大数据采集与清洗

# 数据

## 数据

### 数据和信息

数据的有序排列组成信息，数据是构成信息的基本单位

无需算法，有数据足矣；数据为王；数据是“未来的石油”

### 数据类型

文本、图片、音频、视频

### 数据组织形式

文件（文本网页视频音频……）、数据库

### 数据的价值（曾考）

为人们提供答案

数据的价值不会因使用而削减，反而会因不断重组产生更大的价值

数据应该尽可能地被长时间保存

在一定条件下与全社会分享，并产生价值

### 数据爆炸

数据的产生不以人的意志为转移

生活在数据爆炸（大数据时代的鲜明特征）的时代

## 数据分析



1. 采集与预处理：采用各种技术手段，把外部各种数据源产生的数据实时或非实时地采集、预处理并加以利用，是数据分析的第一步、大数据产业的基石
2. 数据存储与管理：利用计算机硬件和软件技术，对数据进行有效的存储和应用，其目的在于充分有效的发挥数据的作用
3. 数据处理与分析：用适当的分析方法（统计学、机器学习和数据挖掘等领域）对收集的数据进行分析，提取有用信息和形成结论
4. 可视化：将数据以图形图像形式表示，并利用数据分析和开发工具发现其中的未知信息

## 数据采集与预处理任务

### 数据采集

又称：数据获取，是数据分析的入口

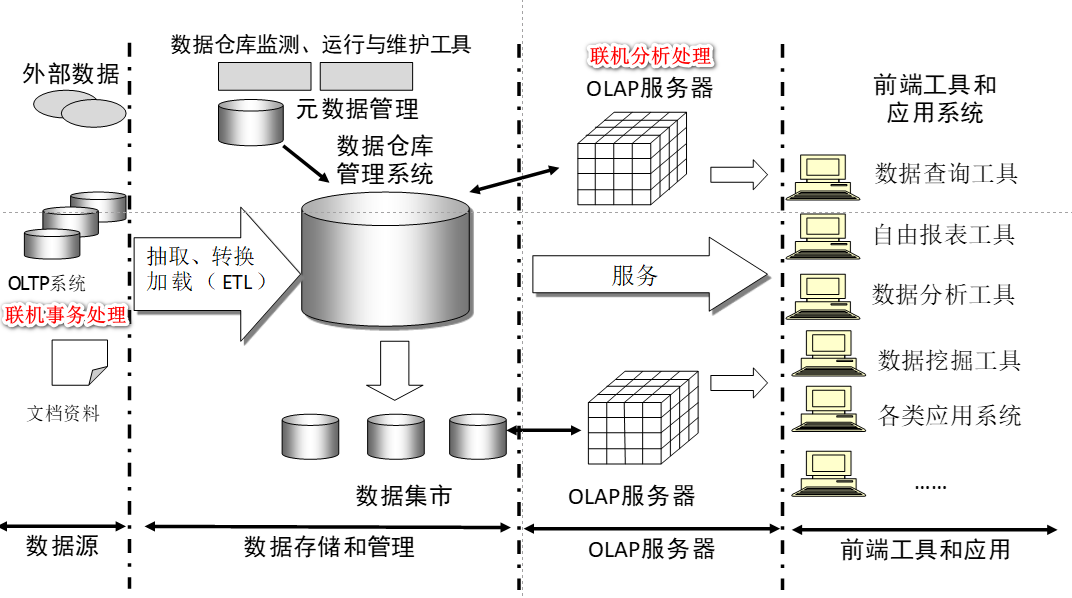
数据源：传感器和智能设备、企业在线/离线系统、社交网络、互联网平台

传感器数据来源：压力传感器、温度计、麦克风、DV录像、手机拍照、声音传感器

互联网数据来源：网络爬虫（抽取非结构化数据并以结构化方式存储为本地数据文件）

日志文件：数据源系统产生，有网络监控的流量管理、金融应用的股票记账、Web服务器记录的用户访问行为，为离线和在线的实时分析提供服务

企业业务系统数据：



数据类型：RFID数据、传感器数据、用户行为数据、社交网络交互数据、移动互联网数据等结构化、半结构化、非结构化数据

是大数据技术的重要组成部分

三大要点：全面性、多维性、高效性

|  |  |  |
| --- | --- | --- |
|  | 传统的数据采集 | 大数据采集 |
| **数据源** | 来源单一，数据量相对较少 | 来源广泛，数据量巨大 |
| **数据类型** | 结构单一 | 数据类型丰富（结构化、半结构化和非结构化） |
| **数据存储** | 关系数据库和并行数据仓库 | 分布式数据库，分布式文件系统 |

### 数据采集方法

数据采集是数据平台的根基

数据采集方法有：系统日志采集、分布式消息订阅分发、ETL、网络数据采集

### 数据预处理

目的：为后续数据提供可靠的和高质量的数据；减少数据集规模；提高数据抽象程度和数据分析效率

任务：降低源数据维度；去除噪声；提供干净、准确且有针对性的数据、减少数据分析算法的数据处理量；改进数据质量，提高分析效率



## 数据清洗

### 应用领域

数据仓库与数据挖掘：DW合并时或多源数据集成时消除二义性、去除冗余等等；DM消除特殊数据异常

数据质量管理：贯穿数据的整个周期并在发现有缺陷数据时及时将数据正确化和规范化

### 数据清洗的实现方式

手工清洗：效率低下，数据量大时不太可能

自动清洗：编写程序实现，不够灵活，没有充分利用数据库提供的强大的数据处理能力

### 数据清洗的内容

缺失值处理：估算、整例删除、变量删除、成对删除

异常值处理

数据类型转换

重复值处理

### 注意事项

优先进行缺失值、异常值和数据类型转换操作，最后进行重复值处理

处理方法要灵活，视情况而定

清洗前要了解表结构

数据量的大小也关系着数据的处理方式

逐列处理，保证数据处理的彻底性

### 基本流程

数据分析（前提和基础）；定义数据清洗的策略和规则；搜寻并确定错误实例；纠正发现的错误；干净数据回流

### 评价标准

数据的可信性：精确性、完整性、一致性、有效性、唯一性

数据的可用性：时间性、稳定性

数据清洗的代价

## 数据集成

将来自多个数据源的数据结合在一起，形成一个统一的数据集，为数据处理工作的顺利完成提供完整的数据基础

需要解决的问题：模式集成问题、冗余问题、数据值冲突检测与消除问题

## 数据转换

将数据进行转换或归并，从而构成合适数据处理的描述形式

### 转换策略

平滑处理、聚集处理、数据泛化处理、规范化处理、属性构造处理

平滑处理：去除数据中的噪声，分箱、回归和聚类等等

聚集处理：汇总操作

数据泛化处理：用更抽象的概念来取代低层次的数据对象

规范化处理：将属性值按比例缩放，使之落入特定的区间，如Min-Max规范化（0~1之间），Z-Score规范化（均值为0，方差为1）和小数定标规范化（将数据变化为-1~1）

属性构造处理：根据已有属性构造新的属性

### 分箱

要理解坐标轴，横轴为属性值大小，纵轴为箱子中的元素个数

等高：元素个数相同

等宽：取值间距相同

### 回归

利用拟合函数对数据进行平滑

### 聚类

通过聚类发现异常元素

### 规范化处理

Min-Max：x=(待转换属性值−属性最小值)/(属性最大值−属性最小值)

新加入的数值导致最大/最小值变化时，需要重新计算全部

Z-Score：z=（待转换属性值−属性平均值）/属性标准差（值越大越优秀）

不能表达数据的真实意义，最好是正态分布；不需要知道数据最大最小值，对离群点规范化效果好，不受数据量级影响

小数定标：x=待转换属性值/(10的k次方)

简单直观但没有消除属性间的权重差异

## 数据脱敏

“数据漂白”，抹去数据中的敏感内容

保持数据原有数据特征、保持数据之间的一致性、业务规则的关联性和数据一致性，保证开发、测试及业务不受脱敏影响，以达成脱敏前后数据的一致性和有效性

## 数据替换

设置固定虚构值

手机号码统一替换为13900010002

### 无效化

截断、加密、隐藏数据值，使其不再有利用价值

将地址替换为“\*\*\*\*\*\*”

### 随机化

用随机数据代替真值以模拟样本

用随机生成的姓和名代替真值

### 偏移和取整

通过随机位移改变数字数据

“2018-01-02 8:12:25”变为“2018-01-02 8:00:00”

### 掩码屏蔽

对账户类数据的部分信息进行脱敏

身份证“220524199209010254”换为“220524\*\*\*\*\*\*\*\*0254”

### 灵活编码

以满足各种脱敏规则

用固定字母和固定位数的数字替代合同编号真值

# 网络爬虫

## 定义

自动提取网页的程序



图表 1爬虫工作原理

## 爬虫类型

### 通用网络爬虫-全网爬虫

对象：从种子URL扩充到整个Web

使用者：门户网站搜索引擎、大型Web服务提供商采集数据

策略：深度优先、广度优先

[主题网络爬虫研究综述\_主题爬虫\_Violent-Ayang的博客-CSDN博客](https://blog.csdn.net/L6666688888/article/details/128226573)

### 聚焦网络爬虫-主题网络爬虫

工作流程：根据网页分析算法，过滤无关链接；保留有用链接并放置到URL；依据搜索策略，选择下一个爬取的URL；重复直到停止；网页被系统存储、分析、过滤并建立索引，以便查询和检索

特点：选择性爬取相关主题页面；节省了网络资源和硬件资源；保存页面少、更新快；满足特定人员对特定领域信息的需求

1. 策略：基于内容评价的爬行策略——Fish Search算法（将文本相似度计算方法引入网络爬虫，将用户输入的查询词作为主题；但无法评价页面与主题相关度的高低；改进算法Sharksearch用空间向量模型计算页面与主题的相关度）
2. 策略：基于链接结构评价的爬行策略——利用Web结构评价链接价值，Web页面为半结构化文档，包含很多结构信息，可用来评价链接的重要性——PageRank算法（对查询结果评价链接重要性，进行排序，每次选取PageRank较大的链接）、HITS算法（计算每个已访问页面的Authority权重和Hub权重，并以此决定链接的访问顺序）
3. 策略：基于增强学习的爬行策略——贝叶斯分类器（利用贝叶斯分类器，根据整个网页文本和链接文本对超链接进行分类，计算出每个链接的重要性，从而决定链接的访问顺序）
4. 策略：基于语境图（Context Graphs）的爬行策略——建立语境图学习网页间的相关度，去训练机器学习系统，通过该系统计算当前页面到相关Web页面的距离，距离越近越优先

IT和IBM：分类器和净化器

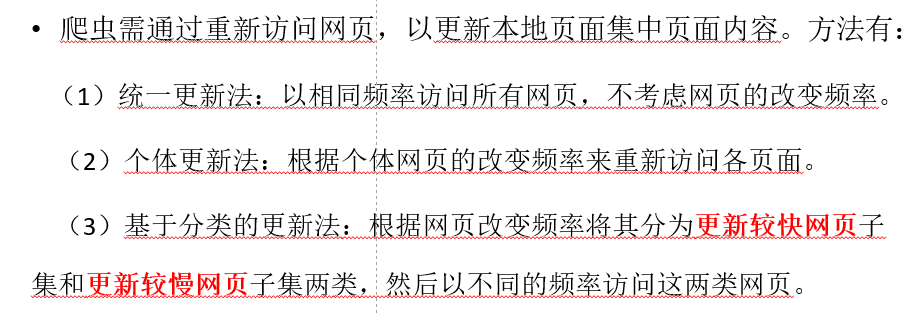
### 增量式网络爬虫

对已下载网页采取增量式更新，只爬取新产生或发生变化的网页。不抓取内容未变的网页，尽量保证爬取页面为尽可能新的页面；有效减少了网页的下载量，减少访问时间和存储时间的消耗；但增加了网页爬取算法的复杂度和实现难度

代表性增量爬虫：IBN开发的WebFountain；北京大学的天网增量爬虫

两个目标：

* 保持本地集中存储的页面为最新页面（统一更新法；个体更新法；基于分类的更新法）



图表 2增量式网页爬虫更新页面方法

* 提高本地页面集中页面的质量（对网页的重要性进行排序——广度优先策略、PageRank优先策略）

### 深层网络爬虫

Web按页面存在方式分为表层网页、深层网页（Deep Web）（从[www.baidu.com](http://www.baidu.com)输入关键字打开的网页就是深网）

[表网，深网，暗网有什么区别？ - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/356668588)

6个基本功能模块：爬行控制器、解析器、表单分析器、表单处理器、响应分析器、LVS控制器

2个爬虫内部数据结构：URL列表、LVS列表

## 反爬机制

原因：简单低级的网络爬虫，数据采集速度快、简单粗暴，请求过多会引起服务器异常，影响业务；企业不愿意自己的数据被免费获取

特定：可保护企业网站和网站数据；会误伤真正的用户请求

## 网页基础知识

[HTML 参考 - HTML（超文本标记语言） | MDN (mozilla.org)](https://developer.mozilla.org/zh-CN/docs/Web/HTML/Reference)

### 网页

基本组成：文本（展示信息的主体）、图片（对网页进行装饰）、超链接（万维网的灵魂）、表格表单音频视频等

### 超文本

使用超链接的方法，把文字和图片等信息相互联结，形成具有相关信息的体系

### 超文本标记语言

HTML（浏览器网页由HTML解析而成）

### HTTP：超文本传输协议

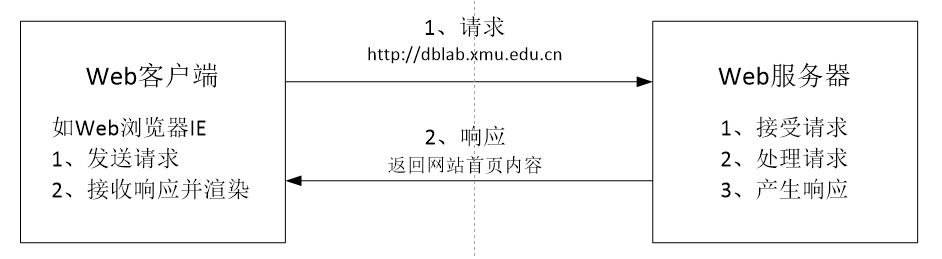
从网络传输超文本数据到本地浏览器的传送协议，能保证高效准的传达超文本内容

C/S通信架构：服务器端（httpd、nginx）和客户端（Firefox、Internet Explorer、Google Chrome、Safari、Opera）

只有get和post两种命令模式

[HTTP 方法：GET 对比 POST | 菜鸟教程 (runoob.com)](https://www.runoob.com/tags/html-httpmethods.html)

浏览器和服务器通过HTTP进行通信



图表 3一个典型的HTTP请求过程

## 用Python实现HTTP请求

要注意http和https的不同

[HTTP 与 HTTPS 的区别 | 菜鸟教程 (runoob.com)](https://www.runoob.com/w3cnote/http-vs-https.html)

### urlib模块

[爬取百度翻译 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/496595010#:~:text=1%E3%80%81%E9%A6%96%E5%85%88%E5%89%8D%E5%BE%80%20%E7%99%BE%E5%BA%A6%E7%BF%BB%E8%AF%91,%EF%BC%8C%E5%9C%A8%E4%B8%8B%E9%9D%A2%E8%BF%99%E4%B8%AA%E9%A1%B5%E9%9D%A2%E8%BE%93%E5%85%A5%E6%83%B3%E8%A6%81%E7%BF%BB%E8%AF%91%E7%9A%84%E6%96%87%E6%9C%AC%202%E3%80%81%E5%9C%A8%E8%BF%99%E4%B8%AA%E9%A1%B5%E9%9D%A2%E6%8C%89%E4%B8%8BF12%E6%89%93%E5%BC%80%E5%BC%80%E5%8F%91%E8%80%85%E9%A1%B5%E9%9D%A2%EF%BC%8C%E7%82%B9%E5%87%BB%E7%BD%91%E7%BB%9C%EF%BC%8C%E5%86%8D%E7%82%B9XHR%E8%BF%9B%E5%85%A5%E5%A6%82%E4%B8%8B%E9%A1%B5%E9%9D%A2%EF%BC%8C%E6%89%BE%E5%88%B0sug%E7%9A%84%E8%AF%B7%E6%B1%82%203%E3%80%81%E9%87%8C%E9%9D%A2%E7%9A%84%E8%AF%B7%E6%B1%82url%E5%B0%B1%E6%98%AF%E6%88%91%E4%BB%AC%E9%9C%80%E8%A6%81%E7%9A%84%EF%BC%8C%E8%AF%B7%E6%B1%82%E6%96%B9%E6%B3%95%E6%98%AFpost)

Python自带模块，提供了一个urlopen()方法，可向指定URL发生HTTP请求获取数据

|  |  |
| --- | --- |
| 模块名称 | 功能 |
| urllib.request | 定义了打开URL（主要是HTTP）的方法和类，如身份验证authenticaton、重定向 redirections 和cookie等 |
| urllib.error | 包含异常类，基本的异常类是URLError |
| urllib.parse | 定义URL解析和URL引用两类功能 |
| urllib.robotparser | 解析页面的robots.txt文件 |

图表 4urllib中的子模块

### urllib3模块

特性：

* 线程安全、连接池、客户端SSL/TLS验证
* 文件分部编码上传
* 协助处理重复请求和HTTP重定位
* 支持压缩编码
* 支持HTTP和SOCKS代理
* 100%测试覆盖率

### reuqests模块

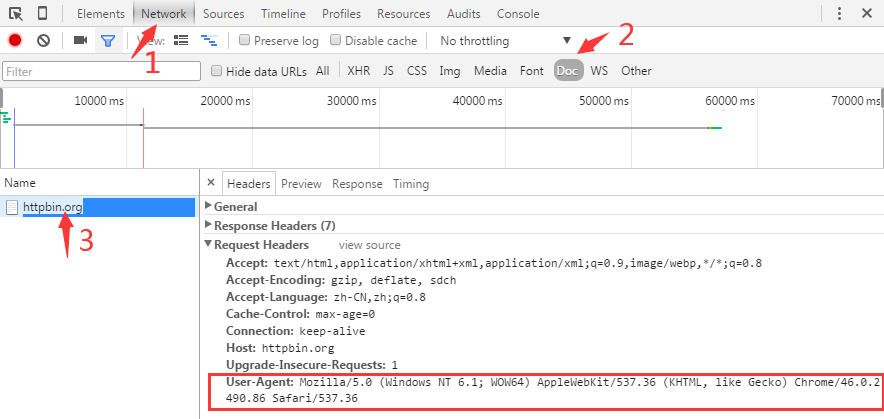
### 定制请求头Headers

爬取网页出现“抱歉，无法访问”等禁止爬取；没定制请求头或请求头和实际网页不一致，可能无法返回正确结果

请求头通过修改 User-Agent 来伪装浏览器，默认的 User-Agent 是 Python-urllib

获取headers

* 1. 浏览器打开一网页，在其上单击右键“查看元素”，然后刷新网页。
  2. 按下图“Network”->“Doc”->“Name”下方的网址，会出现类似如下的Headers信息：User-Agent:Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.86 Safari/537.36



图表 5查看请求头

### 网络超时

可为requests的timeout参数设定等待秒数，服务器若在指定时间内没有应答就返回异常

### 解析网页BeautifulSoup

为HTML/XML的解析器，其功能是解析和提取HTML/XML数据

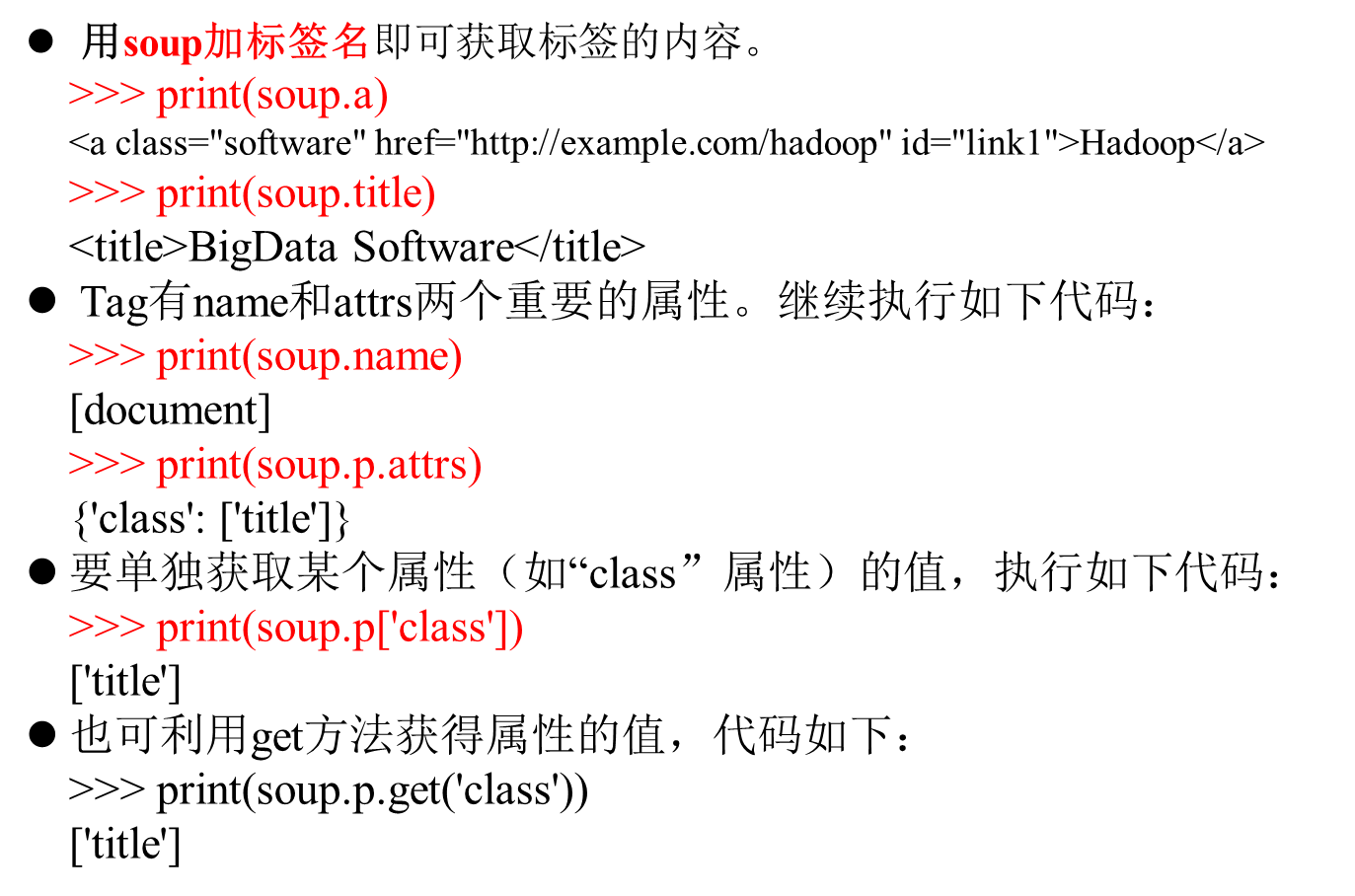
自动将输入文档转换为Unicode编码，输出文档转换为UTF-8编码

|  |  |  |  |
| --- | --- | --- | --- |
| 解析器 | 用法 | 优点 | 缺点 |
| Python标准库 | BeautifulSoup(markup,"html.parser") | Python标准库执行速度适中 | 文档容错能力差 |
| lxml的HTML解析器 | BeautifulSoup(markup,"lxml") | 速度快文档容错能力强 | 需安装C语言库 |
| lxml的XML解析器 | BeautifulSoup(markup, "lxml-xml") BeautifulSoup(markup,"xml") | 速度快唯一支持XML的解析器 | 需安装C语言库 |
| html5lib | BeautifulSoup(markup, "html5lib") | 兼容性好以浏览器的方式解析文档生成HTML5格式的文档 | 速度慢，不依赖外部扩展 |

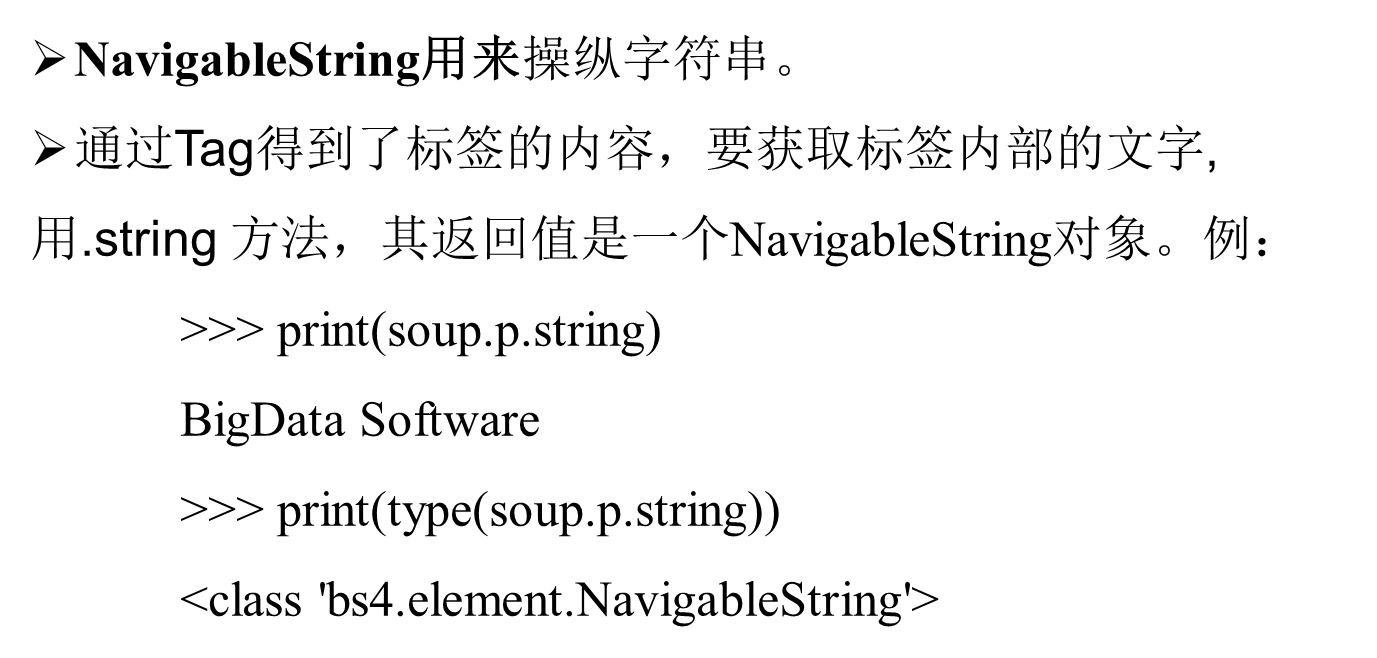
图表 6解析器的优缺点比较

四大对象：[beautifulsoup4教程（二）bs4中四大对象\_bs4.element.navigablestring-CSDN博客](https://blog.csdn.net/chinaltx/article/details/86748757)

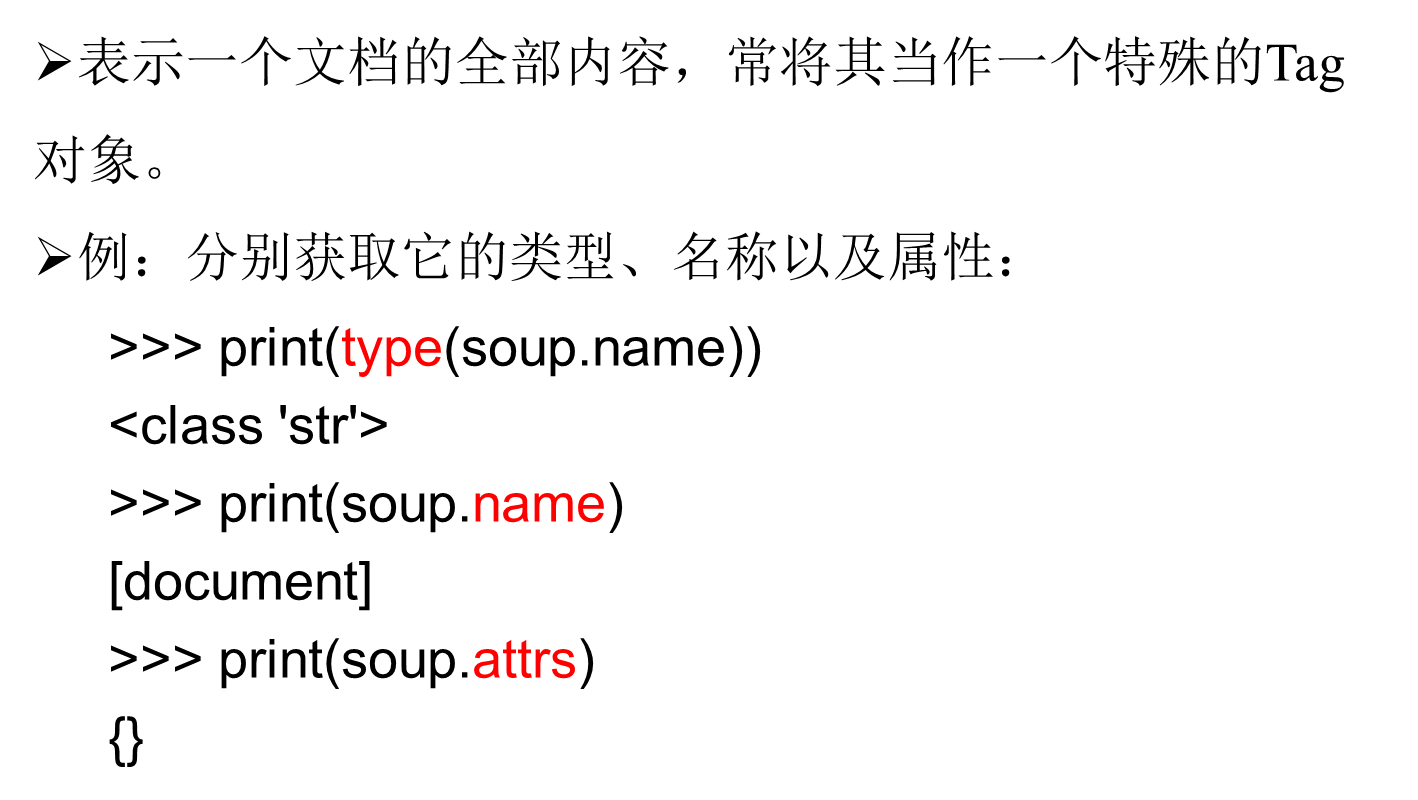
图表 7 Tag对象（HTML中的一个个标签）



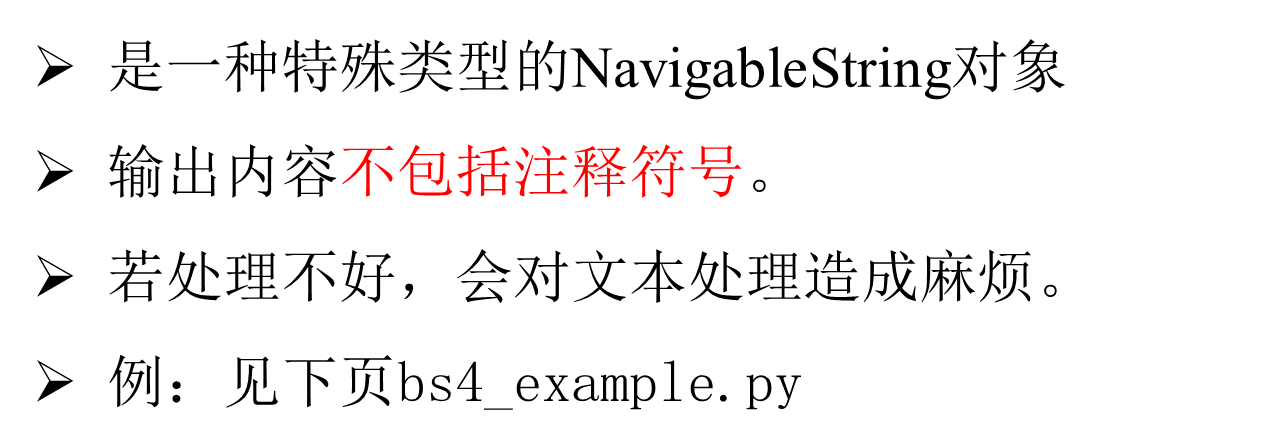
图表 8 NavigableString对象



图表 9 BeautifulSoup对象



图表 10 Comment对象



### 遍历文档树

直接子节点(.contents属性、.children属性)

所有子孙节点(.descendants属性)

节点内容(.string、.stripped\_strings)

直接父节点(.parent、.parents)

兄弟节点(.next\_sibling、.previous\_sibling)

全部兄弟节点(.next\_siblings、.previous\_siblings)

前后节点(.next\_element和.previous\_element属性，用于获得**不分层次**的前后元素)

所有前后节点(.next\_elements和.previous\_elements属性，用于获得**不分层次**的前后元素)

### 搜索文档树

通过指定标签名搜索元素，还可通过指定标签属性值精确定位某个节点元素。

方法为find()和find\_all()，两个方法在BeatifulSoup和Tag对象上均可被调用。

find\_all()搜索当前Tag所有子节点，并判断是否符合过滤器的条件,返回值是一个Tag组成的列表(字符串对象会被自动忽略掉)

[Python BeautifulSoup 中.text与.string的区别 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/30911642)

find()与find\_all()的区别：

1. find\_all()将所有匹配的条目组合成一个列表
2. find()仅返回第一个匹配的条目
3. 除此以外，二者的用法都相同

### CSS选择器

层叠样式表，用来表现html或xml等文本样式、可实现网页和内容分离的一种样式语言。

css样式定义组成：[code] 选择器{样式（作用对象）}[/code]

使用css可对html页面中的元素实现一对一、一对多或多对一的控制；HTML中的元素通过CSS选择器进行控制；选取设置样式的元素的模式

1. 使用soup.select()，返回类型是list，查找方式有：
2. 标签名查找：soup.select('xxx\_标签名')
3. 类名查找：soup.select('.xxx\_类名')
4. id名查找：soup.select('#xxx\_id名')
5. 组合查找，查找xxx\_标签名下的xxx\_类名，两者之间需用空格分开：soup.select('xxx\_标签名 .xxx\_类名')
6. 通过子标签查找：soup.select("head > title")
7. 属性查找：使用标签查找时加入属性元素，属性需要用中括号，属性和标签属于同一节点，所以不需要加空格
8. soup.select('xxx\_标签名[xxx\_属性="xxx"]')
9. soup.select('xxx\_标签名 xxx\_标签名[xxx\_属性="xxx"]')

选择器的选择准则：

1. 准确的选到要控制的标签；
2. 使用最合理优先级的选择器；
3. HTML 和 CSS 代码尽量简洁美观

## Scrapy爬虫框架

### 基本信息

纯Python实现、基于Twisted的异步处理框架

可用于数据挖掘、监测和自动化测试

### 特点

速度快、扩展性强、使用简便

可本地运行、可以部署云端，实现真正的生产级数据采集系统

### 体系架构

Item Pipeline（项目管道）：负责处理蜘蛛从网页中抽取的项目，其主要任务是清洗、验证和存储数据。pipelines.py

Item项目：定义了爬取结果的数据结构，爬取的数据会被赋值成该对象。items.py

Scheduler调度器：接受引擎发来的请求并加入队列中，在引擎再次请求的时提供给引擎。

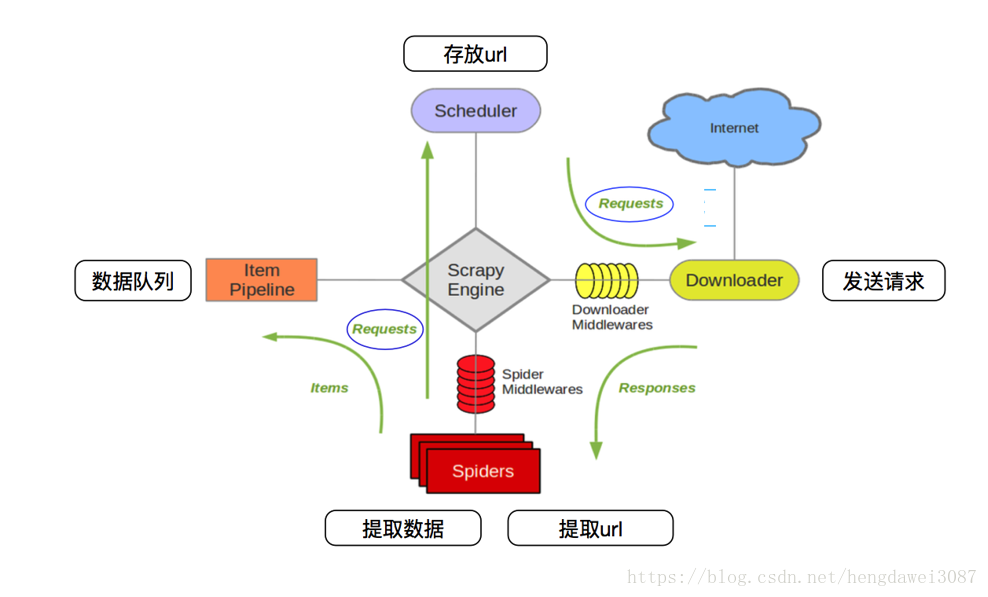
Engine引擎：处理系统的数据流处理、触发事务，是整个框架的核心。

Spider Middlewares（Spiders 中间件）：引擎和蜘蛛间的钩子框架，处理蜘蛛输入的响应和输出的结果及新的请求。middlewares.py

Spiders（蜘蛛）：定义了爬取的逻辑和网页的解析规则，负责解析响应并生成提取结果和新的请求。

Downloader Middlewares（下载器中间件）：引擎和下载器间的钩子框架，处理引擎与下载器之间的请求及响应。middlewares.py

Downloader（下载器）：下载网页内容，并将内容返回给 Spiders。



### Scrapy工作流

引擎从调度器取出URL用于下一步的抓取；

引擎把URL封装成一个请求，并传给下载器；

下载器下载资源，并封装成应答包；

爬虫解析应答包；

若解析出项目，则交给项目管道进一步处理；

若解析出URL，则把URL交给调度器等待抓取。

# Kafka

## 基本信息

高吞吐量的分布式消息订阅分发系统（发布、订阅、消费消息）

优点：分布式（易扩展）、高吞吐量（发布和订阅）、支持多订阅者、支持消息持久化到磁盘（批量消费）

Kafa是一个分布式、分区的、多副本的、多订阅者、基于Zookeeper协调的分布式日志系统

应用于日志收集系统和消息系统

以时间复杂度为O（1）的方式提供消息持久化能力

支持高吞吐量（低配机也支持10万条/秒）

### 组件

Broker：Kafka集群中的一个或多个服务器

Topic：每条消息都有一个类别，被称为Topic（物理上不同的Topic分开存储，逻辑上一个Topic的消息保存于一个或多个Broker上，用户不必关心数据存在何处，只需指定Topic）

Partition：物理上的概念，每个Topic包含一或多个Partition

Producer：负责发布消息到Broker

Consumer：消息消费者，从Broker读取消息的客户端

ConsumerGroup：每个Consumer属于一个Group，同一Topic的消息只能被统一Group中的一个Consumer消费，多个Group可同时消费一条消息

Zookeeper：管理集群配置

### 运行

Producer通过push模式将消息发布到Broker上，Counsumer通过pull模式从Broker订阅并消费消息



### 特性

高吞吐量、低延迟

可扩展性

持久性、可靠性

容错性

高并发

顺序保证

异步通信

日志收集

消息系统

用户活动跟踪

运营指标

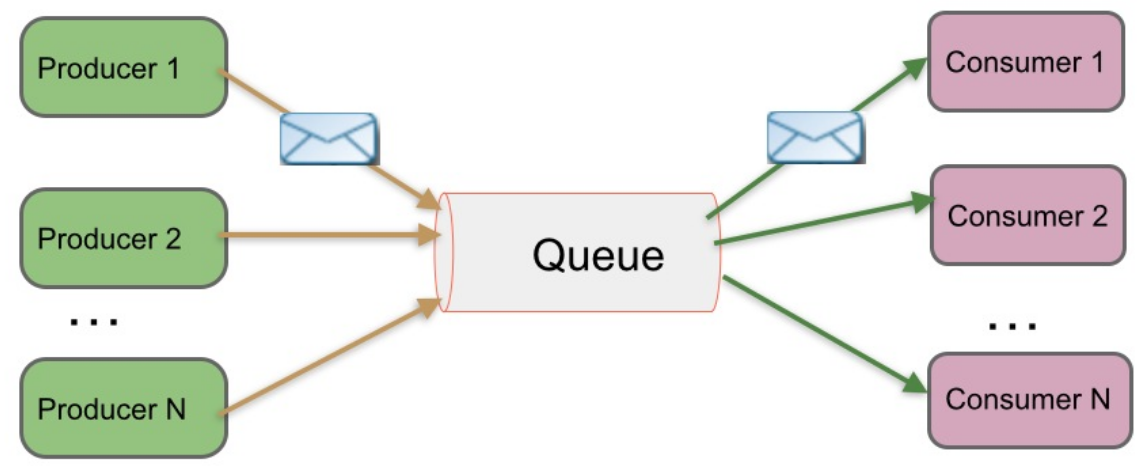
流式处理

### 消息传递模式

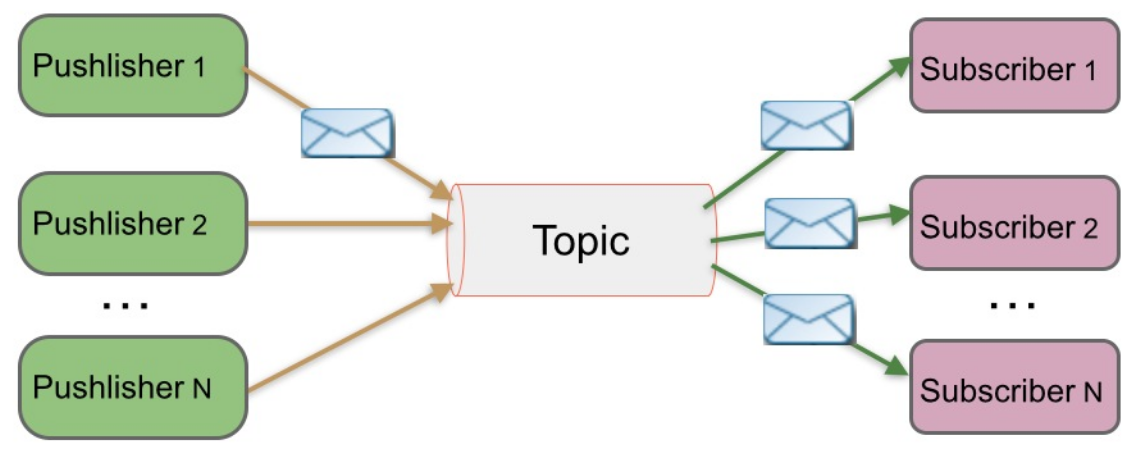
基于可靠的消息队列，在客户端和消息系统之间异步传递消息

点对点模式、发布订阅模式（Kafka是这种模式）

点对点：一条消息被消费一次；消息持久化到队列，将有一/多个消费者消费消息队列中的数据；可保证数据处理的顺序



发布订阅消息：消息被持久化到一个topic，消费者可订阅一/多个topic；消费者可消费topic中的所有数据，同一条数据可被多个消费者消费，数据被消费后不会立刻删除



## Kafka在大数据生态系统中的作用

可以解决不同系统之间的数据产生/消费速率不同的问题



## Kafka与Flume

Kafka是通用型系统，有许多的生产者和消费者分享多个主题，若数据被多个应用程序消费，推荐Kafka；Flume特定的向HDFS和HBase发送数据，为更好的为HDFS服务Flume做了特定优化，并且与Hadoop的安全体系进行整合，若数据只是面向Hadoop，推荐Flume

Kafka拥有非常小的生产者和消费者环境体系，如果需要准备自己的生产者和消费者，需要使用Kafka；Flume拥有许多配置的数据源和数据槽，若数据源确定，不许额外编码，则使用Flume提供的数据源和数据槽

Kafka需要一个外部系统帮助处理数据；Flume可在拦截器里实时处理数据，对于过滤数据非常有用

Kafka和Flume都可保证数据不丢失

Kafka和Flume可一起工作：Kafka是分布式消息中间件，自带存储，更适合做日志缓存；Flume数据采集很好，可作为日志采集；Flume把采集的日志发送到Kafka，再由Kafka把数据传送给Hadoop、Spark等消费者

# Flume

## 基本信息

Cloudera提供的一个高可用、高可靠、分布式的海量日志采集、聚合和传输系统

### 支持在日志系统中

定制各类数据发送方，用于收集数据

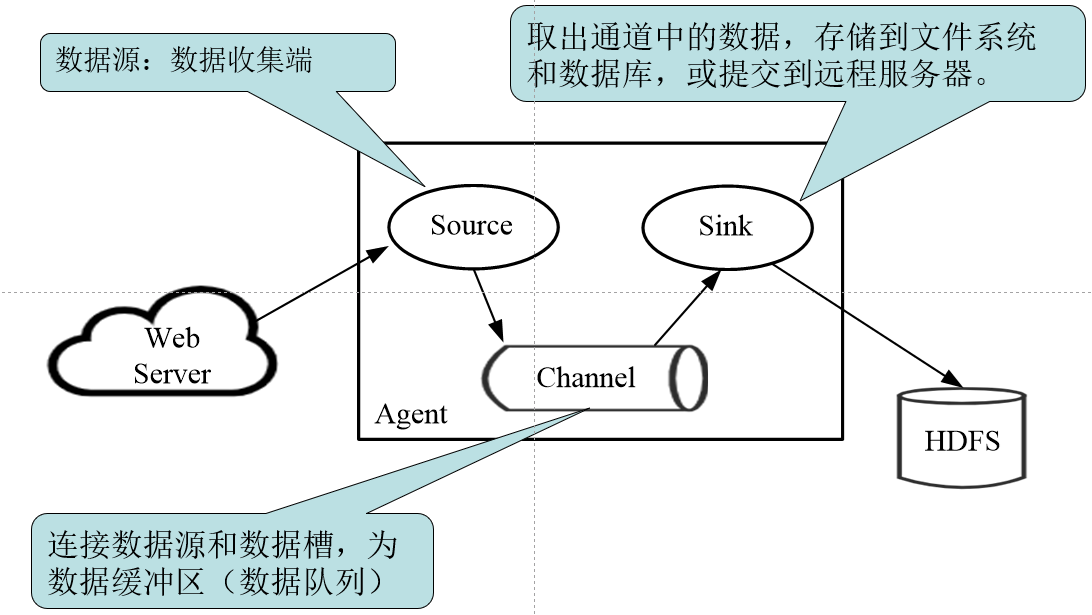
Flume提供对数据的简单处理

将处理结果写到各种数据接收方（可定制）

## 运行核心Agent

Flume以Agent为最小的独立运行单位

一个Agent是一个JVM，是一个完整的数据采集工具，包含三个核心组件（数据源Source、数据通道Channel、数据槽Sink）



### 数据源Source

数据收集端

将数据捕获后进行特殊的格式化；将数据封装到事件（Event）里；将数据推入到数据通道中

常用数据源类型：Avro、Thrift、Exec、JMS、Spooling Directory、Taildir、Kafka、NetCat、Syslog、HTTP等10类

### 数据通道Channel

连接数据源和数据槽的组件，可看作数据缓冲区（数据队列）

数据通道可把事件暂存内存里，也可持久化到本地硬盘上

将事件暂存到内存中；也可持久化到本地磁盘上，直到数据槽处理完事件

常用数据通道类型：Memory、JDBC、Kafka、File、Custom

### 数据槽Sink

取出数据通道中的数据；存储到文件系统和数据库；或提交到远程服务器

数据槽可把日志写入HDFS、Hbase，甚至是另外一个数据源等

常用数据槽：HDFS、Hive、Logger、Avro、Thrift、IRC、File Roll、HBase、ElasticSearch、Kafka、HTTP等11类

# 数据仓库

## 数据仓库

### 概念

面向主题的、集成的、相对稳定的、反应历史变化（不同时间的）的数据集合，用于支持管理决策（是一种管理技术）

传统DB用于事务处理（操作型处理）：对数据库联机进行日常操作；对一个（组）记录的查询和修改；主要为企业特定的应用服务（面向应用）；用户关心响应时间、数据的安全性和完整性

DW用于决策支持（分析性处理）：是建立决策支持系统的基础

大型DW是一个1000GB级数据库问题（需要巨大的硬件平台、一个并行的数据库系统）

### 面向主题

主题是数据归类的标准，可以设置很多主题

每个主题对应一个宏观的分析领域（按主题进行分类）

例如：银行DW的主题为客户，则DW的客户数据来源：从银行储蓄DB、信用卡DB、贷款DB。三个DB中抽取同一客户（主题）的数据整理而成，在DW中分析客户数据，可决定是否继续给予贷款

### 集成

数据进入仓库前必须加工和集成

数据来自于不同数据源，需要统一编码和数据结构；统一原始数据中的所有矛盾之处，如字段的同名异义、异名同异、单位不统一、字长不统一等等

实现原始数据结构从面向应用到面向主题的转变

### 稳定

大量的历史数据

数据进入仓库后，极少或根本不更新

### 随时间变化

数据仓库的数据时限在5-10年，数据键包含时间项

时间项标明数据的历史时期，以便DSS进行时间趋势分析

数据库只包含当前数据，即存取某一时间的正确有效数据

### DB与DW的区别

DB面向事务设计，DW面向主题设计

DB一般存储在线交易数据，DW一般存储历史数据

DB为捕获数据设计，DW为分析数据设计



### 五个发展阶段

报表阶段、分析阶段、预测阶段、实时决策阶段、主动决策阶段

实时主动数据仓库——支持实时主动决策

数据集成是数据仓库建设的关键部分

### 传统数据仓库



### 实时主动数据仓库

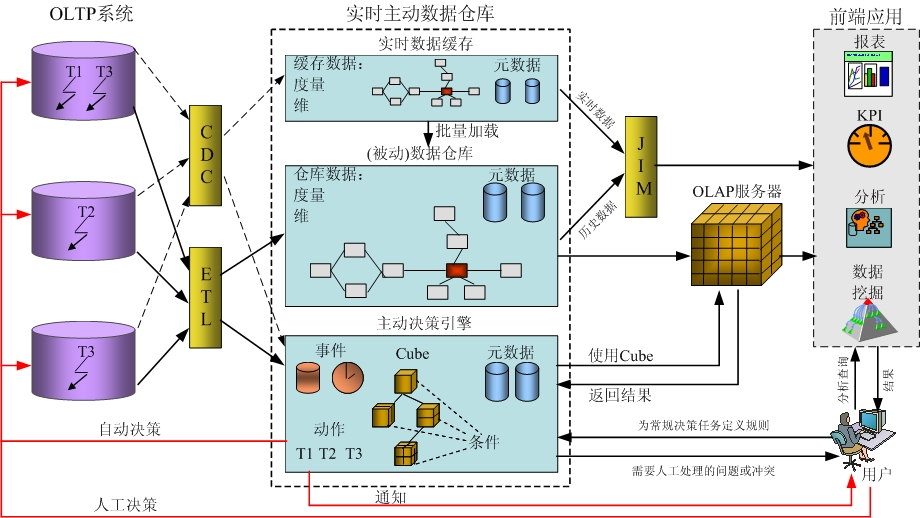
可使用针对传统数据仓库，开发数据集成技术（如脚本、ETL），完成数据批量加载

还使用实时的连续的数据集成技术，使数据源中的数据发生变化，及时反映到DW，保证为实时应用提供最新数据

是一个集成的信息存储仓库,具备批量和周期性的数据加载能力（ETL技术）

具备数据变化的实时探测、传播和加载能力（CDC技术）

能结合历史数据和新颖数据，实现查询分析和自动规则触发，以提供对战略决策和战术决策的双重支持



## 数据集成

### 数据集成方式

数据整合：利用ETL工具把数据源中的数据，批量的加载到数据仓库

数据联邦：在多数据源的基础上，建立统一逻辑视图，对外界应用屏蔽在各个数据源的分布细节（数据整合代价太大时，或为满足一些突发的实时数据需求时，数据联邦建立企业范围内的全局统一数据视图）

数据传播：数据在多个不同应用之间传播（不同应用通过传播消息进行交互，如EAI）

混合方法：不同应用都使用的数据以数据整合方式进行集成，只有特定应用才使用的数据通过数据联邦的方式进行集成

### 视图：

可为物理数据视图：数据来自多个分散的数据源，并被整合存储到一个集成的数据存储中如数据仓库

可为虚拟数据视图：数据分散在多个数据源，而不是集中存储在一个地方。当需要时，临时从多个数据源中抽取数据，并加以适当处理后，提交给数据请求者

### 数据分发方式

数据分发是数据集成过程的一个重要组成部分

数据分发方式：推（push）和拉（pull）；周期和非周期；一对一和一对多

（传统DW采用拉，操作系统负担重，实时DW多用推）

| **推、拉** | **周期、非周期** | **一对一**、**一对多** | **数据分发选择** |
| --- | --- | --- | --- |
| **拉** | 非周期 | 一对一 | 请求/响应 |
| 一对多 | 请求/探测式响应 |
| 周期 | 一对一 | 轮询 |
| 一对多 | 探测式轮询 |
| **推** | 非周期 | 一对一 | ------- |
| 一对多 | 发布/订阅 |
| 周期 | 一对一 | 发送电子邮件 |
| 一对多 | 电子邮件列表 |

### 数据集成技术

实时主动数据仓库的集成技术有：脚本、ETL、EAI、CDC

脚本：最常见的办法，是数据集成的一种快速解决方案，使用灵活且经济，容易开发和修改，几乎任何操作系统和DBMS都可以使用脚本，但开发费时费力，管理和操作不方便，不能满足服务水平协议（SLA）

ETL：实现大规模数据初步加载的理想解决方案，提供了高级的转换能力。ETL任务通常在“维护时间窗口”进行，在ETL任务执行期间，数据源默认不会发生变化，用户不必担心ETL任务对数据源的影响，但对商务用户而言，数据和应用并非任何时候都可用

EAI：实时数据获取和集成的解决方案，场合ETL解决方案并存，从而增强ETL的功能——数据传播方案。EAI在源系统和目标系统间进行连续的数据分发，且保证数据的分发成功，同时提供高级的工作流支持和基本的数据转换，但EAI受数据量的限制（EAI为实现应用集成而非数据集成，他是用来调用应用或分发命令和消息的；EAI在数据集成过程中，可实时分发数据和维护数据一致性，所以也能提供实时数据获取，正式实时主动数据仓库需要的）

CDC：提供连续变化数据的捕捉和分发能力，且只需很低的开销和时间延迟。CDC在提交的数据事务上进行操作，从OLTP系统中捕获变化的数据，再进行基本的转换，最后把数据发送到数据仓库；在体系上属于异步，但表现出类似同步，数据延迟只有不到1s的时间，同时可以维护数据事务的一致性

| **属性** | **脚本** | **ETL** | **EAI** | **CDC** |
| --- | --- | --- | --- | --- |
| **数据量** | 中等 | 很高 | 低 | 高 |
| **频率** | 间歇性 | 间歇性 | 连续性 | 连续性 |
| **延迟** | 中到高 | 中到高 | 低 | 低 |
| **数据集成** | 无 | 无 | 保证 | 保证 |
| **转换** | 中度 | 高级 | 基本 | 基本 |
| **处理开销** | 高 | 高 | 中等 | 低 |

## ETL

将业务系统的数据经过抽取（Extract）、清洗转换（Transform）后加载（Load）到数据仓库的过程

大规模数据初步加载的理想方案，提供了高级的转换能力

ETL任务通常在“维护时间窗口”进行，ETL任务执行期间，数据源默认不变，对商务用户，数据和应用并非任何时候都可用

ETL是企业DW构建过程的一个核心步骤

借助ETL可高效完成数据抽取、转换和加载工作

### 需要ETL的原因

（1）数据来自不同的物理主机，使用SQL语句处理较吃力且开销更大；

（2）需将多源数据整理为统一格式后进行数据处理，此过程用代码实现较麻烦；

（3）存储过程可以去处理数据，但处理海量数据时，存储过程占用较多数据库资源，从而导致数据库资源不足，进而影响数据库的性能

### 目标

将企业中分散、零乱、标准不统一的数据整合到一起，为企业决策提供分析依据

### 功能

早期：以固定的周期进行批处理工作，从平面数文件和数据关系库中捕捉数据并整合到数据仓库

当前拓展的功能有：额外的数据源、额外的目标、改进的数据转换功能、更好的管理、更好的性能、改进的可用性、增强的安全性、支持基于数据联邦的数据集成方法

### 基本模块：数据抽取、数据清洗与转换、数据加载



### 实现模式

触发器模式、增量字段、全量同步、日志对比

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **模式** | **兼容性** | **完备性** | **抽取**  **性能** | **源库**  **压力** | **源库改动量** | **实现**  **难度** |
| **触发器** | **关系型数据库** | **高** | **优** | **高** | **高** | **容易** |
| **增量**  **字段** | **关系型数据库，具有字段结构的其它数据格式** | **低** | **较优** | **低** | **高** | **容易** |
| **全表**  **同步** | **任何数据格式** | **高** | **极差** | **中** | **无** | **容易** |
| **日志**  **比对** | **关系型数据库** | **高** | **较优** | **中** | **中** | **难** |

### 触发器模式

常采取增量抽取机制

根据抽取要求，在被抽取的数据源中建立插入、修改、删除3个触发器，源表中的数据发生变化，相应触发器将变化的数据写入一个增量日志表，从增量日志表（不是源表）中抽取数据，增量日志表中抽取过的数据要及时标记或删除，增量日志表只存储源表名称、更新的关键字值和更新操作类型（insert、update或delete）

抽取进程的工作进程：根据源表名和更新的关键字值，从源表中提取对应的完整记录，再根据更新操作类型，对目标表进行相应的处理

优点：数据的抽取性能高，加载规则简单，速度快；不需要修改业务系统表结构，可实现数据的递增加载

缺点：要求业务表建立触发器，对业务系统有一定影响，容易对源数据库构成威胁

### 增量字段

在源系统业务数据表中增加增量字段（可为时间字段，也可为自增长字段），当源业务系统中数据新增或被修改时，增量字段就会变化，时间戳字段就会被修改为相应的系统时间，自增长字段就会增加

ETL工具获取增量数据时，只需比对最近一次数据抽取的增量字段值，即能判断新增数据和修改数据

优点：抽取性能高、判断过程简单、可实现数据的递增加载

局限：数据库设计时未考虑增量字段，需改造业务系统；可能出现漏数据

### 全量同步（全表删除插入方式）

抽取前先删除目标表数据，抽取时全新加载数据，适用于数据量不大且全量抽取代价小于执行增量抽取的算法和条件代价时

优点：不影响已有业务表结构；不需修改业务操作程序；所有抽取规则由ETL完成，管理维护统一、可是先数据的递增加载

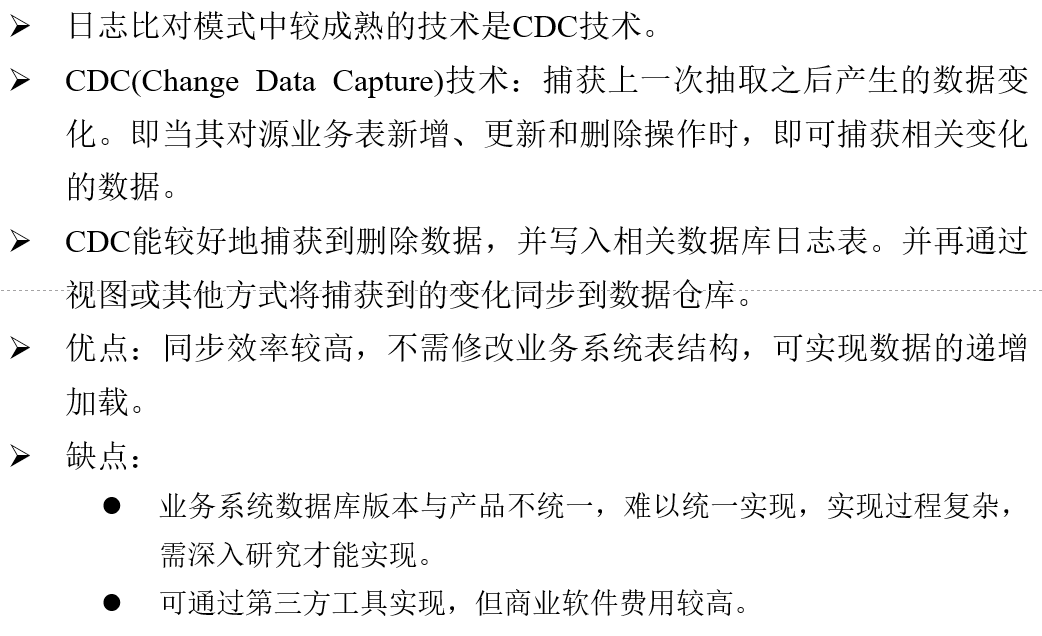
缺点：被动的进行全表数据的对比，性能较差；ETL比对较复杂，设计较为复杂，速度较慢；当表中没有主键或唯一列且含有重复记录时，全表对比的准确性较差

### 日志对比

通过获取数据库层面的日志捕获数据的变化

优点：不需要改变源业务系统数据库相关表结构，数据同步的效率较高，同步的实时性较好

缺点：不同数据库的日志结构差异较大，实施分析难度较大；需要具备访问源业务库日志表文件的权限，风险大



### 选择ETL工具

对平台的支持程度；

抽取和装载的性能，对业务系统的性能影响及侵入性高不高；

对数据源的支持程度；

是否具有良好的集成性和开放性；

数据转换和加工的功能强不强；

是否具有管理和调度的功能

主流工具有：Kettle、DataPipline、Talend、Informatica、Datax、Oracle Goldengate

## CDC（变化数据捕捉）

### 功能

可实现实时高效的数据集成；是实时主动数据仓库连续数据集成的有效解决方案

### 特性

没有宕机时间，不需要专门的时间窗口

保持数据新颖性

只转移变化数据，较少系统开销

### 组成

变化捕捉代理、变化数据服务、变化分发机制

### 变化捕捉代理

是一个软件组件，负责确定和捕捉发生在操作型数据存储源系统中的数据变化

可专门优化变化捕捉代理，使其适用于特定的源系统，如使用数据库触发器或数据日志比较

### 变化数据服务

为实现变化数据捕捉提供过滤、排序、附加数据、生命周期管理和审计等一系列的重要功能

|  |  |
| --- | --- |
| **功能** | **解释** |
| 过滤 | 确保只接收已经提交的数据 |
| 排序 | 接收数据时，基于事务、表或时间戳进行排序 |
| 附加数据 | 为分发的变化增加一些参考数据，以便于对数据进行进一步的处理。 |
| 生命周期管理 | 在多长时间内应用可以得到变化的数据；多长时间以后丢弃所分发的数据 |
| 审计 | 允许对系统的端到端行为的监听和对趋势的检查 |

### 变化分发机制

把变化分发到变化的消费者（常为ETL程序）

支持一个或多个消费者，且提供灵活的数据分发方式：拉（pull）方式需要消费者周期性的发送请求，常用标准接口如ODBC或JDBC实现；推（push）方式需要消费者一直监听和等待变化，一旦捕捉到变化，就立刻转移变化的数据，常采用消息中间件来实现

提供动态返回和请求旧的变化，以满足重复处理和回复处理等任务

### 典型应用场景：面向批处理的CDC（pull CDC）、面向实时的CDC（push CDC）

Pull CDC：

ETL工具周期性地请求变化（变化分发请求可采取2次/天或1次/15min等不同的频度），并接收批量数据（批量数据是上次和这次**请求之间**所捕捉到的变化）

1. 以数据表记录的形式提供变化数据：通过标准接口如ODBC无缝访问变化数据；CDC则需维护上次变化分发的位置和分发新的变化；pull CDC只需转移变化的数据，减少了资源消耗，消除了传统ETL过程的宕机时间
2. 使用现有技术，容易实现
3. 对时间延迟以分钟或小时衡量时，此种方式比较可行

Push CDC：

1. 满足零延迟的要求，变化分发机制一旦探测到变化，就把变化push给ETL程序
2. 常通过可靠的传输机制如事件分发机制和消息中间件（如MQ Series）实现
3. 很多ETL工具厂商提供这种功能，以满足高端、实时的商务应用需求
4. 当BI应用需零延迟和最新的数据时，这种实时的数据集成方法就是必须的

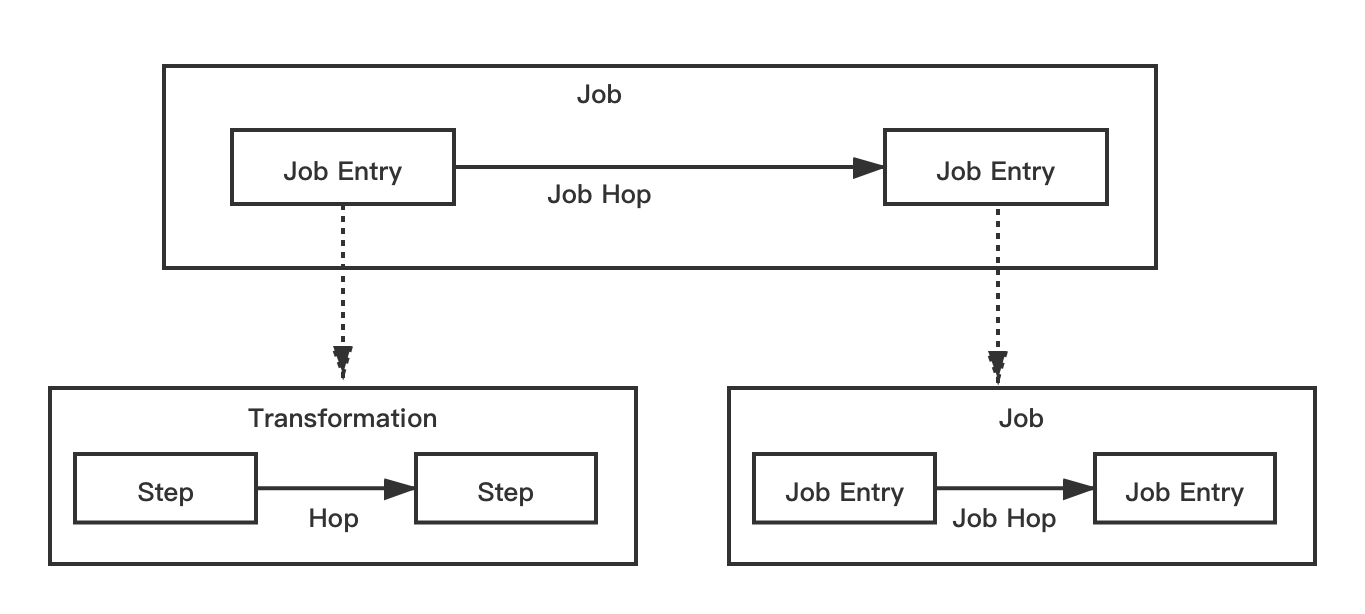
# Kettle

## 基本概念

一个数据抽取多项：创建一个Job，每个作业由一/多项作业项（Job Entry）和连接作业项的作业跳（Job Hop）组成

作业项：可为一个转换（Transformation）或另一个作业

转换：由一/多个步骤（Step）和连接步骤的跳（Hop）组成



图表 11一个数据抽取过程的要素

### 转换

用于数据的抽取（Extraction）、转换（Transformation）及加载（Load）：如读取文件、过滤输出行、数据清洗或加载到数据库等过程

转换包含一或多个步骤，每个步骤都是单独的线程

启动转换时，所有步骤的线程几乎并行执行

步骤间的数据以数据流方式传递：所有步骤都从他们的输入条中读取数据，并把处理过的数据写到输出跳，知道输入跳里不再有数据；当所有步骤终止时，整个转换就终止

转换里的步骤依赖上一步骤获取数据，因此转换里不能有循环

### 作业项

以某种自定义的顺序串行执行

作业项之间：可传递包含了数据行的结果对象

当一个作业执行完成后，在传递结果对象给下一个作业项

作业里可以有循环

### 跳

Step之间带箭头连线，定义了一个单向通道：实现了数据集从一个步骤（写入数据到行集）流向另一个步骤（从行集中读取数据）

跳是两个步骤间“行集”（Row Set）的数据行缓存（可在转换设置中定义行集大小）

### 行集

若行集满了：则停止向行集写数据，直到行集又有空间

若行集空了：则停止从行集读取数据，直到行集又有可读数据行

### 输出跳

向行集写入数据的步骤，一个步骤可拥有多个输出跳

### 输入跳

从行集中读取数据的步骤

### 作业跳

作业项之间带箭头的连接线，它定义了作业的执行路径