**电子科技大学成都学院**

**Spark用户电商行为分析**

项目设计报告

项目名称： Spark用户电商行为分析

项目成员：

专 业：

班 级：

任课教师姓名：

2021年6月制

摘 要

在大数据时代，如何高效地处理海量数据以满足性能需求，是一个需要解决的重要问题。 Apache Spark是专为大规模数据处理而设计的快速通用的计算引擎 现在形成一个高速发展应用广泛的生态系统。首先，在分析内存计算技术特点的基础上对其进行了分类，并分别介绍了各类技术及系统的原理、研究现状及热点问题；其次，对内存计算的典型应用进行了分析；最后，从总体层面和应用层面对内存计算面临的挑战予以分析，并且对其发展前景做了展望。

**关键词：**大数据时代； 计算引擎； 内存计算

ABSTRACT

In the era of big data, how to efficiently process massive data to meet the performance requirements is an important problem to be solved. Apache spark is a fast and general computing engine designed for large-scale data processing. Now it has formed an ecosystem with rapid development and wide application. Firstly, on the basis of analyzing the characteristics of memory computing technology, it classifies them, and introduces the principles, research status and hot issues of various technologies and systems respectively; secondly, it analyzes the typical applications of memory computing; finally, it analyzes the challenges faced by memory computing from the overall level and application level, and prospects its development prospects.

**Keywords:** Big data Era ; Computing engine; Memory computing

目录

[第1章 绪论 1](#_Toc43024473)

[1.1 研究背景 1](#_Toc43024474)

[1.2 Spark技术概述 1](#_Toc43024475)

[1.3 Spark技术的特点分析 2](#_Toc43024476)

[1.4 Spark技术具体应用分析 2](#_Toc43024477)

[第2章 Scala语言综述 3](#_Toc43024478)

[2.1 Scala语言的定义 3](#_Toc43024479)

[2.2 Scala语言的特点 3](#_Toc43024480)

[2.3 Scala与Groovy的对比 4](#_Toc43024481)

[2.4 Scala的适用场景 4](#_Toc43024482)

[第3章 Spark电商用户行为的实现 6](#_Toc43024485)

[3.1 集群环境搭建 6](#_Toc43024486)

[3.2 基于IDEA新建MAVEN工程 10](#_Toc43024489)

[3.3 Coding开发 16](#_Toc43024490)

[第4章 SpringBoot后端框架 18](#_Toc43024492)

[4.1 SpringBoot介绍 18](#_Toc43024493)

[4.2 SpringBoot的特点 18](#_Toc43024494)

[4.3 为何选择SpringBoot 19](#_Toc43024495)

[4.4 Maven仓库 19](#_Toc43024496)

[第5章 总结与展望 20](#_Toc43024500)

[参考文献 21](#_Toc43024501)

[致谢 22](#_Toc43024502)

1. 绪论
   1. 研究背景

Apache Spark 是专为大规模数据处理而设计的快速通用的计算引擎。Spark是UC Berkeley AMP lab (加州大学伯克利分校的AMP实验室)所开源的类Hadoop MapReduce的通用并行框架。尽管创建 Spark 是为了支持分布式数据集上的迭代作业，但是实际上它是对 Hadoop 的补充，可以在 Hadoop 文件系统中并行运行。通过名为 Mesos 的第三方集群框架可以支持此行为。Spark 由加州大学伯克利分校 AMP 实验室 (Algorithms, Machines, and People Lab) 开发，可用来构建大型的、低延迟的数据分析应用程序。

* 1. Spark技术概述

Spark，拥有Hadoop MapReduce所具有的优点；但不同于MapReduce的是——Job中间输出结果可以保存在内存中，从而不再需要读写HDFS，因此Spark能更好地适用于数据挖掘与机器学习等需要迭代的MapReduce的算法。Spark是一种与 Hadoop 相似的开源集群计算环境，但是两者之间还存在一些不同之处，这些有用的不同之处使 Spark 在某些工作负载方面表现得更加优越，换句话说，Spark 启用了内存分布数据集，除了能够提供交互式查询外，它还可以优化迭代工作负载。Spark 是在 Scala 语言中实现的，它将 Scala 用作其应用程序框架。与Hadoop 不同，Spark 和 Scala 能够紧密集成，其中的 Scala 可以像操作本地集合对象一样轻松地操作分布式数据集。

* 1. Spark技术的特点分析

Spark 主要有三个特点。首先，高级 API 剥离了对集群本身的关注，Spark 应用开发者可以专注于应用所要做的计算本身。其次，Spark 很快，支持交互式计算和复杂算法。最后，Spark 是一个通用引擎，可用它来完成各种各样的运算，包括 SQL 查询、文本处理、机器学习等，而在 Spark 出现之前，我们一般需要学习各种各样的引擎来分别处理这些需求。

* 1. Spark技术具体应用分析

Spark 使用了内存分布式数据集，除了能够提供交互式查询外，还优化了迭代工作负载，在 Spark SQL、 Spark Streaming、 MLlib、 GraphX 都有自己的子项目。在互联网领域， Spark在快速查询、实时日志采集处理、业务推荐、定制广告、用户图计算等方面都有相应的应用。国内的一些大公司，比如阿里巴巴、腾讯、 Intel、网易、科大讯飞、百分点科技等都有实际业务运行在 Spark 平台上。下面简要说明 Spark 在各个领域中的用途。快速查询系统，基于日志数据的快速查询系统业务构建于 Spark 之上，利用其快速查询以及内存表等优势，能够承担大部分日志数据的即时查询工作；在性能方面，普遍比 Hive 快 2 ～ 10 倍，如果使用内存表的功能，性能将会比 Hive 快百倍。实时日志采集处理，通过 Spark Streaming 实时进行业务日志采集，快速迭代处理，并进行综合分析，能够满足线上系统分析要求。业务推荐系统，使用 Spark 将业务推荐系统的小时和天级别的模型训练转变为分钟级别的模型训练，有效优化相关排名、个性化推荐以及热点点击分析等。定制广告系统，在定制广告业务方面需要大数据做应用分析、效果分析、定向优化等，借助 Spark 快速迭代的优势，实现了在“数据实时采集、算法实时训练、系统实时预测”的全流程实时并行高维算法，支持上亿的请求量处理；模拟广告投放计算效率高、延迟小，同 MapReduce 相比延迟至少降低一个数量级。用户图计算。利用 GraphX 解决了许多生产问题，包括以下计算场景：基于度分布的中枢节点发现、基于最大连通图的社区发现、基于三角形计数的关系衡量、基于随机游走的用户属性传播等。

1. Scala语言综述
   1. Scala语言定义

Scala是一门多范式的编程语言，一种类似java的编程语言，设计初衷是实现可伸缩的语言、并集成面向对象编程和函数式编程的各种特性。Scala有几项关键特性表明了它的面向对象的本质。例如，Scala中的每个值都是一个对象，包括基本数据类型（即布尔值、数字等）在内，连函数也是对象。另外，类可以被子类化，而且Scala还提供了基于mixin的组合（mixin-based composition）。

与只支持单继承的语言相比，Scala具有更广泛意义上的类重用。Scala允许定义新类的时候重用“一个类中新增的成员定义（即相较于其父类的差异之处）”。Scala称之为mixin类组合。

Scala还包含了若干函数式语言的关键概念，包括高阶函数（Higher-Order Function）、柯里化（Currying）、嵌套函数（Nested Function）、序列解读（Sequence Comprehensions）等等。

2.2 Scala语言的特点

Scala编程语言抓住了很多开发者的眼球。如果你粗略浏览Scala的网站，你会觉得Scala是一种纯粹的[面向对象](https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1)编程语言，而又无缝地结合了命令式编程和[函数式编程](https://baike.baidu.com/item/%E5%87%BD%E6%95%B0%E5%BC%8F%E7%BC%96%E7%A8%8B)风格。Christopher Diggins认为：

不太久之前编程语言还可以毫无疑意地归类成“命令式”或者“函数式”或者“面向对象”。Scala代表了一个新的语言品种，它抹平了这些人为划分的界限。

根据David Rupp在博客中的说法，Scala可能是下一代Java。这么高的评价让人不禁想看看它到底是什么东西。

Scala有几项关键特性表明了它的面向对象的本质。例如，Scala中的每个值都是一个对象，包括基本数据类型（即布尔值、数字等）在内，连函数也是对象。另外，类可以被子类化，而且Scala还提供了基于mixin的组合（mixin-based composition）。

与只支持单继承的语言相比，Scala具有更广泛意义上的类重用。Scala允许定义新类的时候重用“一个类中新增的成员定义（即相较于其父类的差异之处）”。Scala称之为mixin类组合。

Scala还包含了若干函数式语言的关键概念，包括高阶函数（Higher-Order Function）、柯里化（Currying）、[嵌套函数](https://baike.baidu.com/item/%E5%B5%8C%E5%A5%97%E5%87%BD%E6%95%B0)（Nested Function）、序列解读（Sequence Comprehensions）等等。

Scala的设计承认一个事实，即在实践中，某个领域特定的应用程序开发往往需要特定于该领域的语言扩展。Scala提供了许多独特的语言机制，可以以库的形式轻易无缝添加新的语言结构。

2.3 Scala与Groovy的对比

Scala和Groovy之间的核心区别在于前者是静态类型的。有些人可能争辩说这使得达到脚本化目标变得更加复杂了，而脚本化正是Groovy的动机。然而，Scala有完整的体系特征，这使Groovy看上去更像个玩具。比如，Scala有“sequence comprehensions”。该要素导致对算法的表述非常紧凑和强大。Scala还有更多被证明是非常有用的特性，如嵌套类，currying和代数类型模式匹配。它还支持类似于JDK1.5所增加的泛型和注解。这些还都只是冰山一角。

之后，Derek Young撰文“Scala对比Groovy：静态类型是性能的关键”。在文中他举了一个实际的例子，试图说明针对同样的算法，Scala的性能远高于Groovy。然而，Scala并不是尽善尽美的，它也有一些明显的缺陷。Rick Hightower在发表的一篇博客中，尖锐地批评了Scala的语法问题：Scala并不是更好的选择。在阅读了Scala的文档之后，我的想法是：虽然这种语言的特性听起来挺好，但是语法却让我想放弃。为什么事情非要为了不同而不同？Scala让Groovy看起来比以前更加美味可口。Scala和Groovy两种语言都在快速发展的过程中。就情况来看，Groovy的优势在于易用性以及与Java无缝衔接，Scala的优势在于性能和一些高级特性，如果在发展过程中两者能互相借鉴对方的优点来充实自身，对开发者来讲无疑是福音。大家并不想看到一场殊死斗争，而是想看到更注重实效思想的Groovy团队能与更具有学术思想的Scala团队一起合作，制作出一门既强大又易用的语言。

2.4 Scala的特性

[面向对象](https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1)特性

Scala是一种纯面向对象的语言，每一个值都是对象。对象的数据类型以及行为由类和特征(Trait)描述。类抽象机制的扩展有两种途径。一种途径是子类继承，另一种途径是灵活的混入（Mixin）机制。这两种途径能避免[多重继承](https://baike.baidu.com/item/%E5%A4%9A%E9%87%8D%E7%BB%A7%E6%89%BF)的种种问题。

函数式编程

Scala也是一种函数式语言，其函数也能当成值来使用。Scala提供了轻量级的语法用以定义[匿名函数](https://baike.baidu.com/item/%E5%8C%BF%E5%90%8D%E5%87%BD%E6%95%B0)，支持高阶函数，允许嵌套多层函数，并支持[柯里化](https://baike.baidu.com/item/%E6%9F%AF%E9%87%8C%E5%8C%96) 。Scala的Case Class及其内置的模式匹配相当于函数式编程语言中常用的代数类型（Algebraic Type）。

更进一步，程序员可以利用Scala的模式匹配，编写类似[正则表达式](https://baike.baidu.com/item/%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F)的代码处理XML数据。在这些情形中，顺序容器的推导式（comprehension）功能对编写公式化查询非常有用。

由于JVM不支持尾部[递归](https://baike.baidu.com/item/%E9%80%92%E5%BD%92)，Scala也不能完全支持尾部递归优化。不过，在简单的情况下，Scala[编译器](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E5%99%A8)可以把尾部递归优化成循环。

以下代码以函数式风格实现了[快速排序算法](https://baike.baidu.com/item/%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95)，可以与Erlang快速排序的例子做个比较：

def qsort(list: List[Int]): List[Int]=

list match{

case Nil => Nil

case pivot::tail =>

qsort(for(i <- tail if i < pivot)yield i)::: pivot :: qsort(for(i <- tail if i >= pivot)yield i)

}

静态类型

Scala是具备类型系统，通过编译时的检查，保证代码的安全性和一致性。类型系统具体支持以下特性：

[泛型类](https://baike.baidu.com/item/%E6%B3%9B%E5%9E%8B%E7%B1%BB)，型变注释（Variance Annotation），类型继承结构的上限和下限，把类别和抽象类型作为对象成员，复合类型，引用自己时显式指定类型，视图，[多态](https://baike.baidu.com/item/%E5%A4%9A%E6%80%81)方法。

扩展性

Scala的设计承认一个事实，即在实践中，某个领域特定的应用程序开发往往需要特定于该领域的语言扩展。Scala提供了许多独特的语言机制，可以以库的形式轻易无缝添加新的语言结构：

任何方法可用作前缀或后缀操作符，可以根据预期类型自动构造闭包。联合使用以上两个特性，使你可以定义新的语句而无须扩展语法也无须使用宏之类的[元编程](https://baike.baidu.com/item/%E5%85%83%E7%BC%96%E7%A8%8B)特性。 

使用Scala的框架

Lift是一个开源的[Web应用框架](https://baike.baidu.com/item/Web%E5%BA%94%E7%94%A8%E6%A1%86%E6%9E%B6/4262233)，旨在提供类似Ruby on Rails的东西。因为Lift使用了Scala，所以Lift应用程序可以使用所有的Java库和Web容器。

1. Spark电商用户行为的实现
   1. 集群环境搭建

修改hadoop配置文件

core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://node1:8020</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/home/hadoop/appsData/hdpData/tmp</value>

</property>

</configuration>

hdfs-site.xml

<configuration>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>/home/hadoop/appsData/hdpData/namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/home/hadoop/appsData/hdpData/datanode</value>

</property>

</configuration>

mapred-site.xml

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

<property>

<name>mapreduce.jobhistory.address</name>

<value>node1:10020</value>

</property>

<property>

<name>mapreduce.jobhistory.webapp.address</name>

<value>node1:19888</value>

</property>

</configuration>

yarn-site.xml

<configuration>

<!-- Site specific YARN configuration properties -->

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.resourcemanager.hostname</name>

<value>node1</value>

</property>

<property>

<name>yarn.log-aggregation-enable</name>

<value>true</value>

</property>

<property>

<name>yarn.log-aggregation.retain-seconds</name>

<value>640800</value>

</property>

<property>

<name>yarn.nodemanager.resource.memory-mb</name>

<value>4096</value>

</property>

<property>

<name>yarn.nodemanager.resource.cpu-vcores</name>

<value>4</value>

</property>

</configuration>

Hive配置

hive-site.xml

<configuration>

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://node1:3306/hivemetastore?createDatabaseIfNotExist=true&amp;useSSL=false</value>

<description>JDBC connect string for a JDBC metastore</description>

</property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

<value>com.mysql.jdbc.Driver</value>

<description>Driver class name for a JDBC metastore</description>

</property>

<property>

<name>javax.jdo.option.ConnectionUserName</name>

<value>root</value>

<description>username to use against metastore database</description>

</property>

<property>

<name>javax.jdo.option.ConnectionPassword</name>

<value>root</value>

<description>password to use against metastore database</description>

</property>

<property>

<name>hive.cli.print.header</name>

<value>true</value>

</property>

<property>

<name>hive.cli.print.current.db</name>

<value>true</value>

</property>

</configuration>

* 1. 基于IDEA新建Maven工程

pom 文件

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.spark.dianshangusers</groupId>

<artifactId>spark-dianshangusers-analyze</artifactId>

<version>1.0</version>

<name>spark-dianshangusers-analyze</name>

<!-- FIXME change it to the project's website -->

<url>http://www.example.com</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<scala.version>2.11.12</scala.version>

<spark.version>2.2.2</spark.version>

<kafka.version>0.9.0.0</kafka.version>

<hadoop.version>2.6.0-cdh5.7.0</hadoop.version>

</properties>

<repositories>

<!--添加库-->

<repository>

<id>cloudera</id>

<url>https://repository.cloudera.com/artifactory/cloudera-repos</url>

</repository>

</repositories>

<dependencies>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_2.11</artifactId>

<version>${spark.version}</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-sql\_2.11</artifactId>

<version>${spark.version}</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-hive\_2.11</artifactId>

<version>${spark.version}</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming\_2.11</artifactId>

<version>${spark.version}</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming-kafka-0-8\_2.11</artifactId>

<version>${spark.version}</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-client</artifactId>

<version>${hadoop.version}</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.47</version>

</dependency>

<dependency>

<groupId>org.json</groupId>

<artifactId>json</artifactId>

<version>20090211</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>2.4.3</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.4.3</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-annotations</artifactId>

<version>2.4.3</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.1.41</version>

</dependency>

<dependency>

<groupId>fastutil</groupId>

<artifactId>fastutil</artifactId>

<version>5.0.9</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<sourceDirectory>src/java</sourceDirectory>

<testSourceDirectory>src/test</testSourceDirectory>

<plugins>

<plugin>

<artifactId>maven-assembly-plugin</artifactId>

<configuration>

<descriptorRefs>

<descriptorRef>jar-with-dependencies</descriptorRef>

</descriptorRefs>

<archive>

<manifest>

<mainClass></mainClass>

</manifest>

</archive>

</configuration>

<executions>

<execution>

<id>make-assembly</id>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>1.2.1</version>

<executions>

<execution>

<goals>

<goal>exec</goal>

</goals>

</execution>

</executions>

<configuration>

<executable>java</executable>

<includeProjectDependencies>true</includeProjectDependencies>

<includePluginDependencies>false</includePluginDependencies>

<classpathScope>compile</classpathScope>

<mainClass>com.sparkproject.App</mainClass>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

<pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->

<plugins>

<!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean\_Lifecycle -->

<plugin>

<artifactId>maven-clean-plugin</artifactId>

<version>3.1.0</version>

</plugin>

<!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_jar\_packaging -->

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<version>3.0.2</version>

</plugin>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.0</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.1</version>

</plugin>

<plugin>

<artifactId>maven-jar-plugin</artifactId>

<version>3.0.2</version>

</plugin>

<plugin>

<artifactId>maven-install-plugin</artifactId>

<version>2.5.2</version>

</plugin>

<plugin>

<artifactId>maven-deploy-plugin</artifactId>

<version>2.8.2</version>

</plugin>

<!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site\_Lifecycle -->

<plugin>

<artifactId>maven-site-plugin</artifactId>

<version>3.7.1</version>

</plugin>

<plugin>

<artifactId>maven-project-info-reports-plugin</artifactId>

<version>3.0.0</version>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

* 1. Coding开发

测试数据库

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

/\*\*

\* @Description: JDBC 增删改查示范类

\*/

public class JDBCTest {

public static void main(String[] args) {

insert();

}

private static void insert() {

// 数据库连接对象

Connection conn = null;

// SQL 语句执行句柄：Statement 对象

// Statement 对象，其实就是底层基于Connection数据库连接

// 可以让使用者方便针对数据库中的表，执行增删改查的SQL语句

Statement stmt = null;

try {

// 第一步，记载数据库的驱动，面向 java.sql 包下接口的编程

// 要想让 JDBC 代码能够真正操作数据库，必须第一步先加载要操作的数据库

/\*

\* Class.forName() 是Java 提供的一种基于反射的方式，直接根据类的全限定名（包+类）

\* 从类所在的磁盘文件（.class文件）中加载类对应的内容，并创建对应的 Class 对象

\* \*/

Class.forName("com.mysql.jdbc.Driver");

// 获取数据库的连接

conn = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/spark\_project?useSSL=false",

"root",

"root");

// 基于数据库连接对象，创建 SQL语句执行句柄，Statement 对象

stmt = conn.createStatement();

String sql = "insert into test\_user(name,age) values('张三',18)";

// 返回值：SQL 语句影响的行数

int rtn = stmt.executeUpdate(sql);

System.out.println("SQL语句影响了 " + rtn + " 行" );

} catch (Exception e) {

e.printStackTrace();

}finally {

try{

if(stmt != null){

stmt.close();

}

if(conn != null){

conn.close();

}

}catch (Exception e){

e.printStackTrace();

}

}

}

1. SpringBoot后端框架
   1. SpringBoot介绍

每Spring框架是[Java平台](https://baike.baidu.com/item/Java%E5%B9%B3%E5%8F%B0/3793459)上的一种开源应用框架，提供具有控制反转特性的容器。尽管Spring框架自身对编程模型没有限制，但其在Java应用中的频繁使用让它备受青睐，以至于后来让它作为[EJB](https://baike.baidu.com/item/EJB/144195)（EnterpriseJavaBeans）模型的补充，甚至是替补。Spring框架为开发提供了一系列的解决方案，比如利用控制反转的核心特性，并通过依赖注入实现控制反转来实现管理对象生命周期容器化，利用面向切面编程进行声明式的事务管理，整合多种持久化技术管理数据访问，提供大量优秀的Web框架方便开发等等。Spring框架具有控制反转（IOC）特性，IOC旨在方便项目维护和测试，它提供了一种通过Java的反射机制对Java对象进行统一的配置和管理的方法。Spring框架利用容器管理对象的生命周期，容器可以通过扫描XML文件或类上特定Java注解来配置对象，开发者可以通过依赖查找或依赖注入来获得对象。Spring框架具有[面向切面编程](https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%88%87%E9%9D%A2%E7%BC%96%E7%A8%8B/6016335)（AOP）框架，SpringAOP框架基于代理模式，同时运行时可配置；AOP框架主要针对模块之间的交叉关注点进行模块化。Spring框架的AOP框架仅提供基本的AOP特性，虽无法与AspectJ框架相比，但通过与AspectJ的集成，也可以满足基本需求。

* 1. SpringBoot的特点

SpringBoot基于Spring4.0设计，不仅继承了Spring框架原有的优秀特性，而且还通过简化配置来进一步简化了Spring应用的整个搭建和开发过程。另外SpringBoot通过集成大量的框架使得依赖包的版本冲突，以及引用的不稳定性等问题得到了很好的解决。 

SpringBoot所具备的特征有：

（1）可以创建独立的[Spring](https://baike.baidu.com/item/Spring/85061)应用程序，并且基于其Maven或Gradle插件，可以创建可执行的JARs和WARs；

（2）内嵌Tomcat或Jetty等Servlet容器；

（3）提供自动配置的“starter”项目对象模型（POMS）以简化[Maven](https://baike.baidu.com/item/Maven/6094909)配置；

（4）尽可能自动配置Spring容器；

（5）提供准备好的特性，如指标、健康检查和外部化配置；

（6）绝对没有代码生成，不需要XML配置。

* 1. 为何选择SpringBoot

众所周知，后端框架有许多，比如说在之前学习javaweb时候的struts2，以及spring框架，还有python后端的Django框架，以及我们要用的SpringBoot框架。

既然有如此多的后端框架，为什么我们选择的是SpringBoot呢，下面将告诉你SpringBoot的优点。

SpringBoot的优点有：可以创建独立的Spring应用程序，并且基于其Maven或Gradle插件，可以创建可执行的JARs和WARs；内嵌Tomcat或Jetty等Servlet容器；提供自动配置的“starter”项目对象模型（POMS）以简化Maven配置；尽可能自动配置Spring容器；提供准备好的特性，如指标、健康检查和外部化配置；绝对没有代码生成，不需要XML配置。

* 1. Maven仓库

在我们之前学习JavaWeb的时候，当我们需要相关的jar包的时候我们就会去相关的网    站上把我们需要的包给下载下来，然后把需要用到的jar包放在我们的WebRoot目录下WEB-INF目录下的lib文件夹当中。这种方式我们要自己去找相关的依赖包，然后下载下来，放到指定的位置。这样显得比较麻烦，而且浪费时间。

但是有了Maven仓库之后，这一切就变得方便许多，我们只需要在pom.xml下写上相关的依赖，比如说我需要连接mysql数据库的jar包，那么，我们就需要去pom.xml写这个jar的名称，版本，写好了之后我们的maven工具就会自动的去下载我们需要的jar包，就不需要我们去一个一个找了，这样方便了许多。

第5章 总结与展望

Spark技术广泛的应用，但仍存在很大的开发空间，人们对这一领域的研究还存在很大不足。任何一项实用型技术都是在实践中不断革新，在实践中带动技术的进一步发展。总之，以Spark技术的产业革命已经兴起。这个新的经济单元将会是技术比拼的时代，技术壁垒也会越积越高。技术研发会成为企业发展的主要因素，同时商业模式也会因为技术的不断革新而不断演变。以后的技术革命中会衍生更多的新兴技术，而Spark技术的直接导火索，中国也将因为这些技术而真正步入“中国制造”时代。

参考文献

[1]张安站.Spark技术内幕 [J].spark技术处理，2015，22（06）：76-77.

[2]徐云鹏.Spark技术初探 [J].数字技术与应用，2015，18（12）：224-226.

[3]刘峰波.大数据Spark技术研究 [J].Spark技术的分析平台，2000，22（07）：177-179.

[4]祝永志.基于Spark技术的ALS推荐算法的可扩放性研究 [J].电子技术与软件工程，2008，10（01）：15-18.

致谢

经过几个月的学习，我很感谢胡恒老师的辅导与同学的帮助鼓励下，我们互相进步与学习讨论，收获了很多。我们受到了指导老师胡恒老师的细心指导,胡老师严谨的指导态度与深厚的理论知识都让我受益非浅,从他身上我学到了很多的东西,无论是理论还是实践都使我们的知识有很大的提高.借此我们特提出感谢. 还有广大网友提供的知识分享。