列车时刻信息模块负责列车时刻表的维护、余票的查询，用户也可以查询车辆的正晚点信息。当列车发生晚点时，该模块可以向车站推送晚点信息，以便提醒乘客

**步骤一：**

该迭代不需要此步骤

**步骤二：**

1. **选择元素**：该迭代针对购票模块进行分解
2. **选择ASR**：可用性availability与性能performance，原因如下
3. 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。
4. 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。

**步骤三：**

确定框架驱动元素

|  |  |  |  |
| --- | --- | --- | --- |
| # | 架构驱动元素 | 对涉众的重要性 | 实现的难易度 |
| 1 | 场景3：应对系统错误 | 高 | 高 |
| 2 | 场景5：正常的客户端操作 | 高 | 低 |
| 3 | 场景6：高并发的客户流量 | 高 | 高 |
| 4 | 场景7：硬件升级 | 中 | 低 |
| 5 | 场景8：服务器数据库升级 | 中 | 中 |
| 6 | 场景9：系统不可崩溃、无响应 | 高 | 高 |

**步骤四：策略表和决策**

1. **设计关注点**

|  |  |
| --- | --- |
| 设计关注点 | 子关注点 |
| 限制请求 |  |
| 增加处理资源 |  |

1. **备选方案以及选择方案**

**针对性能：**

|  |  |  |  |
| --- | --- | --- | --- |
| 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 | 采用，系统处理的业务比较稳定，相比可修改性，性能更加重要 |
| 限制一个事件的执行时间 |  | 造成计算结果的不精确 | 采用，防止一次请求占用大量处理时间 |
| 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 |
| 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 |
| 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 |
| 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：**

**错误检测、错误恢复、错误预防**

|  |  |  |  |
| --- | --- | --- | --- |
| 策略 | 优点 | 缺点 | 是否采用 |
|  |  |  |  |

**步骤五： 本次迭代结果**

如图所示，本次迭代新增的元素说明如下：