**课程设计报告**

组号：（ 09 ）

|  |  |  |  |
| --- | --- | --- | --- |
| 角色 | 小组成员 | 班级 | 得分 |
| 组长 | 郭柏良 | 2017级2班 |  |
| 组员 | 林墨馨 | 2017级2班 |  |
| 胡映鑫 | 2017级2班 |  |
| 廖惠康 | 2017级2班 |  |
| 邹鹏宇 | 2016级4班 |  |

**软件学院**

**2019年11月**

|  |  |
| --- | --- |
| **批改意见：** |  |
| **批阅老师：** |  |
| **批阅日期：** |  |

# 架构分析文档

**一、采用场景化方法，明确定义目标系统应该具有的质量属性，如下所示：**

**（1）目标系统应该具有availability：**

|  |  |
| --- | --- |
| 场景1（Scenario）：平台可以检测、记录故障，以及系统可以正常运行的时候 | |
| 刺激源（source） | 使用平台的各种用户，如服务大厅的工作人员、数据库的管理人员、负责增删改查信息的技术人员、希望在网上能够查到信息的市民；硬件、软件；物理基础设施，如服务器、线路。 |
| 刺激（stimulus） | 系统发生崩溃、出现信息丢失、超时请求、返回的信息紊乱等等。 |
| 制品（Artifact） | 可靠有效的通讯通道（确保不会产生错误的相应或超时）；强大的处理器（可减少故障产生的频率）；稳定的物理存储器（保证可以记录故障或者相关信息） |
| 环境（Environment） | 系统正常运转时；启动系统或终止系统时；修复故障时 |
| 响应（Response） | 系统崩溃后在日志里生成相关的记录；能够及时终止发生故障的源头；通过向备份服务器请求数据以掩盖故障 |
| 响应度量（Response Measure） | 从宕机或者故障发生到修复的所耗时间；故障发生到发现故障源的耗时；系统发生故障的概率 |

**（2）目标系统应该具有interoperability：**

|  |  |
| --- | --- |
| 场景1（Scenario）：A市退休人员管理系统向A市人力资源管理系统请求数据 | |
| 刺激源（source） | 退休人员管理系统 |
| 刺激（stimulus） | 退休人员管理系统向人力资源管理系统发送请求，希望收到一名退休员工在人力资源系统里的信息：姓名、年龄、性别、工作时长、曾任职的公司、退休方式等信息。 |
| 制品（Artifact） | 人力资源管理系统 |
| 环境（Environment） | 人力资源管理系统在接收到请求之前处于运行状态，或者退休人员管理系统对人力资源管理系统的运行状态已知。 |
| 响应（Response） | 人力资源管理系统把请求的信息，按照要求、有序地发送给退休人员管理系统 |
| 响应度量（Response Measure） | 退休人员管理系统对发过来的信息进行检查，信息格式是否正确，顺序是否对，信息是否缺少或者冗余。 |

**（3）目标系统应该具有Modifiability：**

|  |  |
| --- | --- |
| 场景（Scenario） | 保险管理系统中用户对自己的账户进行修改。 |
| 刺激源（source） | 终端用户——比如保险管理系统中的某用户。 |
| 刺激（stimulus） | 某个保险管理系统中的用户想购买新的保险。 |
| 制品（Artifact） | 一套体现“购买保险”功能机制的代码，包括选择保险具体种类的函数、选择金额的函数、付款的函数、修改数据库购买者账户信息的函数以及相关的设置等等。 |
| 环境（Environment） | 系统正常运行时。 |
| 响应（Response） | 保险管理系统成功受理该用户的购买请求。 |
| 响应度量（Response Measure） | 整个购买过程控制在一小时以内，购买者的购买金额、购买种类等数据会正确的修改与更新，购买者能实时看到自己的购买情况，产生相关购买的电子凭证，记录相关的时间、保险代号、购买者者承诺书等等信息。 |

**（4）目标系统应具有的Performance：**

|  |  |
| --- | --- |
| 场景（Scenario） | 大量用户同时在线活动(比如购买资金，购买保险，查看信息)。 |
| 刺激源（source） | 系统的使用者（终端用户） |
| 刺激（stimulus） | 大量periodic、sporadic、stochastic(周期性 零星的 随机的)事务、事件同时发生、到达，比如某个特殊的节日，平台下各数据库系统的大量用户都登录进行相关的交易，查看信息，更改信息，删除信息等操作。 |
| 制品（Artifact） | 平台下的所有数据库系统 |
| 环境（Environment） | 正常操作时（Normal Operation） |
| 响应（Response） | 事务一件件有序地并行化地被处理 |
| 响应度量（Response Measure） | 系统的平均事务吞吐量、平均事务延迟时间、事务处理时间的抖动(jitter)情况、事务失败率、请求超时的次数等等。为了避免不同事务的处理时长不同的问题，可以将事务归类，在同一类事务下求响应度量并结合过去的数据（昨天同时间、上个月同一时间、去年同时间等等）进行比较。 |

**（5）目标系统应具有的Security：**

|  |  |
| --- | --- |
| 场景（Scenario） | 外部人员使用合法用户身份信息利用脚本进行DDOS攻击 |
| 刺激源（source） | 系统外部人员 |
| 刺激（stimulus） | 频繁访问系统服务，降低系统可用性，试图使系统拒绝向合法用户提供服务 |
| 制品（Artifact） | 平台下所有数据库系统 |
| 环境（Environment） | 系统联网，防火墙处于工作状态，系统处于完全可操作状态 |
| 响应（Response） | 限定用户登录所使用IP地址，拒绝用户同时从多IP地址登录与操作 |
| 响应度量（Response Measure） | 攻击检测响应时间间隔，从攻击成功后恢复的时间间隔，系统承受攻击后数据与被攻击前的一致率，系统抵御攻击的次数 |

**（6）目标系统应具有的Testbility：**

|  |  |
| --- | --- |
| 场景（Scenario） | 资金（基金）综合管理系统向社会保险管理系统请求收集数据。 |
| 刺激源（source） | 单元测试人员，使用自动化测试工具 |
| 刺激（stimulus） | 资金（基金）综合管理系统向社会保险管理系统发送一个查询请求，请求查询参加了社会保险的市民使用社会保险资金的情况 |
| 制品（Artifact） | 资金（基金）综合管理系统 |
| 环境（Environment） | 资金（基金）综合管理系统在社会保险管理系统发送请求前就已经有了本市市民的相关信息。社会保险管理系统需要知道某个人是否参加了社会保险或者需要知道本市今年参加社会保险使用的情况等。 |
| 响应（Response） | 社会保险管理系统把资金（基金）综合管理系统需要知道的合法的信息打包后按资金（基金）综合管理系统提出的要求有序发给资金（基金）综合管理系统。 |
| 响应度量（Response Measure） | 资金（基金）综合管理系统对社会保险管理系统发过来的信息进行评估，响应时间的长短，信息是否正确、齐全，是否有冗余、错误等等。 |

**（7）目标系统应具有的Usability**

|  |  |
| --- | --- |
| 场景（Scenario） | 用户在多个信息系统登记个人信息，办理不同的业务 |
| 刺激源（source） | 终端用户（市民） |
| 刺激（stimulus） | 用户希望能够在一个系统处理全部业务，能够在移动端使用 |
| 制品（Artifact） | 一个信息统一交叉能够办理所有业务的个人信息系统，并且能在移动端使用 |
| 环境（Environment） | 系统正常运行时候，用户正常使用 |
| 响应（Response） | 对信息系统进行信息更新，并且能够即使删除冗余数据 |
| 响应度量（Response Measure） | 短时间内对信息数据系统进行更新，并且进行删除冗余数据来维护数据的交叉 |

**二、实现上述质量属性采用的具体战术与架构模式:**

**（1）实现可用（有效）性（Availability）而采用的战术与架构模式：**

1.错误检测

1.1心跳 heartbeat

各个系统周期性地向其他系统发送一个“心跳”信息，另一个系统收听该信息，以确定与其他系统之间的通信通道是正常且该系统是运转的。

1.2投票（voting）

请求的信息可能是冗余的并存在与多个系统内，只有当请求的冗余信息全部正确且一致时才决定该信息时可靠的。

2.恢复

2.1监测和修复

被动冗余（passive redundancy）。系统对信息增删改事件做出相应后，并通知其他的备用服务器进行必须的状态更新和数据变化，这样可以确保系统的数据和冗余的备用数据时一致的，这样在故障发生的时候可以快速恢复，可用性高、造价不贵，计算起来不是很复杂。这个战术可以用来在系统正常运转的情况下实现错误检测与恢复。

2.2重新引入（reintroduction）

Shadow操作：出现故障的组件可以以“Shadow模式”运行，这样就可以在系统恢复正常之前模仿故障的工作组件的行为。

3. 预防

removal from service：从操作中删除了系统的某个组件，以执行某些活动来仿制预期发生的故障，比如在非工作时间重新启动备用服务器来防止内存泄漏。

**Availability架构模式：**

Client-server pattern:将系统分为客户端和服务器，客户端可周期性地向服务器发送信息，如Heartbeat，以监测客户端服务器之前通讯通道是否正常，保证用户在请求数据的时候，服务器是正常运转的，实现可用性。

**（2）实现互操作性（Interoperability）而采用的战术与架构模式：**

1.Locate战术

平台内的各个系统内存在一个列表，能够知道其他系统的存在并且知道怎么可以找到其中一个系统，可以向其发送请求，并能够识别接收到的请求来自哪个系统，例如：退休人员管理系统知道人力资源管理系统，能向其发送请求，并识别人力资源管理系统的相应。

2.Manage Interface战术

2.1 Orchestrate

由于平台中涉及到了多个系统，所以用户请求的信息可能同时存在与多个系统中，例如：退休人员管理系统需要从人力资源管理请求工作经历信息，但是需要从公安的居民信息系统中请求个人信息。在这样复杂的任务中，引进控制机制来协调向不同系统请求的内容和顺序。

2.2 Tailor interface战术

提升接口的能力：

例如：居民身份信息大部分存储在公安的居民信息系统里，其他系统可能会发送大量的请求以获得不同的居民信息，可增加缓冲应对这种情况。

去除不必要的功能：

例如：由退休人员管理系统里的退休信息不能被其他的系统请求到，如资金管理系统。

**Interoperability架构模式：**

Share-Data Pattern：

将系统存储的分成多个数据模块，其中被多个系统共用的数据存储在一个子系统中，例如：姓名、性别、身份证号等信息，这样当信息被修改时可避免在多个系统中修改冗余的信息，避免造成不必要的操作。而一些系统特有的数据保存在另一些子系统中，如退休人员管理系统中的退休方式信息。

**（3）实现可修改性（Modifiability）而采用的战术与架构模式：**

1. 减小单个模块的大小(Reduce Size of a Module)

1.1 分离模块(split module)

如果整个保险管理系统都没有分层、分子模块，所有信息（比如用户和保险负责人的信息）都是处于同一层次上，那么这个系统查找用户信息的时间成本会很高，从而导致更改成本也很高，修改费用将会很高，时间也要很长。但如果把整个系统分成层次清晰的小系统，比如把保险管理系统分成保险信息子系统、保险负责人子系统、用户系统等子系统，那么用户需要进行购买时可以方便地对系统进行更改。从而使整体的平均修改成本降低。

1. 提高内聚度 (increase cohesion)

2.1 增加语义连贯性 (increase semantic coherence)

提高内聚性可以降低修改带来的影响，也可以降低修改所需要的成本。在购买新保险地场景中，修改用户信息并不会使的保险信息发生更改，也不会是负责人的信息发生更改，所以可以将他们放在不同的子模块中，从而降低修改成本。

1. 降低耦合度(reduce coupling)

3.1提供封装(Encapsulation)

为用户信息系统提供一个授权的API，从而提供一层抽象，降低了对一个模块的更改传播到其他模块的可能性。

3.2 使用一个中间层来打破、代替依赖关系(use an intermediary breaks a dependency)。

用户系统修改用户信息需要依赖于保险信息子系统的一些信息，还需要负责人系统的授权信息等，这些工作可以交给一个中间层（Control），控制它们之间的通信、信息同步等，而不是把这些操作分散在不同的子系统中。

**Modifiability架构模式：**

Layered Pattern：

采用多层体系，在对各个子数据库系统分层（纵向）的同时，也横向对所有系统分层，每个子系统都有表示层（UI层）、应用层（服务层）、业务逻辑层，将子任务根据不同的抽象程度进行分级处理。

**（4）实现性能（Performance）而采用的战术与架构模式：**

1.(控制资源需求)Control Resource Demand

1.1管理采样率(manage sampling rate)

比如，在信号处理系统中，降低网站某些图像、图片、视频的采样率，牺牲一定的保真度，换取更多的处理量。从而在大访问量的前提下网站依旧能维持一定的延迟时间，降低超时率与失败率。应用这种策略时要确认较低的保真度比丢包更可取。

1.2 提高资源效率(improve resource efficiency)

优化数据处理算法，使得数据处理具有更低的时间和空间复杂度，从而提高系统在大访问量下的性能。

1.3 减少开销(reduce overhead)

采用周期性清理低效率的资源的策略，例如定期重计算或重初始化哈希表和虚拟内存映射；使用多个单线程的服务器而不是一个多线程服务器,并在它们之间分配工作负载。

1.4 限制事件响应(limit event response)

当离散事件到达系统（或元素）的速度太快而无法处理时，则必须让事件排队，直到可以对其进行处理。

2.管理资源(Manage Resources)

2.1 引入并行(introduce concurrency)

引入并行计算，比如对不同的系统进行查询可以同时进行，同一个系统下不同用户请求修改自己范围内的信息也可以同时进行等等。在这种情况下，可以引入调度算法来实现目标，并能在最大程度上提供公平性。

2.2 增加资源(increase resources)

增加可用资源，使用更快的处理器，并使用更多的处理器，更大的储存空间，以及更快的网络，都可以在一定程度上提高系统的性能。

2.3 维护数据的多个副本(Maintain multiple copies of data)

可以将一些重要的信息复制几分作为缓存，避免几个请求对同一份资源的争用。在这种情况下，要注意保持数据的一致性。

**Performance架构模式：**

客户端-服务器模式(Client-Server Pattern)：

对于管理好资源以及稳定的提供服务，这种模式具有比较大的优势。这种模式可以整合服务资源，并提供接口供客户端使用，对于提高性能有很大的作用。

**（5）实现安全性（Security）而采用的战术与架构模式：**

1.检测攻击（Detect Attacks）

记录一定时间间隔内用户登录IP数与执行的操作数，大于阈值时判断为攻击。

2.抵制攻击（Resist Attacks）

针对可疑用户限定最后登录IP为可用IP，丢弃来自其他IP的请求。

3.对攻击做出反应（React to Attacks）

记录可疑用户除最后登录IP外的所有IP与请求，生成攻击日志并归档，通知安全人员分析处理。

4.后攻击恢复（Recover from Attacks）

对可疑用户在与检测攻击所用的相同时间间隔内执行的事务全部进行回退，可疑用户后续登录需额外的安全认证以解除IP限制。

**（6）实现可测试性（Testability）而采用的战术**

1. 控制和观察系统状态

1.1特化接口

通过测试工具或通过正常执行来控制或捕获组件的变量值。

1.2 记录回放

捕获跨接口的信息，并将其用作进一步测试的输入。

1.3 沙盒

将需要测试的系统与现实世界隔离开来，从而可以进行测试，而不必担心撤消测试的后果。

2. 限制复杂性

2.1限制结构复杂性

避免或解决组件之间的循环依赖关系，隔离和封装对外部环境的依赖关系，并通常减少组件之间的依赖关系。

**（7）实现易用性（Usability）而采用的战术与架构模式：**

1.聚合（Aggregate）：

将用户的信息统一储存在数据应用系统，不同需求的业务如果需要相同的个人信息，将这些信息聚合到一组，用户在某一个业务中登记了个人信息，聚合在同一个组的个人信息可以同时更新。系统同时可以在移动端使用。

1. **Architecturally Significant Requirement(ASR):**

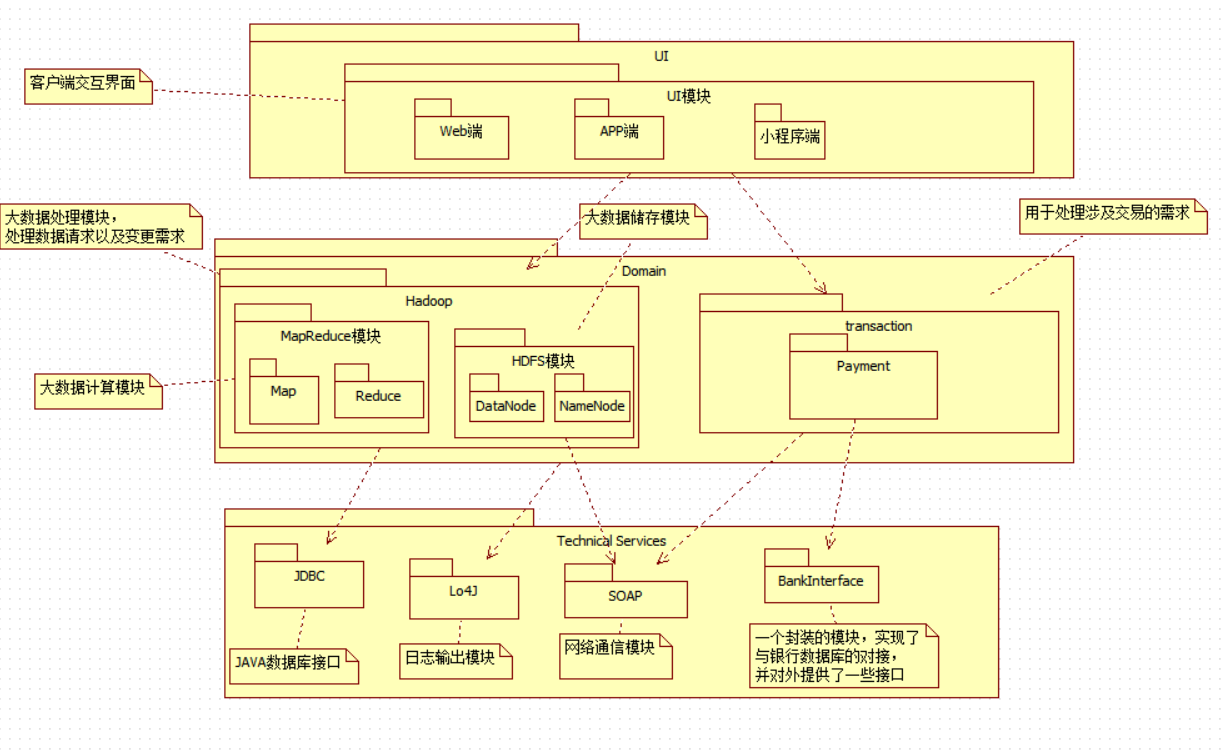
|  |  |  |
| --- | --- | --- |
| **Quality Attributes** | **Attribute Refinement** | **ASR** |
| **Availability** | 发生故障的概率 | 不会出现导致系统长时间崩溃的故障，平均每个月发生故障的数量在8个以下（H,M） |
| 故障修复时间 | 当系统内的某个组件出现故障时，工作人员能通过日志在半个小时内找到错误，并在一个小时内解决故障。（H，M） |
| No downtime | 当系统进行更新的时候，仍然要维持系统的运转，确保能够全年24小时运转（H，L） |
|  |  |  |
| **Interoperability** | 通信传输速度 | 存在通信的不同系统间存在可靠高速的通讯通道，能够快速传输数据，能带给用户良好的使用体验（H，M） |
| 数据正确 | 数据在传输的过程中，格式、顺序是正确无误的，不会因为干扰而出现信息丢失或者不一致（H,M） |
|  |  |  |
| **Modifiability** | 正确率 | 在用户正确操作的前提下，对系统数据库的修改能在10秒钟内完成，且保证100%的准确率(H,L) |
| 涉及规模 | 对于一次修改，应该使修改涉及的范围限制在一个模块之内(M,M) |
| 系统修改成本 | 对于系统前端界面的微调或者后端逻辑的修改，都可以由程序员小组在2h内完成(H,M) |
|  |  |  |
| **Performance** | 响应时间 | 在正常情况下，系统能在10s内对用户提交的事务作出响应，在高压情况下，系统能在15s内对用户的事务作出响应。(H,M) |
| 失败率 | 在高压情况下，系统能保持不大于0.5%的事务失败率(H,M) |
| 事务吞吐量 | 在用户高峰期，系统能够保持这样的平均速度：每秒钟正确处理600个普通的原子事务或者300个比较复杂的原子事务（M,M) |
|  |  |  |
| **Security** | 认证响应 | 用户登录验证身份时，系统短信验证有效时间为60秒 |
| 权限 | 不同用户（如数据库管理员和一般用户）对数据的操作权限与权限范围不同 |
|  |  |  |
| **Testability** | 问题定位 | 当系统增加新功能或者更新时，开发者能够迅速定位到系统出现问题的具体位置。 |
| 维护 | 当有新的需求时，开发者可以在5小时之内快速增加功能，修改界面并维护系统。(M,L) |
|  |  |  |
| **Usability** | 冗余数据处理 | 用户在不同平台办理业务会产生冗余数据，对这些数据进行处理，使数据统一化，聚合冗余信息，减少用户对于相同信息的重复登记。 |
|
|
| 智能平台 | 信息系统能够根据用户的数据，给出一些智能的决策建议 |
|
| 数据清理 | 数据库定时对数据库的信息进行检查，将过一小时的冗余数据删除 |
|

# 

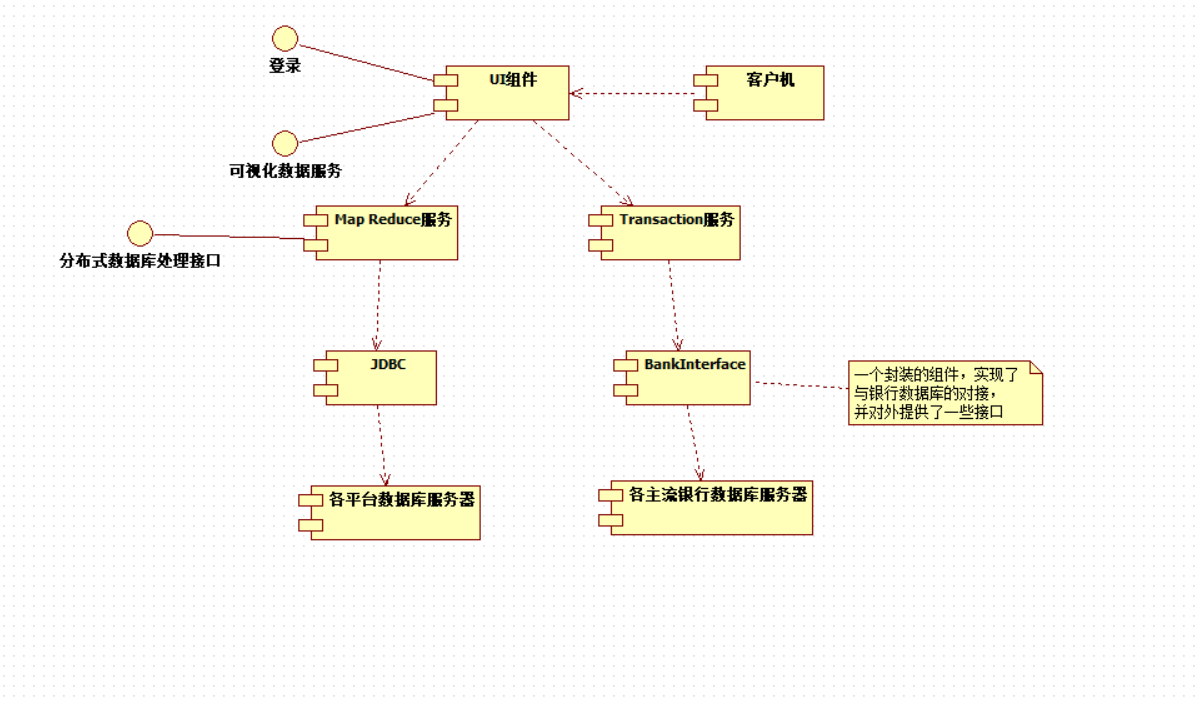
**架构设计文档**

**1.Len Bass 三视图**

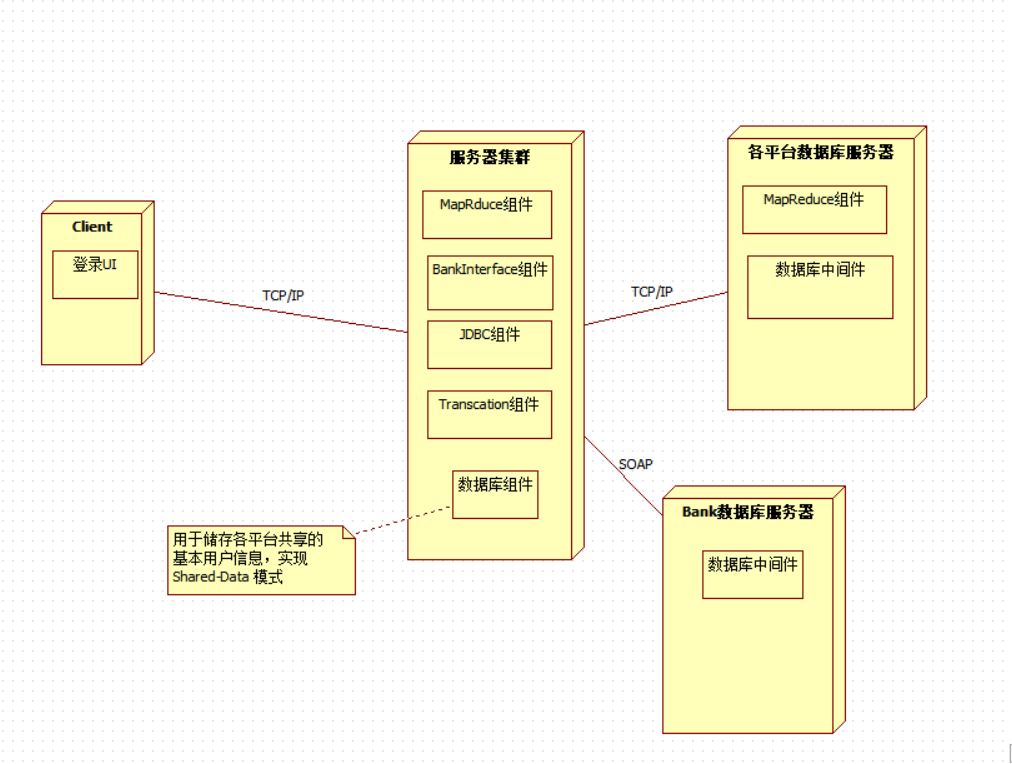
**1.1模块视图**

****

**1.2组件连接图**

****

**1.3分配视图**

****

**2.设计思路**

**2.1采用Shared-Data Pattern 在服务器端增加一个通用数据库**

说明: 由于各平台数据库太多，在用户进行登录操作时，若是一个个数据库去验证，则会使登录操作响应时间很长，严重影响系统的Performance。而且，由于各平台都有各自的信息数据库，数据的更新操作复杂性也成了一个影响Usability的问题。所以我们决定在服务器端建立一个通用的数据库(通用数据库的信息是由各平台共享而非某平台特有的)，储存各平台用户的基本信息，从而保证系统的性能和易用性。

**2.2采用Layer Pattern 将系统分为不同的层次**

说明: 由于系统是一个综合性的系统，发生变更的频率会比较高，为了使变更的影响降低，保证系统的Modifiablity以及Testability ,我们采用了分层模式，对系统进行分层，每个层有独立的模块，这样就可以最大程度上降低变更带来的影响。而且分层模式限制了不同用户的接口，在一定程度上保证了系统的Security。

**2.3采用Client-Server Pattern**

说明: 本系统是综合性的系统，具有许多的服务与资源，于是我们决定使用Client-Server Pattern ，在客户端提高服务接口，而用服务器对服务和资源进行管理和调度，在一定程度保证了性能和易用性。

**2.4服务器使用PSSA**

说明: 本系统的服务器使用云计算服务企业提供的PSSA(Platform as a Service) , 而不必自己购买机器作为服务器。

**2.5使用SOAP**

说明: 为了提高系统间信息交互准确性，保证系统Interoperatability , 我们在部分连接上使用了SOAP。

**2.6在数据处理使用了Hardoop 架构**

说明: 为了提高系统数据处理能力，我们决定采用Hardoop 架构。Hardoop主要分为两部分 HDFS 和 MapReduce。其中，HDFS是一种分布式文件系统，包括了DataNode 和 NameNode ，对于同一数据，通常由几个备份，这样，即时部分数据丢失，也不会对系统有太大影响(类似于Active Redundancy)，确保了系统的Availability。而MapReduce 包括了Map 和 Reduce ,用于并行处理数据，有助于提高系统的性能。

**3.设计目标**

**3.1 解决”数据孤岛”问题**

说明: 系统能够作为用户于各平台数据库之间的中间层，能够接受并分析用户请求，映射到对应数据库的操作；能够快速与各平台数据库进行交互；降低用户相关操作难度，简化获取数据过程，解决”数据孤岛问题”。

**3.2 实现各平台数据库同一接口**

说明: 系统应该实现一个在各平台数据库基础上的同一接口，用于在不改变各平台数据库系统的前提下兼容各平台数据库系统。

**3.3 实现统一平台**

说明: 系统应当提供一个统一的平台，从而解决用户需要重复填写信息以登录不同系统的问题。同时，该平台必须具有处理各平台业务的功能。并且，该平台必须在移动端也可用。

**3.4 解决数据冗余问题**

说明: 系统应当在一定程度上解决数据冗余问题，并简化用户需要填写的信息，使查询与更新操作更加简单。

**3.5 实现可视化数据服务**

说明: 系统应当能够提供自动生成相关数据报表与数据分析结果的服务，并在分析结果基础上提高一些智能化建议。

**4.设计约束条件**

**4.1 系统需在各平台数据库基础上开发**

说明: 平台需要建立在各平台数据库服务器基础上，需要建立统一的数据库接口；系统需要作为一个统一平台；应当尽量避免更改原有数据库。

**4.2 系统需具有Availability**

说明: 需设置适当规模的服务器集群，降低单个服务器宕机带来的影响并且保证服务器能一直进行工作；需保存日志文件，用于维护系统以及修复系统时快速定位错误；对数据进行主动备份和被动备份，避免数据丢失带来的影响以及快速恢复。

**4.3 系统需具有Interoperability**

说明: 需设置高速网络，保证系统间通信速度足够；需使用更安全的数据传输服务(如SOAP) 确保数据的准确传输。

**4.4 系统需具有Modifiability**

说明: 需将系统按功能分成一个个模块，从而降低系统修改带来的影响；系统应该具有良好的架构分层，从而降低系统维护所需的成本。

**4.5 系统需具有Performance**

说明: 系统需具有较高的数据处理能力，能快速对用户的请求作出响应；系统设置一定规模的服务器集群，分摊访问量，提高系统抗压能力，在高峰期也能保持较高的准确率；系统应具有并行处理事务的能力，具有较高的吞吐量；系统能对资源进行及时的管理与重分配。

**4.6 系统需具有Security**

说明: 系统应具有一定的信息保护措施，能验证用户输入验证的时间；系统需具有权限管理功能，能区分用户的权限。

**4.7 系统需具有Testability**

说明: 系统提供模块的接口，便于测试使用沙盒，插桩等策略，快速定位问题。

**4.8 系统需具有Usability**

说明: 系统通过聚合冗余信息，减少重复单一的操作，降低用户的使用难度；系统能生成数据报表和分析，提供一些智能的决策建议。

**架构评估文档**

1. **Availability**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 场景#: A系列 | 场景: 平台可以检测、记录故障，以及系统可以正常运行的时候 | | | |
| 属性 | Availability | | | |
| 环境 | 系统正常运转时；启动系统或终止系统时；修复故障时 | | | |
| 刺激 | 系统发生崩溃、出现信息丢失、超时请求、返回的信息紊乱等等。 | | | |
| 响应 | 系统崩溃后在日志里生成相关的记录；能够及时终止发生故障的源头；通过向备份服务器请求数据以掩盖故障 | | | |
| 架构决策 | 敏感点 | 权衡点 | 风险点 | 非风险点 |
| 心跳(heartbeat) | S1、2 |  | R4 | N5 |
| 投票（voting） | S1 |  |  | N6 |
| 监测和修复  被动冗余（passive redundancy） | S1、4 |  | R4、6 |  |
| 重新引入（reintroduction） | S1、4 |  | R4 |  |
| 预防 | S1、4 | T4 | R4 |  |
| 架构推理 | 心跳在SOAP系统下增大开销  voting可以保证更高的正确响应率  采用重新引入保证全天可获得性保持在高的状态  被动冗余增加测试难度  没有预防机制会使可获得性大大降低 | | | |
| 架构视图 | 检测、预防故障  出现故障  刺激    修复故障  正常响应 | | | |

**2.Interoperability**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 场景#: I系列 | 场景: A市退休人员管理系统向A市人力资源管理系统请求数据 | | | |
| 属性 | Interoperability | | | |
| 环境 | 人力资源管理系统在接收到请求之前处于运行状态，或者退休人员管理系统对人力资源管理系统的运行状态已知。 | | | |
| 刺激 | 退休人员管理系统向人力资源管理系统发送请求，希望收到一名退休员工在人力资源系统里的信息：姓名、年龄、性别、工作时长、曾任职的公司、退休方式等信息。 | | | |
| 响应 | 人力资源管理系统把请求的信息，按照要求、有序地发送给退休人员管理系统。 | | | |
| 架构决策 | 敏感点 | 权衡点 | 风险点 | 非风险点 |
| Locate | S1、2、3 |  |  | N1、2、3 |
| Orchestrate | S2、6 | T6 | R2、6 |  |
| Tailor interface | S2、6 | T2、3、6 | R2、3、6 |  |
| 架构推理 | 确保各系统相互可见  协作机制合理性需要在部署后评估  使用接口能提高可修改性 | | | |
| 架构视图 | 发送信息请求  响应请求  搜索信息源  确认响应  刺激 | | | |

**3.Modifiability**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 场景#: M系列 | 场景: 保险管理系统中用户对自己的账户进行修改。 | | | |
| 属性 | Modifiability | | | |
| 环境 | 系统正常运行时。 | | | |
| 刺激 | 某个保险管理系统中的用户想购买新的保险。 | | | |
| 响应 | 保险管理系统成功受理该用户的购买请求。 | | | |
| 架构决策 | 敏感点 | 权衡点 | 风险点 | 非风险点 |
| 减小单个模块的大小(Reduce Size of a Module) | S2、3 | T3 | R3、6 |  |
| 提高内聚度 (increase cohesion) | S3、7 | T3、7 | R6、7 |  |
| 降低耦合度(reduce coupling | S3、4、6 | T6 | R6 |  |
| 架构推理 | 减小模块大小，降低系统复杂度  高内聚，减少系统间平凡的交互，将业务尽可能封装在单个模块完成，加快业务处理速度  低耦合延长调用链，降低系统效率 | | | |
| 架构视图 | 响应修改  记录日志  修改系统  登录平台，请求办理业务  刺激 | | | |

1. **Performance**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 场景#: P系列 | 场景: 大量用户同时在线活动(比如购买资金，购买保险，查看信息)。 | | | |
| 属性 | Performance | | | |
| 环境 | 正常操作时 | | | |
| 刺激 | 大量periodic、sporadic、stochastic(周期性 零星的 随机的)事务、事件同时发生、到达，比如某个特殊的节日，平台下各数据库系统的大量用户都登录进行相关的交易，查看信息，更改信息，删除信息等操作。 | | | |
| 响应 | 事务一件件有序地并行化地被处理 | | | |
| 架构决策 | 敏感点 | 权衡点 | 风险点 | 非风险点 |
| 管理采样率(manage sampling rate) | S1、4 | T4 |  | N1、4、5 |
| 提高资源效率(improve resource efficiency) | S4、7 | T6 |  | N1 |
| 减少开销(reduce overhead) | S2、4 |  | R7 |  |
| 限制事件响应(limit event response) | S1、3 | T1、4 |  | N6 |
| 引入并行(introduce concurrency) | S4 |  | R4 |  |
| 增加资源(increase resources) | S2、4、6、7 |  |  | N2 |
| 维护数据的多个副本(Maintain multiple copies of data) | S1 |  | R6 | N5 |
| 架构推理 | 采样率与系统响应速度密切相关  提高资源效率保证可用性  减少开销的同时也会限制系统流量，降低usability  限制响应对可用性影响大  并行计算增加系统复杂度  资源增加能提高系统稳定性，保证可用性  数据的备份使得系统能从攻击中恢复，提高安全性 | | | |
| 架构视图 | 系统行为优化  刺激  执行系统稳定策略 | | | |

1. **Security**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 场景#: S系列 | 场景: 外部人员使用合法用户身份信息利用脚本进行DDOS攻击 | | | |
| 属性 | Security | | | |
| 环境 | 系统联网，防火墙处于工作状态，系统处于完全可操作状态 | | | |
| 刺激 | 频繁访问系统服务，降低系统可用性，试图使系统拒绝向合法用户提供服务 | | | |
| 响应 | 限定用户登录所使用IP地址，拒绝用户同时从多IP地址登录与操作 | | | |
| 架构决策 | 敏感点 | 权衡点 | 风险点 | 非风险点 |
| 检测攻击（Detect Attacks） | S3、5 | T3 |  | N4 |
| 抵制攻击（Resist Attacks） | S1 | T4 |  |  |
| 对攻击做出反应（React to Attacks） | S1 | T7 |  |  |
| 后攻击恢复（Recover from Attacks） | S7 |  | R7 |  |
| 架构推理 | 防御机制在不同情况下会提升或降低可用性  对恶意攻击制定抵御恢复措施 | | | |
| 架构视图 | 允许操作  安全检测  刺激    限制用户操作，并抵制攻击  完成检测  抵御后恢复 | | | |

1. **Testibility**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 场景#: T系列 | 场景: 资金（基金）综合管理系统向社会保险管理系统请求收集数据。 | | | |
| 属性 | Testibility | | | |
| 环境 | 资金（基金）综合管理系统在社会保险管理系统发送请求前就已经有了本市市民的相关信息。社会保险管理系统需要知道某个人是否参加了社会保险或者需要知道本市今年参加社会保险使用的情况等。 | | | |
| 刺激 | 资金（基金）综合管理系统向社会保险管理系统发送一个查询请求，请求查询参加了社会保险的市民使用社会保险资金的情况 | | | |
| 响应 | 社会保险管理系统把资金（基金）综合管理系统需要知道的合法的信息打包后按资金（基金）综合管理系统提出的要求有序发给资金（基金）综合管理系统。 | | | |
| 架构决策 | 敏感点 | 权衡点 | 风险点 | 非风险点 |
| 特化接口 | S2、3 |  |  | N1、6 |
| 记录回放 | S2 |  | R3 | N7 |
| 沙盒 | S2、4 |  | R4、6、7 |  |
| 限制结构复杂性 | S2、4 |  | R4 |  |
| 架构推理 | 特化接口不影响系统正常运行  记录回放增强对测试人员可用性  隔离机制对系统整体测试的可测试性不佳  组件依赖关系减少，延长调用链，系统效率下降 | | | |
| 架构视图 | 执行测试  选择测试策略  系统回归正常运行时  刺激 | | | |

1. **Usability**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 场景#: U系列 | 场景: 用户在多个信息系统登记个人信息，办理不同的业务 | | | |
| 属性 | Usability | | | |
| 环境 | 系统正常运行时候，用户正常使用 | | | |
| 刺激 | 用户希望能够在一个系统处理全部业务，能够在移动端使用 | | | |
| 响应 | 对信息系统进行信息更新，并且能够即使删除冗余数据 | | | |
| 架构决策 | 敏感点 | 权衡点 | 风险点 | 非风险点 |
| 聚合（Aggregate） | S3、4 | T4 | R4 | N7 |
| 架构推理 | 信息聚合降低搜索效率 | | | |
| 架构视图 | 响应  代理组件分配业务  请求业务  刺激 | | | |

**注：场景编号-定义**

场景**1** ：平台可以检测、记录故障，以及系统可以正常运行的时候.。

场景**2** ：A市退休人员管理系统向A市人力资源管理系统请求数据。

场景**3** ：保险管理系统中用户对自己的账户进行修改。

场景**4** ：大量用户同时在线活动(比如购买资金，购买保险，查看信息)。

场景**5** ：外部人员使用合法用户身份信息利用脚本进行DDOS攻击。

场景**6**：资金（基金）综合管理系统向社会保险管理系统请求收集数据。

场景**7**：用户在多个信息系统登记个人信息，办理不同的业务。

# 学生工作日志

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **时间** | **地点** | **主要工作内容** |
| 1 | 2019.11.04 | 教室 | 小组全体成员对系统进行分析，对任务进行划分和讨论。 |
| 2 | 2019.11.05-11.06 | 宿舍 | 各组员完成各自任务：  邹鹏宇——负责Interoperability和Availability的质量属性的场景化描述、战术、效用树以及架构模式。  廖惠康——负责Modifiability和Performance的质量属性的场景化描述、战术、效用树以及架构模式。  郭柏良——负责Security的质量属性的场景化描述、战术、效用树以及架构模式，考虑hadoop框架在本系统的应用。  林墨馨——负责Testability的质量属性的场景化描述、战术、效用树以及架构模式，考虑HDFS在本系统的应用。  胡映鑫——负责Usability的质量属性的场景化描述、战术、效用树以及架构模式，考虑MapReduce Model的应用。 |
| 3 | 2019.11.07-2019.11 | 宿舍 | 各组员完成各自任务以及个人心得：  廖惠康、胡映鑫——架构三视图的绘制和架构设计规约编写。  郭柏良——根据架构设计进行架构评估。  林墨馨——与命题相关场景的质量属性相关的实验性研究，以展示大数据分析的价值。  邹鹏宇——管理文档，编写工作日志，整合和排版。 |