# 可视化大屏课堂讲义

# 背景&目的

* 参考网站

<https://ark.analysys.cn/portal/industry-demo.html>

<https://tongji.baidu.com/web/welcome/login>

<https://www.umeng.com/?spm=a211g2.11755511.0.0.117d19e9EpW1I7>

* 案例一：

构建数据仓库，建设覆盖财务、采购、人力、猪/禽/鲜品/熟食零售等各产业全流程的经营分析体系，提供数据监控、预警和决策支持。

通过实时大屏、管理看板和固定报表等不同数据消费形式，满足从集团总裁到部门经理、业务线人员等各层级的日常经营管理数据分析需求，轻松掌握企业经营状况。

* 案例二:

通过构建数据仓库，建设覆盖各个产品线、店铺、品牌、物流、采购、支付等等泉流的经营分析体系，提供数据监控、预警和决策支持。

通过可视化实时大屏和固定报表等不同数据的消费形式，满足本企业不同级别人员的对各个业务线、部门日常经营管理数据分析需求，轻松掌握企业经营状况。

总体把握，第一公司领导决策层希望能能够更快，更准确的把握公司各个层面的运营数据情况；第二方便配合公司进行商务活动，丰富公司发言人的一些数据依据等待。

# 企业级的项目开发流程

项目：广义上说包含：产品+项目；狭义上说就是项目。

1. 项目立项

参与投标(外部进行投标，有可能也是在内部部分或者分公司之间进行投标)，公司内部产品开发。

项目就要成立。

1. 需求分析调研

直接到客户现场去和客户进行交流，询问客户有哪些需求，为了更加准确的记录客户的需求，可以通过视频，音频、图像、文字记录等待一系列方式记录需求。回来之后，就需要将这各个方面转化为书面的表达，最后形成需求说明文档。说白了需求说明文档，就是该项目的大体的设计。

1. 概要设计

概要设计主要就是对需求说明文档的理解，细化。

比如，相关模块的数据库如何设计，表有哪些字段、表与表之间的关联关系是怎样，核心模块使用怎样的算法来完成。

1. 详细设计

详细设计便是对概要设计的补充和具体化。涉及到各个模块之间的工作流程，各个模块间的关联系如何等待。

1. 编码阶段

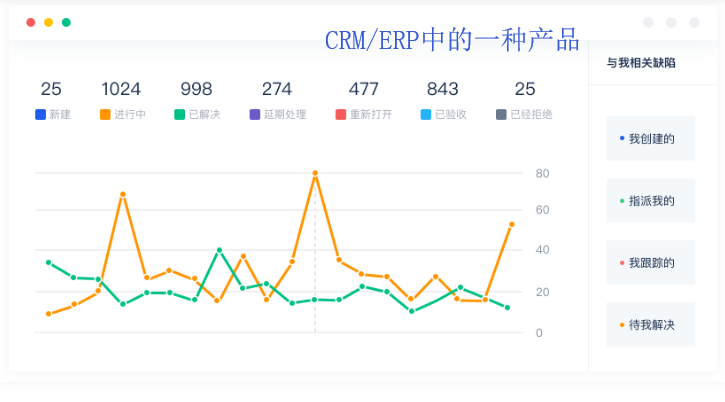
完成自己所负责的各个单元模块的开发工作，一般公司都是对详细设计和编码开发工作是交叉进行。加班。

1. 进行测试阶段

单元测试和项目联调。

单元测试就是自己负责自己开发的单元模块，通过代码测试来完成；项目组进行联调一般都是有专业的测试工程师进行测试。

以测试报告的方式进行呈现。将通过或者未通过的模块列表罗列出来，供各位程序员同事进行进行任务认领——buglist。



1. 项目上线

项目上线，必须要有严格的上线流程，部署说明流程，如果项目是交付给客户的，那么则必须要编写项目安装部署文档。

1. 线上运维

和各种各样的在测试中没有出现的bug进行打交道的过程，持续时间很长很长。

# 项目需求

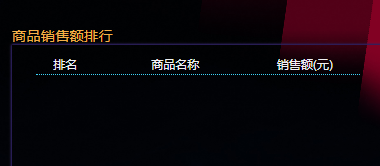


基本的项目实现功能：

## 商品交易额实时统计



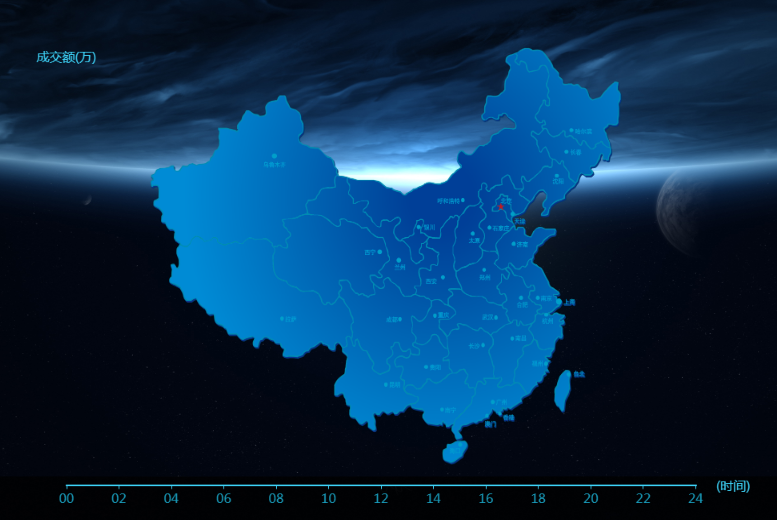
## 商品销售额排行榜



## 最受欢迎商品排行榜



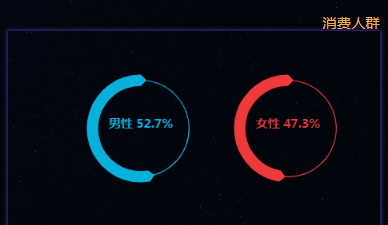
## 实时商品成交额统计分析



## 实时商品成交量统计分析



## 消费人群统计



## 消费区域统计

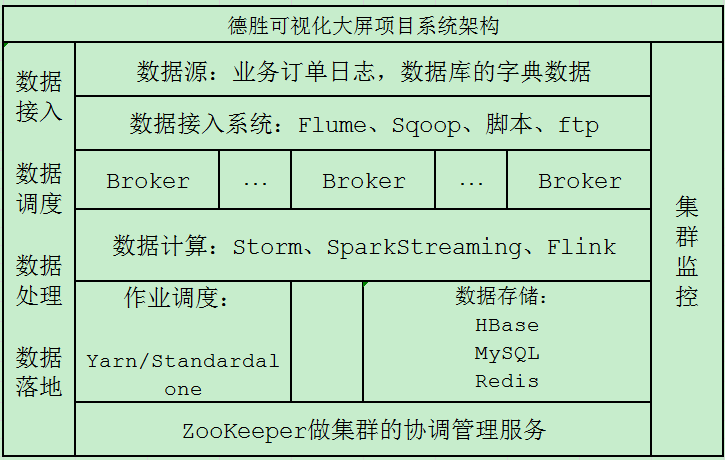


## 无线pc端流量，成交占比

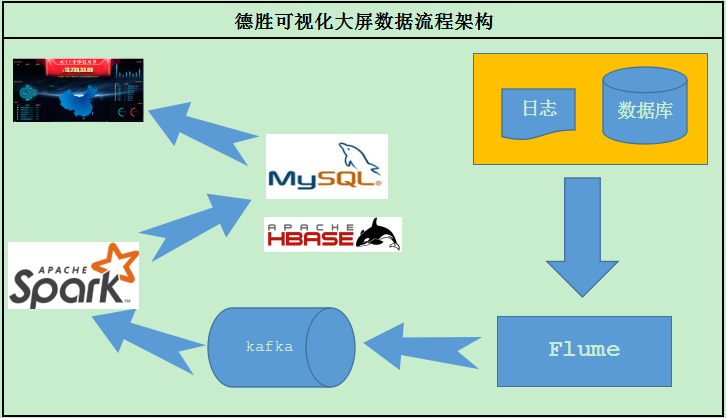
## 各品类各品牌成交额、成交量排行

分组排序TopN

# 项目系统架构



# 项目数据流程

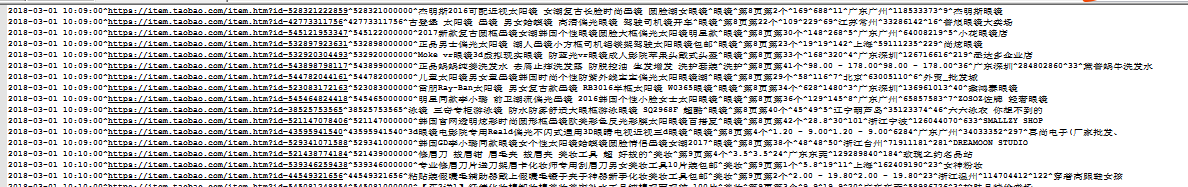


# 项目基础数据准备

## 服务列表



## 基本数据



## **数据结构**





## **项目开发使用的软件**

**SpringBoot**

**Spark**

**Maven**

**HDFS**

**Redis**

**Flume**

**Kafka**

**MySQL**

**Echarts**

**Yarn**

**CDH**

**需要大家注意的是：软件版本切不可使用最新版本。**

# 编码过程

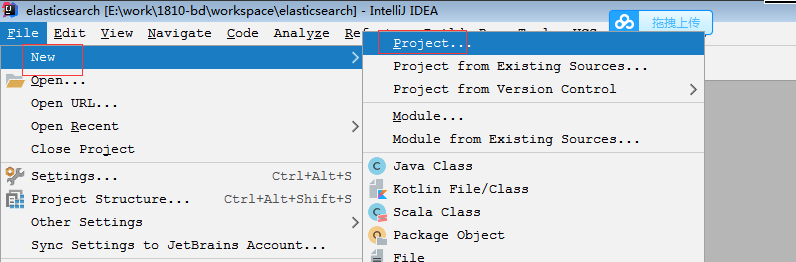
## 项目工程的构建

SpringBoot+Maven+MySQL+HBase+Kafka+Flume+Spark+Redis等等。

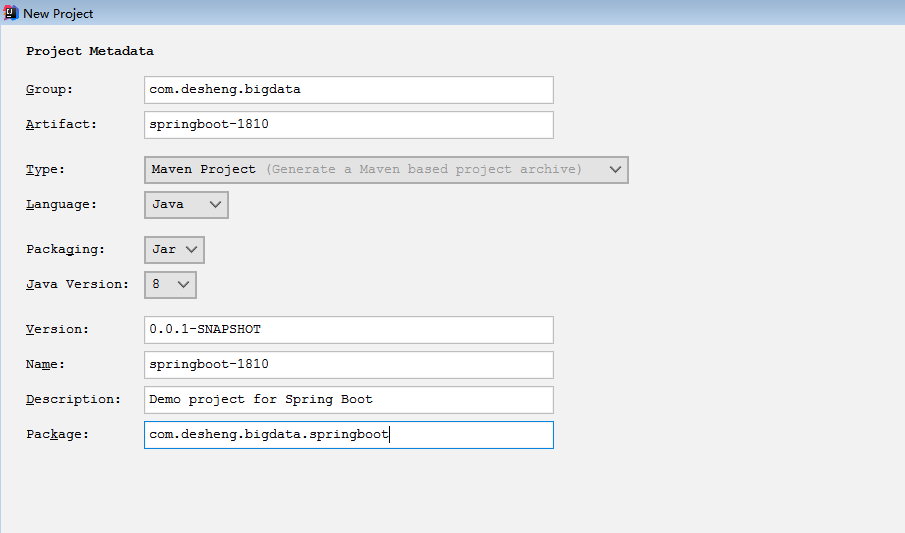
项目构建使用maven的来进行管理，是maven的模块化开发。

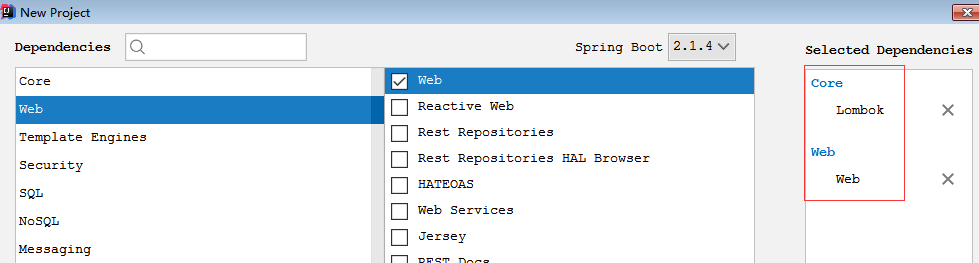
使用Scala进行spark作业的计算，使用SpringBoot进行前端界面管理。

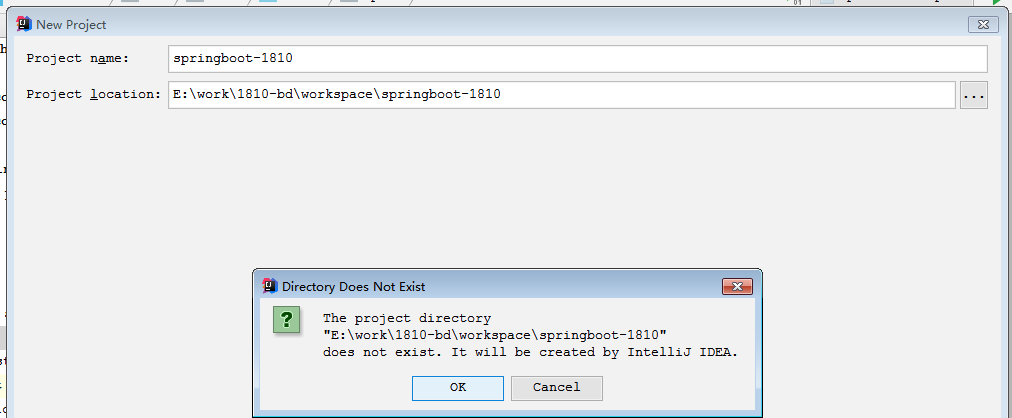
### Idea创建SpringBoot项目











### 本可视化的项目工程

通用模块---Common

计算模块---calc

展示模块---springboot+js

====>maven的继承和聚合

## 数据采集

### 使用java程序模拟用户点击

#### Log4j的配置文件

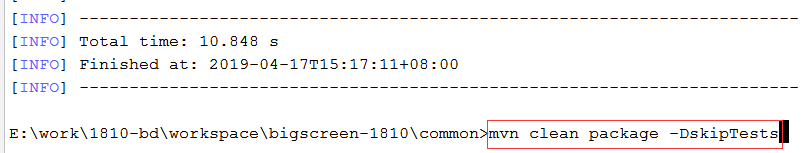
*##自定义日志的输出级别***log4j.rootLogger**=**INFO, stdout***##自定义日志***log4j.logger.access**=**INFO, access  
  
log4j.appender.stdout** = **org.apache.log4j.ConsoleAppender  
log4j.appender.stdout.Target** = **System.out  
log4j.appender.stdout.layout**=**org.apache.log4j.PatternLayout   
log4j.appender.stdout.layout.ConversionPattern**=**%d{yyyy-MM-dd HH:mm:ss} [%t] [%c] [%p] - %m%n***### 输出到日志文件 ###***log4j.appender.R** = **org.apache.log4j.DailyRollingFileAppender  
log4j.appender.R.File** = **logs/bigscreen.log  
log4j.appender.R.Append** = **true  
log4j.appender.R.Threshold** = **DEBUG  
log4j.appender.R.DatePattern** = **'.'yyyy-MM-dd  
log4j.appender.R.layout** = **org.apache.log4j.PatternLayout  
log4j.appender.R.layout.ConversionPattern** = **%d{yyyy-MM-dd HH:mm:ss} [%t] [%c] [%p] - %m%n***### 业务日志###***log4j.appender.access** = **org.apache.log4j.FileAppender  
log4j.appender.access.File** = **/home/bigdata/data/projects/bigscreen/1810-bd/access.log  
log4j.appender.access.Append** = **true  
log4j.appender.access.layout** = **org.apache.log4j.PatternLayout  
log4j.appender.access.layout.ConversionPattern** = **%m%n**

#### 用户acess日志生成代码

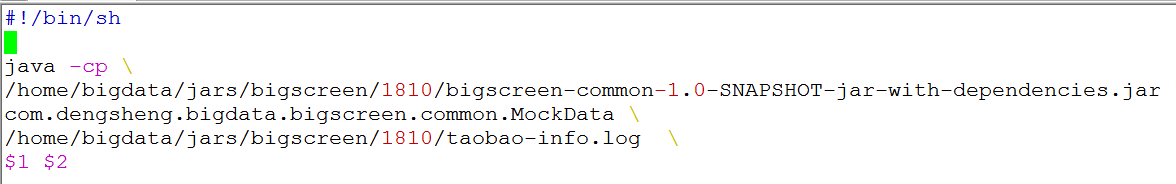
*/\*\*  
 \* 此类，用户模拟用户点击数据  
 \* 日志的方式-----log4j  
 \*  
 \* 配置文件  
 \* 第一次统计1， 5  
 \* 第二次统计5， 10  
 \* 分页操作  
 \* 假如每页M条记录  
 \* 读取第N条的记录  
 \*/***public class** MockData {  
 **private static** Logger *logger* = Logger.*getLogger*(**"access"**);  
 **public static void** main(String[] args) **throws** IOException {  
 **if**(args == **null** || args.**length** < 2) {  
 System.***err***.println(**"Parameter Errors! Usage: <page> <size>"**);  
 System.*exit*(-1);  
 }  
 **int** page = Integer.*valueOf*(args[0].trim());  
 **int** size = Integer.*valueOf*(args[1].trim());  
 LineNumberReader lnr = **new** LineNumberReader(**new** FileReader(**"common/data/taobao-info.log"**));  
  
 String line = **null**;  
 **int** start = (page - 1) \* size + 1;  
 **int** end = page \* size;  
 *//读取的数据在[start, start+size]* **while** ((line = lnr.readLine()) != **null**) {  
 *//拿到改行数据，通过log4j输出到指定的日志文件中* **int** lineNumber = lnr.getLineNumber();  
 **if**(lineNumber >= start && lineNumber <= end) {  
 *logger*.info(line);  
 } **else if**(lineNumber > end) {  
 **break**;  
 }  
 }  
 lnr.close();  
 }  
}

#### 打包上传到服务器运行

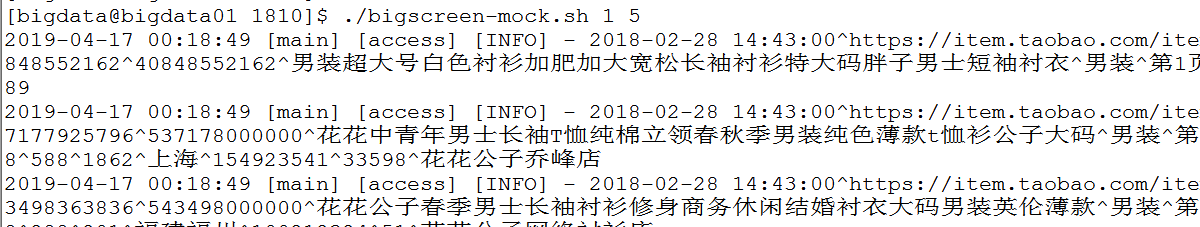
编译打包：



运行脚本：



运行测试：



### 通过flume采集用户生成的日志数据

使用flume采集的配置文件：

a1.sources = r1

a1.sinks = k1

a1.channels = c1

# 监听日志中的新增数据

a1.sources.r1.type = exec

a1.sources.r1.command = tail -F /home/bigdata/data/projects/bigscreen/1810-bd/access.log

# Describe the sink

a1.sinks.k1.type = logger

# 使用file做channel临时存储

a1.channels.c1.type = file

a1.channels.c1.checkpointDir = /home/bigdata/logs/checkpointDir/bigscreen/chk

a1.channels.c1.dataDirs = /home/bigdata/logs/checkpointDir/bigscreen/data

a1.channels.c1.transactionCapacity = 1000

a1.channels.c1.capacity = 10000

# Bind the source and sink to the channel

a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1

启动命令：

[bigdata@bigdata01 flume]$ bin/flume-ng agent -n a1 -c conf -f conf/flume-kafka-sink-1810.conf -Dflume.root.logger=INFO,console

### 通过kafka进行数据缓存

#### **创建topic：**

kafka-topics.sh --create --topic bigscreen-1810 --zookeeper bigdata01:2181/kafka --partitions 3 --replication-factor 3

#### Flume和kafka的整合

1. 修改配置文件

修改上述7.2.2中flume-kafka-sink-1810.conf 的sink部分，如下：

# Describe the sink

a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink

a1.sinks.k1.kafka.topic = bigscreen-1810

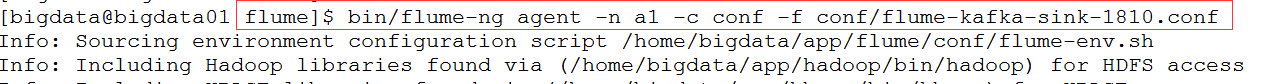
a1.sinks.k1.kafka.bootstrap.servers = bigdata01:9092,bigdata02:9092,bigdata03:9092

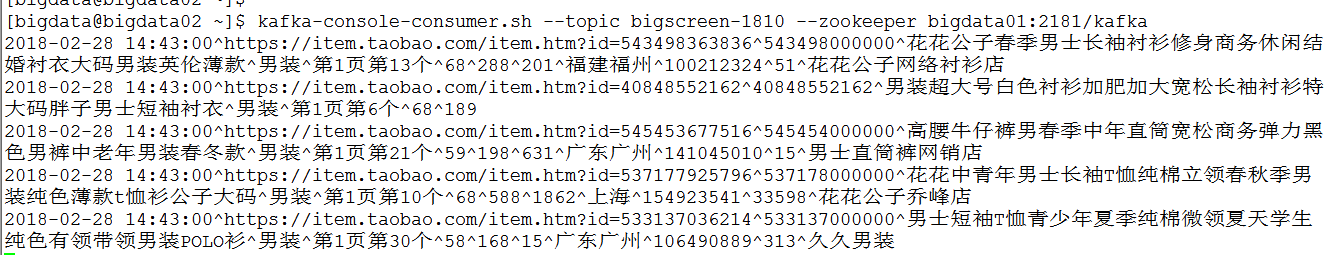
a1.sinks.k1.kafka.flumeBatchSize = 20

a1.sinks.k1.kafka.producer.acks = 1

a1.sinks.k1.kafka.producer.linger.ms = 1

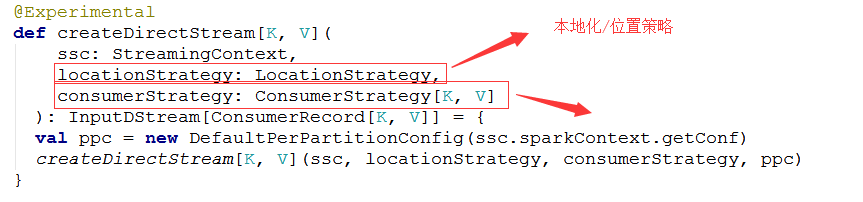
1. 测试





## 统计分析

### SparkStreaming和kafka的整合



#### 本地化策略

**LocationStrategy。**新版被API每次都会预先拉取kafka中的数据到缓冲区buffer中。因为相比较于在每个批次中都创建消费者，这种在每一个executor中缓存消费者对于提升作业的性能显得非常重要；为不同的机器中的partitition指定恰当的消费者去拉取数据。这种方式要比之前的api显得比较只能。

大多数情况下，我们选择**PreferConsistent**(优先、偏好一致性)即可。PreferConsistent的特点是，将kafka的分区partition，分发给可用的所有的executor。如果executor(计算)和kafka的分区(数据)都在同一台机器(之间是一一对应的在相关机器上，类似DataNode和NodeManager)，请选择**PreferBrokers**（偏好于节点的策略）。如果在加载数据的时候，会有一些倾斜现象，请选择**PreferFixed。**

在**locationStrategy这种情况下，最多可以缓存64个consumer**。如果想获取更多的缓存的consumer，可以通过配置参数spark.streaming.kafka.consumer.cache.maxCapacity来实现。

#### 消费者策略

**ConsumerStrategy**：选择如何在executor和driver上创建，以及配置consumer的相关策略。该接口接收两个泛型参数[K, V],其中K对应的是Kafka Message的Key，V对应的Message的value。

ConsumerStrategy，目前有三个实现的方式：

* Subscribe：就像0.8的版本中提供的订阅某些topic集合即可。
* SubscribePattern：通过对多个topic进行正则匹配来进行订阅。
* Assign：直接进行指派。

## 前端呈现

调试：

Springboot模块的运行方式

