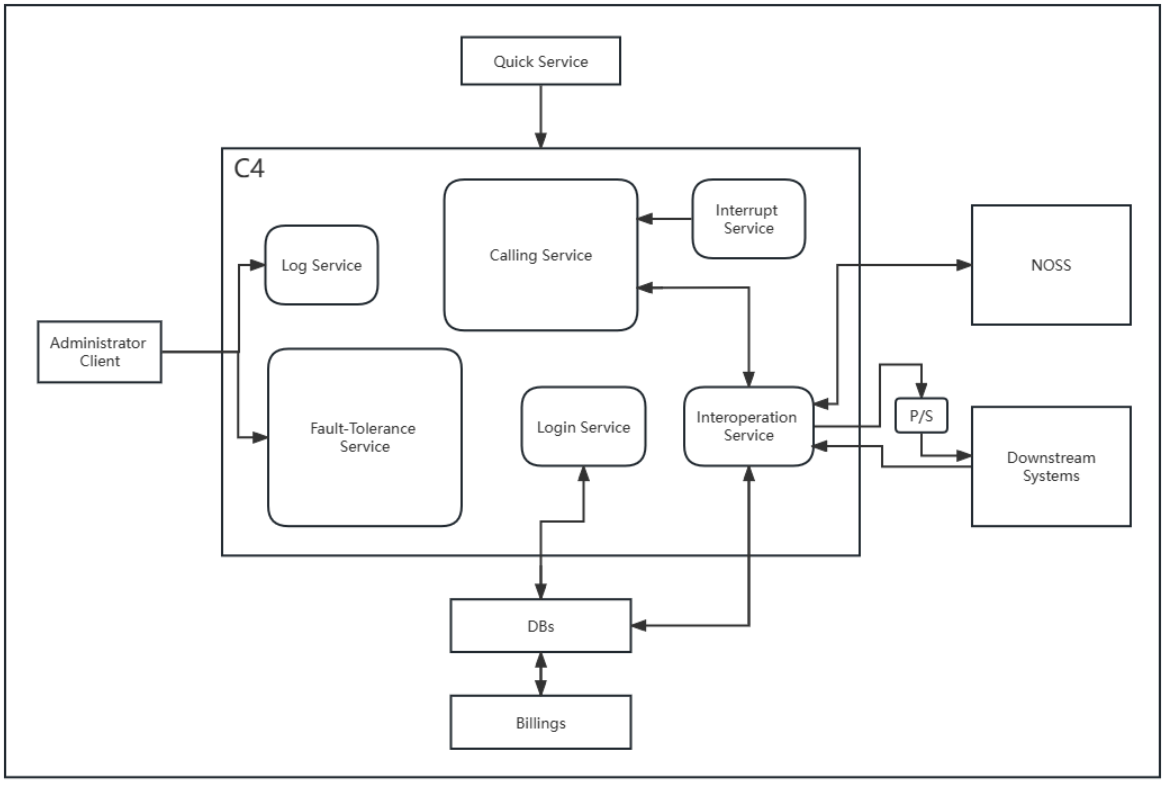
			同时满足更加精简的增长（设计约束 2） 满足系统可以长时间的可用（设计约束 4）
7	中断恢复	自动重连机制	保证系统的可恢复性，用户可以恢复中断的会话（场景 7）

2.3.4 捕获初步的架构视图

2.3.4.1 元素表

#	元素	是否在本次 ADD 迭代中涉及
1	Login Service	1
2	Log Service	1
3	Interrupt Service	1
4	Fault-Tolerance Service	1
5	Interoperation Service	1
6	Calling Service	1

2.3.4.2 架构元素视图



2.4 实例化架构元素并分配职责

1. Login Service：接受客户端发送的登录请求，然后调用内部的 Interoperation Server 来核对用户身份，校验成功则允许用户登录，否则不允许用户进行下一步的系统交互。

2. Log Service：Log Service 会将用户在任何时刻对系统执行的所有操作记录在操作日志中；管理员则可以查询用户曰志以显示特定用户或者用户组的活动记录，管理员可以使用诸如日期、操作类型等各种有效的条件来过滤信息，以此来保证系统的互操作性。
3. Interrupt Service：Interrupt Service 会在网络中断时保持会话的连通性，并在网络异常结束之后在短时间内重新恢复通话的可用性。
4. Fault-Tolerance Service：该服务用于应对在正常处理过程中可能会发生的故障。提供错误发现、错误恢复和错误预防的功能，通过自我检测的方式在第一时间发现系统出现的问题并提供有效的解决方案然后对尚未处理的信息进行重新处理，并采用动态请求窗口的方式进行错误预防，显著提高系统的可靠性和可用性。
5. Interoperation Service：C4 系统通过该服务与。。。进行互操作，对 DB 和 Billing 系统进行增、删、改和服务的调用，对 NOSS 系统进行进行网络请求的处理，对下流系统进行冲突的解决，这些系统也会对 C4 进行操作的反馈，无论操作成功与否。
6. Calling Service：该服务负责支持长时间通话的需求，并且支持用户在同一通电话中使用不同服务，并在冲突发生时，使用优先级排队与回拨机制解决冲突

2.5 为实例化的元素定义接口

来源元素	目的元素	接口	时间条件
Client	Login Service	服务请求	0.2s
Login Service	User Dictionary Service	服务应答	0.2s
Client	Interoperation Service	服务请求	0.1s
Interoperation Service	DB，NOSS，Downstream System	服务应答	0.5s
Client	Calling Service	服务请求	长时间
Calling Service	NOSS	服务应答	长时间
Client	Interrupt Service	服务请求	0.5s
Interrupt Service	NOSS	服务应答	1s
Administrator Client	Log Service	服务请求	0.5s
Administrator Client	Fault-Tolerance Service	服务请求	<1s

3、迭代二

3.1 选择系统的一个元素并分解

我们第二次迭代的元素选择主要是根据第一次迭代的 Calling Service 和 Fault-Tolerance Service

3.2. 架构关键需求（ASRs）

	Architectural Drivers	重要性	难度
1	场景 1: 用户可以在同一通电话中选择不同服务	高	中

2	场景 2: 用户可以在通话遭遇故障时进行恢复	高	高
3	场景 3: 用户可以在发生冲突时排除冲突	高	高
4	需求 1: 对故障现场信息的记录	高	中
5	需求 2: 对不同功能的切换	中	低
6	需求 3: 优先级队列和回拨机制解决冲突	中	中
7	设计约束 1: 恢复的时间不能超过 15 秒	中	高
8	设计约束 2: 不同服务之间要避免冲突	中	低

3.3 设计关注点与模式(pattern)/策略(tactic)选择

3.3.1 识别设计关键点

设计关注点	从属关注点
可用性保持	错误预防
故障恢复	对用户透明
	发生故障后系统故障状态处理
性能优化	恢复性能优化
	处理冲突性能优化

3.3.2 为从属设计关注点列举可选战略并选择合适的战略

1. 错误预防

#	模式名称	模式介绍
1	引入冗余和备份系统	确保在系统故障或错误发生时能够无缝切换到备用系统。这可以通过使用冗余服务器、备用网络连接和数据备份等方式来实现。
2	建立有效的监控机制	实时监测系统的性能和健康状态。通过监控关键指标，例如服务器负载、响应时间和错误率等，可以及时发现潜在问题并采取相应措施，以避免系统故障。

选择：引入冗余和备份系统

理由：容错机制可以帮助提高系统的可用性，并在错误发生时保持系统的稳定性。通过引入冗余和备份系统，可以确保即使主系统出现故障或错误，也能够无缝切换到备用系统，从而避免服务中断和用户受到影响。

2. 对用户透明

#	模式名称	模式介绍
1	保持一致的用户体验	在故障发生时，努力保持一致的用户界面和交互体验。用户不应该在故障恢复过程中遇到任何明显的界面变化或操作流程的改变。确保用户能够继续使用熟悉的界面和工作流程，减少对系统变化的适应成本。
2	提供实时状态更新	向用户提供实时状态更新，使他们了解系统的运行状况。这可以通过在用户界面或客户端应用程序中显示系统状态信息、故障通知或警报来实现。用户可以清楚地了解系统是否正常运行，以及是否存在任何故障或恢复操作。

选择：提供实时状态更新

理由：提供实时状态更新可以帮助保持用户对系统运行状况的了解，增强用户对系统的信任，并提供对用户透明性。可以提高用户对系统的了解和信任，保持用户的透明性，并减少用户可能因系统故障而遇到的困惑和不便。通过及时传达系统状态和相关信息，您能够与用户建立更紧密的沟通和合作关系，提升用户体验。

3. 发生故障后系统故障状态处理

#	模式名称	模式介绍
1	故障状态记录和评估	对故障状态处理过程进行记录和评估，以便今后的改进。收集有关故障处理的数据、操作步骤和结果。通过回顾和分析故障状态处理的绩效，识别改进的机会，并采取措施以减少类似故障的发生和处理时间。
2	紧急响应计划	建立紧急响应计划，明确团队成员在系统故障状态处理期间的角色和职责。确保团队成员了解应对措施和步骤，以便迅速、有效地处理故障状态。包括在计划中进行紧急通信、备份系统的启动、故障修复和恢复操作等。

选择：故障状态记录和评估

理由：故障状态记录和评估为持续改进提供了基础。通过分析记录的数据和评估结果，可以识别潜在的改进机会，包括流程优化、技术增强或资源调整。这有助于不断提高系统的可靠性、效率和性能。可以发现潜在的故障模式或趋势。这有助于预测将来可能出现的故障，并采取相应的预防措施。

4. 恢复性能优化

#	模式名称	模式介绍
1	并发处理和负载均衡	优化系统的并发处理能力和负载均衡机制，以确保在高负载或突发流量情况下的性能稳定性。考虑采用分布式架构、增加服务器容量、实现负载均衡策略等，以分担负载并提高系统的处理能力。
2	缓存和数据存储优化	通过合理的缓存机制和数据存储优化来提高系统的性能。使用适当的缓存策略，减少对数据库和外部资源的频繁访问，以提高响应速度。优化数据存储方案，包括数据索引、分区和压缩等，以提高数据的读写性能。

选择：并发处理和负载均衡

理由：可以将请求均匀地分发到多个服务器上，从而避免单个服务器过载。这样可以确保系统能够处理更多的请求，并保持较高的性能水平。减少了单个服务器的压力。如果其中一个服务器发生故障或停机，负载均衡可以自动

将流量转移到其他正常工作的服务器上，避免了单点故障导致的服务中断。这提高了系统的可用性和容错性。

5. 处理冲突性能优化

#	模式名称	模式介绍
1	冲突检测和解决机制优化	改进系统的冲突检测和解决机制，以提高处理冲突的性能。考虑采用更高效的算法或技术来检测冲突，并实施更快速和准确的解决方法。通过减少冲突检测和解决的时间和成本，可以提高系统的响应能力和用户满意度。
2	异步处理和消息队列	引入异步处理和消息队列机制，将冲突处理的任务异步化，提高系统的并发性能。通过将冲突处理任务分离出来，并使用消息队列将任务进行排队和分发，可以提高系统的吞吐量和响应能力。这样可以避免冲突处理阻塞主线程或影响其他任务的执行。

选择：异步处理和消息队列

理由：使用消息队列可以将冲突处理任务进行排队和分发，使系统能够以较低的延迟处理冲突。任务进入消息队列后，可以立即返回响应，而实际的冲突处理可以在后台进行。这样可以大大减少用户等待的时间，提高系统的响应速度和用户体验。当系统遭遇冲突时，将冲突处理任务放入消息队列中，即使系统出现故障或重启，任务也不会丢失。任务可以在系统恢复正常后继续处理，确保冲突得到适当解决，提高系统的可靠性和容错性。

3.3.3 决定所选模式/战略和 ASRs 的关系

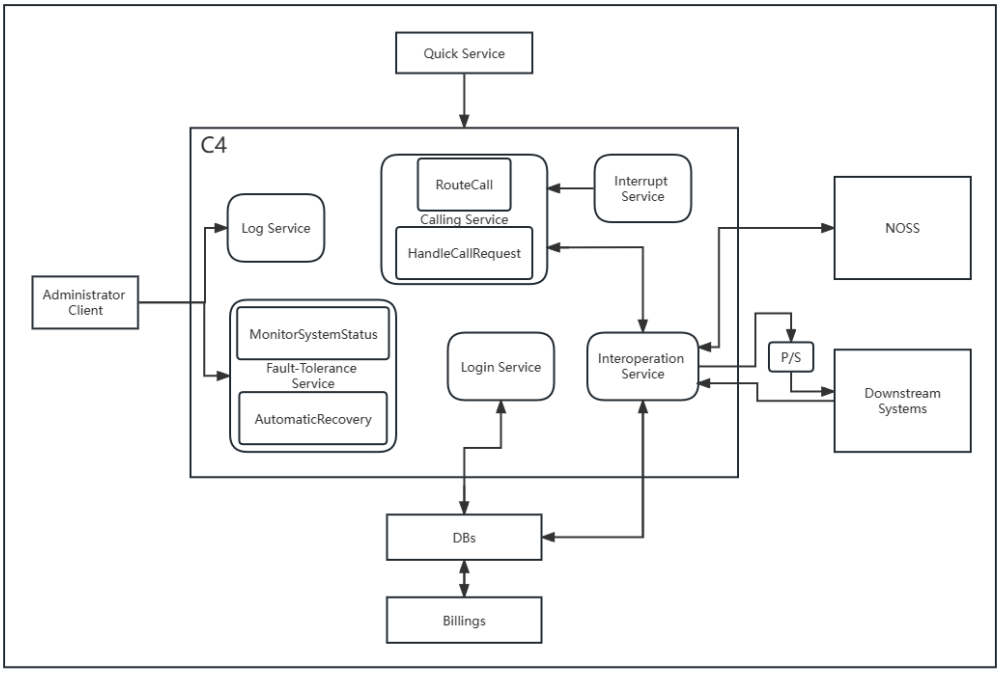
#	模式类型	模式选择	架构驱动
1	错误预防	引入冗余和备份系统	对故障现场信息的记录（需求 1） 不同服务之间避免冲突（设计约束 2）
2	对用户透明	提供实时状态更新	用户可以在同一电话中选择不同服务（场景 1） 对不同功能的切换（需求 2）
3	发生故障后系统故障状态处理	故障状态记录和评估	用户可以在通话遭遇故障时进行恢复（场景 2） 对故障现场信息的记录（需求 1）
4	恢复性能优化	并发处理和负载均衡	对故障现场信息的记录（需求 1）
5	处理冲突性能优化	异步处理和消息队列	用户可以在发生冲突时排除冲突（场景 3） 优先级队列和回拨机制解决冲突（需求 3） 不同服务之间避免冲突（设计约束 2）

3.3.4 捕获初步的架构视图

3.3.4.1 元素表

#	元素	是否在本次 ADD 迭代中涉及
1	HandleCallRequest	2
2	RouteCall	2
3	MonitorSystemStatus	2
4	AutomaticRecovery	2

3.3.4.2 架构元素视图



3.4 实例化架构元素并分配职责

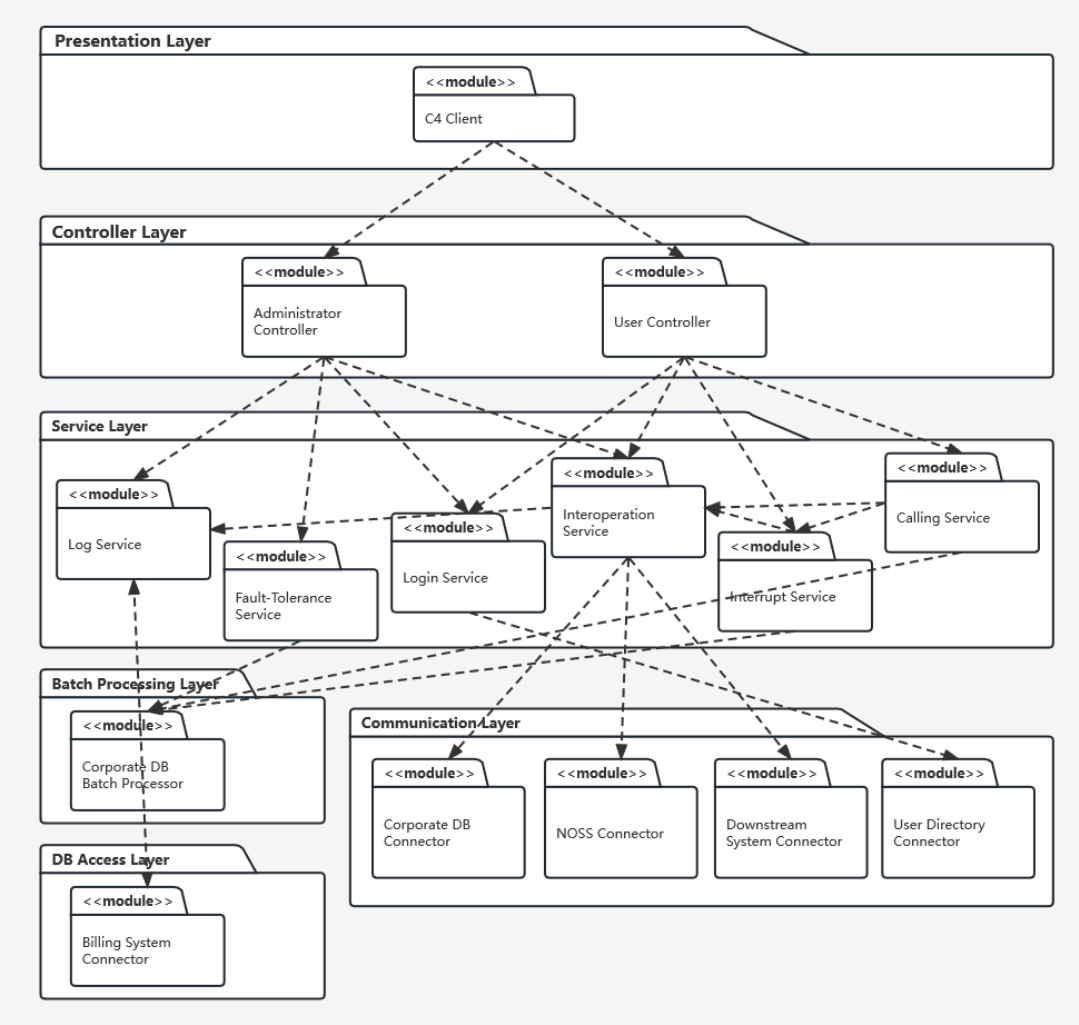
1. 处理呼叫请求（HandleCallRequest）：接收和处理来自客户的呼叫请求，包括请求新服务、更改服务配置或报告问题。
2. 路由呼叫（RouteCall）：将呼叫路由到适当的公司代表或其他适当的处理机制。
3. 监控系统状态（MonitorSystemStatus）：定期检查系统的运行状态和健康状况，包括服务器、网络连接和关键组件的正常运行。
4. 自动故障恢复（AutomaticRecovery）：在检测到故障时，自动采取措施以保证系统的可用性。这可能涉及到自动重启故障组件、切换到备用服务器或其他恢复机制。

3.5 为实例化的元素定义接口

来源元素	目的元素	接口	时间条件
CustomerCall	CompanyRepresentative	HandleCallRequest	0.1s
CompanyRepresentative	CustomerCall	RouteCall	0.1s
SystemStatus	SystemComponent	MonitorSystemStatus	全天候
SystemComponent	SystemStatus	AutomaticRecovery	2s

二、最终的软件架构文档

1、模块视图



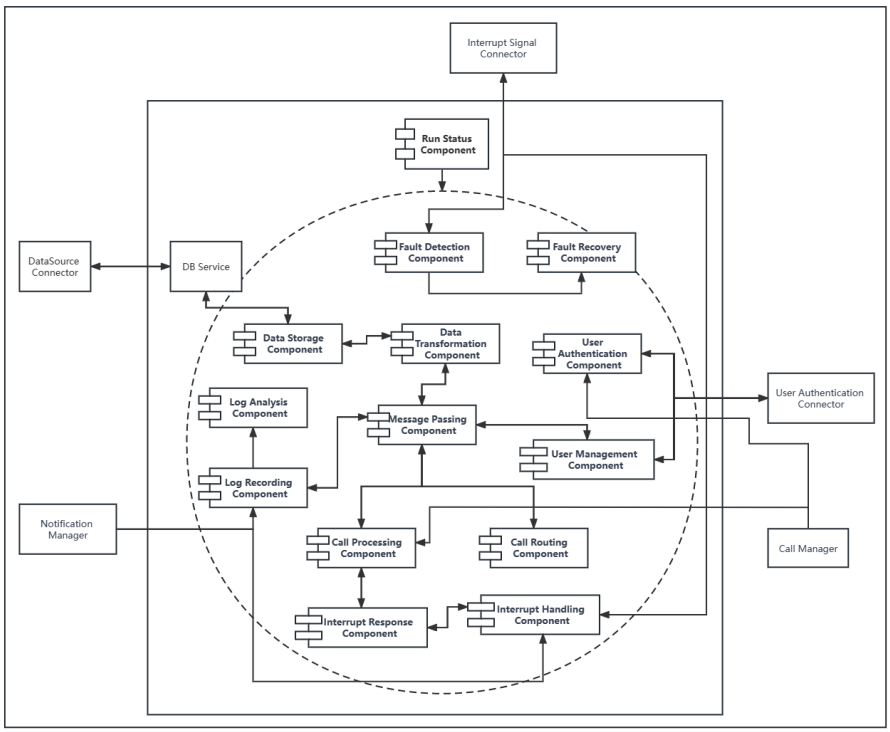
Element	Responsibility
Service Layer	该层包含用于包装 C4 系统向用户与管理员提供的服务的模块。这些模块使用 spring 框架进行开发组合。
Controller Layer	该层包含用于包装 C4 系统向用户与管理员提供的服务的模块。这些模块使用 spring 框架进行开发组合。
Presentation Layer	该层包含用于展示模块。
User Directory Service Connector	该模块负责连接用户数据库，提供用户的账户和权限信息。
Client	该模块负责界面展不，以 web 网页的形式与用户或管理员交互，并将其操作包装为对应请求与控制器层进行交互。

Administrator Controller	该模块负责处理网络请求，提供面向管理员的日志服务。
User Controller	该模块负责处理网络请求，并提供面向用户的登录、进行通话等服务。
Log Service	该模块负责记录保存用户操作曰志，并允许管理员查看用户用志。用户操作曰志将被直接保存至本地数据库。
Fault-tolerance Service	该模块负责自动监视定期批处理过程中银行对账单处理状态，收集、记录处理发生错误的对账单与信息，并等待管理员请求重新处理错误对账单。
Interoperation Service	该模块负责与其他模块进行复杂的互相操作，比如 Db 和 Billing 的数据的增删改查，NOSS 系统的相互调用服务，和下流系统进行互相操作等
Interrupt Service	该模块负责进行通话中断的恢复，采用保留通话状态并提供恢复选项和中断恢复机制
Calling Service	该模块负责提供高质量的通话服务，比如长时间通话、短休息时间、短间隔时间、支持通话并发任务等服务

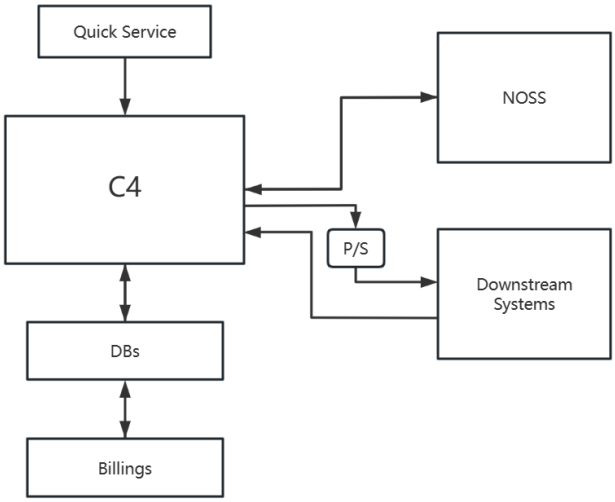
职责分配表(只包含新增部分)

2、 组件-连接器视图

2.1 主表示图



2.2 上下文图

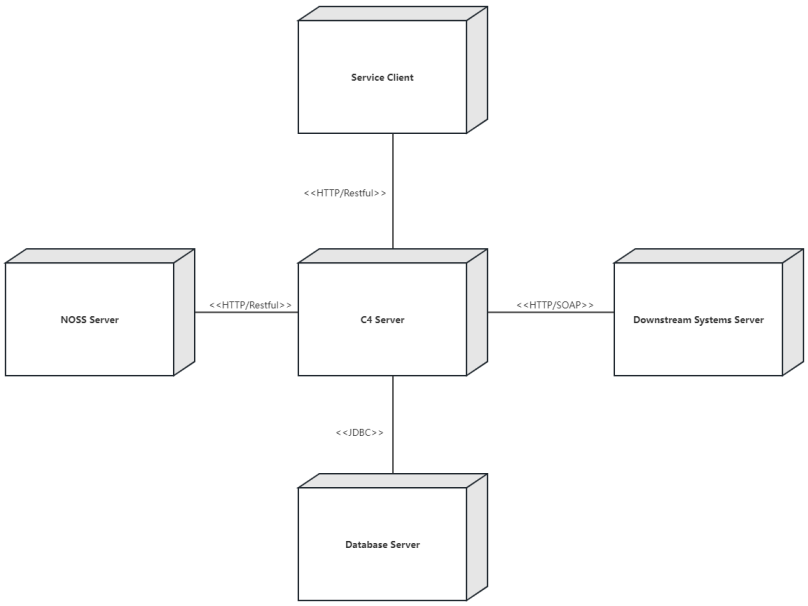


2.3 元素解析

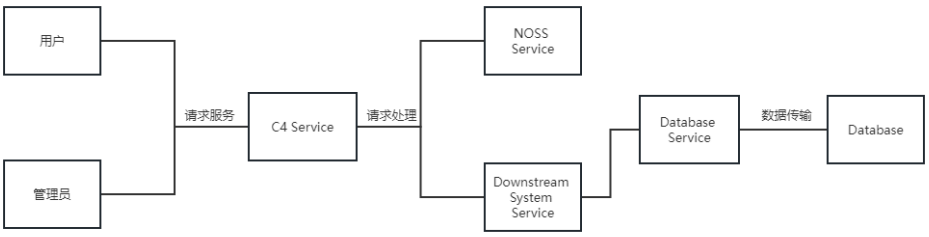
1. 用户认证组件（User Authentication Component）：负责验证用户身份和授权访问系统。
2. 用户管理组件（User Management Component）：处理用户账户的创建、更新和删除操作。
3. 日志记录组件（Log Recording Component）：负责记录系统的操作日志和事件。
4. 日志分析组件（Log Analysis Component）：对系统日志进行分析和提取有用的信息。
5. 中断处理组件（Interrupt Handling Component）：捕捉系统的中断事件，并根据需要进行相应的处理。
6. 中断响应组件（Interrupt Response Component）：对中断事件作出及时的响应，保证系统的稳定性和可用性。
7. 容错检测组件（Fault Detection Component）：监测系统中的故障或错误，并及时发现问题。
8. 容错恢复组件（Fault Recovery Component）：在出现故障或错误时，采取相应的措施进行恢复或修复。
9. 数据转换组件（Data Transformation Component）：处理来自不同系统的数据，进行格式转换和映射。
10. 消息传递组件（Message Passing Component）：负责系统之间的消息传递和通信。
11. 呼叫处理组件（Call Processing Component）：处理与客户的呼叫交互，包括请求新服务、配置更改和报告问题等。
12. 呼叫路由组件（Call Routing Component）：根据客户需求和系统状态，将呼叫路由到适当的处理单元。
13. 呼叫管理器（CallManager）：管理与客户的呼叫交互过程，包括呼叫路由、呼叫状态管理和呼叫记录的管理。
14. 中断信号连接器（Interrupt Signal Connector）：将中断处理组件与硬件系统连接，接收和处理来自硬件的中断信号。
15. 日志记录连接器（Log Recording Connector）：将日志记录组件连接到日志存储系统，将系统操作日志进行持久化存储。

3. 分配视图

3.1 主表示图



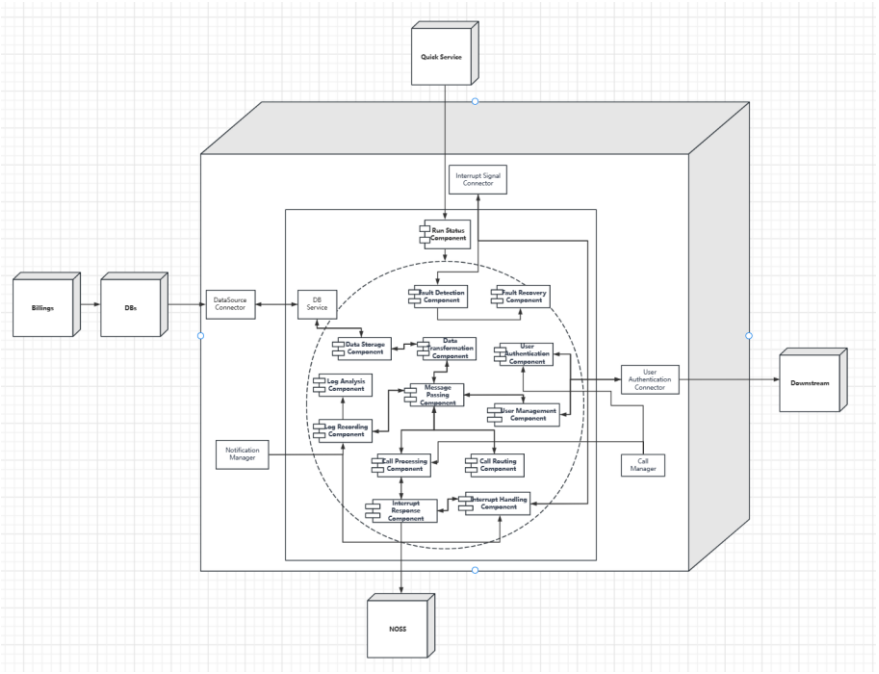
3.2 上下文图



3.3 元素解析

Service Client	分配给负责完成服务接通与交互的组件
C4 Server	分配给系统的主要处理调度系统
NOSS Server	分配给用于与 NOSS 交互的服务
Downstream Server	分配给负责与数据库交互的服务
Database Server	分配给负责存储处理过程中产生数据和系统自带的数据库

4. 交叉视图



三、 个人评语

1、 陈子凡 201250072

在本次软件架构设计的小组作业中，主要确定了文档的结构，在详细的设计过程中我主要负责了 C4 系统中重要的非功能性需求和约束的识别与提取，并根据 ASR 的选取过程确定了项目的 ASRs。在之后主要负责了迭代一的部分编写工作，确定了设计关注点，并分析了从属关注点，提出策略并根据列出的场景和约束确定较好的策略。

在使用 ADD 方法进行架构设计的过程中，我得到了以下的经验和总结:

- C4 的 reading 材料上已经给出了初步的架构，在给出基础架构的基础上提出了需求和约束，而 ADD 方法在已选取基本架构的基础上对软件架构进行迭代和完善的情况下更加具适用
- ADD 方法的迭代对于整体架构的分布推进是十分有效的，他选择采用逐步细化的方式来将软件系统中的每个元素细节化，从而让我逐渐深入对整个 C4 系统的需求、功能以及组件的理解；在完成了关注点的分离后，我可以放心并专注地完成眼下地设计工作。
- 在使用 ADD 方法地过程中，我发现 ADD 分析法可以帮助我有效地辨识出系统地质量属性模块。在对整体地系统架构有一个初步夫人了解之后，可以围绕制定地元素和 ASRs 进行细节地设计，确定一个个设计关注点并为从属设计关注点列举可选战略，最后做出更好地选择

2. 邢佳勇 201250071

在本次软件架构设计的小组作业中，我主要负责迭代二的捕获初步架构视图、实例化元素并分配职责、为实例化的元素定义接口部分，以及最终软件架构文档中的第 2 部分和第 4 部分。参与了整个架构设计的讨论，选取迭代二元素。

在使用 ADD 方法进行架构设计的过程中，我得到以下的经验和总结:

1. 迭代式开发：将架构设计过程分解为多个迭代，并逐步完善和细化架构。每个迭代都可以关注特定的属性和设计问题，并逐步构建出更完善的架构。这种迭代式的方法有助于在整个过程中不断验证和改进设计。

2. 利用视图：使用不同的视图来表示架构的不同方面，例如组件视图、连接器视图、部署视图等。通过这些视图，可以更清晰地描述系统的组织结构、组件之间的交互和依赖关系，以及系统的部署情况。
3. 模块化和解耦：将系统划分为模块化的组件，并通过明确定义的接口实现解耦。这样可以提高系统的可维护性、可测试性和可扩展性，同时降低组件之间的耦合度。

3. 郁博文 201250070

在本次软件架构设计的小组作业中，我主要负责确定了迭代二的设计关注点，并分析了从属关注点，提出策略并根据列出的场景和约束确定策略。同时在最终软件架构文档中完成了分配视图的构建。

在使用 ADD（Attribute-Driven Design）方法进行架构设计时，以下是一些经验和总结：

1. 理解需求：首先，深入理解系统的功能需求和非功能需求。这包括对系统目标、用户需求、业务需求和技术约束的全面了解。只有在清楚地定义需求之后，才能更好地进行架构设计。
2. 确定关键属性：识别和确定系统最重要的属性。这些属性可以是性能、可靠性、安全性、可维护性等方面。通过了解关键属性，可以更好地理解系统的关注点，并将其纳入到架构设计中。
3. 划分系统：将系统划分为模块或子系统，以支持解耦和模块化。模块之间的划分应基于功能、职责和相互之间的依赖关系。使用合适的架构模式（如分层、微服务、事件驱动等）来组织和管理模块。

4. 郑启睿 201250229

在本次软件架构设计的小组作业中，我主要负责确定文档的整体结构，在架构驱动设计中完成了部分非功能性需求分析，在迭代一中完成了捕获初步架构视图、实例化架构元素并分配职责，并为实例化出来的元素定义接口。同时在最终软件架构文档中完成了分模块的构建和职责分配表的编写。

在使用 ADD 方法进行架构设计的过程中，我得到以下的经验和总结：

1. 针对功能性需求：ADD 方法比较重视在分析前对系统的功能性需求，非功能性需求和约束的思考。需要对系统与使用场景有一个比较全面的认识。
2. 多次迭代：ADD 方法要求多次迭代，每次关注一个元素的方法能够很好的找出每一个模块的不足以及问题所在，这是一个非常好的自我更新方法。每次战略的选择就是一个决定具体设计方法的流程，在设计的时候必须结合实际情况与其他因素进行具体的设计方法选择。
3. 难点分析：我个人认为 ADD 比较难的地方是确定分析那个系统，在系统比较错综复杂、需求交互频繁的系统中，确定一个核心系统，并要对其进行准确的分析是需要非常仔细地思考系统所有部分的。