主机游戏市场数据分析系统

作业报告

|  |  |
| --- | --- |
| 系别 | 软件工程 |
| 学生 | 陈飞炯 |
| 学号 | 201992274 |
| 班级 | 软测 1905 |
| 成绩 |  |

2022 年 1 月 3 日**《主机游戏市场数据分析系统》**

**一、系统概述**

近年来，随着社会文化产业的不断发展，人们电子娱乐需求越来越多，主机游戏市场越来越大，对主机游戏市场数据分析也愈发重要。互联网拥有几乎所有主机游戏的数据，是收集主机游戏市场数据的最佳渠道。同时，大数据技术的创新和应用，为人们对主机游戏市场数据进行大数据统计分析进一步提供了便利，让我们可以更清楚地统计和分析主机游戏市场数据。

本系统将利用MongoDB+Spark+Java Web技术开发主机游戏市场数据分析系统，用于对主机游戏市场数据分析系统分析并展示。

**图示, 示意图

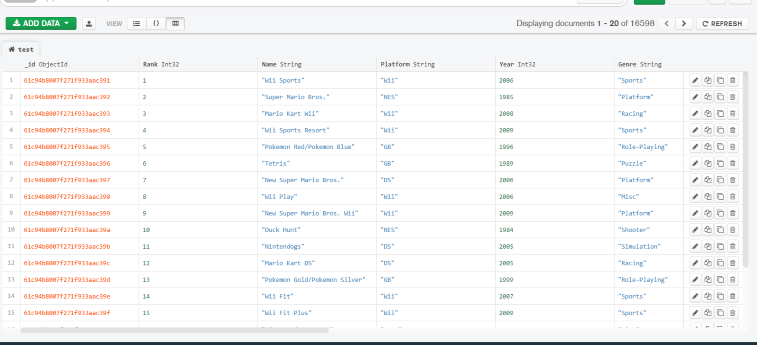
描述已自动生成二、系统架构**

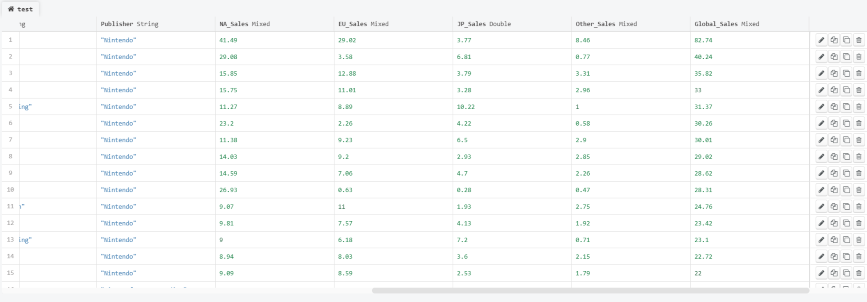
如图所示本系统所需的数据是来源于数据科学社区。采集对于主机游戏这块市场的游戏产量以及销售额度，选取了vgsales作为我们本次数据分析的数据集，并将采集的数据存储到MongoDB数据库中，采用Spark计算框架对采集的数据进行统计分析，将分析的数据存入到MongoDB中。为了可以更加直观的查看分析结果，我们通过Web系统获取MongoDB数据库中存储的分析结果生成多个Echarts图，实现数据的可视化。

**四、系统实现**

**1.** 数据采集模块

经过我们仔细的分析，认为对于当下的游戏市场的数据分析中，往往主机游戏的数据分析不够充分全面，人们往往只知道任天堂，PS等等一些已经众所周知的游戏厂商，但是对于主机游戏这块市场的游戏产量以及销售额度没有一个概念，于是我们本次的数据分析从[Kaggle：您的机器学习和数据科学社区](https://www.kaggle.com/)上选取了vgsales作为我们本次数据分析的数据集，并将数据集插入数据库表，共16598条数据进行分析。





**2．**数据分析模块

采用Spark计算框架对采集的数据进行统计分析，将分析的数据存入到MongoDB中。

1. 对于各大游戏平台的上发布游戏数量进行分析

import com.mongodb.spark.MongoSpark  
import org.apache.spark.rdd.RDD  
import org.apache.spark.sql.{DataFrame, SparkSession}  
import org.bson.Document  
  
object SparkToPlatform {  
  
 def main(args: Array[String]): Unit = {  
  
 //连接mongodb数据库，并指定访问的数据集合和写入的数据集合  
 val spark: SparkSession = SparkSession.*builder*().appName(getClass.getName).master("local[\*]")  
 .config("spark.mongodb.input.uri", "mongodb://localhost/work1.test")  
 .config("spark.mongodb.output.uri", "mongodb://localhost/dataset.GamePlatform")  
 .getOrCreate()  
 //加载访问数据库中的数据集合的数据，并转换成dateframe形式  
 import spark.implicits.\_  
 val df: DataFrame = MongoSpark.*load*(spark)  
 //实例化df,把它命名为vgsales,用于后续的集合名字  
 df.createOrReplaceTempView("vgsales")  
  
 //SQL  
 val resDf: DataFrame = spark.sql(  
 """  
 |select Platform ,count(\*)  
 |from vgsales  
 |group by Platform  
 |order by count(\*) DESC  
 |limit 10  
 """.stripMargin)  
  
 val resultRDD: RDD[(String, String)] = resDf.as[(String,String)].*rdd* //转化document对象  
 val docRDD: RDD[Document] = resultRDD.map(  
 x => {  
 val document = new Document()  
 document.append("Platform", x.\_1).append("sum", x.\_2)  
 document  
 }  
 )  
  
 //写入mongo  
 MongoSpark.*save*(docRDD)  
  
 }  
  
  
}

1. 对于各大游戏发行商的出产的游戏在北美，欧洲和日本的销售额进行统计

import com.mongodb.spark.MongoSpark  
import org.apache.spark.rdd.RDD  
import org.apache.spark.sql.{DataFrame, SparkSession}  
import org.bson.Document  
object SparkToGamesales {  
 def main(args: Array[String]): Unit = {  
 val spark: SparkSession = SparkSession.*builder*().appName(getClass.getName).master("local[\*]")  
 .config("spark.mongodb.input.uri", "mongodb://localhost/work1.test")  
 .config("spark.mongodb.output.uri", "mongodb://localhost/dataset.GAMESALES")  
 .getOrCreate()  
 import spark.implicits.\_  
 val df: DataFrame = MongoSpark.*load*(spark)  
 df.createOrReplaceTempView("vgsales")  
 //SQL  
 val resDf: DataFrame = spark.sql(  
 """  
 |select Publisher , sum(NA\_Sales),sum(EU\_Sales),sum(JP\_Sales)  
 |from vgsales  
 |group by Publisher  
 |order by sum(NA\_Sales) DESC  
 |limit 10  
 """.stripMargin)  
  
 val resultRDD: RDD[(String, String,String,String)] = resDf.as[(String,String,String,String)].*rdd* val docRDD: RDD[Document] = resultRDD.map(  
 x => {  
 val document = new Document()  
 document.append("Publisher", x.\_1).append("NA\_sales", x.\_2).append("EU\_sales", x.\_3).append("JP\_sales", x.\_4)  
 document  
 }  
 )  
  
 //写入mongo  
 MongoSpark.*save*(docRDD)  
  
 }  
  
  
}

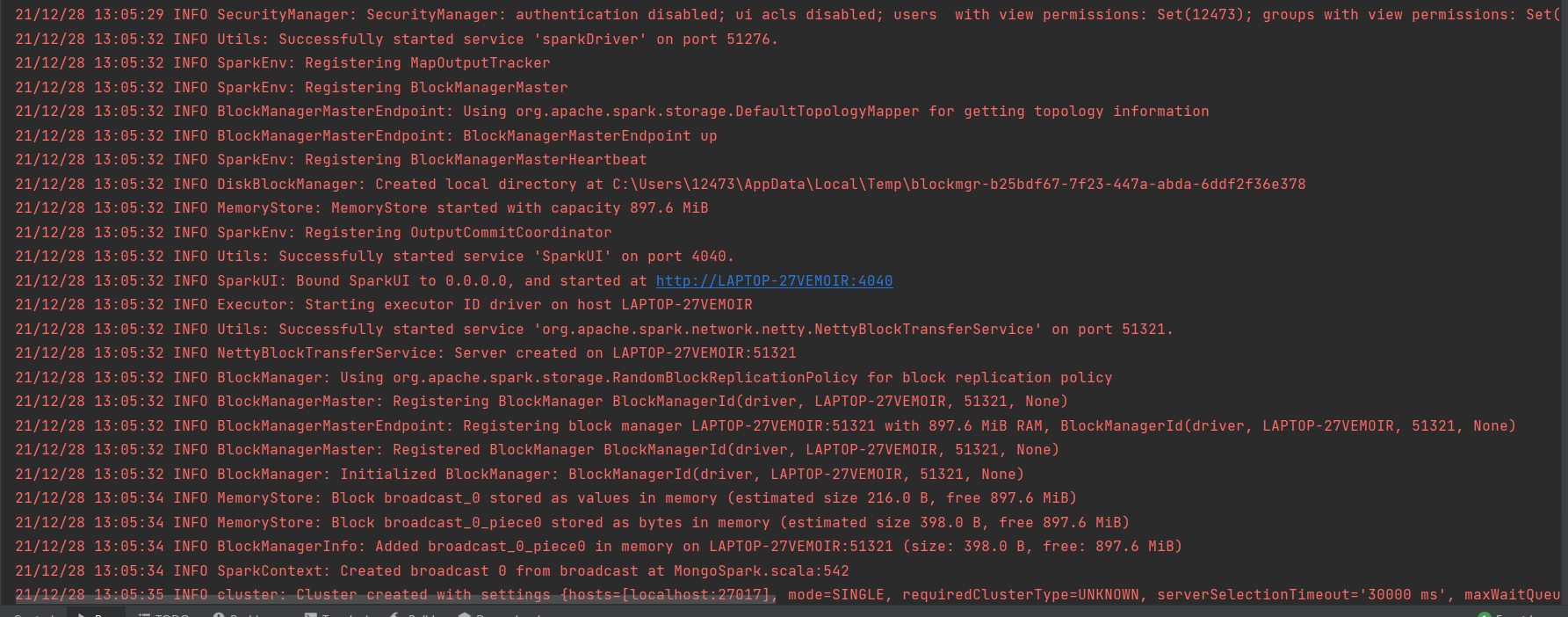
3.对于任天堂公司的每年的游戏发行数目进行统计

import com.mongodb.spark.MongoSpark  
import org.apache.spark.rdd.RDD  
import org.apache.spark.sql.{DataFrame, SparkSession}  
import org.bson.Document  
object SparkToNintendo {  
  
 def main(args: Array[String]): Unit = {  
  
 //连接mongodb数据库，并指定访问的数据集合和写入的数据集合  
 val spark: SparkSession = SparkSession.*builder*().appName(getClass.getName).master("local[\*]")  
 .config("spark.mongodb.input.uri", "mongodb://localhost/work1.test")  
 .config("spark.mongodb.output.uri", "mongodb://localhost/dataset.Nintendo")  
 .getOrCreate()  
 //加载访问数据库中的数据集合的数据，并转换成dateframe形式  
 import spark.implicits.\_  
 val df: DataFrame = MongoSpark.*load*(spark)  
 //实例化df,把它命名为vgsales,用于后续的集合名字  
 df.createOrReplaceTempView("vgsales")  
  
 //SQL语句进行数据查询  
 val resDf: DataFrame = spark.sql(  
 """  
 |select Year ,count(\*)  
 |from vgsales  
 |where Publisher = "Nintendo"  
 |group by Year  
 |order by Year DESC  
 """.stripMargin)  
 //通过rdd进行数据转换，将查询到的数据写会数据库  
 val resultRDD: RDD[(String, String)] = resDf.as[(String,String)].*rdd* val docRDD: RDD[Document] = resultRDD.map(  
 x => {  
 val document = new Document()  
 //设置数据属性  
 document.append("Year", x.\_1).append("count", x.\_2)  
 document  
 }  
 )  
 MongoSpark.*save*(docRDD)  
  
 }  
  
  
}

运行截图

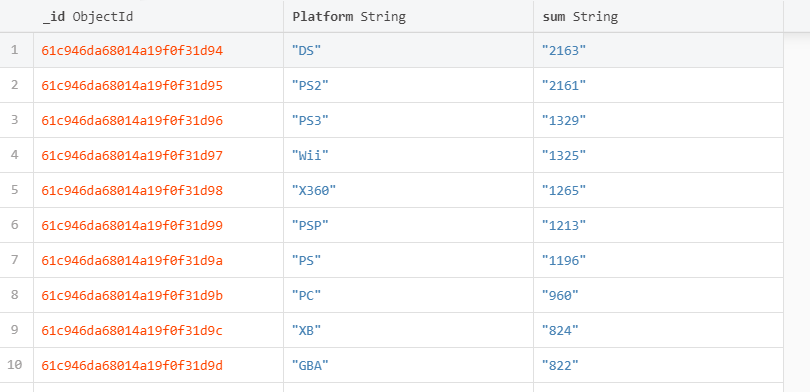
Intellij下运行截图

可在在此成功的启动了spark



分别在mongodb的数据中导入了数据

1.



2.



3.



**三、数据展示模块**

Index.html

<!DOCTYPE html>

<html>

<head>

    <meta charset="UTF-8">

    <title>游戏数据集</title>

    <link rel="icon" href="favicon.ico">

    <link rel="stylesheet" href="css/index.css">

    <script type="text/javascript" src="scripts/xlsx.core.min.js"></script>

    <script type="text/javascript" src="scripts/index.js"></script>

    <script type="text/javascript" src="scripts/echarts.min.js"></script>

</head>

<body>

<div id="header">

    <h1>销售额</h1>

    <span class="fileinput-button" id="btnLine">

        <span>上传表格</span>

        <input type="file" id="inputLine"

               accept="application/vnd.ms-excel,application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"

               onchange="excelToECharts(this)" onclick="this.value=''">

    </span>

    <span class="fileinput-button" id="btnPie">

        <span>制作饼图</span>

        <input type="file" id="inputPie"

               accept="application/vnd.ms-excel,application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"

               onchange="excelToECharts(this)" onclick="this.value=''">

    </span>

</div>

<!-- 为 ECharts 准备一个具备大小（宽高）的 DOM -->

<div id="ECharts\_main">

    <div id="select\_tip">上传表格</div>

</div>

</body>

</html>

Index.css

html, body {

    height: 100%;

    margin: 0;

}

/\*图表\*/

#ECharts\_main {

    width: 70%;

    height: 70%;

    border: 3px solid #565a58;

    margin: 5px auto;

    padding: 10px 0;

    position: relative;

}

#select\_tip {

    width: 300px;

    height: 70px;

    font-style: italic;

    font-size: 60px;

    font-weight: 200;

    color: #dbbcbc;

    border: 3px dotted;

    text-align: center;

    margin: auto;

    position: absolute;

    top: 0;

    left: 0;

    right: 0;

    bottom: 0;

}

#header {

    width: 70%;

    height: 100px;

    margin: 0px auto;

}

h1 {

    margin: 0px;

    padding: 10px 0px;

}

.fileinput-button {

    position: relative;

    display: inline-block;

    overflow: hidden;

    border: 1px solid #eaeaea;

    padding: 7px 10px;

    border-radius: 4px;

    background: #4d5b8b;

    color: #fff;

    font-weight: bold;

}

.fileinput-button input {

    position: absolute;

    right: 0;

    top: 0;

    opacity: 0;

    -ms-filter: 'alpha(opacity=0)';

    font-size: 150px;

}

.fileinput-button :hover {

    cursor: pointer;

}

Index.js

function excelToECharts(obj) {

    excelToData(obj);

}

// 读取Excel转换为json

function excelToData(obj) {

    let inputId = obj.id;

    let files = obj.files;

    if (files.length) {

        let reader = new FileReader();

        let file = files[0];

        // 看下文件格式是否为xls或xlsx

        let fullName = file.name;

        let filename = fullName.substring(0, fullName.lastIndexOf("."));    // 文件名

        let fixName = fullName.substring(fullName.lastIndexOf("."), fullName.length);   // 后缀名

        // 处理excel

        if (fixName == ".xls" || fixName == ".xlsx") {

            reader.onload = function (ev) {

                let data = ev.target.result;

                // 获取excel

                let excel = XLSX.read(data, {type: 'binary'});

                // 获取第一个标签页名

                let sheetName = excel.SheetNames[0];

                // 获取第一个标签页的内容

                let sheet = excel.Sheets[sheetName];

                // 转换为JSON

                let sheetJson = XLSX.utils.sheet\_to\_json(sheet);

                // 转换成json后,转换线图格式

                if (inputId == 'inputLine') {

                    // 线图

                    getLineChartFromJson(sheetJson, filename);

                } else if (inputId == 'inputPie') {

                    // 饼图

                    getPieChartFromJson(sheetJson, filename);

                }

            }

        } else {

            alert("只支持excel")

        }

        reader.readAsBinaryString(file);

    }

}

// 获取列名，返回列名的数组

function getColName(sheetJson) {

    // 遍历json的第一行从而得到key

    let keys = [];

    for (let key in sheetJson[0]) {

        keys.push(key)

    }

    return keys;

}

// 数据封装及显示

function getLineChartFromJson(sheetJson, filename) {

    // 如果有结果，处理结果

    if (sheetJson.length) {

        // 获取所有列名

        let keys = getColName(sheetJson);

        // 处理一下作为x轴的列名和数据

        let xZhou = {};

        xZhou.name = keys.splice(0, 1);

        let xDatas = [];

        for (let i in sheetJson) {

            xDatas.push(sheetJson[i][xZhou.name]);

        }

        xZhou.data = xDatas;

        // 主体数据

        let datas = [];

        for (let i in keys) {

            let one = {};       // 一组

            one.name = keys[i];

            one.type = 'line';  // 图表类型

            one.smooth = true;  // 平滑的线

            let point = [];     // 记录这一组的所有点

            for (let idx in sheetJson) {

                // 把这组的点push到数组中

                point.push(sheetJson[idx][one.name]);

            }

            one.data = point;

            // 把这组数据添加到主体数据中

            datas.push(one)

        }

        // 调用展现的方法

        dataToLineChart(filename, keys, xZhou, datas);

    }

}

// 数据可视化

function dataToLineChart(title, keys, xZhou, datas) {

    document.getElementById('ECharts\_main').removeAttribute('\_echarts\_instance\_');

    // 基于准备好的dom，初始化echarts实例

    var myChart = echarts.init(document.getElementById('ECharts\_main'));

    // 指定图表的配置项和数据

    var option = {

        title: {

            text: title,

            x: 'center',

            y: 'bottom'

        },

        tooltip: {

            trigger: 'axis'

        },

        legend: {

            data: keys,

            orient: 'vertical',

            x: 'right',

            y: 'center'

        },

        xAxis: xZhou,

        yAxis: {},

        series: datas,

        toolbox: {

            show: true,

            left: 'right',

            feature: {

                dataView: {},

                magicType: {

                    type: ['line', 'bar', 'stack', 'tiled']

                },

                saveAsImage: {}

            }

        }

    };

    // 显示图表。

    myChart.setOption(option);

}

// 饼图数据封装及显示

function getPieChartFromJson(sheetJson, filename) {

    let keys = getColName(sheetJson);

    let items = [];

    for (let i in sheetJson) {

        items.push(sheetJson[i][keys[0]]);

    }

    // 获取数据

    let sheetData = [];

    for (let i in sheetJson) {

        sheetData.push({'name': sheetJson[i][keys[0]], 'value': sheetJson[i][keys[1]]});

    }

    // 构造series要的数据

    let datas = {};

    datas.name = keys[0];

    datas.type = 'pie';

    datas.radius = '50%';

    datas.center = ['45%', '50%'];

    datas.data = sheetData;

    datas.itemStyle = {

        emphasis: {

            shadowBlur: 10,

            shadowOffsetX: 0,

            shadowColor: 'rgba(0, 0, 0, 0.5)'

        }

    };

    dataToPieChart(filename, items, datas);

}

// 饼图数据展现

function dataToPieChart(title, items, datas) {

    document.getElementById('ECharts\_main').innerHTML = "";

    document.getElementById('ECharts\_main').removeAttribute('\_echarts\_instance\_');

    // 基于准备好的dom，初始化echarts实例

    let myChart = echarts.init(document.getElementById('ECharts\_main'));

    // 指定图表的配置项和数据

    let option = {

        title: {

            text: title,

            x: 'center',

            y: 'bottom'

        },

        tooltip: {

            trigger: 'item',

            formatter: "{a} <br/>{b} : {c} ({d}%)"

        },

        legend: {

            type: 'scroll',

            orient: 'vertical',

            right: 10,

            top: 30,

            bottom: 20,

            data: items,

        },

        series: datas,

        toolbox: {

            show: true,

            left: 'right',

            feature: {

                dataView: {},

                magicType: {},

                saveAsImage: {}

            }

        }

    };

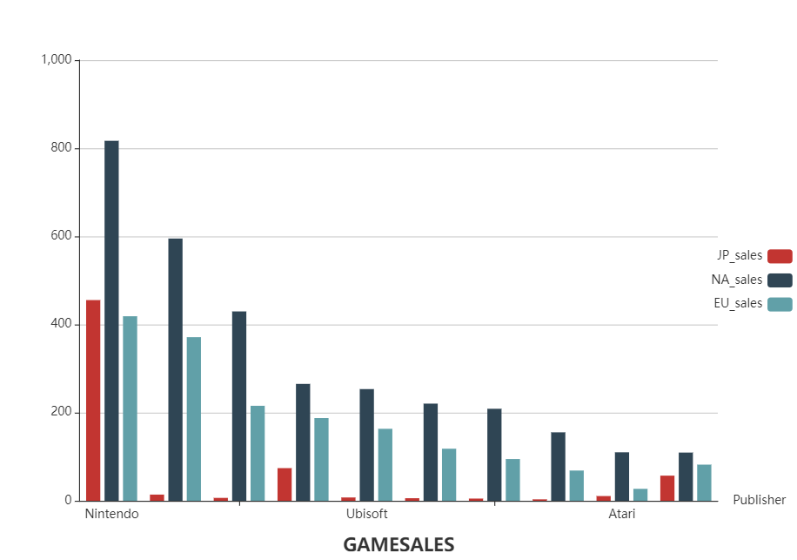
    // 使用刚指定的配置项和数据显示图表。

    myChart.setOption(option);

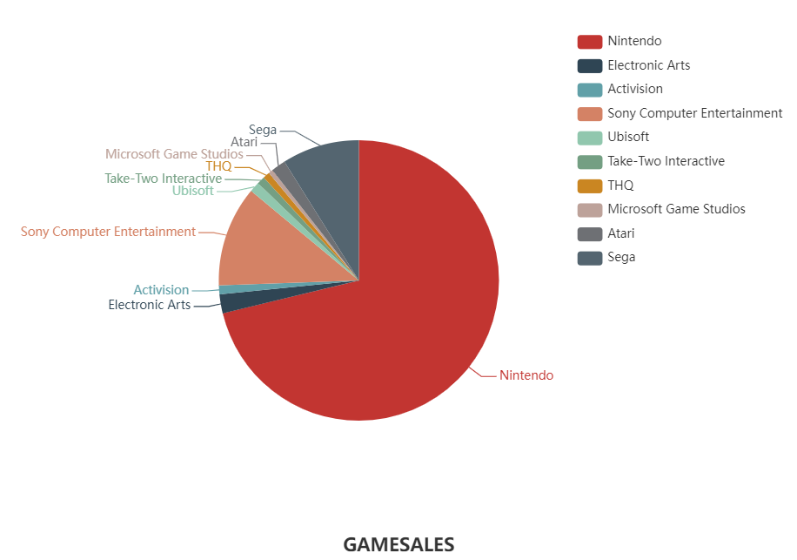
}

**五、系统运行结果**

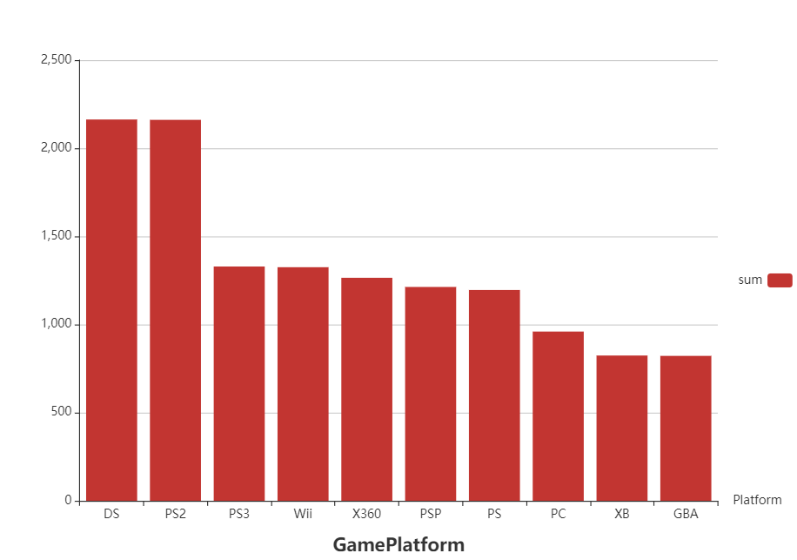
这个表是统计了各大游戏生产商其下所产的所有游戏的在北美，欧洲，日本的总销售额，

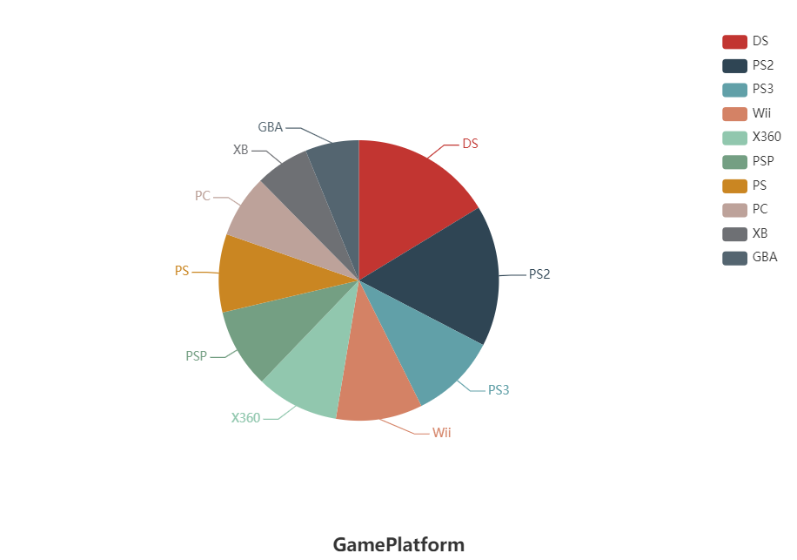


单位：3million



这个表是统计了各大游戏平台上所发布的游戏总数





统计了任天堂在每年发布的游戏数量

