体系结构设计

目录

系统主要功能

- 余票查询:根据时间、地点等输入信息检索车次和剩余座位等信息
- 车票购买:从检索到的车辆中选定指定车次，购买车票、选择座位
- 车票改签、退票:对未出行的车票进行改签、退票等操作
- 查询订单:查询用户历史已出行订单和未出行订单
- 账户管理:对个人账户信息进行管理，增删常用乘车人
- 列车时刻表查询:提供列车运行时刻表
- 正晚点信息查询与推送：提供实时列车运行正晚点的信息，在列车晚点时向车站和乘客推送晚点信息

ASRs scenarios

A1　　安全性：场景 1：阻止恶意抢火车票行为

场景组成部分	可能的值
源	外部恶意软件
刺激	以用户身份持续购票或者快速购票
制品	购票模块
环境	系统运行时
响应	系统冻结此用户的账号，阻止其购票行为
响应度量	当恶意软件发出恶意请求时，要在 1s 内检测。 在 1s 内冻结其账户操作。

A2　　安全性：场景 2：阻止黑客攻击

场景组成部分	可能的值
源	对系统的攻击行为
刺激	黑客对系统的数据攻击、DOS 攻击等
制品	系统
环境	系统运行时
响应	系统检测到攻击，同时采取相应保护措施，避免攻击范围扩大，消除攻击带来的影响
响应度量	在 1s 之内检测到正在进行的攻击。 在 10s 内采取安全措施。 在 60s 内消除攻击带来的影响。

A3　　可用性：场景 1：应对系统错误

场景组成部分	可能的值
源	系统内部的运行或者是用户的操作
刺激	大量用户的操作导致系统压力增大或者是系统自身运行时出现错误
制品	系统
环境	系统运行时
响应	如果压力过大则将压力转移到一些设备上。如果是出错，那么快速修复错误。
响应度量	诊断异常状况的时间不超过 2s 将请求压力转移到其他设备上的时间不超过 3s 快速修复错误的时间不超过 2s 错误修复成功率不低于 98%

A4　　可延展性：场景 1：硬件升级

场景组成部分	可能的值
源	开发人员
刺激	系统硬件需要升级，如可能增加服务器数量，或者对单个服务器进行硬件的修改
制品	系统硬件
环境	系统在构建中或者已经上线
响应	对系统硬件进行淘汰或更新

	修改项目使之适应新的硬件环境 测试更改对于系统造成的影响 发布维护变更
响应度量	升级时间不应超过 8 小时 升级所影响的代码量不应超过 2%

A5 可延展性：场景 2：服务器数据库升级

场景组成部分	可能的值
源	开发人员
刺激	服务器数据库需要进行扩容或者分布式处理
制品	服务器，数据库
环境	系统在构建中或者已经上线
响应	对数据库进行升级处理 修改服务端响应代码 测试修改后性能
响应度量	升级时间不应超过 8 小时 升级所影响的代码量不应超过 3%

A6 互操作性：场景 1：和第三方系统进行交互

场景组成部分	可能的值
源	系统
刺激	系统请求和第三方系统交互
制品	第三方系统
环境	系统运行时或构建时
响应	系统成功拿到第三方系统的服务
响应度量	交互请求成功率大于 99.999%

A7 易用性：场景 1：正常操作

场景组成部分	可能的值
源	终端用户
刺激	想要使用系统完成操作
制品	系统

环境	系统运行时
响应	能够正确引导用户完成操作 提供撤销或取消功能，在用户操作过程中识别并纠正用户错误 错误发生时的恢复
响应度量	用户完成单项操作的时间小于等于 2min 用户操作成功率应该大于等于 99% 错误恢复时间小于 5min

A8　　性能：场景 1：系统运行

场景组成部分	可能的值
源	用户，第三方系统
刺激	非周期性的数据请求，操作请求
制品	Broker 模块，服务器模块，request bus
环境	系统正常运行时，系统高负载时
响应	系统执行用户请求的操作；处理用户数据；返回数据
响应度量	系统在满负载运行时，请求响应延迟时间不超过 5s 任何情况下，请求丢失的比例不超过 0.001%

Broker 架构

ADD 过程

第一次迭代

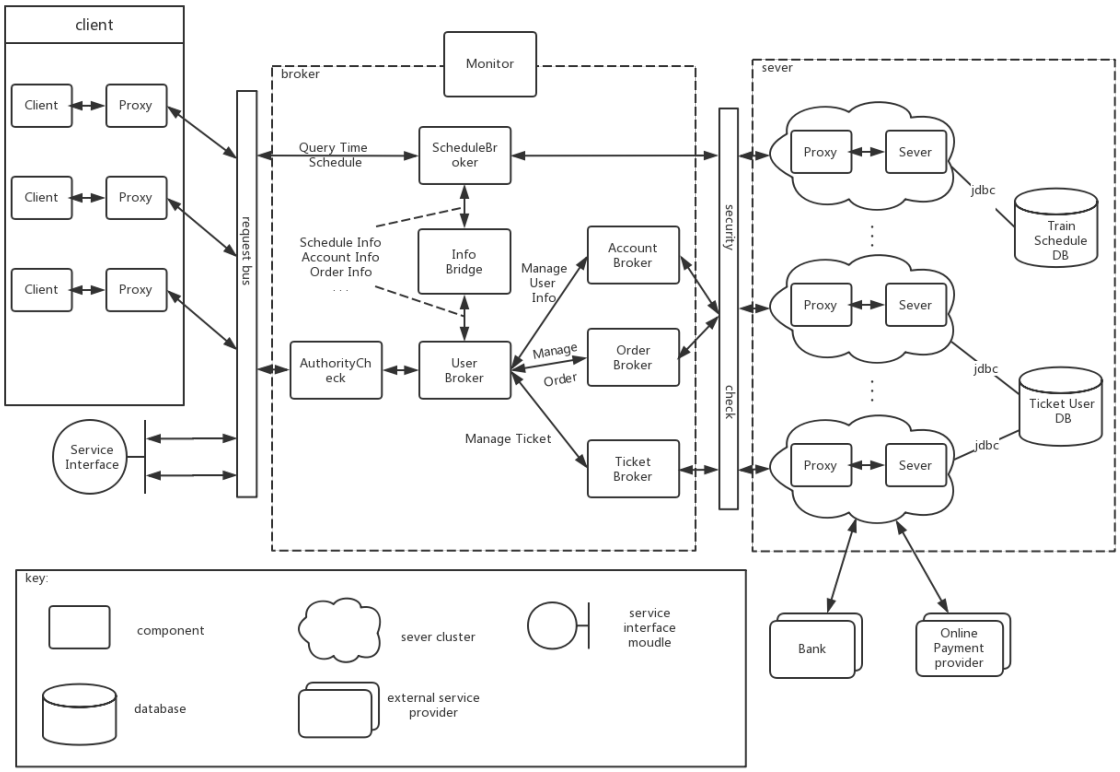
分解系统如图所示。

UserBroker 组件负责有关用户的请求转发处理 ， 该 broker 向 AccountBroker(处理账户信息)、OrderBroker(处理订单信息)、TicketBroker (处理票务请求)三个二级 broker

分送请求。

Monitor 负责监控 Broker 模块，保证可靠性。

AuthorityCheck 和 Security 用于过滤请求保证安全性。



第二次迭代

选择元素

选择的元素是 user-broker 模块，负责用户账户、订单和购票请求的转发分派。

选择 ASR

第二次迭代选择的 ASR 是性能、可用性。

对于该系统，由于该系统的用户基数较大且流量高峰时间比较集中，就要求系统在正常情况下和高并发模式下能做出正确而快速的响应，另外由于 broker 在系统中的枢

纽地位，一旦出错当机，对系统的影响是难以估量的，因此系统的可用性和性能显得尤为重要。

候选策略表和决策

	优点	缺点	是否采用
主动冗余	通过对重要broker的备份，可以在主 broker 出现问题时切换到备份的broker。增强了系统的可用性和安全性；另外由于 broker 架构的灵活性，可以比较方便地切换主从broker，回复速度较快	冗余的 broker 增加了系统的负担，当系统切换 broker 时不可避免地会增加系统的负载压力和复杂度，耗费资源	不采用，耗费资源太多
被动冗余	通过从 broker 中的某些信息来恢复主broker。增强了系统的可用性，两个broker各自有不同的功能，功能不完全冗余，系统的负担较小	主broker和从broker之间的信息同步可能对系统造成一定的压力，而且错误恢复的速度比较慢	采用，可以对于较重要的或者有前后顺序逻辑的 broker 使用被动冗余来加强可用性
移除可疑broker	错误恢复后将发生错误的 broker 移除，可以防范 broker 再次出错	某些 broker 的功能是无法替代的，移除后可能造成系统瘫痪	不采用
模块细化	User-broker 可以将自己的功能分解成几个较独立的功能模块，分派给二级broker。User-broker 负载压力减小，能够更好更快地处理高并发问题	Broker 之间的通信更加复杂，增加了系统的复杂度	采用
优先级调度	User-broker 通过识别优先级对优先级更高的请求进行优先转发，可以保证系统	请求优先级设置问题，难以决定哪些请求更重要。另外优先级调度也	不采用，对于本系统而言用户的请求并没有很明显的优先级

	对于重要功能的快速响应	会增加系统的复杂度	
建立缓存模块	通过把请求结果缓存到缓存模块中，对于重复请求可以直接响应而不用向 server 进行请求，解决一部分并发问题，可以减少等待时间，提高系统性能	需要对用户请求结果进行缓存，可能造成一定的负载压力，另外某些实时变化的信息并不适合缓存	不采用，因为不会有多个用户对 user-broker 中的信息同时进行访问（账户信息、订单信息、票务信息），没有缓存的必要

第二次迭代结果

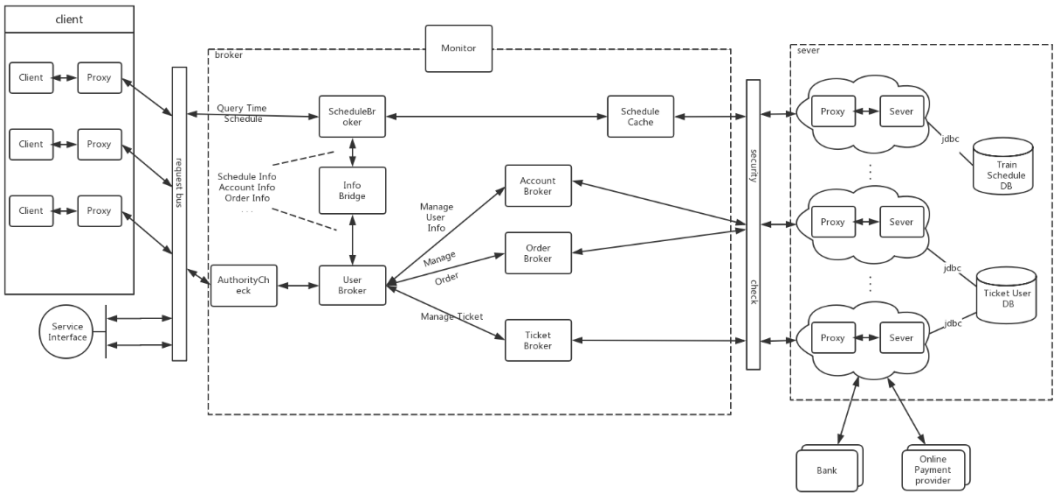
如图所示，新增元素说明如下

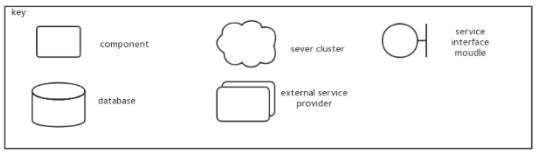
account-broker 模块：负责 user-broker 中的账户请求服务的二级转发

order-broker 模块：负责 user-broker 中的订单请求服务的二级转发

ticket-broker 模块：负责 user-broker 中的票务操作请求服务的二级转发

schedule-cache 模块：负责对列车调度表进行缓存





第三次迭代

选择元素

选择的元素是安全模块，包括 broker 部分的 authorityCheck，位于 broker 和 server 之间的 security Check。

选择 ASR

第三次迭代选择的 ASR 是安全性。这个模块主要负责系统的安全控制。

策略表和决策

	优点	缺点	是否采用
检测请求模式	通过检查正常服务请求的模式和机器指令，来控制访问权限。避免抢票插件不正常的操作数据。避免刷票程序占据带宽，增强性能。	增加系统复杂度，需要专门的模块来记录请求模式，比较请求模式。可能增加每一条请求的时间，降低性能。	采用
检测服务拒绝情况	通过检查同一身份认证多次失败的情况，能够防止恶意的测试用户密码。	同上	采用
识别信息一致性	通过检验 hash 值或者 checksum 来确定数据一致性。避免数据不一致。	增加复杂度，降低性能	不采用，主要检测持久存储的一致性，当前模块无法采用
识别角色	对外部输入系统的资源加以识别，防止脚本注入攻击等。	增加系统复杂度。	采用
用户认证	通过识别用户身份，避免不当的数据操作，其他用户修改了某个用户的个人信息。	需要与其他组件交互，存储用户数据，增加系统复杂度。	采用

用户授权	通过给用户授予合适的权限，限制用户的能力，避免用户对重要数据的不当操作。	增加系统复杂度。	采用
信息加密	增强信息的安全性，即使数据暴露，也不会立即造成极大的影响	需要额外的处理增加复杂度，加密操作可能影响性能，数据加密后仍然可能被破解。	采用。
限制对计算机资源的访问	通过限制直接访问计算机资源，避免暴露系统内部细节。减少了黑客攻击的可能。	增加系统复杂度。	采用
最小化系统的攻击面	通过减少系统访问面，要求所有的请求必须经过安全处理模块，能够集中处理所有的攻击。	因为所有的请求都必须经过安全检查部分，所以该组件一旦遭到攻击失效，会影响系统的可用性安全性。	采用
当攻击发生时删除敏感的资源	系统已经遭到潜在的攻击，与其使重要数据被窃取，不如直接销毁，减小损失的影响。	对于不是特别敏感的资源，删除导致的损失过大	不采用，该系统没有危险到直接删除的敏感资源

迭代结果

如图所示，新增元素说明如下。

请求拒绝监测模块：在请求验证失败后，记录失败信息。再次请求时检查验证失败记录，做出判断并且拒绝请求。避免恶意尝试破解用户密码

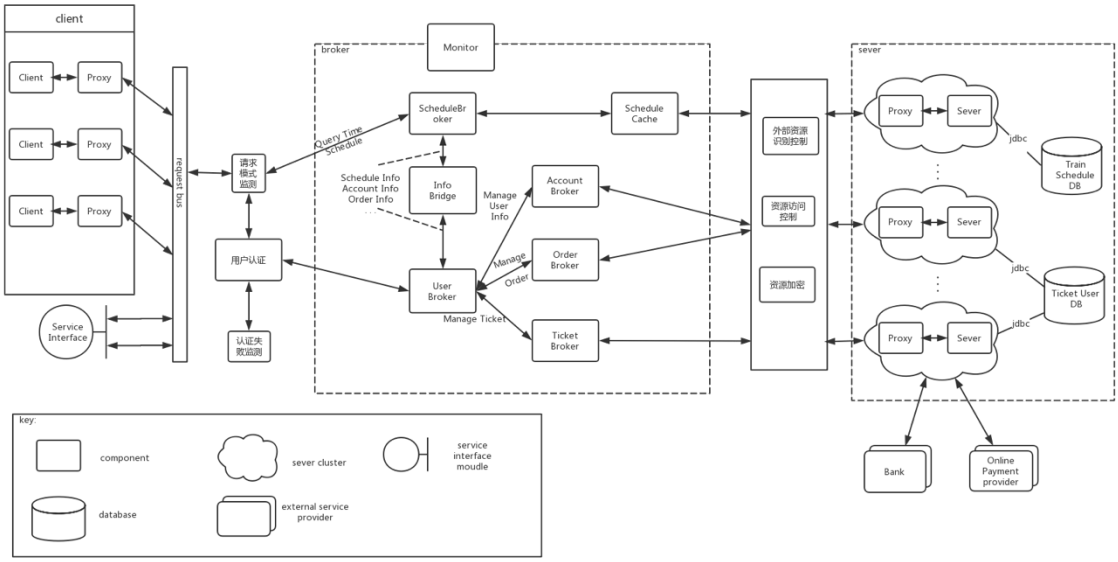
请求模式监测模块：记录请求情况，对比请求模式。在

用户认证模块：确认用户身份，认证权限

外部资源识别控制模块：对输入系统内资源进行识别和转化，包括文件提交，输入数据等，避免注入攻击。

资源访问控制模块：控制对系统资源的访问请求

资源加密模块：对需要加密的输入资源进行加密



第四次迭代

选择元素

第四次迭代选择的元素是 monitor 监测模块。

选择 ASR

第四次迭代选择的 ASR 是可用性、性能。

对于该系统架构， broker 的引入带来了系统错误的集中产生点，此处产生的错误必然给项目整体带来较大的影响。Monitor 作为监控组件，需要保证核心 broker 模块正常运行，固然可用性必不可缺。当某错误产生时，该模块同样需要具有应对错误的能力，在最短时限内完成错误处理任务，所以说性能同样重要。

候选策略表和决策

可用性

Heartbeat	持续周期性监测，并且可以进行少量数	对于监测组件的性能要求高，实时监听	采用。周期性的通信检查，可以有效保证被监测组件

	据传递交换组件状态信息	的情况下还要有能力应对可能出现错误	可能出现的错误。
消极冗余	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	占用一定资源，当前组件和冗余组件之间存在频繁的信息交流	采用。在不占用太多资源的情况下，能够有效处理错误出现的情况。
Ping/echo	通过时长和通信检查模块之间的连通性，常用于远程模块之间	模块之间的主动连通性检查，不定期，模块间联系复杂，压力增大	不采用。Ping/echo 主要适用于远程模块之间的通信检查，不定期检查。
投票	对于逻辑控制模块有有效的监测作用，通过对比不同投票者的输出侦测错误	适用场景有限，投票逻辑定义困难，消耗资源较多	不采用。投票逻辑难以定义，因为 broker 模块内组件职责差异较大，且消耗资源过多。
积极冗余	所有组件均响应请求，只有一个组件的响应被采用，动态调节哪个响应被采用	占用资源大	不采用。本系统对于性能要求高，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。操作复杂，本系统对于宕机时长要求高。
事务	对于错误的恢复身份有效，处理操作能被回滚	事务的记录占用资源较大	不采用。消耗资源过大，并且监控模块的任务不适合事务化。

性能

限制操作时间	控制一项请求的操作时间，保证处理资源的合理分配	对于实时记时组件的性能要求较大	采用。对 broker 处理请求的时长进行上限监控，保证 broker 资源被合理分配，不会因为处理时间过

			长而导致请求阻塞。
增加资源	从基本层面增强模块的整体处理能力	产生预算	采用。增强 monitor 模块的处理能力、资源空间，保证其能高效完成监测任务。
引入并行	并行处理，减少阻塞等待时间	对于并行处理的组件性能要求高，可能引入并发问题	采用。引入不同的线程并行处理不同的监控任务。

第四次迭代结果

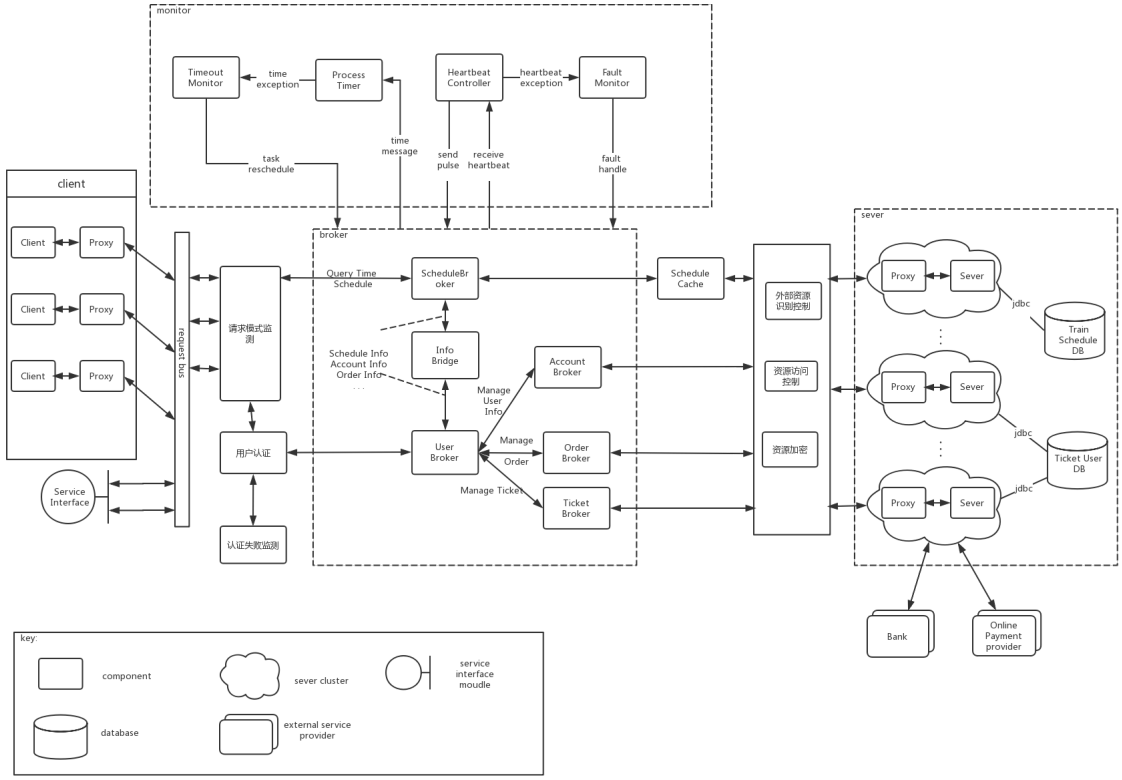
新增元素如下：

心跳控制模块（HeartbeatController）：周期性负责发送脉冲信号并且对 broker 的响应做出解析，当解析异常时发送异常报告给错误监控模块

错误监控模块（FaultMonitor）：接受来自心跳控制模块的异常报告，并且处理异常情况

进程记时模块（ProcessTimer）：记录 broker 传来的时间信息，控制每次请求的响应时间，如果判断超时，将发送超时异常给超时监控模块

超时监控模块（TimeoutMonitor）：接受来自进程记时模块的超时异常，对 broker 等待任务集进行动态调度



第五次迭代

选择元素

第五次迭代选择的元素是服务器模块。

选择 ASR

第五次迭代选择的 ASR 是安全性、互操作性、可用性、易修改性。

1. 因为 12306 的用户群体非常广大，所以安全性尤为重要，如果 12306 的服务器模块被黑客成功攻击，那么会对整个系统都会是非常大的打击，而且可能引发更大的隐患。
2. 因为 12306 系统设计财务流通、身份识别，所以需要与银行系统与公安系统进行交互，用于财务转账、身份识别，所以互操作性对于 12306 系统来说很重要。
3. 因为 12306 的用户群体非常广大，所以系统要保持高可用性，这就需要服务器

端一直要保持高可用的状态。

4. 因为 12306 业务逻辑可能会发生变化，，所以这使得在服务器端进行修改造成的对其他的模块的影响要尽可能小，所以 12306 的易修改性要尽可能好。

候选策略表和决策

安全性

策略	Pros	Cons	是否采用
攻击发生时收回数据访问权限	在遭受攻击时，启动自我防御机制。极大增加了系统的安全性。	增加了失效时间，降低了系统的可用性。	采用。增加攻击侦测模块，当侦测到攻击发生时，服务器模块将会拒绝一切外界访问，直到确认安全。
数据加密	对数据加密，保证了数据的安全性。	对数据的加密需要消耗额外的资源。	采用。因为与银行系统的交互的信息可能被拦截和监听，所以加密是必须的。
用户认证	能够有效阻止一些外部的攻击，增加了安全性。	验证工作消耗额外资源。同时可能会有不短猜测以获取验证身份的行为。	不采用，因为此时处理的是服务器模块的安全性事务，用户认证会由客户端进行工作，这边再做就是重复工作。
限制访问	限制访问范围，增加了对敏感数据的保护，提高了安全性。	消耗大量额外的资源，同时也需要系统构建者额外的工作量。	采用。对于查询的用户，将其权限在只能查看自己想找的信息之内。对于其他操作的用户，同样如此。不透露多余的任何信息。

可用性

保证冗余的服务器和网络连接	保证了系统失效的时候能够立即采取措施，	需要大量的计算资源作为候补。	采用。如果当前正在运行的服务器突然失效，则立即使用冗余的服务器和网络连接。
ping/echo	能随时对系统发起询	通讯量是 heartbeat 方	不采用。因为采用

	问请求。	案的双倍。	ping/echo 的机制的检查消耗通讯量大。
heartbeat	定期检测系统是否已经失效。消耗资源较少。	在系统内部要实现额外的 heartbeat 相关逻辑。	采用。因为采用 heartbeat 的机制所进行的检查是定期的，可靠，而且消耗通讯量较少。
monitor	能即时监控整个系统，有利于立即应用系统失效的情况。	monitor 一直监测整个系统，消耗了大量的系统资源。	不采用。因为 monitor 机制占用了太多的系统资源。

互操作性

服务定位（locate）	采用目录命名机制，方便了与外界系统的交互行为。	维护服务目录需要大量的资源。	采用。在一个已知的目录服务里面，去定位自己想找到的服务，比如银行消费服务、公安人员监测服务。
裁剪接口	通过接口很好地限制地系统与外界的交互范围。	对接口的修改影响整个全局的功能。	不采用。12306 系统的服务器端主要是作为服务的使用者而不是服务的提供者。

易修改性

模块分解	将职责不同的部分分解出来，使得他们相对于其他部分独立，更加容易独立于其他的部分进行修改。	分解要消耗一定的成本，同时可能会增加复杂度。	采用。可以将服务器中不同逻辑部分拆分来进行分解。
延迟绑定	到最后才决定执行的动作。	越往后就越有多种可能性，会使得复杂度提高。使得成本变高，测试效率变低。	不采用。提高了复杂度。
重构（refactor）	通过抽象出共同的 service，可以降低内聚性，使得代码的效	成本较大。	不采用。成本太大。

	率更高。		

第五次迭代结果

如图所示，新增元素说明如下。

攻击侦测模块：侦测非正常访问，在攻击发生时禁止外界访问服务器模块。

数据加密模块：对服务器与外部系统交互时传送的数据进行加密和解密工作。

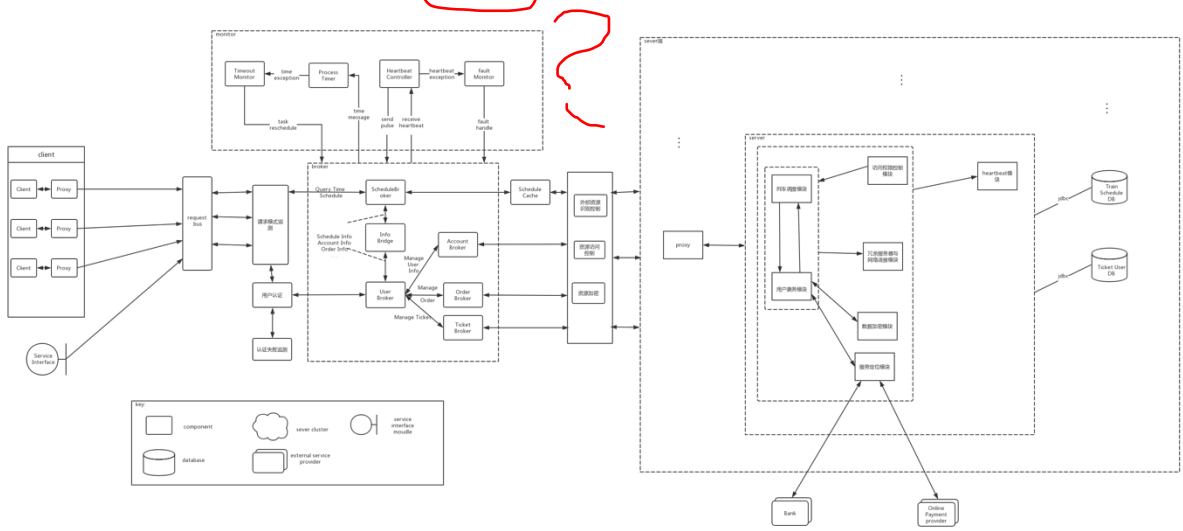
冗余的服务器与网络连接模块：作为候补，等待当前运行的服务器失效或者网络连接失效而发挥作用。

heartbeat 模块：监测服务器模块是否失效，以尽快做出反应。

访问权限控制模块：监测用户的访问范围，使用户的访问范围始终被限制在必须需要访问的范围内。

服务定位模块：用于处理与外部系统的交互，将外部系统视为服务的提供者，从而简化系统的工作。

服务器端内部拆分为列车调度模块与用户票务模块。



第六次迭代

选择元素

第六次迭代选择的元素是请求总线模块。

选择 ASR

第六次迭代选择的 ASR 是性能、可用性。

1.因为 12306 的用户群体非常广大，所以经常会出现高并发的情况，这时候要考查系统的性能，在压力情况下，系统要能够正常运行而不是崩溃，所以请求总线需要对于请求能够合理分派，保持负载均衡。

2.因为 12306 购票系统不能出现长时间失效的情况，而如何请求总线一旦失效，那其实整个系统也就因为这一个组件而失效。因此确保请求总线的可用性是非常重要的。

候选策略表和决策

性能

策略	Pros	Cons	是否采用
智能分派	能够识别出请求对应的具体 broker，将其直接分派到对应的 broker 上，从而实现智能分派，避免造成单个 broker 的压力过大。	需要额外的工作量。	采用。
队列分类	对于 user broker 再构建一个 user broker 的请求队列，对于 schedule broker 构建一个 schdule broker 的请求队列。提高了效率。	耗费资源。	采用。
增加资源	提高了请求总线的效	耗费资源。	采用。

	率，从而提高处理请求效率。		
限制执行时间。	给每一个请求的处理确定一个执行时间的上限，避免单个请求耗费时间过多，造成性能下降。	可能会错失一些请求造成一些请求没有回应。	不采用。错失请求的范围扩大会演变为失效情况。

可用性

请求总线冗余	防止正在运行的请求总线的失效情况。如果其失效，则用冗余的总线来代替它。	耗费资源。	采用。
ping/echo	能随时对系统发起询问请求。	通讯量是 heartbeat 方案的双倍。	不采用。
heartbeat	定期检测系统是否已经失效。消耗资源较少。	在系统内部要实现额外的 heartbeat 相关逻辑。	采用。在 server 对请求响应时，不发出 heartbeat 以免占用资源，在 server 不处理请求的空闲时刻，发出 heartbeat 来确保其没有失效。

第六次迭代结果

如图所示，新增元素说明如下。

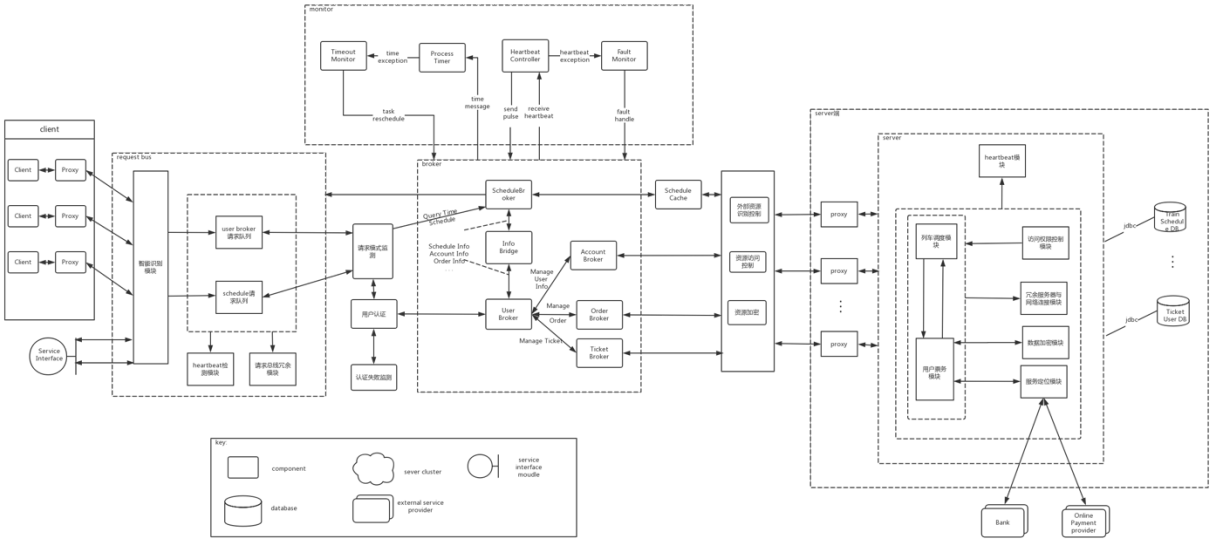
智能识别模块。用于识别正在处理的请求它对应的是 user broker 还是 schedule broker，并把请求放入对应的 broker 的请求队列中。

user broker 请求队列。用来给对于 user broker 的请求进行排队。

schedule broker 请求队列。用来给对于 schedule broker 的请求进行排队。

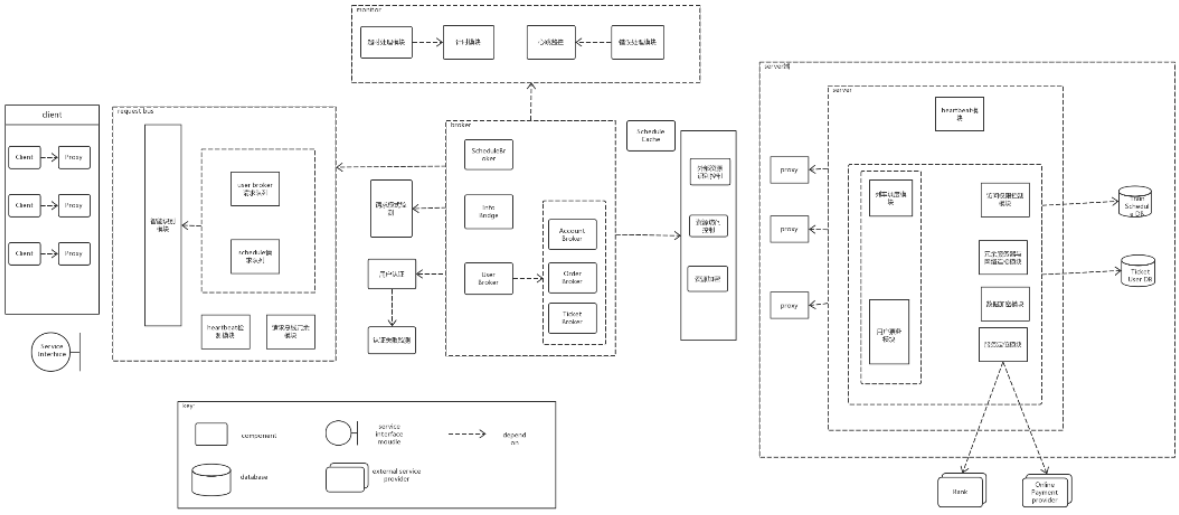
总线冗余模块。提供冗余，防止当前总线失效。

heartbeat 监测模块。用于检测系统是否失效，帮助其在失效情况下采取相应措施。



Module View

1.1 视图的主要表示(分解视图)



1.2 元素目录

本架构为 Broker 架构, 主要模块有 broker 模块(可细分为 monitor 模块、request bus 模块、安全模块、核心 broker 模块)、server 模块（可细分为列车调度模块、用

户票务模块、服务定位模块、数据加密模块、冗余模块、访问权限控制模块、heartbeat 模块、安全模块）

Broker 模块

1.monitor 模块

monitor 模块主要是用于监测整个 broker 模块的情况，主要采用 heartbeat 机制和绑定执行时间机制，来监测确保整个 broker 模块能够正常工作。

2.request bus 模块

request bus 模块主要负责对 client 端的请求进行识别，然后智能分派到不同的请求队列。不同的请求队列可以对应于后续的不同的 broker，减轻 broker 识别请求的压力。

3.安全模块

可以筛选从 request bus 模块进入核心 broker 模块的请求，去除危险的请求，同时记录可疑的请求，当可疑的请求次数累积到一定数量，采取相应措施。

4.核心 broker 模块

核心 broker 模块获取请求队列中的请求，定位此请求对应的服务器位置，进行发送。同时获取服务器端的响应，将其交给 client 端。

Server 模块

1.列车调度模块

列车调度模块负责与列车调度信息有关的操作处理。如查询列车调度信息等。

2.用户票务模块

用户票务模块负责与购票、用户订单相关的操作处理。如购买车票等。

3.服务定位模块

服务定位模块主要用于帮助 server 端与外部系统进行通信，服务定位模块可以帮助定位接口的位置，使得 server 端使用银行、公安系统的接口。

4.数据加密模块。

server 端与外部系统交互时，要传递一些关键数据，所以需要一个数据加密模块来加密数据，使得与外部系统交互的数据是绝密的。

5.冗余模块

为了防止服务器或者网络连接失效，冗余模块存在冗余的网络连接和服务器，在服务器失效的时候可以发挥作用，防止失效时间过长。

6.访问权限控制模块

为了限制请求对于服务器的访问范围，避免其能访问到自己不需要访问的资源，访问权限控制模块可以限制用户的访问范围，保证其只获取必需的资源。

7.heartbeat 模块

为了检测 server 端是否失效，heartbeat 模块一直监听从服务器端的信号，当信号没有被发送，heartbeat 发现 server 端失效，启动警报以及后续处理机制。

8.安全模块

过滤进入 server 端的请求，发现危险的请求并拦截。

接口包括内部接口和外部接口，描述如下表：

接口名称	接口职责	接口类别
外部服务接口	向服务注册中心查询需要的外部服务	内部接口
请求模式监测接口	监测客户端发来的请求	内部接口
用户认证接口	监测访问权限	内部接口
监控服务接口	监测内部组件是否正常	内部接口
票务服务服务接口	给应用客户端提供查票订票退票等系统业务服务	外部接口
列车时刻表数据服务接口	提供列车时刻表的数据库信息检索服务	内部接口
订单和用户数据服务接口	提供订单和用户信息的数据库信息检索和更新服务	内部接口
队列调度接口	请求队列调度	内部接口
Broker 调度接口	请求定位转发	内部接口
资源安全接口	服务端资源的加密，权限控制	内部接口
备份数据服务接口	提供备份数据库信息检索和更新服务	内部接口

1.3 可变性指南

系统的可变性主要体现在：

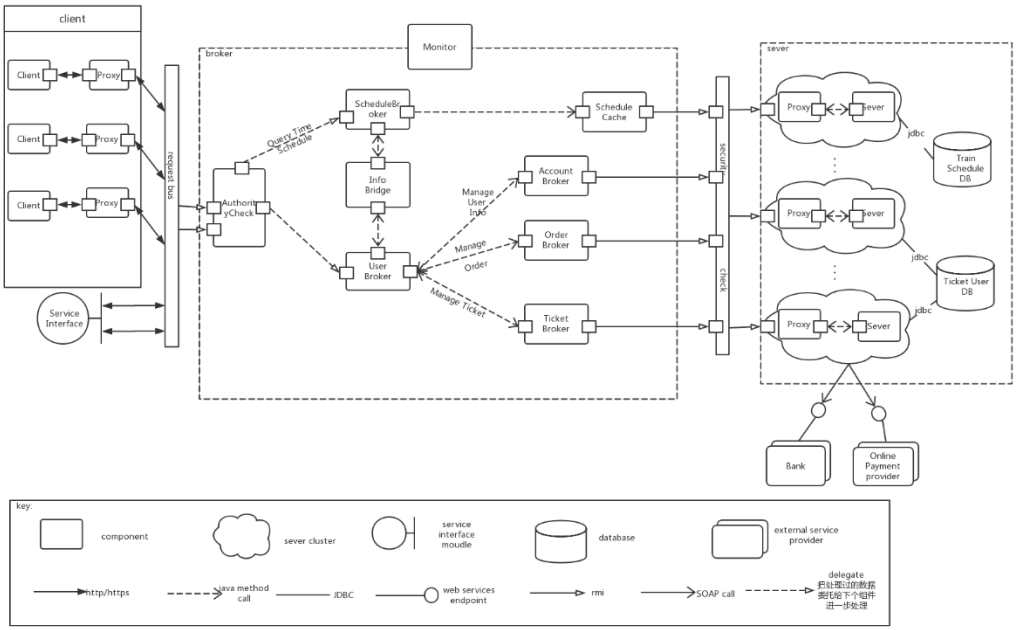
1. Client 和 Server 不存在耦合，知识掌握在 Broker 中，对于系统来说，功能服务

的改动将变得十分方便，面向变更的需求

2. RequestBus 模块、Broker 接口模块在运行时有 monitor 对状态进行监测，应对可能变化的环境对系统造成影响。
3. 业务处理模块对每一项服务都进行了冗余处理，可以在某一个处理器异常的情况下开启其冗余处理器。
4. 对数据进行备份，保证在发生变化时数据可恢复。

C&C 视角

1.1 视图的主要表示（分解视图）



1.2 元素目录

本架构为 Broker 架构，主要元素由 client、client proxy、broker、server proxy、server 组成以及之间的 Connector 组成。

图中 request bus 负责客户端请求的分派处理，可以识别请求类型并为其指派不同的 broker 进行处理，分派完成之后请求会先发送给 AuthorityCheck 模块进行安

全验证和数据加密（主要包括拦截 dos 攻击和非法请求），之后再通过它和 broker 之间的接口分发给指定好的 broker 模块；monitor 为系统对 broker 整个模块的监控器，负责周期性发送 heartbeat 对 broker 的健康状况进行判断，并处理异常情况，同时它也可以根据请求响应时间来对 broker 等待任务集进行动态调度，可以降低 broker 任务阻塞的可能性；broker 任务转发完成之后通过和 security check 之间的接口将请求转发给对应的服务器， security check 负责对 broker 和服务器之间传递的信息进行加密以防止信息泄露；Authority check 和 security check 这两个安全验证模块加强了 broker 的入口和出口的安全性，有效的降低了 broker 单点失效的可能性。

在 request bus 将不同的请求识别并分派给不同的 broker 之后，schedule broker 负责处理有关列车调度时刻表等信息的请求，user broker 负责处理用户账户、订单以及车票管理等信息的请求，由于 user broker 负责的功能较多，又对 user broker 的请求进行了二次分派，实现了负载均衡，具体信息可以查看 broker 部分的模块分解；server 端负责客户端请求数据的查找与存储，以及系统对外提供的接口，该模块在内部也进行了分解提高了性能、可用性、安全性等质量属性，具体分解可以查看 server 端的模块分解。

元素的接口和行为：

接口名称	接口职责	接口类别
客户端请求分派接口	给内部 requestbus 进行请求分派	内部接口
Request bus 请求分派接口	给内部权限验证模块先进行安全验证和数据加密，并提前指定每个请求的分派 broker	内部接口
Authority 安全模块接口	将通过安全验证的请求分发给 user broker 和 schedule broker	内部接口
列车调度 broker 接口	将列车调度相关请求先发送给服务器安全模块进行处理，并提前指定提供列车时刻表检索服务的 server 端	内部接口
用户 broker 接口	将到达的请求分发给 user account broker、order broker、ticket broker 进行二次分派	内部接口
用户账户 broker 接口	将用户账户相关请求先发送给服务器安全验证模块进行验证，并提前指	内部接口

	定所要分发得提供对应服务的 server 端	
订单 broker 接口	作用同上，只是它操作的是订单相关请求	内部接口
票务 broker 接口	作用同上，只是它操作的是票务相关请求	内部接口
Security check 服务器安全接口	将从 broker 拿到的信息进行加密处理，并发送给对应的 server	内部接口
server 端服务接口	提供银行相关信息和在线支付订单的服务	外部接口

1.3 可变性指南

系统的可变性主要体现在：

1. broker 架构中的 client 和 server 端可能会经常发生变化，但是他们不依赖于彼此，他们只依赖于 broker，这个设计使得他们可以通过在 broker 注册和解注册进行灵活地修改和扩展
2. broker 在运行时有 monitor 对状态进行周期性监测，如果出现崩溃或异常，可以快速利用冗余组件恢复 broker 状态。
3. server 端和 request bus 都有 heartbeat 模块进行周期性监测，也有冗余处理，可以快速处理系统的异常，提高系统可用性
4. 安全模块在运行时对客户端请求进行验证加密，可以通过请求次数验证等方式防御 dos 攻击，也可以通过记录用户登录次数进行用户访问权限的控制，大大提高了系统的安全性
5. Request bus 对请求进行智能分派和排队调度，可以避免个别请求占用系统资源而造成阻塞调度的情况发生
6. 分布式数据处理存储，可以方便的增加或者删除 server 端

SOA 架构

ADD 过程

第一次迭代

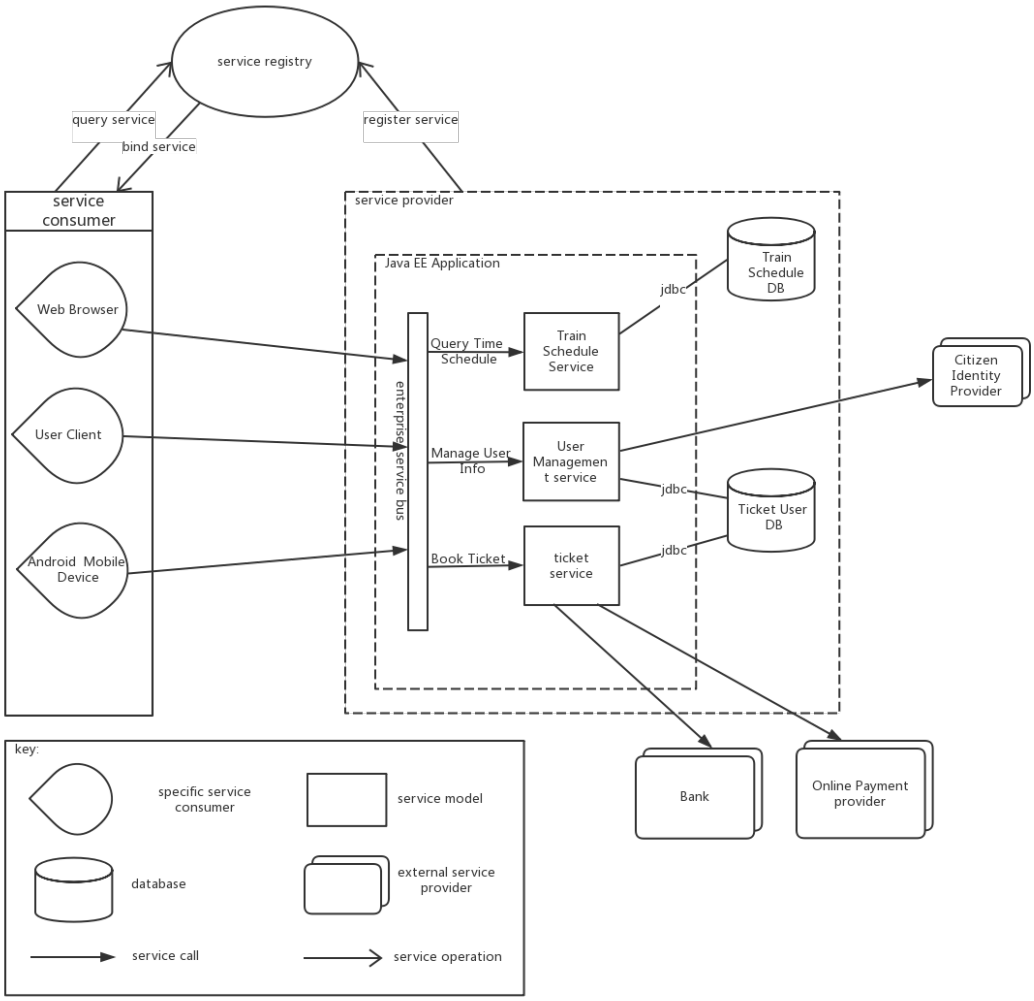
分解系统如图所示。服务端分为 ESB 模块、列车时刻模块、用户管理模块、购票模块以及相关的数据库，分别为列车时刻数据库和用户购票记录数据库。同时，注册中心用于服务提供者向其注册服务，以及服务消费者查找需要的服务。

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。

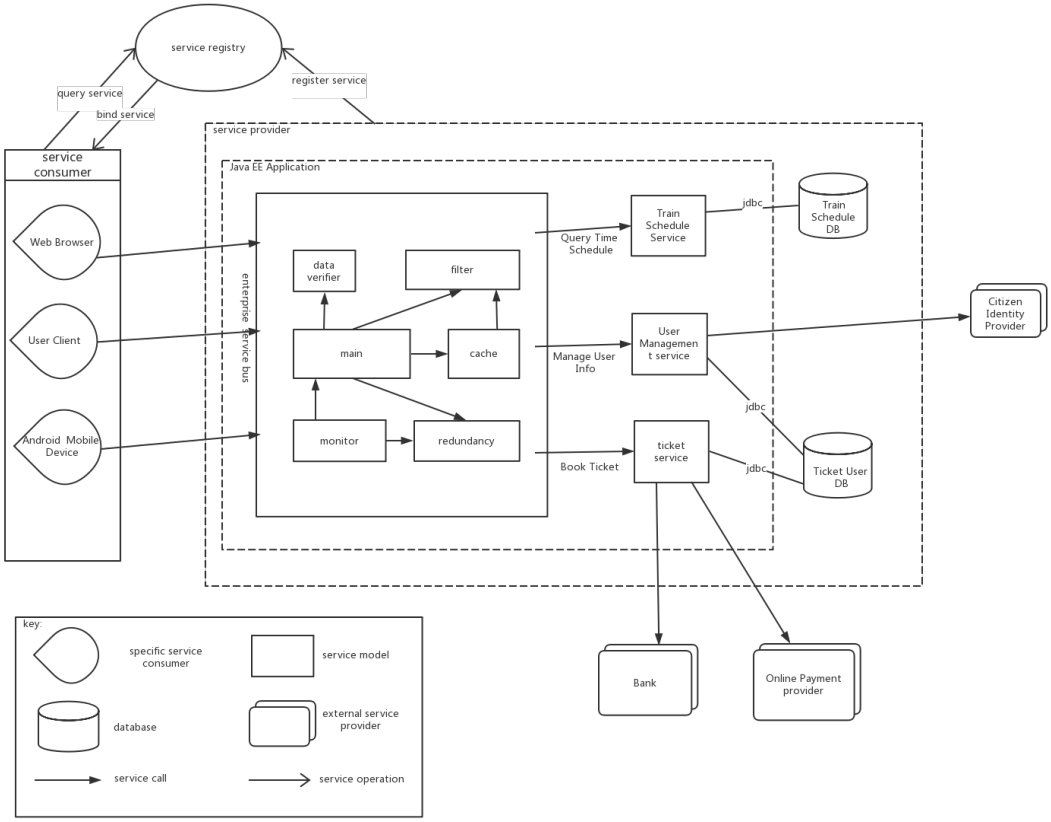
冗余备份模块 (Redundancy): 冗余存储请求队列中的信息，在主服务失去响应后进行恢复。

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	更多的处理资源、内存、更快的网	增加物理资源需要资金开支	采用。增加资源可以带

源	络带宽会提高系统的高并发处理能力。		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。

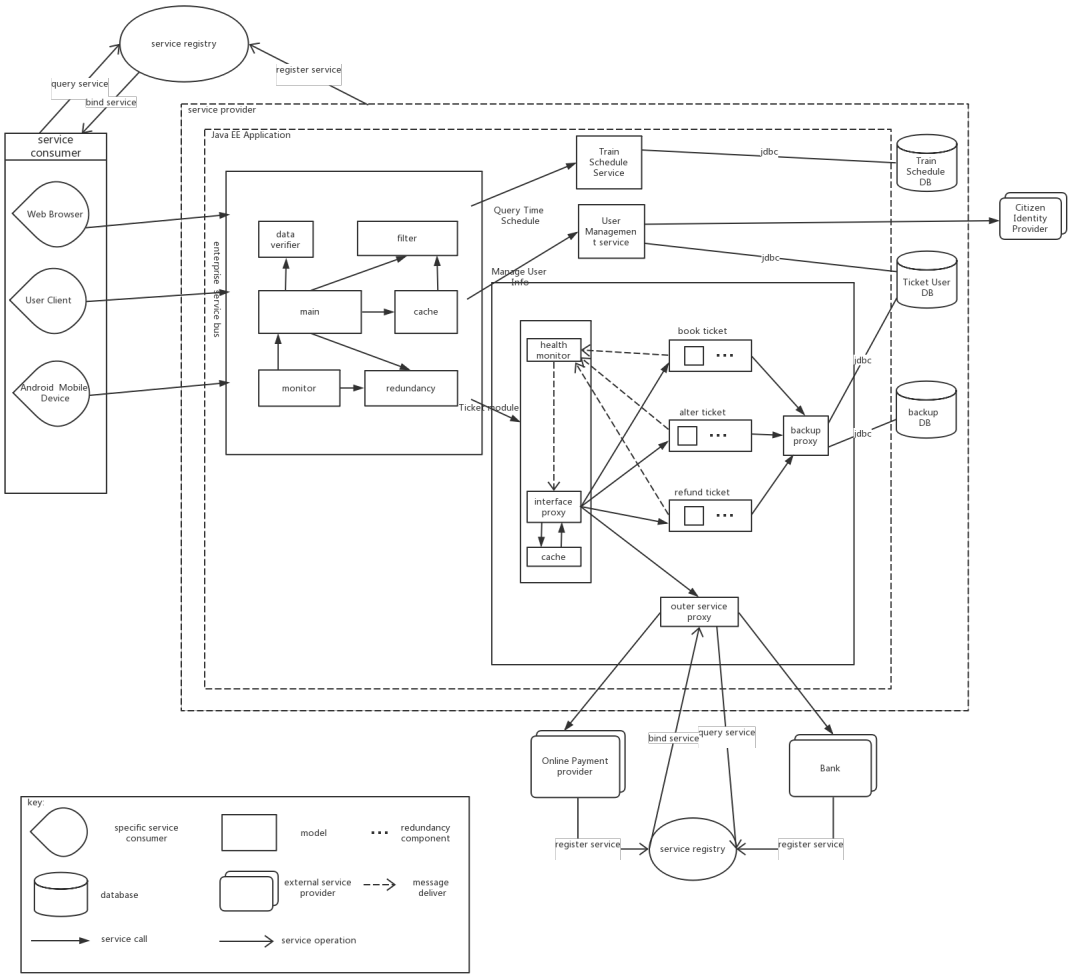
接口代理模块 (Interface Proxy): 接受监测模块的处理器状态消息，决定响应请求调用的处理器。高频的请求和响应存放在 Cache 中提高性能。同时决定各个服务接口的协调顺序，包括系统外部的服务。

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

	并行的调度算法达到期望的公平性、吞吐量		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

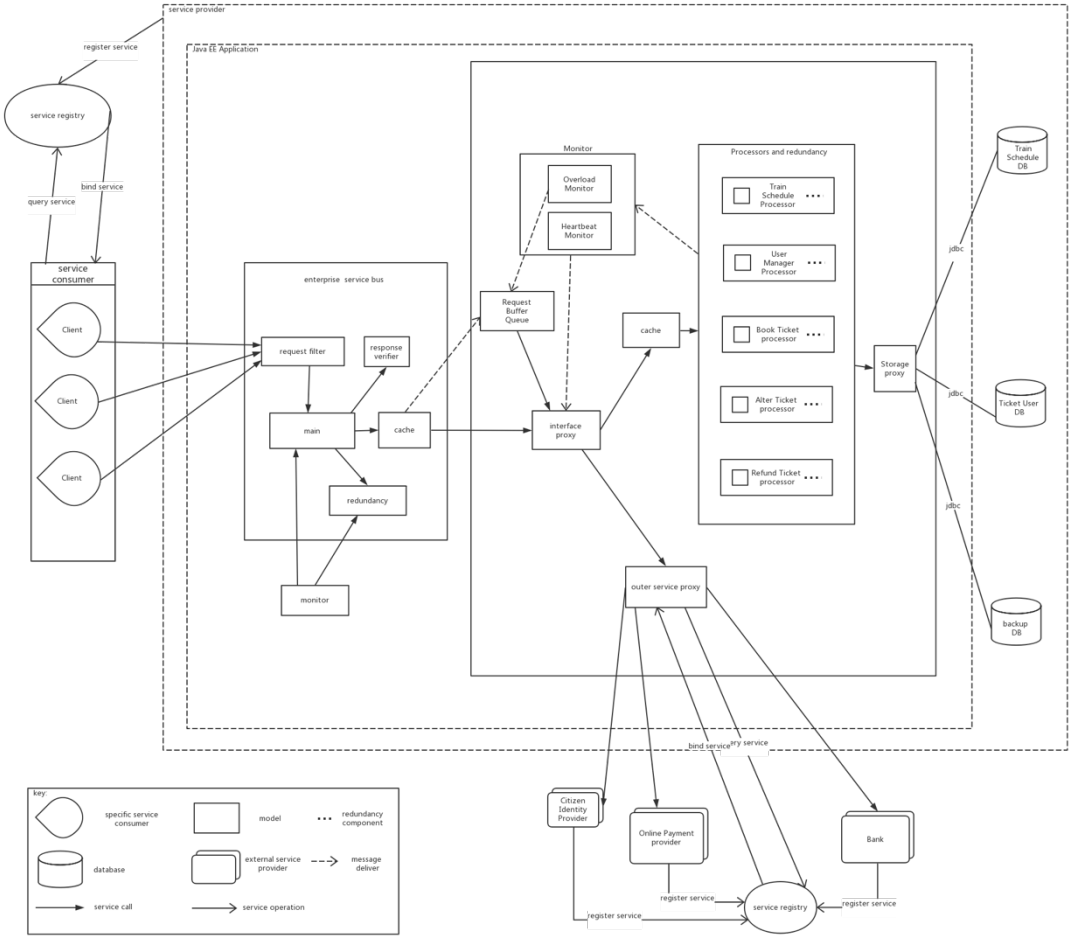
针对可用性：参见购票模块的处理

第四次迭代结果

新增元素如下：

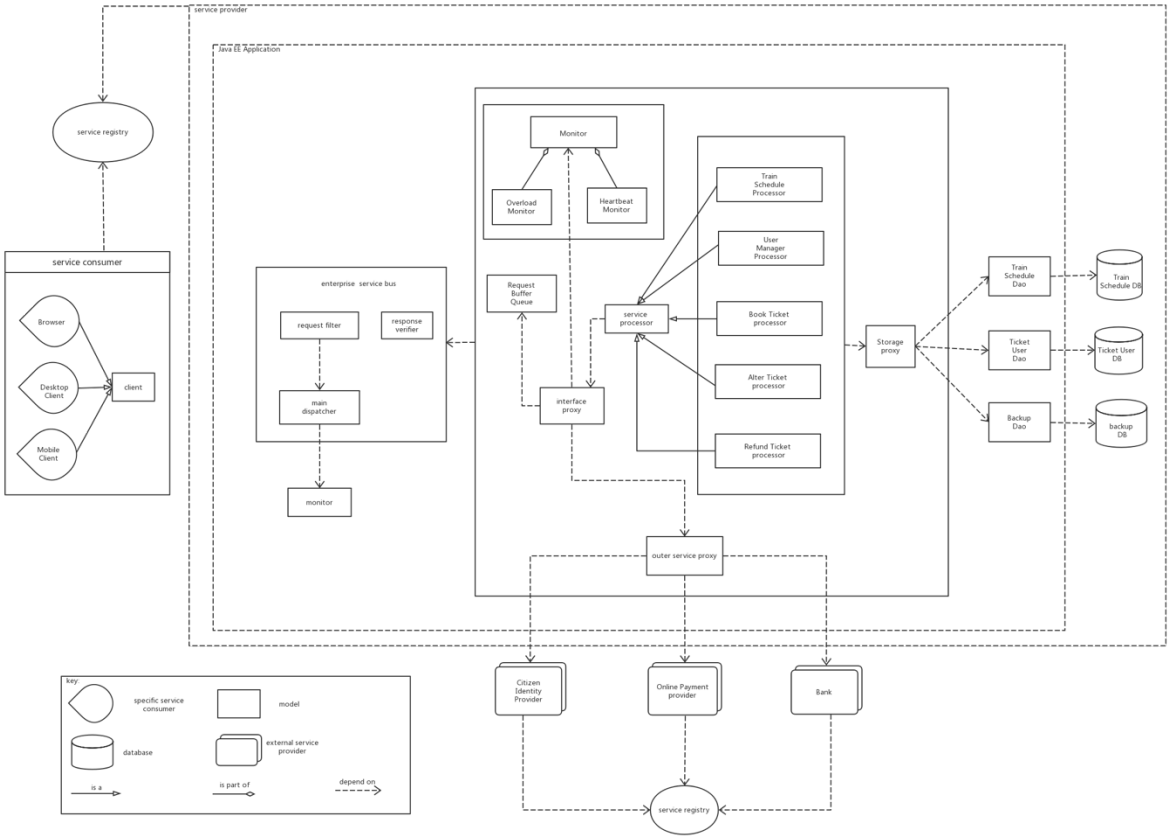
请求缓存模块 (RequestBufferQueue): 限制请求的处理，缓存到达的请求，服务器可以异步、按照优先级的顺序处理这些请求。

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。


service provider
register service
Java EE Application
service registry
bind service
query service
service consumer
Client
Client
Client
enterprise service bus
request filter
response verifier
main
cache
redundancy
monitor
Request Buffer Queue
Monitor
Overload Monitor
Heartbeat Monitor
interface proxy
cache
Processors and redundancy
Train Schedule Processor
User Manager Processor
Book Ticket processor
Alter Ticket processor
Refund Ticket processor
Storage proxy
Train Schedule DB
Ticket User DB
backup DB
jdbc
jdbc
jdbc
outer service proxy
bind service
ery service
Citizen Identity Provider
Online Payment provider
Bank
register service
register service
register service
service registry
key:
specific service consumer
model
redundancy component
database
external service provider
message deliver
service call
service operation

Module 视角

1.1 视图的主要表示(分解视图)



1.2 元素目录

本架构为 SOA 架构，主要模块有 ESB 模块、代理模块、业务模块以及监测模块。

ESB 模块用于实现路由选择的作用，连接服务端和客户端，对请求进行分发、过滤、缓存等处理，以减轻服务器负担。

代理模块包括接口代理、外部服务代理以及存储代理。接口代理用于协调不同服务之间的调用以完成整体任务，外部服务代理用于访问需要的外部服务，存储代理用于实现数据存储以及备份。

业务模块包括了系统提供的各项服务。将不同的服务放置在不同的处理器上运

行，并设置冗余以提高业务处理的能力。

监测模块用于监测其他模块是否正常工作以及核心组件能否正常运行，并对出现的故障在第一时间进行处理，提高系统的可靠性。

接口包括内部接口和外部接口，描述如下表：

接口名称	接口职责	接口类别
外部服务接口	向服务注册中心查询需要的外部服务	内部接口
安全过滤服务接口	过滤客户端发来的请求	内部接口
监控服务接口	监测内部组件是否正常	内部接口
票务服务服务接口	给应用客户端提供查票订票退票等系统业务服务	外部接口
列车时刻表数据服务接口	提供列车时刻表的数据库信息检索服务	内部接口
订单和用户数据服务接口	提供订单和用户信息的数据库信息检索和更新服务	内部接口
备份数据服务接口	提供备份数据库信息检索和更新服务	内部接口

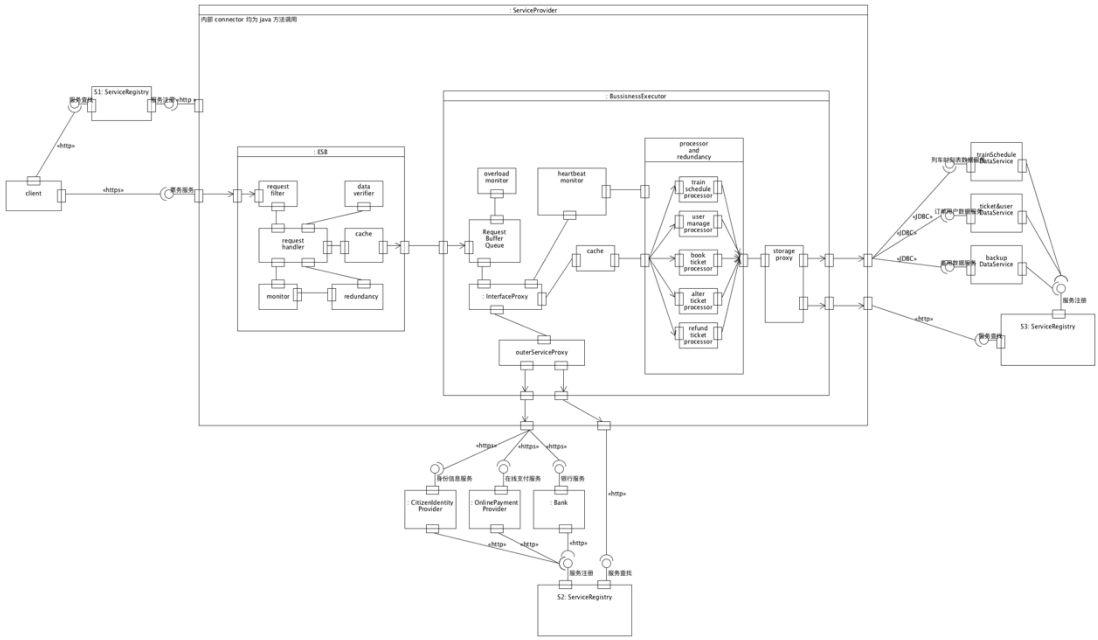
1.3 可变性指南

系统的可变性主要体现在：

1. 服务注册中心允许客户端在运行时动态绑定相关服务，系统可以为不同的客户端提高不同的服务。
2. ESB 模块、接口模块在运行时有 monitor 对状态进行监测，应对可能变化的环境对系统造成影响。
3. 业务处理模块对每一项服务都进行了冗余处理，可以在某一个处理器异常的情况下开启其冗余处理器。
4. 对数据进行备份，保证在发生变化时数据可恢复。

C&C 视角

1.1 视图的主要表示(分解视图)



1.2 元素目录

本架构为 SOA 架构，所以主要元素由 Service Provider, Service Consumer, Service Registry 以及之间的 Connector 组成。

上图中 S1 为 Client 端和后台服务端之间的服务注册点，可以提供不同的后台 service provider 注册，然后可以为 client 指定不同的 provider，实现负载均衡；S2 为后台服务端（作为 consumer）和外部服务之间的服务注册点；S3 为后台服务端（作为 consumer）和数据服务之间的服务注册点，可以为数据库的分布式管理提供支持。

后台服务端 Service Provider 在运行时由两大模块组成，一个是 ESB（企业服务总线），负责统一管理 consumer 发来的多种请求，包括过滤非法请求和格式错误请求等，而且可以利用 cache 减轻业务逻辑部分的压力，具体各个功能的分解可以查看 ESB 组件的分解部分；另一个是 Business Executor，负责处理业务逻辑，包括管理乘车人, 列车时刻表查询, 买票退票等, 该模块在内部也进行了分解提高了性能、可靠性等质量属性，具体各个功能的分解可以查看该模块的分解部分。

接口包括内部接口和外部接口，描述如下表：

接口名称	接口职责	接口类别
S1 及 S3 服务注册接口	给内部 service provider 进行服务注册	内部接口
S2 服务注册接口	给外部 service provider 进行服务注册	外部接口
服务查找接口	给 service consumer 进行服务查找	内部接口
票务服务接口	给应用客户端提供查票订票退票等系统业务服务	内部接口
身份信息服务接口	提供个人身份信息的核对和备案等服务	外部接口
在线支付服务接口	提供在线支付订单的服务	外部接口
银行服务接口	提供银行转账、网上银行等服务	外部接口
列车时刻表数据服务接口	提供列车时刻表的数据库信息检索服务	内部接口
订单和用户数据服务接口	提供订单和用户信息的数据库信息检索和更新服务	内部接口
备份数据服务接口	提供备份数据库信息检索和更新服务	内部接口

1.3 可变性指南

系统的可变性主要体现在：

1. SOA 架构决定了不同的 service consumer 可以在运行时绑定不同的 service provider，这允许我们对后台在运行时进行更新并让 client 重新绑定。
2. ESB 模块在运行时有 monitor 对状态进行监测，若出现崩溃或异常，可以进行重新启动并利用 redundancy 模块中的冗余信息进行状态恢复。
3. Business Executor 同样有 2 中的 monitor 模块和 redundancy 模块可以进行重启和恢复。
4. Processor 模块中可以新增不同的处理组件动态提高处理能力。

SOA & Broker

基于 ASR 的比较

security	Broker	SOA
Pros	因为 client 对于 server 的服务请求和 server 对于 client 的服务请求回应都是要通过 broker 的，安全模块可以过滤处理服务请求与数据请求。增强系统安全性。	对 ESB 的过滤设计可以防止恶意请求，提高系统的安全性
Cons	如果安全控制模块被入侵，那么整个系统的安全性将大大降低。	SOA 架构的安全性依赖于 ESB 等中间件的设计。一方面会对系统造成安全隐患，另一方面也会对系统的架构设计带来负担。

availability	Broker	SOA
Pros	在 Monitor，server，request bus 模块采取心跳监测，采用消极冗余的方式在系统发生故障时加快恢复速度，增强了系统的可用性。	对 ESB 模块以及业务核心模块采用心跳监测的方式保证系统的正常工作。
Cons	Broker 的存在容易引起单点失效，系统可用性较低。	各个服务模块更加分散，没有易失效的单点，该架构本质上可用性更高

Interopreability	Broker	SOA
Pros	在服务端和客户端都有 proxy，负责编码、解码、发送的工作，由 broker 负责定位，向合适的服务器发生请求，增强互操作性。	客户端只需要通过 Service registry 查询服务的接口就可以访问服务，所以系统一方面可以很方便地向 Service registry 注册自己可以提供的服务，比如购票服务，另一方面又可以向 Service registry 查询自己需

		要的外部服务，比如支付服务等
Cons	无	无

Performance	**Broker**	**SOA**
Pros	Request bus 可以对请求进行排队调度，避免个别请求占用系统资源，阻塞其他请求的处理。采用分布式数据处理，负载平衡，引入并发机制，有效的利用系统资源分配请求，增强系统的性能。	ESB 的存在可以提高系统的高并发处理能力。
Cons	由于请求需要经过 broker 的处理转发，经过很多模块，性能较差。在高并发时，request bus 可能成为性能瓶颈。	SOA 的性能一方面依赖于中间件，另一方面如果需要调用的子服务出现故障，会影响整体的性能，出现性能瓶颈

Scalability	**Broker**	**SOA**
Pros	采用分布式数据处理存储，增强系统的 scalability	采用计算资源冗余以及分布式数据处理，有利于 scalability
Cons		

Usability	**Broker**	**SOA**
Pros	决定于客户端的实现，客户端可以自由的进行交互设计。	决定于客户端的实现。
Cons	无	无

总结

SOA 架构

Pros：SOA 架构对系统内部来说降低了系统内部组件的耦合程度，也提高了系统的可修改性。但对系统外部来说 SOA 架构提高了系统的互操作性。客户端只需要通过 Service registry 查询服务的接口就可以访问服务，所以系统一方面可以很方便地向 Service registry 注册自己可以提供的服务，比如购票服务，另一方面又可以向 Service registry 查询自己需要的外部服务，比如支付服务等。

Cons：在外部访问服务时，SOA 架构需要对内部相关子服务的协作进行调节，在设计上提高了复杂性。同时，和服务相关的内部组件可能会对整体服务本身产生影响，即某个子服务的性能也会影响整个系统的性，造成瓶颈。此外，SOA 架构的安全性、性能等也要依赖于中间件的设计，给系统架构的过程带来额外的设计成本。

Broker 架构

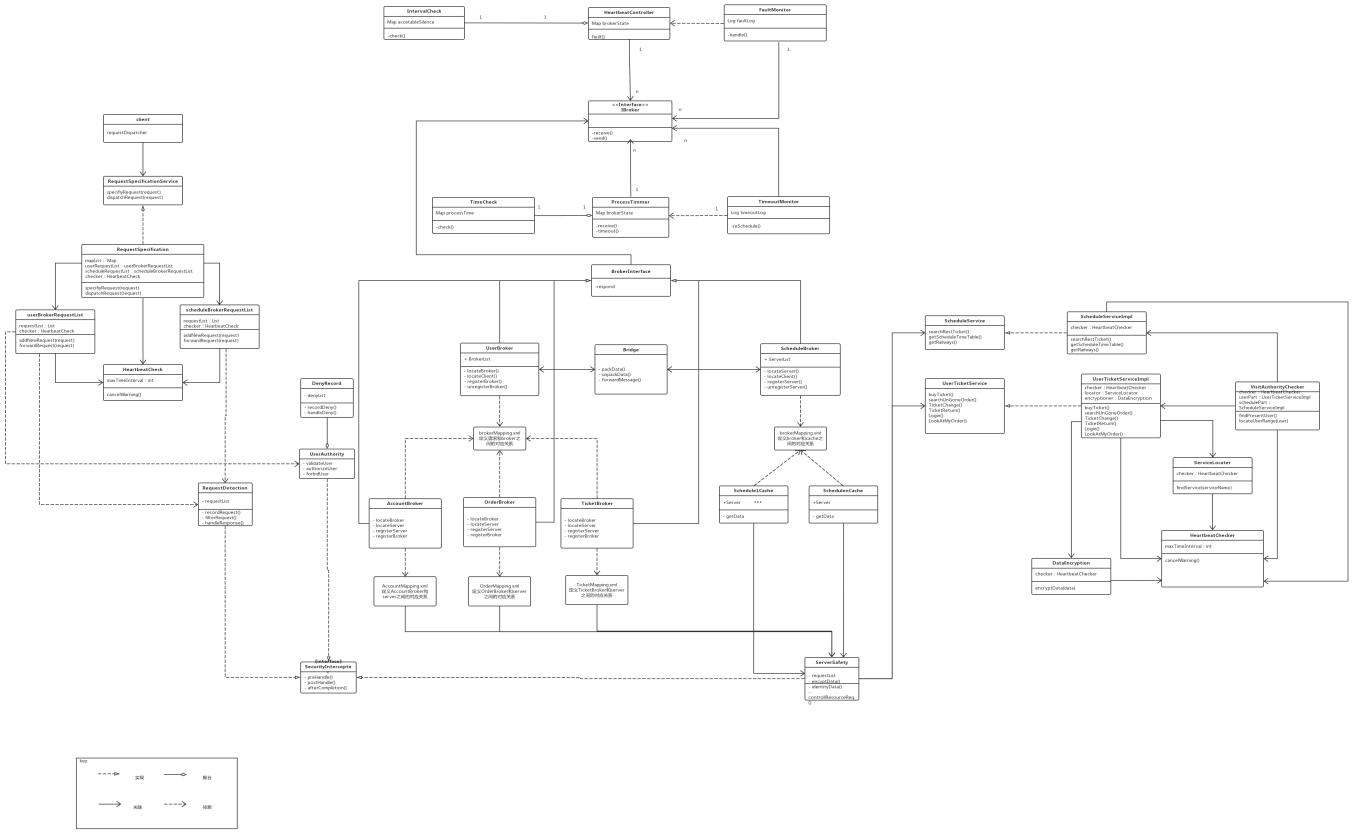
Pros：Broker 架构允许将服务端部署到实现不同功能的服务器，可以减轻 server 端的压力，将 server 端进行分布化，也使得 server 端更容易维护。同时也可以做到分发控制，负载平衡。Broker 架构也有利于系统的安全性，可以过滤恶意请求。当系统需要变化时，由于客户端和服务端都只依赖 broker，使得系统的可修改性和可扩展性都很好。

Cons：由于 broker 作为客户端和服务端的中间层，所以对系统的通信造成一定的影响。同时，当 broker 出现问题时，会对整个系统造成严重的影响。

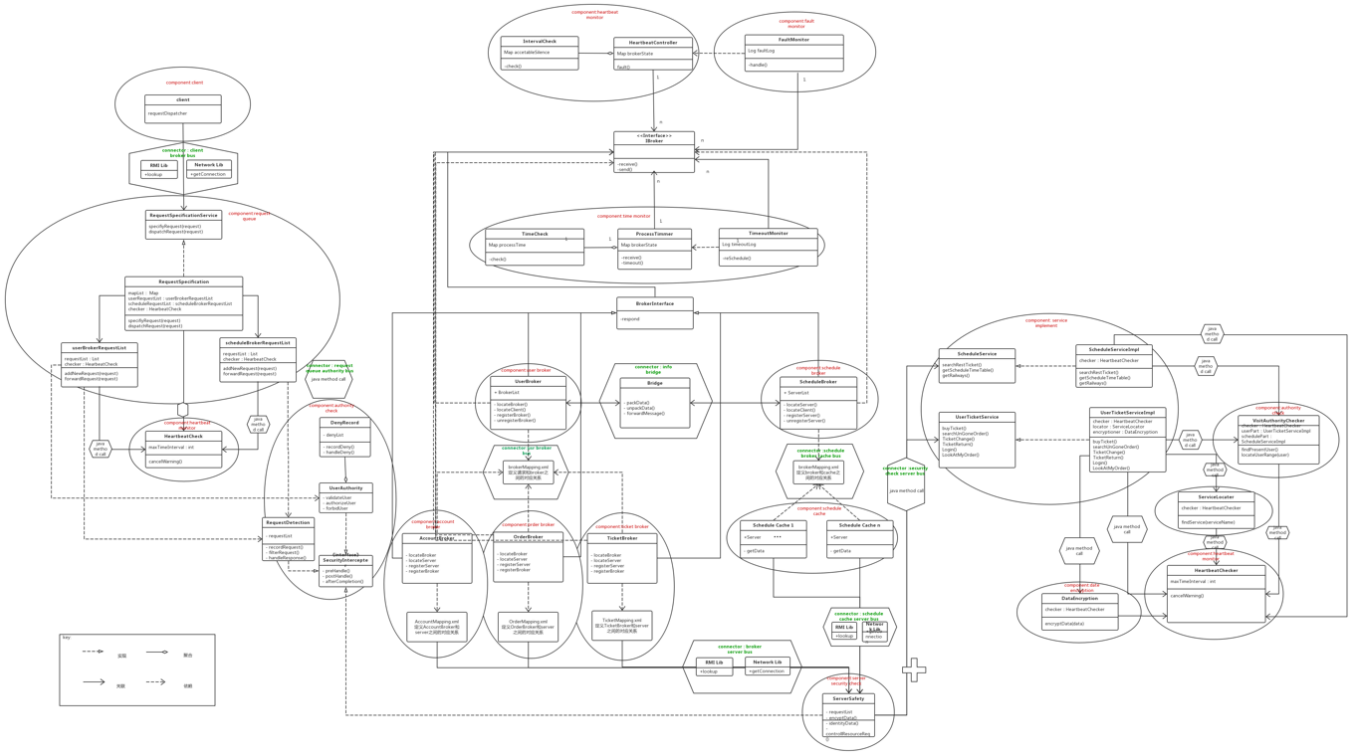
结论

对于使用本系统的用户而言，性能和安全性是最需要的。Broker 架构可以进行分发控制和负载平衡，减轻服务器压力，有效提高了系统的性能。同时 broker 可以有效截取并过滤客户端请求，提高系统的安全性。而 SOA 架构只能通过中间件来提高系统性能以及安全性。为了更好地实现 ASR，我们决定采用 Broker 架构对系统进行设计。

Broker 类图


IntervalCheck
HeartBeatController
FaultMonitor
client
RequestSpecificationService
RequestSpecification
TimeCheck
ProcessTimmer
TimeoutMonitor
BrokerInterface
userBrokerRequestList
scheduleBrokerRequestList
HeartBeatCheck
UserBroker
Bridge
ScheduleBroker
DenyRecord
UserAuthority
RequestDetection
AccountBroker
OrderBroker
TicketBroker
ScheduleICache
SchedulesCache
ScheduleService
SchedulerviceImpl
UserTicketService
UserTicketServiceImpl
VisitAuthentityChecker
ServiceLocater
HeartBeatChecker
DataEncryption
SecurityInterrepte
ServerSafety

与 C&C 组件的映射关系



其他的设计文档

Utility tree 质量属性效用树

质量属性	属性精化 (Attribute Refinement)	ASR (QA scenarios)
安全性	机密性(confidentiality)	身份验证通过的用户可以进行购票等操作，但是系统应当阻止外部恶意软件伪装合法用户进行的操作，并将伪装用户账号冻结（**H, M**）
	完整性(integrity)	系统在对用户身份进行验证的完成比率不小于 99.999%（**L, L**）
	应对恶意攻击	系统可以检测到外部软件的恶意攻击，同时在 10s 内采取相应保护措施，并自动上报攻击信息

		（M, H）
可用性	错误修复	系统自身运行时出现错误，快速修复错误的时间不超过 2s，错误修复成功率不低于 90%**（H, H）**
	无停机时间	系统可以向用户提供 7*24*365 标准的购票服务**（M, H）**
可延展性	硬件升级	系统服务器进行硬件上的升级，硬件升级队伍可以在一个工作日内完成任务，升级所影响的代码量不应超过 2%**（L, M）**
互操作性	与外部系统交互	系统调用银行系统提供的服务完成购票支付的操作，交互请求成功率大于 99.999%**（H, H）**
易用性	新用户学习	一个从未使用过系统的用户，可以在系统的指导下，在 15 分钟内成功完成一次购票操作**（M, M）**
	正常操作	一个使用过系统数次的用户可以熟练进行查票、购票的操作，在整个过程中不需要系统给出操作提示**（H, M）**
性能	用户请求的响应时间	在系统正常运行的情况下，当用户请求查询余票情况时，系统应当在 2s 之内给出查询结果**（H, M）**
	用户请求的响应时间	在系统满负载运行的情况下，当用户请求查询余票情况时，系统应当在 5s 之内给出查询结果**（H, H）**
	吞吐量	在满负载时，系统应当可以在一秒之内处理完 10000 次查询与 5000 次更新操作**（H, M）**

敏感点与权衡点列表

敏感点

编号	决策	质量属性	分析
S1	broker 被动冗余	可用性	能够有效处理错误，但被动冗余的 broker 作用不完全相同，需要额外信息才能恢复主 broker，占用一定资源，Broker 的数目

			会影响系统可用性
S2	限制操作时间	性能	控制一项请求的操作时间，保证处理资源的合理分配，但是检测操作时间本身消耗资源，过高的检测频率会影响处理性能
S3	引入并行	性能	并行处理请求可以减少阻塞等待时间，但并行数目过多也会导致并发处理问题
S4	队列分类	性能	对不同的 broker 分别构建请求队列，可以提高处理效率，请求队列的大小决定了耗费资源的多少，影响到 broker 处理性能

权衡点

编号	决策	质量属性	分析
T1	模块细化	性能，可修改性。	将 Broker 分为不同层级，可以提升系统的并发能力，但是增加了系统组件之间的通讯和系统复杂度
T2	检测请求模式	安全性，性能	检测请求可以避免抢票插件不正常的操作数据，但是增加了额外的检测模块，降低了每次请求的响应时间
T3	检测服务拒绝情况	安全性，性能	同上
T4	识别信息一致性	安全性，性能	能够保证用户数据的安全与一致，但增加了需要传输的数据总量，在带宽一定的情况下降低性能
T5	最小化系统的攻击面	安全性，可用性	请求必须经过安全处理模块，能够集中处理所有的攻击，但有可能导致单点失效
T6	Heartbeat	可用性，性能	发送 Heartbeat 的频率决定了检测错误的时间，但是过高的频率可能消耗过多的处理资源和带宽
T7	攻击发生时收回数据访问权限	安全性，可用性	侦测到攻击发生时，服务器模块拒绝外界访问直到确认安全，但是降低了可用性
T8	服务定位	互操作性，性能	采用目录命名机制，方便了与外界系统的交互行为，但维护服务目录需要消耗大量

			的资源
T9	模块分解	可修改性，可维护性	分解可以独立模块的职责，使得它们可以独立修改，但过多的模块数目会增加系统的层次与复杂度

3.风险决策

编号	决策	风险分析
R1	最小化系统的攻击面	安全处理模块能够集中处理所有的攻击，但如果安全模块收到集中攻击，可能导致单点失效
R2	Heartbeat	Heartbeat 频率可以影响系统检测到错误的时间，频率过低可能导致系统无法在规定的时间内发现错误
R3	被动冗余	被动冗余的 broker 数目过少会导致系统在错误发生时不能够及时恢复正常操作状态
R4	攻击发生时收回数据访问权限	攻击发生后系统在一段时间内收回数据访问权限，导致用户和外部系统无法请求系统资源，相当于系统下线

挑战和经验

在这次详细的架构设计中，我们分成了两个小组分别进行 SOA 和 Broker 的架构设计。我们遇到的第一个麻烦是，对于 ADD 方法的理解上，两个小组存在理解上的不一致，这是我们在分工之前的准备工作没有做好。最终我们通过会议对两个小组的文档进行了评审，并最终选择了当前的文档形式。

我们在这次实践中最大的收获是对文档进行了充分的评审，例如上面提到的统一了 ADD 方法的运用，此外，SOA 小组内部也对各个组件的分解进行了评审，在一些可以合并的地方进行了合并，这样子我们的组件分解更加清晰不冗余，也能确保组员对整体架构和小架构的了解。

在最终两个架构的选择中，我们进行了充分的讨论，罗列了两个架构的优缺点，在权衡点上进行了详细的讨论，最终我们从高铁票务系统的关键质量属性出发选择了最终方案。

在代码实现上，我们由两位组员先搭建总体框架，进行组件的划分和接口的定义。得益

于之前充分的讨论和评审，组内成员对于架构都有一些基本的认识，最后我们成功在非常有限的时间里面完成了票务系统的后台的实现。

组员和分工

姓名	学号	工作
孙康	141250117	编写安全性与可用性系统操作场景，参与 Broker 架构的 ADD 设计，具体负责服务器模块与请求总线模块的迭代设计，绘制相应模块的 UML 类图。搭建系统原型的主要接口与基本架构。
张文玘	141250192	编写互操作性与易用性系统操作场景，参与 Broker 架构的 ADD 设计，具体负责 user-broker 模块的迭代设计，绘制相应模块的 UML 类图。负责最终文档汇总与排版。参与系统原型的编码实现。
周小帆	141250209	编写性能系统操作场景，参与 Broker 架构的 ADD 设计，具体负责安全模块的迭代设计，绘制 Broker 设计的 C&C 视图，绘制相应模块的 UML 类图，负责最终文档汇总与排版。参与系统原型的编码实现。
王嘉琛	141250137	编写 scalability 系统操作场景，参与 Broker 架构的 ADD 设计，具体负责检测模块的迭代设计，绘制 Broker 设计的 Module 视图，绘制相应模块的 UML 类图。搭建系统原型的主要接口与基本架构。
吴嘉荣	141250148	参与 SOA 架构的 ADD 设计，具体负责 ESB 模块的迭代设计，负责绘制 SOA 设计的 C&C 视图。负责编写项目总结与经验分析部分的文档。参与系统原型的编码实现。
余旻晨	141250177	参与 SOA 架构的 ADD 设计，具体负责第一次整体迭代与购票模块的迭代设计，负责整理 SOA 架构设计和评估。负责编写两个体系结构优缺点分析与决策。参与系统原型的编码实现。
王梦麟	141250140	汇总系统主要功能，参与 SOA 架构的 ADD 设计，具体负责列车时刻模块的迭代。绘制 SOA 设计的 module 视图，负责 utility tree、敏感点、权衡点的编写。参与系统原型的编码实现。负责安排团队分工协作