Argus Technology Selection Report

# 1. System Design

## 1.1 Overview



Figure 1.1 System Design

Argus智能后台维护系统主要用来管理现场服务工程师提交的缺陷问题单，对问题解决的全流程提供跟踪域数据支持。同时能够在后台分析自动采集的后台日志数据，利用统计学手段提取缺陷的一般性特征。做到对未来新问题的辅助定位和对潜在问题的预判。

## 1.2 Description

整个系统由以下几个部分构成：

Dispatcher，调度器。对现有的所有未关闭的缺陷问题单进行调度和跟踪，保存问题解决所需的必要信息，并确保所有问题能及时闭环。

Data Analyzer，数据分析器。对定时采集的日志数据进行分析。重点提取用户的操作行为和问题关联日志的特征。帮助开发人员及时解决问题，同时找到问题的深层规律。

Reporter，系统报告。将后台的数据及时上报给不同环节的用户。提供丰富的数据表格和图表，帮助用户直观理解数据，挖掘背后的联系，对未来的工作决策提供数据支持。

Code Analyzer，代码分析器。与开发人员的代码库关联，分析代码的静态和动态结构，帮助开发人员解决一些无法通过人力发现或复现的问题。

Log Comparer，日志比较器。通过对两台用户设备日志的对比，用户日志与错误日志之间对比，用户日志与测试日志之间的对比。找出错误日志发生规律，对Data Analyzer提供必要数据，并帮助测试人员复现问题。

Test Generator，自动化测试用例生成器。根据Data Analyzer提取的用户行为特征（时间间隔，操作等），自动生成复现问题所需的测试用例。帮助开发人员复现问题，以及问题解决之后的回归测试。

# 2. Architecture Design

## 2.1 Hierarchical Design



Figure 2.1 Hierarchical Design

整个系统采用B/S模式，前台使用Web网页作为数据呈现平台，通过WebAPI与后台的数据逻辑进行交互。后台的Dispatcher组件负责工作流的调度跟踪，Analyzer负责对日志数据的分析提取。他们分别有一个独立的数据库，Business DB用于存放工作流相关的数据，LogDB则专注于对原始Log日志和分析结果的持久化。

## 2.2 Components Design

# 3 Technology Selection

## 3.1 Frontend

#### 备选方案：

React, VUE, Angular 2

#### 方案比较：

React

优点：采用虚拟DOM进行快速更新，使用FLUX架构实现单向更新，在服务器上预渲染，减少SPA的浏览器压力。

缺点：目标只是是UI组件，需要和Redux等其他框架结合形成Web应用，第三方组件远不如Angular组件丰富。

VUE

优点：简单的API实现响应数据搞定，中文文档非常完善。组织结构灵活，可以自己确定如何构建，且构建出的尺寸很小。有类似React的虚拟DOM，性能表现很好。

缺点：个人主导的项目，服务周期相对不稳定。全世界范围内支持的生态环境不够丰富。只关注Web，对于移动设备无法很好支持。

Angular

优点：模板功能丰富，整体框架完善。有不输React的性能。较高的抽象层次，让开发者更加专注于具体业务实现。Angular-CLI提供意见是构建，易于上手。Google+微软的官方支持，整体生态环境非常完整，全球社区能提供丰富的资源。

缺点：size较大，构建起来较重。学习曲线较陡，TS具有和原有JS不同的语言特性，内置依赖注入，路由，装饰器，服务等给原JS开发人员带来较大学习成本。

|  |  |  |  |
| --- | --- | --- | --- |
|  | React | VUE | Angular |
| Performance | 3 | 3 | 3 |
| Build Tool | 1 | 2 | 3 |
| Build Size | 3 | 3 | 1 |
| Perfection | 1 | 2 | 3 |
| Document | 2 | 3 | 2 |
| Environment | 2 | 2 | 3 |
| Study | 2 | 2 | 3 |
| **Total** | **14** | **17** | **18** |

Table 3.1 Frontend Technology Selection

#### 方案选择：

从性能上看，三种框架都支持服务器端渲染和虚拟DOM，能够极大提升性能。

Angular的框架比较完善，提供了功能强大的构建工具，虽然构建的尺寸较大，但其依赖关系能够很好地解决，对于初学者非常友好。另外，由于Angular框架本身提供了足够高的抽象，对于设计者而言，也省去了很多的时间。

在文档支持领域，虽然不如VUE的中文团队提供的支持丰富。Angular在全球范围内仍然具有巨大的社区和来自google与微软的支持。

对于Angular学习成本高的劣势，对我们而言反而是优势。不存在原有的JS技术栈，使得学习任何知识都是新知识，反而有较高的效率。另外，在WPF的开发上具有丰富的经验，使得Angular+TypeScript组件化的开发更容易上手。

**综上，前端开发建议选择Angular作为框架，并辅以jQuery，Bootstrap，Chart.JS等组件提供数据，UI，图表相关的功能。**

## 3.2 Web API

#### 备选方案：

#### 方案比较：

#### 方案选择：

## 3.3 Data Analyzer

#### 备选方案：

#### 方案比较：

#### 方案选择：

## 3.4 Backend

#### 备选方案：

#### 方案比较：

#### 方案选择：

## 3.5 Database

#### 备选方案：

MySQL（关系型），MongoDB（非关系型，键-文档），Neo4j（非关系型，图）

#### 方案比较：

MySQL

优点：关系型数据库拥有几十年的发展历史，具有庞大丰富的生态环境和支持工具。对于关系丰富的业务具有很强的处理能力。内容存于硬盘上，不需要大量占用内存。在复杂事务处理中保持数据一致性。数据更新开销较小。

缺点：其存储数据在硬盘之上，又使用SQL语句进行查询，使得在大数据量下join多表查询性能较差。同时由于其结构在早期设计建模阶段已经确定，具有较大的修改成本，很难进行横向扩展。由于数据表是严格固定的，一对多关系中反查非常困难。

MongoDB

优点：性能基于存储于缓存的键值对，有极高的查询性能。由于数据之间没有耦合，所以非常容易水平扩展。

缺点：内容存于缓存中，需要占用较大空间。技术较新，业内没有统一的数据查询标准（JSON的接受程度较高）。由于数据之间没有耦合，对于业务关系较为紧密的数据也没有足够强大的操作能力。对于一对多关系也无法简单实现反查。

Neo4j

优点：在存储实体信息为节点的同时，保留了关系数据库中的关系作为连边，使其易于在面向事务的应用中处理数据。由于本身是NoSQL的，对于数据节点本身的建模变化，影响较小，便于快速扩展业务。使用图模型作为存储结构，无论正反方向都具有很高的查找效率。图结构具有很强的表现力，在大数据分析领域可以发挥巨大潜力。

缺点：属于内存数据库，由于存在连边，其占用内存比MongoDB要大。由于依然保存了连边，使得修改节点时，数据修改量不如关系型数据库小。

|  |  |  |  |
| --- | --- | --- | --- |
|  | SQL | MongoDB | Neo4j |
| Query Performance | 1 | 2 | 3 |
| Modify Performance | 3 | 2 | 2 |
| Schema Modify | 1 | 3 | 3 |
| Business Handler | 3 | 1 | 3 |
| Reverse Query | 1 | 1 | 3 |
| Expressive | 2 | 1 | 3 |
| Memory Use | 3 | 1 | 2 |
| Environment | 3 | 2 | 1 |
| Development | 1 | 3 | 3 |
| Cost | 1 | 3 | 3 |
| Total | 19 | 19 | 26 |

Table 3.4 Database Technology Selection

#### 方案选择：

从系统整体设计而言，包含于业务流程强相关的部分和数据之间相对解耦的日志部分。单一选择某一种数据库都无法兼容所有需求。SQL数据库和NoSQL数据库在各种参数上几乎针锋相对。SQL数据库非常接近底层，而文档型NoSQL数据库非常接近后台代码实现。但他们也都有比较明显的短板。

SQL数据库更适用于面向复杂业务关系的数据库，而NoSQL则适用于关系解耦的数据库。但图数据库虽然属于NoSQL，却没有抛弃关系，而是将其作为减少冗余的方法。在内存占用和事务处理上都优于其他NoSQL数据库。另外由于其双向边关系网的丰富表达力，和对数级的查询效率在大量依赖反向查询的数据挖掘领域相对于SQL数据库也具有很强的优势。

**综上，建议在业务流中和提取出的用户特征采用图数据库精心存储，方便使用大数据的方法进行深度挖掘。使用MongoDB存储Log日志的原始信息。在所有模块之间的数据传输中使用JSON格式，统一接口。**