第N次迭代

* 步骤一

第N次迭代可以不用此步骤

* 步骤二

我们选择GPS连接服务模块进行分解

* 步骤三

确定架构驱动因素，如下表所示

|  |  |  |  |
| --- | --- | --- | --- |
| # | 架构驱动因素 | 重要性 | 难易度 |
| 1 | 场景X：用户数量增加 | 高 | 中 |
| 2 | 场景X：单车数量增加 | 高 | 中 |
| 3 | 场景X：用户结束骑行 | 高 | 中 |
| 4 | 场景X：与GPS全球定位系统进行交互 | 高 | 低 |

* 步骤四

选择满足架构驱动因素的架构模式

1. 针对可用性相关的架构驱动因素

i. 设计关注点

|  |  |
| --- | --- |
| 设计关注点 | 子关注点 |
| 正确性 | 精确地定位用户的位置 |
| 精确地定位用户周边单车的位置 |
| 解锁单车时发生故障的概率 |

ii. 各子关注点的候选架构模式

1. 精确地定位用户的位置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 模式名称 | 成本开销 | 难易程度 | 稳定性 |
| 1 | 多次定位取均值 | 低 | 低 | 低 |
| 2 | 使用高精度的GPS定位服务 | 高 | 低 | 高 |

* 选择理由：多次定位取均值虽然成本开销低、比较容易实现，但稳定性差，效果不好。使用高精度的GPS定位服务虽然成本开销高，但开发难度低，足够稳定。由于系统需要精确、稳定的定位服务，因此我们选择使用高精度的GPS定位服务来精确定位用户的位置。

b) 精确地定位用户周边单车的位置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 模式名称 | 成本开销 | 难易程度 | 稳定性 |
| 1 | 单车端内置LocalSense定位 | 高 | 低 | 高 |
| 2 | 使用高精度的GPS定位服务 | 高 | 低 | 高 |

* 选择理由：单车端内置LocalSence定位服务的成本比使用高精确的GPS定位服务成功略高，且稳定性不如GPS定位，因此我们选择使用高精度的GPS定位服务来精确定位用户周边单车的位置。

c) 解锁单车时发生故障的概率

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 模式名称 | 成本开销 | 难易程度 | 可实施性 |
| 1 | 使用Observe模式，单车第一次出现故障时，发送维修请求给维修人员 | 低 | 中 | 中 |
| 2 | 使用备忘录模式，单车第一次发生故障时，单车嵌入式系统自动恢复到一个正确的状态 | 中 | 中 | 中 |
| 3 | 使用observe模式+备忘录模式 | 中 | 中 | 高 |

* 选择理由：三种解决方案的成本开销、难易程度基本一致。使用observe模式，单车发生故障后，需要维修人与手动维修，中间花的时间过长，会影响单车的使用。使用备忘录模式，单车发生的故障若是物理器件的损坏，单车不能自动修复，可行性不高。使用observe+备忘录模式，单车在第一次发生故障时，使用备忘录模式尝试单车自行恢复，若自行恢复失败，则通知维修人员，可行性高。因此我们选择使用observe+备忘录模式。

iii. 候选架构模式综合评估

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 精确地定位用户的位置 | | 精确地定位用户周边单车的位置 | | 解锁单车时发生故障的概率 | |
| 优点 | 缺点 | 优点 | 缺点 | 优点 | 缺点 |
| 使用高精度的GPS定位服务 | 易实现  稳定性高 | 成本开销高 | 易实现  稳定性高 | 成本开销高 | 可用性高 | 无显著缺点 |
| observe模式+备忘录模式 | 无明显影响 | | | | 大大降低单车发生故障的概率 | 复杂度偏高，实现难度稍大 |

* 选择理由：精确地定位用户的位置、精确地定位用户周边单车的位置可同时使用高精度的GPS定位服务，即简化了系统的复杂度，又能降低系统成本。且高精度的GPS定位服务有极高的可用性、稳定性、易于实现。因此我们采用高精度的GPS定位服务。Observe模式+备忘录模式具有较高的可行性，且成本低，因此我们采用此方案。

1. 针对可伸缩性相关的架构驱动因素

i. 设计关注点

|  |  |
| --- | --- |
| 关注点 | 子关注点 |
| 高并发的处理能力 | 适应突然暴涨的用户人数 |
| 适应突然暴涨的单车数量 |

ii. 各子关注点的候选架构模式

1. 适应突然暴涨的用户人数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 模式名称 | 成本开销 | 难易程度 | 稳定性 |
| 1 | MapReduce | 高 | 中 | 低 |
| 2 | BigTable | 高 | 高 | 高 |

* 选择理由：MapReduce的实现难易程度虽然不高，但是其稳定性差，出错被认为是很寻常的事，因此我们在这里选择BigTable

1. 适应突然暴涨的单车数量

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 模式名称 | 成本开销 | 难易程度 | 稳定性 |
| 1 | MapReduce | 高 | 中 | 低 |
| 2 | BigTable | 高 | 高 | 高 |

* 选择理由：同上选择理由

iii. 候选架构模式综合评价

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 适应突然暴涨的用户人数 | | 适应突然暴涨的单车数量 | |
| 优点 | 缺点 | 优点 | 缺点 |
| MapReduce | 实现较为容易 | 成本高且不稳定 | 实现较为容易 | 成本高且不稳定 |
| BigTable | 稳定性高 | 成本高且难以实现 | 稳定性高 | 成本高且难以实现 |

* MapReduce、BigTable都适合处理高并发，都能满足可伸缩性的要求。两种架构的成本开销都较高。但由于系统需要较好的稳定性，而MapReduce又经常容易发生错误，因此，在这里我们选择BigTable架构。