技术栈

目录

[Overview 5](#_Toc491794884)

[Scala 6](#_Toc491794885)

[Java 7](#_Toc491794886)

[Spark 8](#_Toc491794887)

[Spark sql 8](#_Toc491794888)

[Data warehouse 8](#_Toc491794889)

[Spark Streaming 8](#_Toc491794890)

[Structured Streaming 8](#_Toc491794891)

[Memory db 10](#_Toc491794892)

[Alluxio 10](#_Toc491794893)

[Graph db 11](#_Toc491794894)

[TinkerPop3 11](#_Toc491794895)

[Neo4j 12](#_Toc491794896)

[TitanDB 12](#_Toc491794897)

[JanusGraph 12](#_Toc491794898)

[RPC & serialization 13](#_Toc491794899)

[Thrift 13](#_Toc491794900)

[Avro 13](#_Toc491794901)

[Protocol Buffers 13](#_Toc491794902)

[引用 14](#_Toc491794903)

[附录 15](#_Toc491794904)

# Overview

# Statistic learning

# 概率论

# 线性代数与微积分

# Deep learning

# 开发架构设计

## 执行入口参数解析

## 功能路由调度

## 功能逻辑封装

## 公共模块抽象

## 面向对象开发应用

### 设计模式

### 继承

共用

### 封装

复用

### 多态

多种实现

### Single Responsibility Principle

一个类有且仅有一个职责，只有一个引起它变化的原因

### Open Closed Principle

软件实体如模块、类、函数应该对扩展开放，而对修改关闭

### Liskov Substitution Principle

所有引用基类的地方必须能透明地使用其子类的对象

### Interface Segregation Principle

不能强迫用户去依赖那些他们不使用的接口

使用多个功能单一、高内聚的接口总比使用一个庞大的接口要好

### Dependence Inversion Principle

高层模块不应该依赖底层模块，两者都应该依赖其抽象；抽象不应该依赖细节；细节应该依赖抽象

### Law Of Demeter

亦称为“最少知识原则（Principle of Least Knowledge）”

一个对象应该对其他对象保持最少的了解

### omposite/Aggregate Reuse Principle CARP

要尽量使用合成/聚合,尽量不要使用继承.

* 聚合表示的是整体和部分的关系，表示“含有”，整体由部分组合而成，部分可以脱离部分作为一个独立的个体而存在
* 组合是更强的“拥有”，部分组成整体，且不可分割，部分不能脱离整体而单独存在。组合关系中，部分和整体的生命周期一样，组合的新的对象完全支配其组成部分，包括它们的创建和湮灭等。一个组合关系的成分对象是不能与另一个组合关系共享的

### 对象模型要素

* 主要要素：
  + 抽象；
  + 封装；
  + 模块化；
  + 层次结构；
* 次要要素：
  + 类型
  + 持久
  + 并发

# 开发日志

经验法则

* 日志必须覆盖功能逻辑的“奇经八脉”
* 日志要有层次感
* 理想境界：输出的日志可以用代码进行分析
* 实践：代码记录日志，日志统一代码实现使用 slf4j

**private val** **LOG** **=** **LoggerFactory.**getLogger**(this.**getClass**)**

**private** **final** **static** Logger LOG **=** LoggerFactory**.**getLogger**(**WorldCountMR**.**class**);**

# 开发异常处理

经验法则

* 功能逻辑开发异常捕获

# Scala

# Java

## 设计架构

### 参数解析及功能调度

### 业务逻辑开发

#### 业务最小粒度处理

比如数据仓库数据导入，层次分别为dataWareHouse、database、table；此时业务最小处理逻辑为table，添加异常处理，使程序更具有健壮性；



### 公共层抽象

# Spark

## Spark sql

### Data warehouse

#### Metadata

## Spark Streaming

## Structured Streaming

# 数据库设计

## 理论

### 3大范式

# Memory db

## [Alluxio](https://github.com/Alluxio/alluxio)

# Graph db

## [TinkerPop3](https://github.com/apache/tinkerpop)

## [Neo4j](https://github.com/neo4j/neo4j)

## [TitanDB](https://github.com/thinkaurelius/titan)

## [JanusGraph](https://github.com/JanusGraph/janusgraph)

# RPC & serialization

## Thrift

## Avro

## Protocol Buffers

# 认知构建

## 时间管理

* 进程切换非常昂贵，避免多任务，保持单进程。
* 研究表明，集中注意力、高效工作，每天最多3小时。
* 划分任务的优先级，不要把‘急切’当做‘重要’。
  + 

### 番茄工作法

# 引用

# 附录