**SOA架构设计和评估ADD过程**

## 第一次迭代

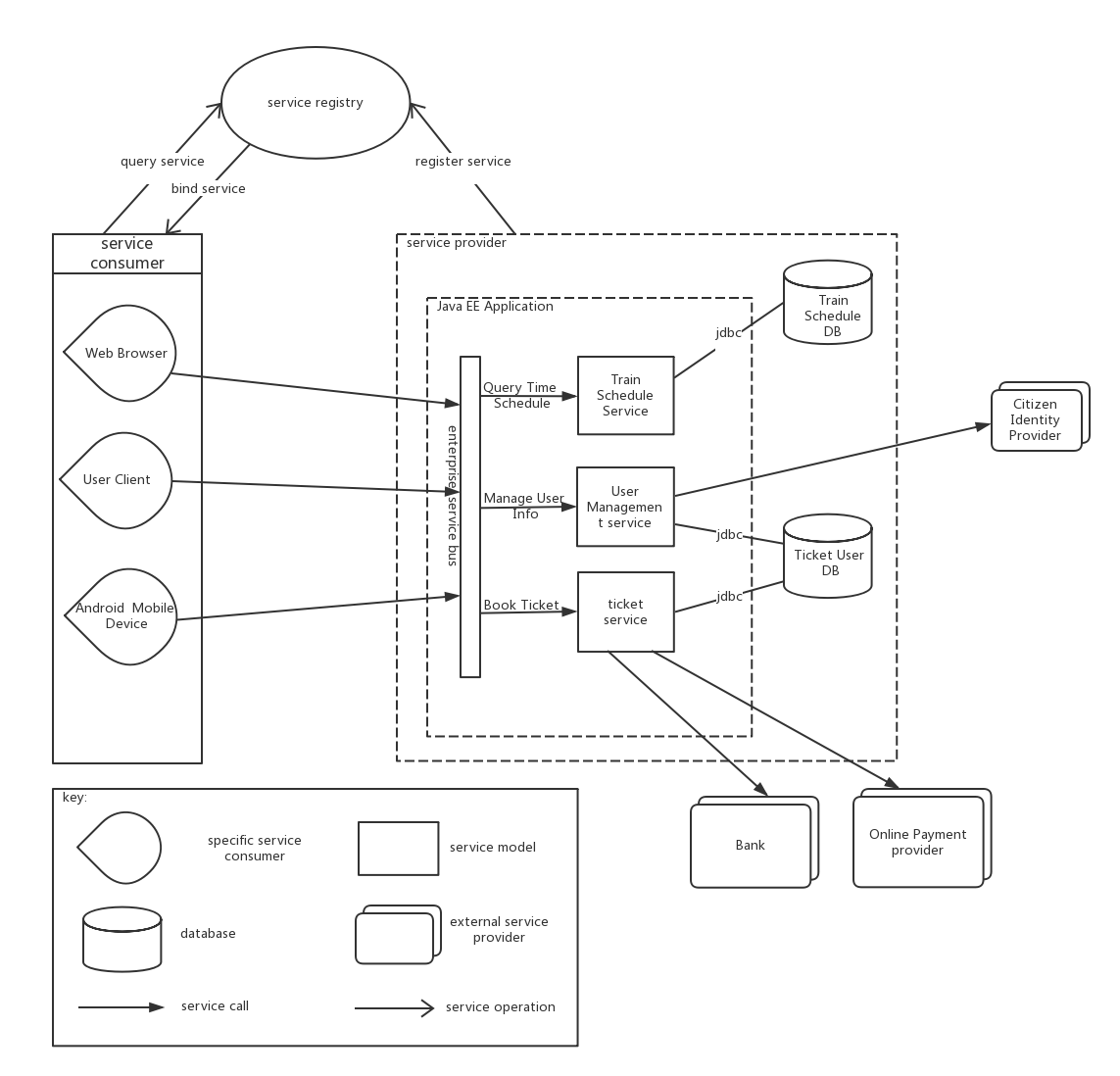
分解系统如图所示。服务端分为ESB模块、列车时刻模块、用户管理模块、购票模块以及相关的数据库，分别为列车时刻数据库和用户购票记录数据库。同时，注册中心用于服务提供者向其注册服务，以及服务消费者查找需要的服务。

ESB模块：在服务的消费者和提供者中间起到路由选择的作用，可以实现防火墙、参数检测与统一、 消费者请求分发、管理理事务等功能。

列车时刻模块：提供查询列车运行时刻以及余票的功能。

用户管理模块：提供用户管理账户信息的功能。

购票模块：提供购票、改签、退票的功能。



## 第二次迭代

#### 选择元素

第二次迭代选择的元素是ESB模块。

#### 选择ASR

第二次迭代选择的ASR是性能、安全性。

对于该系统架构，ESB模块是前后端连接的最重要渠道，前端来的所有流量都要经过ESB才能调用到后端的其他服务，所以性能对于ESB模块极为重要，因为低性能的ESB可能成为整个系统的瓶颈部分。此外，由于此处是前端和后端交互的地方，所以还需要防护从外部来的攻击，保证信息安全和保证系统被攻击后能正常运行。综上，我们选择性能和安全性作为ASR。

#### 候选策略表和决策

#### 性能

|  |  |  |  |
| --- | --- | --- | --- |
| 策略 | 优点 | 缺点 | 决策 |
| 增加资源（硬件能力） | 对其他的质量属性没有太多的影响 | 增加硬件成本（租用购买成本及维护成本） | 采用。比起其他质量属性的降低，经济成本的投入更值得。首先应该保证ESB不成为整个系统的瓶颈部分。 |
| 引入并行 | 并行处理，提高吞吐量，减少平均阻塞等待时间 | 需要对架构作较大的改动，增加实现难度 | 采用。部署多个ESB并注册好，在客户端绑定服务时进行负载均衡 |
| 限制请求执行时间 | 控制一项请求的操作时间，保证处理资源的合理分配 | 丢失请求，降低用户的体验。另一方面计算请求执行时间也会带来额外的负担。 | 不采用。缺点比较明显，可以通过其他的策略提高ESB模块的性能。 |
| 提高处理效率（更高效的库，优化算法等） | 能提高单个请求的处理速度；对其他的外部质量属性没有太多的影响。 | 实现难度大，投入不一定有产出 | 采用。缺点基本可以忽略，但是成功后带来的优点很大，能够整体上根本上提高ESB的性能。 |
| 缓存请求结果 | 减少了对其他服务的访问次数，服务的访问往往是性能的瓶颈。而且还能减少服务提供者的负担。 | 信息的实时性有一定程度的损失。 | 采用。对服务进行访问本身就有一定程度的延时，且系统的实时性需求不是很高。 |

#### 安全性

|  |  |  |  |
| --- | --- | --- | --- |
| 策略 | 优点 | 缺点 | 决策 |
| 主动攻击检测 | 及时发现攻击方，避免更大的损失 | 带来一定的性能负担 | 采用。对攻击的主动识别是有必要的，否则ESB模块作为前后端的通道，一旦被攻陷会导致整个系统无法正常运行。 |
| 数据验证 | 防止数据被恶意修改 | 带来一定的性能负担 | 采用。系统牵涉个人身份信息，银行信息等敏感信息，需要保证信息的integrity |
| 身份验证 | 可以防止资源被恶意方访问 | 带来一定的性能负担 | 采用。系统需要保证个人身份信息，银行信息等敏感信息不被其他人访问 |
| 数据加密 | 可以防止信息传输过程中被窃听，从而泄露信息 | 带来一定的性能负担，且需要其他的组件的配合，减弱了互操作性 | 采用。系统需要保证个人身份信息，银行信息等敏感信息不被其他人访问 |
| 备份恢复 | 可以从被攻击的状态较快恢复 | 提高了硬件成本。 | 采用。可以作为应急措施，使得ESB模块的沦陷不会带来全局的瘫痪。 |

#### 第二次迭代结果

新增元素如下：

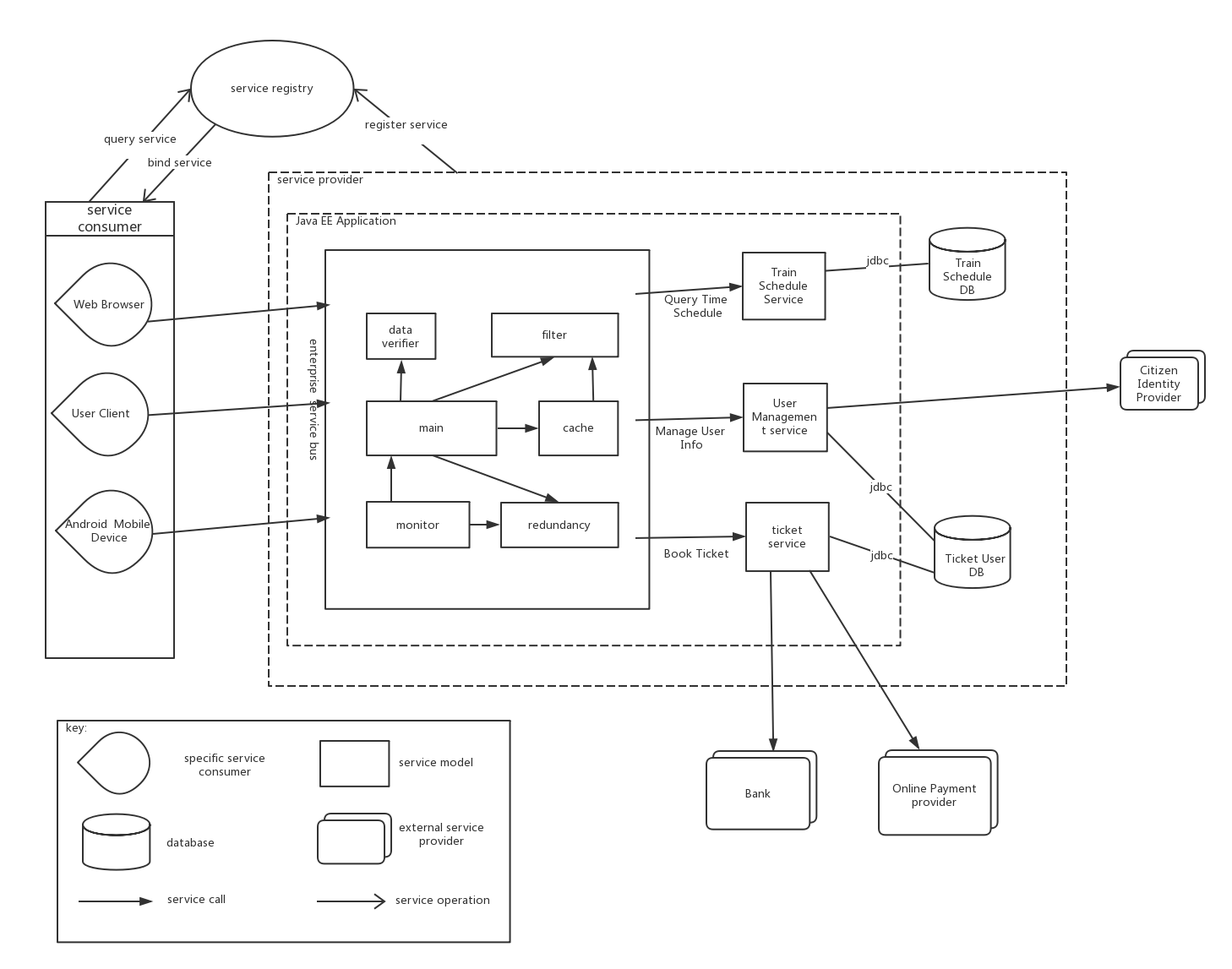
缓存模块 (Cache): 缓存请求调用后端服务后得到的结果，在碰到相同请求的时候将缓存的结果返回，在一定时间后将过时的数据删除。

冗余备份模块 (Redundancy): 冗余存储请求队列中的信息，在主服务失去响应后进行恢复。

请求过滤模块 (Filter): 对请求进行身份验证，攻击检测，丢弃不合法的请求。同时在收到的时候解密，在发出信息时加密。

响应数据验证模块 (Data Verifier): 验证数据是否正常，是否被修改过。

监测重启模块 (Monitor): 监测主服务的状态，若无响应，则从冗余备份模块中进行恢复。



## 第三次迭代

#### 选择元素

第三次迭代选择的元素是购票模块。

#### 选择ASR

第三次迭代选择的ASR是可靠性、互操作性。

选择可靠性：购买车票、改签以及退票是该系统最核心的功能，也是用户最需要的功能。系统应该保证其可靠性需求，在运行时期不能崩溃，不响应。

选择互操作性：购票模块的实现需要频繁访问外部服务，比如外部支付平台等，所以对外需要提高系统的访问效率。同时，作为SOA架构的一部分，购票模块也是服务的提供者，为第三方平台提供服务接口。所以互操作性是系统的ASR。

#### 候选策略表和决策

#### 可靠性

|  |  |  |  |
| --- | --- | --- | --- |
| 策略 | 优点 | 缺点 | 是否采用 |
| 心跳模式 | 实现简单，用于监测组件是否正常工作时每次传输的数据量少。可以实现周期性的检测。 | 需要持续占用部分计算资源用于监测组件，即存在资源的等待。 | 采用。可以通过周期性的监测有效保证被监测组件正常工作。 |
| Ping/Echo模式 | 可以通过中断的方式减少计算资源的等待，监测方式更具有可控性。 | 监测组件每次传输的数据量较多。实现较为复杂，且该方式不适合周期性的组件监测。 | 不采用。Ping/echo不适用于当前情况下的周期性检查。 |
| 同步更新备份数据 | 同步更新备份数据能保证备份数据的实时性。当需要使用备份数据时，同步更新的模式能提供最新的数据。 | 数据吞吐量大时可能增加传输负担。 | 采用。为了防止数据丢失，数据的实时性更为重要。 |
| 定时更新备份数据 | 可以在高吞吐量时降低传输负担。 | 定时更新备份数据可能会因为积累了大量未更新数据而造成数据需要传输的量过大的情况，增加传输负担，影响正常数据的传输。 | 不采用。同步更新更适合保证系统的可靠性， |
| 使用缓冲 | 将调用频率高的服务比如查询某两地之间的车票数据缓存，以提高高并发的处理能力。 | 需要额外的存储空间。 | 采用。可以提高SOA中组件的高并发处理能力 |
| 增加物理资源 | 更多的处理资源、内存、更快的网络带宽会提高系统的高并发处理能力。 | 增加物理资源需要资金开支 | 采用。增加资源可以带来性能上显著的提示，由此带来的成本可以接受 |
| 消极冗余 | 在存在多个冗余的计算资源的条件下，仅有一个组件响应请求，其他冗余组件与之保持状态匹配，随时可以进行替换 | 当前组件和冗余组件之间存在频繁的信息交流 | 采用。在不占用太多计算资源的情况下，能够有效处理错误出现的情况。 |
| 积极冗余 | 在存在多个冗余的计算资源的条件下，所有组件均响应请求，最终决定一个组件的响应被采用。 | 占用的计算资源大 | 不采用。本策略增大了请求处理的任务量。 |
| 冷冗余 | 对发生错误组件进行替换，保证错误的出现不影响系统 | 系统宕机时间略长，多用于可以人工操作的组件 | 不采用。系统需要不间断的持续提供服务。 |

#### 互操作性

|  |  |  |  |
| --- | --- | --- | --- |
| 策略 | 优点 | 缺点 | 是否采用 |
| 注册中心服务查询 | 通过向服务注册中心查询相关服务，可以灵活地调用需要的服务，在SOA架构中也符合服务调用思想。 | 每次查询服务需要消耗一定时间。 | 采用。在SOA架构中适用。 |
| 服务绑定 | 可以高效调用服务，不需要查询。 | 服务的接口固定，不利于服务扩展。 | 不采用。不利于SOA架构的实现。 |
| 定制接口 | 实现较为简单。 | 定制接口针对一个接口进行功能的增加或删除。所以定制接口会增加代码的复杂度，导致系统不利于维护。 | 不采用。不利于系统的维护。 |
| 组织协调接口调用 | 组织协调接口调用通过对不同接口进行协调排序，使接口之间分工合作完成功能。 | 需要考虑接口之间的协调，增加了设计的复杂度。 | 采用。有利于系统的扩展性。 |

#### 第三次迭代结果

新增元素如下：

心跳监测模块 (Health Monitor): 监测各个处理器是否正常工作，并向接口代理提供处理器的状态消息。

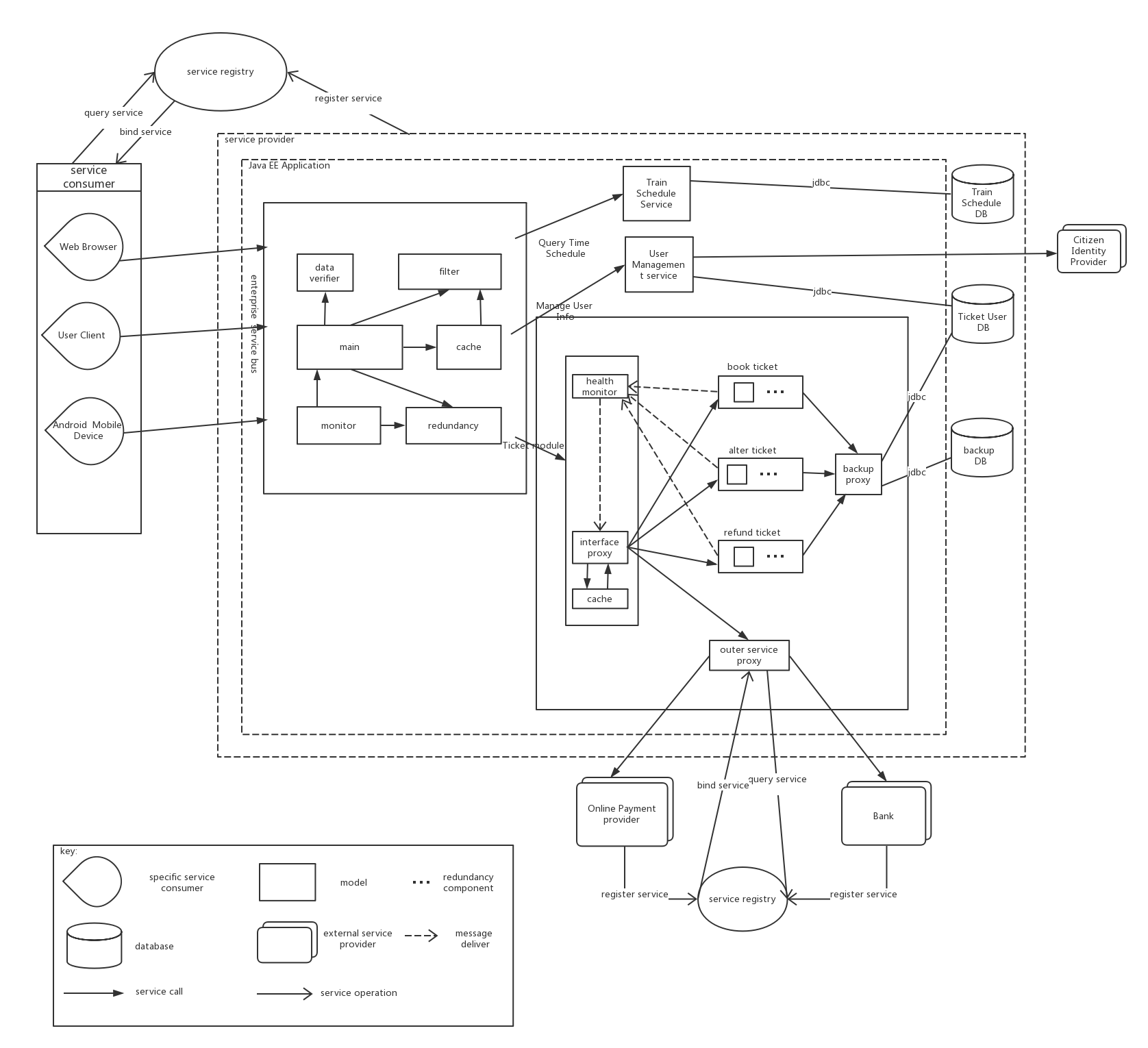
接口代理模块 (Interface Proxy): 接受监测模块的处理器状态消息，决定响应请求调用的处理器。高频的请求和响应存放在Cache中提高性能。同时决定各个服务接口的协调顺序，包括系统外部的服务。

Cache模块 (Cache): 缓存高频的请求和响应，提高系统的响应效率，减少系统负担。

冗余模块 (Redundancy): 对每一个业务处理模块都进行计算资源冗余，提高系统的高并发处理能力，提高可靠性。

外部服务代理 (Outer Service Proxy): 用于向注册中心查询系统需要外部的服务接口，提高互操作性。

数据备份代理 (Backup Proxy): 用于统一进行数据库的操作以及数据备份。



## 第四次迭代

#### 选择元素

第四次迭代选择的元素是列车时刻模块。

#### 选择ASR

第四次迭代选择的ASR是可用性和与性能。

选择可用性：查询列车时刻表与余票信息是用户进行购票操作之前的必要步骤，必须保证列车时刻表是24\*7\*365可用的。一旦列车发生晚点情况，晚点信息对于用户出行比较重要，系统不能崩溃导致用户无法得知列车最新的运行情况。

选择性能：在节假日等用户量较大的情况下，系统应保持高效的查询速度。当列车发生晚点时，系统应能在第一时间预测并推送晚点信息。

#### 候选策略表和决策

#### 性能

|  |  |  |  |
| --- | --- | --- | --- |
| 策略 | 优点 | 缺点 | 是否采用 |
| 限制事件响应 | 当超过处理能力的多个请求同时到达系统的时候，请求会被先缓存再统一处理，可以保证对多个请求更稳定的响应速度 | 当请求缓存到达上限的时候可能会丢失用户请求 | 采用，用户高并发查询时保证系统处理请求的稳定性更加重要，偶尔的查询请求丢失后果不严重 |
| 事件优先级处理 | 当系统资源不足的时候，可以忽略低优先级的请求，保障高优先级请求的处理 | 造成低优先级请求的丢失 | 采用 |
| 减少中间件开支 | 减少中间件会减少用户请求被传递、处理的次数，减少不同中间件之间信息的传递造成的资源耗费 | 如果将多个处理的过程集中在一个组件中，它的可修改性将减低 | 不采用，中间件的减少不利于系统的维护与修改 |
| 限制一个请求的执行时间 | 可以防止一次请求占用大量处理时间 | 造成计算结果的不精确 | 采用，为了防止一个请求阻塞整个系统的极端情况发生 |
| 优化核心算法 | 对核心算法（查询余座、时刻表）的优化可以显著减少延迟 | 无 | 采用 |
| 增加物理资源 | 更多的处理资源、内存、更快的网络带宽一定会显著减少延迟 | 增加物理资源需要资金开支 | 采用，增加资源可以带来性能上显著的提示，由此带来的成本提升可以接受 |
| 并行处理请求 | 请求被并发地处理，可以减少请求的阻塞时间。可以采用不同的并行的调度算法达到期望的公平性、吞吐量 | 并发处理调度的计算与资源成本 | 采用，并发处理是提升计算系统性能的常用方法 |
| 计算资源冗余 | 提供多个计算模块可以减少单个模块上资源的竞争情况。负载均衡器根据不同冗余的运行情况分派请求 | 增加处理资源占用的空间 | 采用，冗余带来的计算性能的提升甚至是翻倍的利益大于资源空间的成本 |
| 数据冗余，使用缓存 | 数据资源的冗余可以缓解多个同时到达的请求之间的竞争情况，将数据缓存到内存中可以提高响应的速度 | 缓存的数据通常是已有的数据，服务器需要保持不同数据之间的一致性和同步，增加了计算负载。 | 采用，同上 |
| 限制处理资源上限 | 限制对一个请求的处理资源上限，防止部分请求占用过多的处理资源，降低系统整体性能 | 当请求对应的处理资源达到上限之后，有可能会造成用户请求的丢失 | 采用，防止一次请求占用过多资源足阻塞其他请求，查询信息请求偶尔丢失的结果并不严重 |

**针对可用性：参见购票模块的处理**

#### 第四次迭代结果

新增元素如下：

请求缓存模块 (RequestBufferQueue): 限制请求的处理，缓存到达的请求，服务器可以异步、按照优先级的顺序处理这些请求。

超载检测模块 (Overload Monitor): 接受监测模块的处理器是否超载的状态消息，决定响应请求调用的处理器。高频的请求和响应存放在Cache中提高性能。同时决定各个服务接口的协调顺序，包括系统外部的服务。

