教你成为全栈工程师(Full Stack Developer) ○-什么是全栈工程师

发表于 2016-04-04 07:30

作为一个编码12年的工程师老将，讲述整段工程师的往事，顺便把知识都泄露出去，希望读者能少走一些弯路。

这段往事包括：从不会动的静态网页到最流行的网站开发、实现自己的博客网站、在云里雾里的云中搜索、大数据一统江湖.....

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

你可以把这个看做一个全栈工程师教程，因为看过“网站开发”部分的几篇文章你基本上就可以开发一个小型的博客网站了，而看过第二部分的几篇你也许对信息检索充满兴趣和信心了，大数据部分还是得花点功夫的，你可以选择放弃，因为比较耗费生命，当然如果你对自己有信心看下去，那么相信你会有收获

你也可以把这个看做是一个从对计算机丝毫不懂到可以和业界牛人谈笑风生一路披荆斩棘的过来人的一些经验

总之，希望对你有收获

为什么从“○”开始？

第一，因为这是一个综述；第二，因为这是我的第一篇；第三，计算机里面不都是从0开始的吗？

12年里我趟过哪些路？罗列如下：

学习和使用过的语言：Q-BASIC, Visual Basic, c, delphi, matlab, html, css, javascript, asp, c++, bash, awk, Fortran, cuda, Qt, lua, Object-C, php, java, python,

学习和使用过的技术：网页设计（ASP+ACCESS，PHP+MYSQL），软件开发（MFC、Qt），并行计算（MPI、hadoop、cuda），网站搭建（IIS、nginx/httpd+mysql+php-fpm），高性能网络服务（c++，libevent，protobuf，负载均衡，低耦合，一致性，10+模块，200+机器部署，7*24工作），搜索架构（c++网络爬虫、lucene、rank策略、时效性、日志统计、机器学习），博客网站开发（symfony2、bootstrap），app（thinkphp、ios、mongodb、memcache、redis），大数据（hive、hbase、map-reduce、storm、zookeeper、redis、mongodb、机器学习）

有什么体会？

语言不过是个工具，没有好坏之分，什么场景用什么语言。不要听网上的一面之词：C++比java牛逼，php是世界上最好的语言，swift比OC好.....

语言学习跟英语一样，就靠练习和实践，除非你还在上学或者刚毕业，否则没必要集中花时间系统学习一门语言，现用现查就好了，有问题百度一下

同一个项目我可能用10种语言，该谁上场就谁上场，不是那块料没必要勉强

我也在BAT做过5年的系统架构，什么样的场景、解决什么样的问题，就用什么样的架构，如果是小业务场景也没必要杀鸡用牛刀，不追求牛逼，简单为上，唯快不破

对于开源项目，如果是小业务场景，无论是存储/计算系统还是各种框架，能用开源就用开源，重复造轮子可能证明了你的技术实力，但也同时证明了你智商不高

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

比较推荐学习的语言和技术？

网站开发：后端语言推荐php，后端框架推荐symfony2，后端服务器推荐nginx，php服务器推荐php-fpm，前端语言html+css+javascript是必须的，前端框架推荐bootstrap，数据库推荐mysql

服务器端开发：语言推荐C/C++, java, python, bash, awk, 框架推荐libevent, protobuf, 开源系统推荐hadoop, hive, hbase, zookeeper, redis, mongodb

搜索技术：语言推荐java，框架推荐pyspider、lucene、solr

什么是全栈工程师？

没什么固定的定义，一个创业公司的技术团队，哪里缺人你都能顶上，你就是全栈工程师。

教你成为全栈工程师(Full Stack Developer) 一-各显神通总结八大类编程语言的区别

发表于 2016-04-06 19:02

为了能在最快的时间里理解更多语言的相同点和不同点，我用大家最熟悉的Hello World来展示一下各个语言的奥妙

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

注意：整个教程是基于linux系统的(为了方便讲解，我选择用vmware虚拟机装了CentOS 7来演示，其他linux均可)，包括编译器、依赖库安装过程、命令行，如果是使用windows或mac系统稍有差别，CentOS 7安装方法百度一下你就知道

万物之源的C语言

简述：C语言是编译执行的语言，linux下常用的编译器是gcc，c语言源代码可以直接被编译成可执行程序（机器码），说它是万物之源是因为其他各种语言都是用C语言写的，如java、php、python……

源代码：hello_world.c

```
#include <stdio.h>
int main(int argc, char* argv[])
{
    printf("hello world\n");
    return 0;
}
```

编译执行：

```
[root@centos7vm code]# gcc -o hello_world hello_world.c
[root@centos7vm code]# ./hello_world
hello world
```

最流行的java语言

简述：java语言曾一度家喻户晓，有人可以昂起头说我会java，如今虽然潮流已过，但java跨平台的优势以及强大的类库着实不容小觑，其他语言很难超越，java衍生除了非常多周边产品（如jsp），非常多的开源系统都是基于java（hadoop、hive、hbase、lucene等），java和c一样也是编译执行的语言，区别在于java编译出的字节码文件运行在一层java虚拟机之上，而虚拟机可以架设在各种操作系统上，所以java也就有了跨平台的优势，一处编译多处执行

环境准备：

```
[root@centos7vm code]# yum install java*
```

源代码：hello_world.java

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

编译执行：

```
[root@centos7vm code]# javac HelloWorld.java
[root@centos7vm code]# java HelloWorld
hello world
```

互联网时代的html语言

简述：html是超文本标记语言，通俗来说就是展示网页的，是一种解释执行的语言（读一句展示一句，不需要编译成其他形式），html通过一个个标签来指示浏览器怎样展示网页

源代码：hello_world.html

```
<html>
    <head>
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

用浏览器打开展示效果：

解释执行的脚本语言bash

简述：bash是shell脚本的一种（也叫bshell，类似还有csh和ksh），其实类似于dos里的bat批处理程序，把一堆顺序执行的命令写到一个文件里一起执行，同时扩展了一些分支、循环、函数等语言特性，在服务器端开发和运维中经常用到

源代码：hello_world.sh

```
#!/bin/bash
echo "hello world"
```

执行效果：

```
[root@centos7vm code]# sh hello_world.sh
hello world
```

即能编译执行又能解释执行的python语言

简述：python的强大在于它的不伦不类，说它是解释执行，它却会编译成文件，说它是脚本语言，它却具有面向对象的所有性质，如今python开源的类库已经非常强大了，什么功能都有，python也成了大数据方向必要的工具

源代码：hello_world.py

```
#!/usr/bin/python
print "hello world"
```

执行效果

```
[root@centos7vm code]# python hello_world.py
hello world
```

ps：python还可以交互式执行，也就是像在终端里执行python语句一样，如下：

```
[root@centos7vm code]# python
Python 2.7.5 (default, Nov 20 2015, 02:00:19)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "hello world"
hello world
```


号称全世界最好的语言的php

简述：动态网页后端语言有很多，asp、asp.net、jsp、c#、python、php……，php可以说是最容易上手的语言

环境准备：

```
[root@centos7vm code]# rpm -Uvh https://mirror.webtatic.com/yum/el7/epel-release.rpm
[root@centos7vm code]# rpm -Uvh https://mirror.webtatic.com/yum/el7/webtatic-release.rpm
[root@centos7vm code]# yum install php55w.x86_64 php55w-cli.x86_64 php55w-common.x86_64 php55w-gd.x8
6_64 php55w-ldap.x86_64 php55w-mbstring.x86_64 php55w-mcrypt.x86_64 php55w-mysql.x86_64 php55w-pdo.
x86_64
```

源代码：hello_world.php

```
<?php
print("hello world\n")
?>
```

执行效果：

```
[root@centos7vm code]# php hello_world.php
hello world
```

ps：实事上php程序不是这样用的，而是和http服务器一起通过接收http请求，并执行响应的php脚本，然后返回html标签给用户，实现动态网站，后面会详细讲解

网页样式语言css

简述：css是html的一个辅助语言，用来描述网页样式，html没有css也可以工作（可以通过标签的各种属性以及style属性定制样式），但css可以把样式做抽象，和html剥离，这样html单纯用作布局

源代码：

hello_world.html

```
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="style.css" />
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

style.css

```
h1 {
    color: red
}
```

用浏览器打开展示效果：

网页前端脚本语言javascript

简述：javascript是可以运行在网页前端的脚本语言，可以基于html之上实现更丰富的交互、异步回调、多线程、定时器、动画等

源代码：hello_world.html

```
<html>
    <head>
    </head>
    <body>
        <script type="text/javascript">
            document.write("<h1>Hello World!</h1>")
        </script>
    </body>
</html>
```

用浏览器打开展示效果：

总结

综上，挑选了几种典型语言来说明运行过程，针对不同的业务场景选用不用的语言，每种编程语言可以边用变查，不建议一开始就系统学习，而应该运用一段时间后再系统学习，从而补充一些没用过的特性。

有关编程语言更高阶的内容可以看看《编译原理》、《lex & yacc》

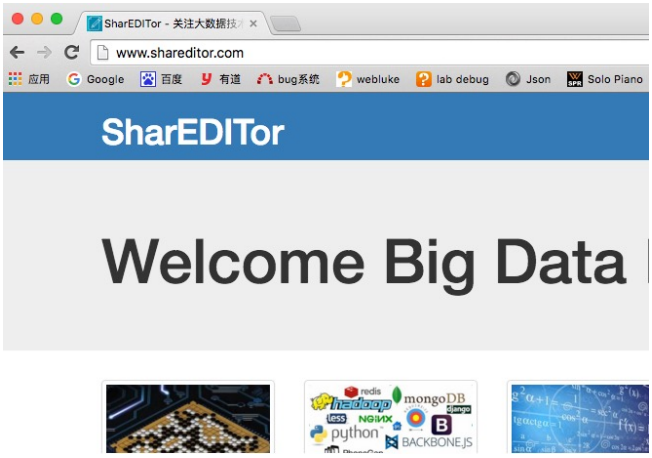
教你成为全栈工程师(Full Stack Developer) 二-半小时学会网站开发

发表于 2016-04-08 09:11

互联网领域常见的产品形式有：网站、移动app、pc软件，网站是最典型的涉及前后端并同时应用在pc和移动端的产品类型，所以咱就拿网站来说事，来跟着我开发一个属于你自己的网站吧

网站是怎么工作的？

打开浏览器输入：www.shareditor.com试一下



这个过程就是：我们用浏览器访问了一个网站。那么到底浏览器和网站分别都做了什么呢？我们再来做一个试验

在试验之前我们要安装nc工具（nc是一个网络工具，通过nc命令可以模拟客户端或服务器端程序，非常方便）

```
[root@centos7vm ~]# yum install nc
```

接着我们来尝试一下nc的网络功能，先打开一个终端输入：

```
[root@centos7vm ~]# nc -l -p 80
```

回车后光标停止了，这条命令的含义是：模拟一个服务端程序监听在80这个tcp端口上，等待客户端连接

然后再打开一个终端输入：

```
[root@centos7vm ~]# nc 127.0.0.1 80
```

回车后光标也停止了，这条命令的含义是：模拟一个客户端通过tcp协议连接127.0.0.1这个ip地址所在的机器（也就是本机）的80端口

然后在第二个终端里直接输入hello，如下：

```
[root@centos7vm ~]# nc 127.0.0.1 80
hello
```

这时看第一个终端出现了：

```
[root@centos7vm ~]# nc -l -p 80
hello
```

如果你这时在第一个终端里输入一些字符，第二个终端里也会看到，这其实就是一个简单的QQ聊天，是不是很神奇

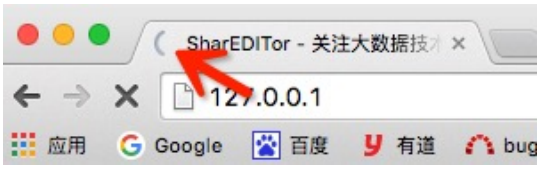
好了，现在你是否理解了nc的功能了呢？

言归正传，现在我们就用nc来说明网站是怎么工作的

我们重新监听80端口（网站默认使用的是http协议，http协议默认都是80端口，其实也可以选用其他端口比如8080，那么浏览器访问时就要输入www.shareditor:8080）

```
[root@centos7vm ~]# nc -l -p 80
```

这时我们同浏览器来访问127.0.0.1，如下：



我们发现浏览器一直在转圈圈，但是发现终端上出现了下面一些内容：

```
[root@centos7vm ~]# nc -l -p 80
GET / HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/49.0.2623.110 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
```

这其实就是浏览器给服务器发的请求内容，这时我们手工来给浏览器回个信，直接在请求下面输入：

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Connection: close

hello
hi
<html>
<body>
<h1>hello</h1>
</body>
</html>
```

这时回过来看浏览器，是不是这样了？

我们刚才其实就做了一个小网站呢有木有！！！当然我们的网站运行的成本有点大，因为需要人来看着，并时刻敲出要返回的网站内容，想象一下如果同时有1000个人访问我们的网站，我们得敲多长时间，这1000个人会不会疯掉

实际上有人已经帮我们做了这个工作，那就是网站服务器程序，比较流行的是nginx和apache（这两个原理区别不大，主要是配置文件写法不同，我个人习惯用nginx），现在我们就来用nginx搭建一个小网站

先来安装nginx的1.8版本：

```
[root@centos7vm ~]# yum install nginx18
```

安装好后可以看到系统里多了以下目录和文件：

```
[root@centos7vm ~]# ls /usr/sbin/nginx
/usr/sbin/nginx
[root@centos7vm ~]# ls /etc/nginx/
conf.d  fastcgi.conf  fastcgi.conf.default  fastcgi_params  fastcgi_params.default  koi-utf  koi-win  mime.types
mime.types.default  nginx.conf  nginx.conf.default  scgi_params  scgi_params.default  uwsgi_params  uwsgi_pa
rams.default  win-utf
[root@centos7vm ~]# ls /usr/share/nginx/
html
[root@centos7vm ~]# ls /var/log/nginx/
```

这些分别是nginx的可执行文件、配置文件、网页文件目录、运行日志目录

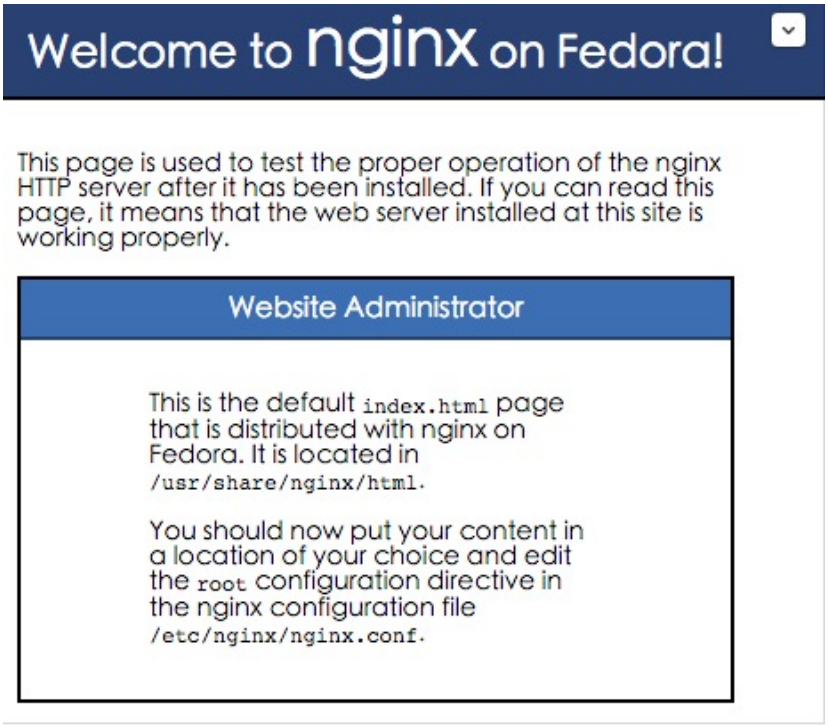
不用犹豫，直接来启动nginx

```
[root@centos7vm ~]# service nginx start
Redirecting to /bin/systemctl start  nginx.service
```

可以看到nginx进程已经启动：

```
[root@centos7vm ~]# ps aux | grep nginx
root      21238  0.0  0.1 109552  2000 ?        Ss   05:13   0:00 nginx: master process /usr/sbin/nginx
nginx     21239  0.0  0.3 109992  3064 ?        S    05:13   0:00 nginx: worker process
root      21243  0.0  0.0 112660   960 pts/1    R+   05:13   0:00 grep --color=auto nginx
```

这时我们再打开浏览器访问127.0.0.1可以看到下面的效果：



这个页面其实就是nginx帮我们输入了一些文字，返回给浏览器的，这些内容就放在：

```
[root@centos7vm ~]# cat /usr/share/nginx/html/index.html
```

我们尝试按照我们自己的想法制作一个网页，编辑/usr/share/nginx/html/test.html文件，写入内容如下：

```
[root@centos7vm ~]# cat /usr/share/nginx/html/test.html
<html>
   <body>
      <h1>hello</h1>
   </body>
</html>
```

这时我们用浏览器访问：127.0.0.1/test.html，效果如下：

现在大功告成，你可以充分发挥你的想象来制作你自己的网站了，具体html语法可以参考网上的html教程，也有很多软件能方便制作漂亮的网页

最后留一个悬念，当你制作好你的网站希望其他人能看到的时候，你会发现在我们虚拟机里运行的网站服务是无法让外部互联网上的其他人看到的，这个问题会在下一篇中详细解释

教你成为全栈工程师(Full Stack Developer) 三-网站后端服务器那些事

发表于 2016-04-10 07:04

当你访问百度的时候就一个简单的页面呈现在你面前，你觉得这个简单的页面几个人就搞定了，实事上也是这样，但隐藏在百度的后端服务器程序却有数千工位程师在维护着。网站后端服务器就是这众多重要的后端服务器的一种。

互联网上的机器之间是怎么相互访问的？

你打开电脑就能相互访问吗？显然需要先接入互联网，也就是常说的“能上网”（插网线、连wifi），这样至少数据信号有途径传播。那么你怎么知道你要连到哪台电脑呢？通过IP地址（门牌号）。每一个接入互联网的电脑都会有一个ip地址，知道某个机器的ip地址就能通过网络协议连接它跟它通信了。

但对于我们来说有一个最大的问题，我们的电脑都是在局域网里的（办公网络、小区宽带都是），在局域网里的电脑比较特殊，是一个局域网的ip地址，事实上整个局域网里只有一台电脑（NAT服务器）是有真正的外网ip的，其他电脑都是通过这个有真正ip地址的特殊的服务器转发网络数据，这就存在一个问题就是从外界无法精确定位到一个局域网的机器，这就像是你可以找到北京市昌平区1024号楼一单元110门牌号的人家，但你找不到北京市某个叫做“主卧”的地方。所以很遗憾，我们的个人电脑做不了网站服务器。

怎么样搞到网站服务器呢？

传统的做法：申请一个固定ip，自己的电脑用这个ip，走出局域网。如今这个方案已经很难而且非常昂贵了，因为32位ip地址几乎被用光了，64位ip地址还没有普及，租用ip费用昂贵到个人是没办法承受的，所以还是考虑下面的方案吧。

现在流行的方案：云服务器供应商，比如阿里云。其实阿里早就囤积了一大批外网ip地址，阿里云服务器就是一个帮你绑定好外网ip地址的远程机器，7*24小时开机提供服务，并把机器的管理员权限全权交给你，就相当于这是你的电脑了，把网站部署在这台机器上就可以了。费用也是很便宜的（流量不大的小网站每个月70元左右）。您现在看到的www.shareditor.com就是我部署在阿里云服务器上的网站。具体阿里云服务器怎么申请怎么用百度一下你就知道。

域名是怎么定位到机器的？

域名就是指www.shareditor.com这种网址，我们一般在浏览器只输入网址不输入ip，怎么就能访问到网站内容呢？因为自动做了一次域名解析，也就是浏览器自动根据域名解析成对应的ip。这就是电脑里经常需要调整配置的DNS服务。DNS服务器属于互联网基础建设，接入互联网就能使用，不必担心。那么我们做的网站的域名是什么呢？

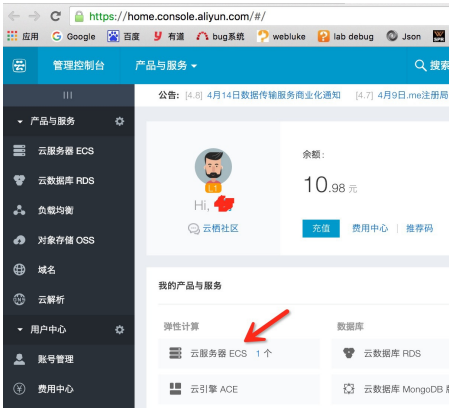
如何指定我的网站的域名？

域名和给小孩子起名不太一样，不可以重名。比如人家叫baidu.com，你不能也叫baidu.com，不然大家访问baidu.com就不知所向了。每个国家有专门的域名注册机构，比如中国的新网和万网，你也要去这两处申请。大家应该经常听说过某某公司重金买域名的案例吧，这就是在数年前有人看到了域名的商机，抢先低价注册短域名，n年后就会有人要高价买，从而赚差价。所以现在已经很难申请到短域名了。不管怎么样，还是得起一个容易记得住的域名，然后去新网或万网提交申请，然后需要拿着各种证件去现场确认审核等，国家备案之后才能批下来，整个过程预计半个月吧。不管是万网还是新网，申请成功之后你都可以登录上去并修改你域名对

应的ip地址，修改完之后就大功告成啦，互联网上任何人只要浏览器输入你的域名就自动连接你设定的ip地址啦，搞定！

给大家看下我的案例吧

1. 注册并登陆阿里云https://www.aliyun.com/，点开“管理控制台”



查看云服务器：

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

登陆阿里云服务器如下：

```
[root@centos7vm ~]# ssh root@182.92.80.220
root@182.92.80.220's password:
Last login: Sun Apr 10 07:26:16 2016 from 222.129.54.231

Welcome to aliyun Elastic Compute Service!

[root@MYAY ~]# ifconfig |grep inet
        inet 10.162.223.224  netmask 255.255.240.0  broadcast 10.162.223.255
        inet 182.92.80.220  netmask 255.255.252.0  broadcast 182.92.83.255
        inet 127.0.0.1  netmask 255.0.0.0
```

这里面有三个ip地址，第一个是阿里云局域网内部ip，第二个是外网ip（网站用），第三个是本地回路ip

我的网站服务情况如下：

```
[root@MYAY ~]# ps aux|grep nginx
root     15743  0.0  0.1  48012  1164 ?        Ss   4月01   0:00 nginx: master process /usr/sbin/nginx -c /etc/nginx/
nginx.conf
nginx    15744  0.0  0.2  50572  2248 ?        S    4月01   0:08 nginx: worker process

[root@MYAY shareditor2.0]# egrep "listen|server" /etc/nginx/conf.d/shareditor2.0.conf
server {
    listen       80;
    server_name  www.shareditor.com;

[root@MYAY shareditor2.0]# netstat -npal|grep "80 "
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      15743/nginx: master
tcp        0      0 182.92.80.220:42483     110.75.102.62:80        ESTABLISHED 1035/AliYunDun
```

我的域名备案通过阿里代备案系统https://beian.aliyun.com/管理


主体信息

备案号：[redacted]	
主办单位或主办人全称：[redacted]	主办单位性质：个人
主办单位证件类型：身份证	主办单位证件号码：[redacted]
主办单位证件住所：[redacted]	主办单位所属区域：北京市昌平区
主办单位通信地址：[redacted]	投资人或主管单位：[redacted]
负责人姓名：[redacted]	负责人证件类型：身份证
负责人证件号码：[redacted]914	办公室电话：
手机号码：[redacted]	电子邮件地址：[redacted]9126.com
备注信息：	

我的DNS解析配置如下：

因为配了，所以能解析到：

```
[root@MYAY ~]# host www.shareditor.com
www.shareditor.com has address 182.92.80.220
[root@MYAY ~]# ping www.shareditor.com
PING www.shareditor.com (182.92.80.220) 56(84) bytes of data.
64 bytes from MYAY (182.92.80.220): icmp_seq=1 ttl=64 time=0.023 ms
```

有了以上这些服务器相关的知识，就可以把你的网站推向互联网啦！

教你成为全栈工程师(Full Stack Developer) 四-浏览器的那些事

发表于 2016-04-13 06:27

如果说windows图形界面是个人电脑得到普及的基础，那么浏览器就是互联网得到普及的基础，那么浏览器是怎么工作的，和远端的网站之间又是怎么交互的呢？

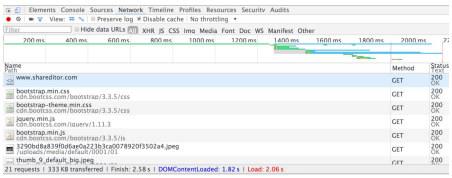
浏览器都干了什么

用浏览器打开http://www.shareditor.com/，在网页空白处点右键选“显示网页源代码”，如下：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scal
    <!-- 上述3个meta标签*必须*放在最前面，任何其他内容都*必须*随后！ -->
    <title>SharEDITor - 关注大数据技术</title>

    <!-- 新 Bootstrap 核心 CSS 文件 -->
    <link rel="stylesheet" href="//cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap.min

    <!-- 可选的Bootstrap主题文件（一般不用引入） -->
    <link rel="stylesheet" href="//cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap-the

    <!-- jQuery文件，务必在bootstrap.min.js 之前引入 -->
    <script src="//cdn.bootcss.com/jquery/1.11.3/jquery.min.js"></script>

    <!-- 最新的 Bootstrap 核心 JavaScript 文件 -->
    <script src="//cdn.bootcss.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>

        <script>
        var _hmt = _hmt || [];
        (function() {
            var hm = document.createElement("script");
            hm.src = "//hm.baidu.com/hm.js?ec8e80cb5a0c012024fffd91e3614edb";
            var s = document.getElementsByTagName("script")[0];
            s.parentNode.insertBefore(hm, s);
        })();
    </script>

</head>
<body>
<div class="row bg-primary">
    <div class="col-sm-1 col-xs-1"></div>
    <div class="col-sm-11 col-xs-11"><h1><a href="/" style="text-decoration: none;co
</div>
```

这才是网页的真正内容——一堆标签。浏览器向www.shareditor.com的网站服务器发送请求后，由网站服务器把这一堆标签发回给浏览器，浏览器就是根据这一堆标签的指令“渲染”成漂亮的页面的。

让我们换个视角重新看一遍这个过程。以google chrome浏览器为例，打开浏览器，从菜单里找到“开发者工具”（不同浏览器名称不一样），重新输入http://www.shareditor.com/打开，开发者工具中会展示多条链接的请求过程，第一条是www.shareditor.com这条链接，再下面是很多的css、js、jpeg等。点开第一条www.shareditor.com，会展示出这一条请求的详细信息如下：



Request URL展示了要请求的链接，Request Method是请求方法：GET（HTTP协议中还有POST、HEAD等方法），Status Code是返回码（200表示正常返回页面），Remote Address是服务器的ip:port（其中的ip就是通过dns服务根据域名获取到的ip地址，port默认都是80端口）

再下面是http请求和响应的头部信息，都是一些域和值

这整个请求和响应的处理过程是一次HTTP协议的处理过程，这种通信协议就叫HTTP协议，那么这样的服务器也叫http服务器（如nginx、apache等）

以上都是铺垫，现在开始说干货，对于HTTP协议返回的结果浏览器都会做哪些处理呢？

1. 响应头字段的分析和处理

响应头里的字段都告诉了浏览器很多要求，比如Cache-Control字段说明了浏览器要不要把这个页面缓存起来（下一次访问是不是不再请求服务器直接展示），再比如Transfer-Encoding说明了返回结果是不是分片的，详细的可以参考官方HTTP协议

2. 标签布局

http服务器返回的结果都是标签的形式，标签主要为了说明页面布局，比如：

<html>

<head>

</head>

<body>

<table>

<tr>

<td>第一行第一列<td><td>第一行第二列<td>

<td>第二行第一列<td><td>第二行第二列<td>

</tr>

</table>

</body>

</html>

这其实就定义了一个2行2列的布局表格，表格里放的不一定是文字，很可能是图片、视频，也可能是另一个表格，也可能是更复杂的内容

标签的特点是每一个标签都是成对出现的（表达开始和结束），标签都是可嵌套的，html、head、body是html网页固定的标签，各种内容都放到body里，专门表示布局的标签有table、div、iframe等，布局形式很丰富，只有你想不到的，没有你做不到的

3. 样式渲染

浏览网页会看到文字都是有字体、大小、颜色的，图片都是有长度宽度的，元素之间都是有距离、对齐的，这些样式方面的展示就是浏览器样式渲染的功能。

html里有两种表达样式的方式，一种是内嵌到html里，如：

<img width="500" height="200" src="http://www.shareditor.com/uploads/4.jpeg">

或者<img style="width:500px; height:200px;" src="http://www.shareditor.com/uploads/4.jpeg">

另一种是剥离出样式文件，如：

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

my.css

```
img {
    width:500px; height:200px;
}
```

4. 链接提取

这里面有两种情形，一种是直接获取链接内容用来展示，一种是识别链接用来响应点击

像类似于

<img src="http://www.shareditor.com/uploads/4.jpeg">

这种标签，就表示要在这里显示一张图片，图片地址是http://www.shareditor.com/uploads/4.jpeg，这是需要浏览器在显示网页的同时再一次发送一个http请求，把图片下载到本地并展示

而类似于

SharEDITor

这种标签，就表示这是一个超链接，你在浏览器里点“SharEDITor”就自动跳转到http://www.shareditor.com/，不点的时候浏览器是不会自动获取这个网页的

5. javascript渲染

javascript（简称js）和样式一样分为内嵌形式和外挂文件的形式。它是在浏览器加载网页时自动执行的一种脚本，它弥补了网页文件只能静态显示内容的缺陷，可以实现网页内容的动态加载（ajax）、动态修改（如新浪首页出现几秒钟后又自动消失的大广告）、异步事件（定时弹出个窗口）、特殊事件处理（如：注册时输入用户名密码的合法性校验）等

前端框架

框架是什么，框架就是前人帮我们做好了大部分工作，我们只需要在此之上稍作修改就能实现强大功能。一般前人都是在开发比较繁琐的时候才会想到抽象出一层框架，前端最繁琐的就是js和样式，那么就这两部分框架最多了，js框架应用广泛的有jquery、angular JS等，样式框架比较知名的是bootstrap。当然还有非常多的控件可以复用，只要你能想到的，基本就有人帮你做了。

cookies是什么？

无论是浏览器使用还是网页开发，经常遇到的一个词就是cookie。cookie是浏览器用来记住你的用户、登陆、浏览等信息的数据。平时你有没有发现过这样的问题：我登陆一个网站后关掉浏览器，明天再打开浏览器能记住登陆状态。这就是cookie在起作用，当你填写账号密码登陆后，服务器返回网页内容同时会带上一串cookie让你的浏览器保存下来，以后每当你再访问这个网站的时候会自动把cookie带在请求里，这样服务器就知道你已经登陆过了，就像是一个令牌一样。在大数据领域，cookie也经常被用来作为服务器端数据统计分析的重要数据基础。

浏览器的种类

大家常用的浏览器有：IE、chrome、firefox、opera、mozilla、腾讯浏览器、百度浏览器、遨游浏览器、360浏览器、搜狗浏览器等，这些浏览器功能基本上相同，都是对html做渲染和展示，不同的地方在于某些浏览器对某些标签解释方式不太一样，所以在开发网页前端代码时需要做一些兼容性的考虑。

浏览器内核

浏览器纵有千万种，内核只有那么几个，因为对网页的渲染是一个非常底层而又繁琐的工作，众多浏览器基本上都是基于三种内核：ie内核（ie、百度浏览器）、火狐内核（mozilla、firefox）、webkit内核（chrome）。火狐和webkit内核都是开源的，所以很多编程语言的基础库里面已经包含了浏览器控件，开发的时候就拿浏览器当做一个简单的控件使用，这是一个很神奇的事情。

没有图形界面的浏览器

有一些特殊场景会遇到一些特殊而又有趣的问题：在linux文本界面下（没有图形界面）想要运行有浏览器控件的程序并且希望截个屏保存成图片，怎么破？有这么一个工具叫做xvfb（Xvirtual frame buffer），可以帮你模拟一个图形界面的外壳，可以执行图形界面的程序（虽然我们看不到，但在linux里真的在运行），那么截屏就是编程库里调用一个方法的事情了

教你成为全栈工程师(Full Stack Developer) 五-世界上最好的网站后端语言 php

发表于 2016-04-14 18:44

php是一种实现动态网页的服务器端语言，具有简单、轻量、上手快的优点，而且可以和其他很多语言结合使用。有人说php是世界上最好的语言，虽然说得不对，但从php工程师的角度来讲确实是这样的，那么就让我们一起领略一下php的神奇之处。

环境准备工作

打开我们的centos 7虚拟机，确保nginx已经安装好并启动了服务（在前面的教程里有安装和启动方法）。下面我们来安装php-fpm（php和nginx之间的一个桥梁），执行：

```
[root@centos7vm ~]# yum install php55w-fpm
```

执行

```
[root@centos7vm ~]# service php-fpm start
```

启动php-fpm服务

修改nginx配置来让nginx把php代码转发给php-fpm服务解析，修改文件/etc/nginx/nginx.conf里的server组后如下：

```
    location ~ \.php$ {
        root           /usr/share/nginx/php/;
        fastcgi_pass   127.0.0.1:9000;
        fastcgi_index  index.php;
        fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
        include        fastcgi_params;
    }
```

执行

```
[root@centos7vm ~]# service nginx reload
```

重新加载配置文件

顺便我们看一眼这个配置文件的其他部分来做一些解释：

```
error_log  /var/log/nginx/error.log
……
access_log  /var/log/nginx/access.log  main;
```

这些表示http访问日志的存放的地方

```
include /etc/nginx/conf.d/*.conf;
```

这一句表示你可以在conf.d目录下创建更多的网站配置文件（虚拟主机），nginx会自动加载进来

listen 80;表示监听http默认的80端口

root /usr/share/nginx/html表示你所有网页文件存放的地方，nginx只会解析你放在这里的网页

location /.....表示当你访问http://www.shareditor.com/时服务器要怎样处理

location /path.....表示当你访问http://www.shareditor.com/path时服务器要怎样处理

```
    location ~ \.php$ {
        root           /usr/share/nginx/php/;
        fastcgi_pass   127.0.0.1:9000;
        fastcgi_index  index.php;
        fastcgi_param  SCRIPT_FILENAME  $document_root$fastcgi_script_name;
        include        fastcgi_params;
    }
```

这几行表示当你访问http://www.shareditor.com/*.php时，服务器会把对应找到的php文件转发给本机的9000端口（就是php-fpm监听的端口）来解析，并把解析好的内容发回给你

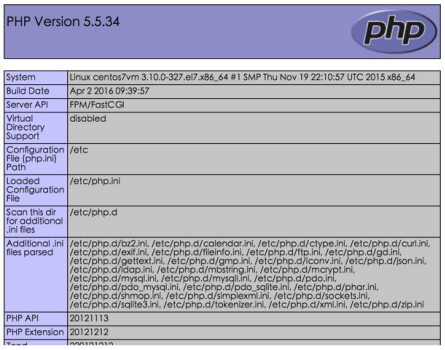
OK，大功告成，现在可以开始php之旅了

一句代码看清楚一切

在/usr/share/nginx/php/目录下创建一个php文件：index.php，内容如下：

```
<?php
phpinfo()
?>
```

这时打开浏览器访问127.0.0.1/index.php结果如何？



展示了php的信息

事实上这都是phpinfo这一句代码在起作用

ps：通过我本机的浏览器看虚拟机的网站为什么访问不了？

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

因为默认情况下centos 7会开启防火墙，导致外部ip无法访问，关闭防火墙方法如下：

```
[root@centos7vm php]# systemctl stop firewalld.service
[root@centos7vm php]# systemctl disable firewalld.service
```

假设你虚拟机ip地址是172.16.142.130那么在你主机浏览器里访问http://172.16.142.130就可以啦

看到这我还是不会写php啊！！

下面让我们来看看php的语法有多么的简单，一分钟你就学会了。我们来用php开发这样一个网页，网页显示一组倒数的数字，如果是偶数就把样式做成h1标题，如果是奇数就做成h3，如下：

```php
<?php
$i=10;
for ($i=10; $i > 0; $i=$i-1)
{
    if ($i % 2 == 0)
    {
        echo "<h1>" . $i . "<h1>";
    }
    else
    {
        echo "<h3>" . $i . "<h3>";
    }
}
?>
```

整个效果是这样子的

现在是不是明白了php的工作原理了呢？php其实就是按照普通编程语言的逻辑来动态输出html标签，让他看起来像个静态html文件

php支持函数、类吗？

支持的，php函数像这样：

```php
<?php
function writeMsg() {
  echo "Hello world!";
}

writeMsg(); // 调用函数
?>
```

php的类像这样：

```
class test
{
    var $b;
    function test() { $this->b=5; }
    function addab($c) { return $this->b+$c; }
}

$a = new test(); echo $a->addab(4); // 返回 9
```

有关php语言的更高级功能可以系统学习php教程，但个人觉得以上内容就足够了，遇到不会的直接百度

教你成为全栈工程师(Full Stack Developer) 六-总结php框架的功能

发表于 2016-04-18 08:00

可以成为一个产品的网站的业务逻辑都有一定的复杂性，哪怕一个小小的博客网站也会有用户管理、新闻发布、分类管理和多媒体资源管理等，但是有没有办法像堆积木一样实现想要的功能呢？有！开源框架及其海量组件可以帮你实现梦想，本节主要是php框架的介绍。

什么是php框架？

php框架就是把php开发过程中通用而繁琐的部分抽象出来，约定出一些固定的套路来供你配置，这样协作开发的人也更容易相互理解。

所有php框架一般都包含以下内容：model、视图（view）、控制器（controller）、路由（router）、工具集、扩展组件。

model：就是数据库表的类表达，这个class对应着数据库表结构，比如数据库表里有一列叫做name，那么这个类就有一个成员变量name，当然model可以附加getters和setters以及其他自己实现的方便操作的成员函数。框架抽象出model的目的是让我们像对待内存中的变量一样对待数据库表，这样我们就不用关注数据库了

视图：就是展现给用户的部分，通常是网页（html），比如要把某个model的某个成员值展现出来，那么就传给视图来通过html显示出来。当然样式也是在视图里面实现的。为了开发的方便性，视图一般结合着模板使用

模板：模板包含三部分内容，一个是视图的继承，一个是传递变量，一个执行逻辑。视图继承就是我们可以写一个base模板，定义好通用的页面布局大框架，并把每个可定制的部分预留出接口来给子类视图实现。传递变量的功能实现了从php到html的数据传递，比如在html里用<h1>{{ name }}</h1>来展示php里的name这个变量的值。执行逻辑就是定义了一些分支、循环等逻辑来动态输出html

控制器：控制器是用来实现业务逻辑的部分，控制器可以理解为事件触发时的执行逻辑，比如当你访问某个

url（如: http://www.shareditor.com/bloglist/2）可以触发bloglist对应的控制器执行“列出博客列表”的逻辑，那么这个控制会怎么做呢？可能是这样的：它获取blog这个model里的所有数据，并把数据全部传给视图，而视图通过循环逻辑取出每个blog的name展示出来。

路由：就是定义了什么pattern的url对应执行哪个控制器逻辑，其实就是一个映射关系。除此之外还有变量传递、路由通配等功能。

工具集：除了上面有固定套路的部分之外就是一堆工具集了，这部分不同框架也不太一样，所以用着再说

扩展组件：一般框架都会配有一些扩展组件，当然强大的网友会帮你开发各种各样的组件，git上只有你想不到的，没有你找不到的。

都有哪些流行的php框架？

国外比较流行的php框架有laravel、symfony2、yii2等，国内比较流行的有phalcon、thinkphp、zend framework等。其实所有的php框架不外乎上面讲的几大部分，真正的业务逻辑都是得我们自己来实现。我个人用symfony2有一段时间了，相对比较熟悉，所以后续章节都以symfony2来带大家领略php的美妙。当然这里我也给symfony2打个广告，我当前发文章的博客网站就是用symfony2框架搭建的，功能包括：用户管理、新闻发布、分类管理和多媒体资源管理，自己开发的代码行数不到800行，其余全是用symfony2各种组件修改配置搭起来的，是不是amazing？

展示一下我的800行代码的网站效果

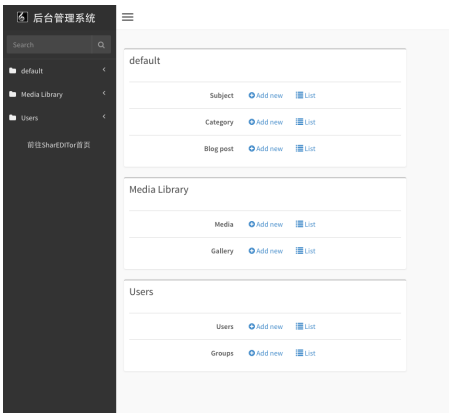
首页效果

文章列表页

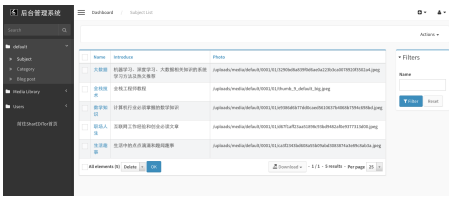
文章详情页

底部评论（多说）

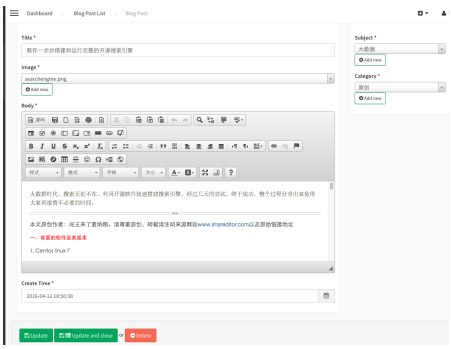
后台管理首页



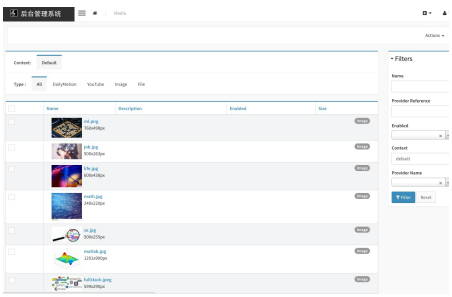
后台管理类别管理



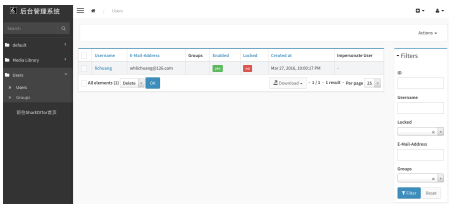
文章编辑



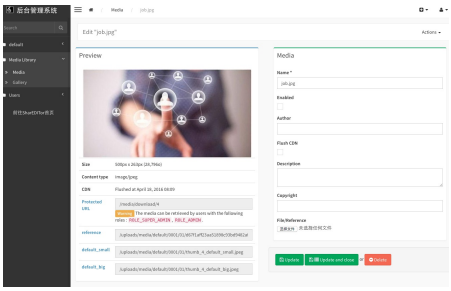
多媒体资源管理



用户管理



图片详情



教你成为全栈工程师(Full Stack Developer) 七-带你进入神奇的symfony世界

发表于 2016-04-20 07:48

一步一步手把手讲解symfony2的安装和配置，并快速搭建网站第一个页面。

安装symfony2

我们默认你使用的是centos7操作系统并已经安装好了php和nginx（如果还没有请回过头看前面几篇）。

首先，我们来安装symfony安装器。假设我们要安装到/usr/local/bin下，那么执行

```
sudo curl -LsS https://symfony.com/installer -o /usr/local/bin/symfony
sudo chmod a+x /usr/local/bin/symfony
```

执行一下symfony命令看看是否正常显示帮助信息

下面找到一个放置自己网站的目录，比如/data/httpdir/，创建好目录后，在httpdir下执行如下命令下载并初始化网站目录

```
[root@centos7vm httpdir]# symfony new mywebsite 2.8
```

这里的2.8是symfony2的版本，现在已经发布到3了，但一些流行的组件兼容性可能不好，所以依然用最稳定的2.8版

《《《《《《《《《《《《《《《《《《《《《《《《《《《《

创建mywebsite时你可能会遇到这样的警告

```
PHP Warning:  date(): It is not safe to rely on the system's timezone settings. You are *required* to use the date.timezone setting or the date_default_timezone_set() function. blublu...
```

这是因为你的php.ini文件没有指定时区，修改/etc/php.ini文件，把

```
;date.timezone =
```

改成

```
date.timezone = Asia/Shanghai
```

注意：要去掉前面的";"

重启php-fpm

```
[root@centos7vm httpdir]# service php-fpm restart
```

》》》》》》》》》》》》》》》》》》》》》》》》》》》》》》》

有没有发现在httpdir目录下出现了mywebsite目录？走进去瞧瞧symfony2已经为你做了什么

```
[root@centos7vm mywebsite]# ls
app  bin  composer.json  composer.lock  README.md  src  vendor  web
```

web目录是网站的入口，假设你的网站叫做www.shareditor.com，那么www.shareditor.com/test.php一般会通过nginx配置为对应web/test.php文件（当然这都需要你的ngixn配置文件配合，如果你非要配置成web目录的上一级也可以，只不过网站不能用而已，呵呵）。一般会把可以通过url访问的文件放到这里，比如css、js、图片、静态html文件等

app目录是全局应用的运行相关文件（包括入口函数、组件加载、全局配置、log文件、cache文件等），这里面还有一个重要二进制文件console，用来方便我们做自动化操作，后面会讲到

src是你自己开发php代码的地方，一般以组件形式整理，symfony2里叫做bundle，一会再说bundle是什么

vendor是symfony2自带组件放置的地方，也是最庞大的地方，一般不需要我们修改里面的内容，使用里面哪个功能就可以在app目录下的配置文件里配置，不使用也无妨，缺什么组件也可以手工安装组件到vendor下，安装方法马上便知

bin目录里是symfony2提供给我们的一些二进制工具

composer.json是这个网站工程里用到symfony2中组件的配置和版本等信息

好！什么都不用改，我们现在去改下nginx的配置，让网站指向web目录，这里需要看一下如何配置nginx实现虚拟主机

nginx.conf如下：

```
user  nobody;
worker_processes  1;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;
    client_max_body_size 20M;
    include /etc/nginx/conf.d/*.conf;
}
```

mywebsite.conf如下：

```
log_format logformat '$remote_addr - $remote_user [$time_local] "$request" '
'$status $body_bytes_sent "$http_referer" '
'"$http_user_agent" "$http_x_forwarded_for"';

server {
    listen       80;
    server_name  172.16.142.130;
    index app.php;
    root /data/httpdir/mywebsite/web;
    location / {
        if (!-e $request_filename){
            rewrite ^/(.+)$ /app.php/$1 last;
        }
        client_max_body_size 20M;
    }
    location ~ ^/(app|app_dev)\.php(/|$) {
        fastcgi_pass   127.0.0.1:9000;
        fastcgi_split_path_info ^(.+\.php)(/.*)$;
        include fastcgi_params;
        fastcgi_param  SCRIPT_FILENAME    $document_root$fastcgi_script_name;
        fastcgi_param  HTTPS              off;
        client_max_body_size 20M;
    }
    location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
    {
        expires 1h;
        client_max_body_size 20M;
    }

    access_log /data/httpdir/logs/mywebsite.log logformat;
    error_log /data/httpdir/logs/mywebsite.error_log;
}
```

因为系统自带的service命令起动服务有些无法理解的现象，所以我们以后都采取直接启动nginx，强制杀掉所有nginx进程后，执行

```
[root@centos7vm mywebsite]# nginx
```

以后如果想重启nginx可以用命令

```
[root@centos7vm mywebsite]# nginx -s restart
```

其实如果只是改配置，可以调用reload命令

```
[root@centos7vm mywebsite]# nginx -s reload
```

为了简单，把php-fpm的子进程用户也改到nobody，修改/etc/php-fpm.d/www.conf，把里面的

```
; RPM: apache Choosed to be able to access some dir as httpd
user = apache
; RPM: Keep a group allowed to write in log dir.
group = apache
```

改成

```
; RPM: apache Choosed to be able to access some dir as httpd
user = nobody
; RPM: Keep a group allowed to write in log dir.
group = nobody
```

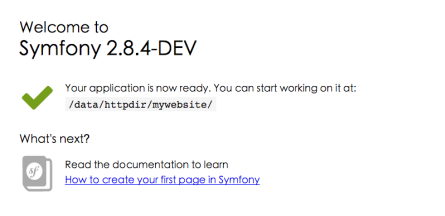
因为系统自带的service命令起动服务有些无法理解的现象，所以我们以后都采取直接启动php-fpm，强制杀掉所有php-fpm进程后，执行

```
[root@centos7vm mywebsite]# php-fpm
```

以后如果想重启php-fpm可以用命令：

```
[root@centos7vm mywebsite]# kill -USR2 `cat /var/run/php-fpm/php-fpm.pid`
```

OK，现在可以打开http://127.0.0.1/啦，看到了什么？



恭喜你，你已经进入了神奇的symfony2的世界

教你成为全栈工程师(Full Stack Developer) 八-10行代码建起你的网站

发表于 2016-04-20 08:59

从这节开始，我们来完成一个完善的博客网站，我们每一节都会实现一点小功能，逐步完成。基于symfony2搭建网站都是采用MVC设计模式，本文用最少的代码带你感受一下symfony2里的MVC是怎么使用的。

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

小插曲1——开发模式与发布模式

我们做c开发或java开发都会提到debug和release，symfony2也是一样有debug和release，区别在于这么几点：

1. debug模式在执行出错时提示到页面上，release没有，这对调试程序非常有帮助

2. debug模式不用cache缓存，release用，开发时用debug可以避免改了代码不生效的奇怪问题

3. debug模式附加了问题分析工具小插件，在页面上可以看到，可以借助它做优化

4. debug模式访问的url一般是http://127.0.0.1/app_dev.php/***，release模式直接就是http://127.0.0.1/***

小插曲2——日志目录和cache目录

日志目录一般在/data/httpdir/mywebsite/app/logs/目录下，cache目录一般在/data/httpdir/mywebsite/app/cache下

一般当我们代码升级用debug模式调试ok了之后，需要清理一下cache才能用release模式访问，清理方式有两种：

1.

```
[root@centos7vm mywebsite]# cd /data/httpdir/mywebsite
[root@centos7vm mywebsite]# php app/console cache:clear
```

2.

```
[root@centos7vm mywebsite]# rm -rf /data/httpdir/mywebsite/app/cache/*
```

因为这两个目录是会在网站运行过程中写入内容的，所以需要把这两个目录的权限改成：

```
[root@centos7vm mywebsite]# chmod -R 777 /data/httpdir/mywebsite/app/cache/
[root@centos7vm mywebsite]# chmod -R 777 /data/httpdir/mywebsite/app/logs/
```

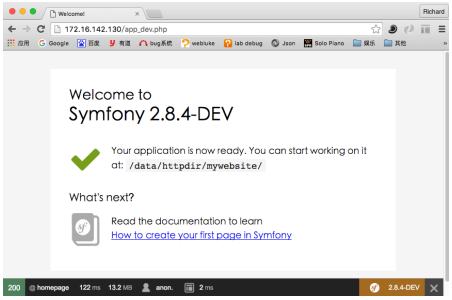
小插曲3——外网访问debug模式

debug模式默认只支持本地访问，不支持外部访问的，原因就是安全的考虑啦，那么对于我们在虚拟机里开发又想在宿主机访问（相当于外部访问）怎么才能开启debug模式呢？

修改/data/httpdir/mywebsite/web/app_dev.php文件，注释掉如下几行代码就好啦：

```
if (isset($_SERVER['HTTP_CLIENT_IP'])
    || isset($_SERVER['HTTP_X_FORWARDED_FOR'])
    || !(in_array(@$_SERVER['REMOTE_ADDR'], array('127.0.0.1', 'fe80::1', '::1')) || php_sapi_name() === 'cli-serv
er')
) {
    header('HTTP/1.0 403 Forbidden');
    exit('You are not allowed to access this file. Check '.basename(__FILE__).' for more information.');
}
```

下面在宿主机访问虚拟机的ip试一下，比如我的虚拟机ip是172.16.142.130，那么访问效果如下：



小插曲4——什么是Bundle

bundle英文释义是捆，也就是把一些内容捆绑在一起的意思，在symfony2中它就相当于一个组件，各种自带的组件也都是以bundle形式组织的，我们自己开发的内容也可以组织成bundle，便于复用。bundle里一般包含MVC的所有内容以及Resource和config

做一个静态页面

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

进入到/data/httpdir/mywebsite/app/Resources/views/目录并创建一个blog目录，并在blog里创建list.html.twig文件内容如下：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">
<!-- 上述3个meta标签*必须*放在最前面，任何其他内容都*必须*跟随其后！ -->
<title>MyWebSite</title>
<!-- 新 Bootstrap 核心 CSS 文件 -->
<link rel="stylesheet" href="//cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap.min.css">
<!-- 可选的Bootstrap主题文件（一般不用引入） -->
<link rel="stylesheet" href="//cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap-theme.min.css">
<!-- jQuery文件。务必在bootstrap.min.js 之前引入 -->
<script src="//cdn.bootcss.com/jquery/1.11.3/jquery.min.js"></script>
<!-- 最新的 Bootstrap 核心 JavaScript 文件 -->
<script src="//cdn.bootcss.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
</head>
<body>
<div class="row bg-primary">
    <div class="col-sm-1 col-xs-1"></div>
    <div class="col-sm-11 col-xs-11"><h1><a href="" style="text-decoration: none;color: white;">MyWebSite</a
></h1></div>
</div>
<div class="row jumbotron">
    <div class="col-md-1 col-xs-1"></div>
    <div class="col-md-10 col-xs-10"><h1>Welcome to MyWebSite!</h1></div>
    <div class="col-md-1 col-xs-1"></div>
</div>
</body>
</html>
```

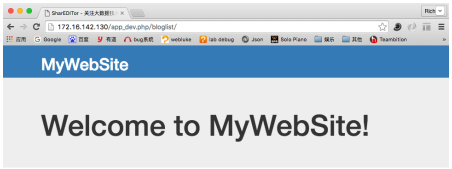
修改/data/httpdir/mywebsite/app/config/routing.yml文件，添加如下行：

```
blog_list:
    path:     /bloglist/
    defaults: { _controller: AppBundle:Blog:list }
```

创建/data/httpdir/mywebsite/src/AppBundle/Controller/BlogController.php文件内容如下：

```
<?php

namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class BlogController extends Controller
{
    public function listAction(Request $request)
    {
        return $this->render('blog/list.html.twig');
    }
}
```

下面访问http://172.16.142.130/app_dev.php/bloglist/看下效果



欣赏一下你的高大上的网站吧

讲解一下symfony2里的MVC

注：本节内容只包括了MVC里的V和C，下一节会涉及到M的部分

先从app/config/routing.yml说起，这里面定义了网站不同的url路径所对应的controller，比如

```
blog_list:
    path:     /bloglist/
    defaults: { _controller: AppBundle:Blog:list }
```

blog_list：表示这个配置组名字叫做blog_list(名字其实无所谓，只要不和其他组起成相同名字就行)

path：网站后面的url路径，比如/bloglist/指的就是http://127.0.0.1/bloglist

defaults：访问bloglist路径时响应你的controller是谁，这里设置的是执行AppBundler下的BlogController的listAction方法

接下来就看这个AppBundler下的BlogController（这就是MVC里的C）的listAction方法干了什么，打开src/AppBundle/Controller/BlogController.php文件，方法如下：

```
    public function listAction(Request $request)
    {
        return $this->render('blog/list.html.twig');
    }
```

render的含义是显示/渲染/输出，也就是说这个action要用blog/list.html.twig这个模板的内容来返回给你数据展示出来

下面继续看blog/list.html.twig（这就是MVC里的V），这是一个静态html文件，具体标签含义可以尽情百度一下，涉及到的一些前端知识后面章节也会详细讲，大体内容就是：展示一个菜单栏和一个大宣传文本

留个悬念：为什么这个静态网页一定要以twig作为扩展名呢？预知原因如何，且听下回分解。

教你成为全栈工程师(Full Stack Developer) 九-让模板文件帮你快速开发网页

发表于 2016-04-21 08:45

模板顾名思义是用来复制的样本，那么模板文件同样具有这样的特性：可继承、可重载、可嵌套，同时具备一些其他优点：流程控制、变量传递、链接自动生成等。html模板给我们开发带来怎样的好处呢？这一节我们继续改进我们的网站来说明模板的作用

再做一张页面

新增/data/httpdir/mywebsite/app/Resources/views/blog/show.html.twig文件，内容如下：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">
<!-- 上述3个meta标签*必须*放在最前面，任何其他内容都*必须*跟随其后！ -->
<title>博客内容</title>

<!-- 新 Bootstrap 核心 CSS 文件 -->
<link rel="stylesheet" href="//cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap.min.css">

<!-- 可选的Bootstrap主题文件（一般不用引入） -->
<link rel="stylesheet" href="//cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap-theme.min.css">

<!-- jQuery文件。务必在bootstrap.min.js 之前引入 -->
<script src="//cdn.bootcss.com/jquery/1.11.3/jquery.min.js"></script>

<!-- 最新的 Bootstrap 核心 JavaScript 文件 -->
<script src="//cdn.bootcss.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>

</head>
<body>

<div class="row bg-primary">
    <div class="col-sm-1 col-xs-1"></div>
    <div class="col-sm-11 col-xs-11"><h1><a href="" style="text-decoration: none;color: white;">MyWebSite</a
></h1></div>
</div>

<div class="row jumbotron">
    <div class="col-md-1 col-xs-1"></div>
    <div class="col-md-10 col-xs-10"><h1>我的第一篇博客</h1></div>
    <div class="col-md-1 col-xs-1"></div>
</div>

</body>
</html>
```

在/data/httpdir/mywebsite/app/config/routing.yml中新增如下内容(注意缩进)：

```
blog_show:
    path:     /blogshow/
    defaults: { _controller: AppBundle:Blog:show }
```

在/data/httpdir/mywebsite/src/AppBundle/Controller/BlogController.php的BlogController类中增加一个新方法，如下：

```
    public function showAction(Request $request)
    {
        return $this->render('blog/show.html.twig');
    }
```

这时打开http://172.16.142.130/app_dev.php/blogshow/可以看到一张新网页（注意：这里的172.16.142.130对应改成你虚拟机的ip）

那么这时候你会发现show.html.twig和list.html.twig除了<title>标签里的内容和body里的一个div内容不同之外完全一样，如果一个网站有成百上千张网页都大体相同，你会想到什么？对！把一样的部分包装成一个组件，

使用它的网页把他include进来，但是html是一层一层标签的嵌套结构，include适合面向函数的语言。所以就产生了模板这种抽象方式。

模板抽象

我们修改app/Resources/views/base.html.twig文件内容如下：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">
<!-- 上述3个meta标签*必须*放在最前面，任何其他内容都*必须*跟随其后！ -->
<title>{% block title %}自定义标题{% endblock title %}</title>

<!-- 新 Bootstrap 核心 CSS 文件 -->
<link rel="stylesheet" href="//cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap.min.css">

<!-- 可选的Bootstrap主题文件（一般不用引入） -->
<link rel="stylesheet" href="//cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap-theme.min.css">

<!-- jQuery文件。务必在bootstrap.min.js 之前引入 -->
<script src="//cdn.bootcss.com/jquery/1.11.3/jquery.min.js"></script>

<!-- 最新的 Bootstrap 核心 JavaScript 文件 -->
<script src="//cdn.bootcss.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>

</head>
<body>

<div class="row bg-primary">
    <div class="col-sm-1 col-xs-1"></div>
    <div class="col-sm-11 col-xs-11"><h1><a href="" style="text-decoration: none;color: white;">MyWebSite</a
></h1></div>
</div>

{% block body %}
{% endblock body %}

</body>
</html>
```

可以看到这里面相比前面两个文件出现了这样两句：

```
<title>{% block title %}自定义标题{% endblock title %}</title>
```

和

```
{% block body %}
{% endblock body %}
```

这里的block就是可以重载的部分，那么我们现在可以修改list.html.twig了，改成如下：

```
{% extends "base.html.twig" %}

{% block title %}MyWebSite{% endblock title %}

{% block body %}
<div class="row jumbotron">
    <div class="col-md-1 col-xs-1"></div>
    <div class="col-md-10 col-xs-10"><h1>Welcome to MyWebSite!</h1></div>
    <div class="col-md-1 col-xs-1"></div>
</div>
{% endblock body %}
```

修改show.html.twig，改成如下：

```
{% extends "base.html.twig" %}

{% block title %}博客内容{% endblock title %}

{% block body %}
<div class="row jumbotron">
    <div class="col-md-1 col-xs-1"></div>
    <div class="col-md-10 col-xs-10"><h1>我的第一篇博客</h1></div>
    <div class="col-md-1 col-xs-1"></div>
</div>
{% endblock body %}
```

这时候在浏览器重新浏览http://172.16.142.130/app_dev.php/bloglist/和http://172.16.142.130/app_dev.php/blogshow/效果是不是和之前一样，但是代码简单了许多

变量传递

在MVC三个部分之controller里面我们调用render来渲染html的内容，那么你不禁会问：怎么把controller里的变量传递到html里来使用呢？下面我们来做个试验，修改src/AppBundle/Controller/BlogController.php文件，把

```
return $this->render('blog/show.html.twig');
```

改成

```
return $this->render('blog/show.html.twig', array('title' => '博客标题', 'content' => '博客内容'));
```

多传递一个php数组，数组里存了title和content

然后在show.html.twig中block body部分改成

```
<div class="row jumbotron">
    <div class="col-md-1 col-xs-1"></div>
    <div class="col-md-10 col-xs-10"><h1>{{ title }}</h1></div>
    <div class="col-md-1 col-xs-1"></div>
</div>
<div class="row">
    <div class="col-md-1 col-xs-1"></div>
    <div class="col-md-10 col-xs-10"><h4>{{ content }}</h4></div>
    <div class="col-md-1 col-xs-1"></div>
</div>
```

重新浏览http://172.16.142.130/app_dev.php/blogshow/看到我们在php代码里指定的title和content展示了出来

变量传递通过 {{ }} 这样的符号来传递

tips 1: 以刚刚传递的content变量为例，如果传递的content是一个array结构，加入里面有time、category、text等字段，那么，可以这样读取：{{ content['time'] }}

tips 2: 如果content是一个类结构的实例，这个类有time、category、text等属性（也就是有getTime()、getCategory()、getText()方法），那么可以这样读取：{{ content.time }}

tips 3: {{ }} 里还可以调用php原生方法，如：去掉 html标签方法{{ content | strip_tags }}、以原始html形式展示方法{{ content | raw }}

流程控制

有想过在html里面使用if else或者for循环吗？模板可以帮你，再次修改src/AppBundle/Controller/BlogController.php文件，把

```
return $this->render('blog/show.html.twig', array('title' => '博客标题', 'content' => '博客内容'));
```

改成

```
        $content = array();
        $content[] = array('category' => '类别1', 'text' => '内容1');
        $content[] = array('category' => '类别2', 'text' => '内容2');
        $content[] = array('category' => '类别1', 'text' => '内容3');
        $content[] = array('category' => '类别2', 'text' => '内容4');
        return $this->render('blog/show.html.twig', array('title' => '博客标题', 'content' => $content));
```

修改app/Resources/views/blog/show.html.twig，把第二组div改成：

```
{% for cont in content %}
{% if cont['category'] == '类别2' %}
<div class="row">
    <div class="col-md-1 col-xs-1"></div>
    <div class="col-md-10 col-xs-10"><h4>{{ cont['text'] }}</h4></div>
    <div class="col-md-1 col-xs-1"></div>
</div>
{% endif %}
{% endfor %}
```

浏览http://172.16.142.130/app_dev.php/blogshow/看到了什么效果？

传递过来的content是个数组，首先用{% for cont in content %}对content做了循环遍历，之后在循环体里用{% if cont['category'] == '类别2' %}做了逻辑判断，只展示类别为“类别2”的内容

链接生成

如果我们希望点击“内容”都能跳转到bloglist页面，怎么做呢？首先想到我们可以增加个超链接，像这样：

```
<a href='http://172.16.142.130/app_dev.php/bloglist/'>{{ cont['text'] }}</a>
```

但是这有一个问题：当网站发布时是不是还要改成

```
<a href='http://www.****.com/bloglist/'>{{ cont['text'] }}</a>
```

这是不够灵活的，所以模板为我们提供了链接生成的功能，还记得router.yml的内容吗？

```
app:
    resource: "@AppBundle/Controller/"
    type:     annotation

blog_list:
    path:     /bloglist/
    defaults: { _controller: AppBundle:Blog:list }

blog_show:
    path:     /blogshow/
    defaults: { _controller: AppBundle:Blog:show }
```

这里面的blog_list、blog_show这种名字开始起作用了，我们可以把“内容”改成如下：

```
<a href='{{ path('blog_list') }}'>{{ cont['text'] }}</a>
```

这样便可以自动根据实际环境生成blog_list对应的正确链接，amazing！

以上模板功能对于基本的网站开发足够了，如果想了解更多内容可以百度一下twig

教你成为全栈工程师(Full Stack Developer) 十-MVC模式中的Model使用方法

发表于 2016-04-24 17:49

model是MVC里的M，它是MVC里最简单而又复杂的部分，简单是因为有了它你就不必关注数据层，让你把精力集中在业务逻辑，说它复杂是因为要想实现这样的机制是比较困难的，好在symfony2帮我们实现了最困难的部分，连在model中编写业务需要的字段都做成了自动化，当然为了学习我们暂时不用自动化的工具。这一节我们就用model来把博客真正管理起来。

创建一个model定义

创建src/AppBundle/Entity/BlogPost.php文件并编写如下内容：

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * BlogPost
 *
 * @ORM\Table()
 * @ORM\Entity
 */
class BlogPost
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="title", type="string", length=255)
     */
    private $title;

    /**
     * @var string
```

```
     *
     * @ORM\Column(name="body", type="text")
     */
    private $body;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="create_time", type="datetime")
     */
    private $createTime;

    /**
     * Get id
     *
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * Set title
     *
     * @param string $title
     * @return BlogPost
     */
    public function setTitle($title)
    {
        $this->title = $title;

        return $this;
    }

    /**
     * Get title
     *
     * @return string
     */
    public function getTitle()
    {
        return $this->title;
    }

    /**
     * Set body
     *
     * @param string $body
     * @return BlogPost
     */
    public function setBody($body)
    {
        $this->body = $body;

        return $this;
    }

    /**
     * Get body
     *
     * @return string
     */
    public function getBody()
    {
        return $this->body;
```

```
        return $this->body;
    }

    /**
     * @return \DateTime
     */
    public function getCreateTime()
    {
        return $this->createTime;
    }

    /**
     * @param \DateTime $createTime
     */
    public function setCreateTime($createTime)
    {
        $this->createTime = $createTime;
    }
}
```

BlogPost这个model从含义上表达了一篇博客，从实现上表达了数据库表的一行，有id、title、body、createTime几个属性，其中id是数据库的主键（自增1，不需要setId方法），title是博客的标题，body是博客内容，createTime是博客创建时间

下面我们就要自动化地根据这个model定义生成数据库表，在此之前，我们要把数据库配置好

安装和配置mysql见《RHEL7或centos7安装mysql5.7方法和配置》

配置网站数据库连接

修改app/config/parameters.yml文件，按照你自己的数据库配置改成如下：

```
parameters:
    database_host: 127.0.0.1
    database_port: 3306
    database_name: mywebsite
    database_user: root
    database_password: shareditor@126.COM
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_password: null
    secret: 1a0cb131fb193436d0f6ce467f2d8b6c7c5b02da
```

mailer部分和secret不知道是做什么的就保持不动

注意：这里我们设置了一个数据库名字：mywebsite，后面会创建这个数据库

下面该是隆重介绍app/console工具的时候了，这是symfony2工程都会带的一个自动化工具，直接执行

```
[root@centos7vm mywebsite]# php app/console
```

可以看帮助信息，它的功能包括：自动管理缓存、配置、路由、entity等

这次我们用它来初始化数据库，执行

```
[root@centos7vm mywebsite]# php app/console doctrine:database:create
```

通过mysql查看发现已经建立了mywebsite数据库

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| mywebsite          |
| performance_schema |
| sys                |
+--------------------+
```

还记得之前我们之前创建的entity吗？对，就是那个class BlogPost，现在我们用自动化工具把它直接映射成mysq的一张表，执行

```
[root@centos7vm mywebsite]# php app/console doctrine:schema:update --force
```

通过mysql查看发现已经成功建立了

```
mysql> use mywebsite;
mysql> show tables;
+--------------------+
| Tables_in_mywebsite |
+--------------------+
| blog_post          |
+--------------------+
mysql> desc blog_post;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| id          | int(11)      | NO   | PRI | NULL    | auto_increment |
| title       | varchar(255) | NO   |     | NULL    |                |
| body        | longtext     | NO   |     | NULL    |                |
| create_time | datetime     | NO   |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
```

而且表的列和BlogPost中的成员变量一致，(*@ o @*) 哇～amazing!有木有

下面我们手工往数据库里插入一行数据

```
mysql> insert into blog_post(title,body,create_time) values('这是标题','这是内容',now());
```

model的读取

修改src/AppBundle/Controller/BlogController.php中的showAction方法如下：

```
    public function showAction(Request $request)
    {
        $blogPostRepository = $this->getDoctrine()->getRepository('AppBundle:BlogPost');
        $blogposts = $blogPostRepository->findAll();
        return $this->render('blog/show.html.twig', array('title' => $blogposts[0]->getTitle(), 'content' => $blogposts[0
]->getBody()));
    }
```

这里面的getRepository('AppBundle:BlogPost')就是隐藏了和数据库的交互细节，暴露给你的就是$blogPostRepository这个对象，直接可以做各种find操作，直接能拿到blog_post表的数据

findAll是取出所有结果，所以我用了$blogposts[0]来只拿第一个结果演示

我们还要修改一下app/Resources/views/blog/show.html.twig如下：

```
{% extends "base.html.twig" %}

{% block title %}博客内容{% endblock title %}

{% block body %}
<div class="row jumbotron">
    <div class="col-md-1 col-xs-1"></div>
    <div class="col-md-10 col-xs-10"><h1>{{ title }}</h1></div>
    <div class="col-md-1 col-xs-1"></div>
</div>

<div class="row">
    <div class="col-md-1 col-xs-1"></div>
    <div class="col-md-10 col-xs-10"><h4>{{ content }}</h4></div>
    <div class="col-md-1 col-xs-1"></div>
</div>

{% endblock body %}
```

模板里面展示{{ title }}和{{ content }}，对应的就是array('title' => $blogposts[0]->getTitle(), 'content' => $blogposts[0]->getBody())取出来的title和content

现在看下网页的效果吧：

这是内容

上面展示了model的读取流程：数据库表=>model类实例=>controller透传=>前端展示

model的写入方式也类似，后面章节会通过更高级的功能来继续介绍

教你成为全栈工程师(Full Stack Developer) 十一-轻松搭建网站管理后台

发表于 2016-04-25 08:12

互联网上见到的多数网站都有一个你看不见的更强大的管理后台支持，比如一个新闻网站的管理后台一定有新闻编辑、发布、审核、管理等，一个论坛网站的管理后台一定有用户管理、板块管理、帖子审核等。这一节我们见识一下php框架的真正魅力：数行代码实现的强大管理后台

sonata介绍

sonata扩展是symfony2众多扩展中应用最广泛的扩展之一，它的主要功能是帮你建立一个强大的管理后台，除此之外还有很多附加功能你可以深入挖掘，官方文档在https://sonata-project.org/bundles/admin/2-3/doc/index.html

composer扩展管理工具

为了安装symfony2的扩展，我们需要一个composer工具，它的安装方法（参考https://getcomposer.org/download/，以下命令如若失效，请以官方最新文档为准）为连续执行以下命令：

```
[root@centos7vm ~]# php -r "readfile('https://getcomposer.org/installer');" > composer-setup.php
[root@centos7vm ~]# php -r "if (hash_file('SHA384', 'composer-setup.php') === '7228c001f88bee97506740ef0888240bd8a760b046ee16db8f4095c0d8d525f2367663f22a46b48d072c816e7fe19959') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
[root@centos7vm ~]# php composer-setup.php
[root@centos7vm ~]# php -r "unlink('composer-setup.php');"
```

然后你会发现当前目录下有一个composer.phar文件，我们把它移动到/usr/local/bin下并重命名：

```
[root@centos7vm ~]# mv composer.phar /usr/local/bin/composer
```

ok，安装完成，执行

```
[root@centos7vm ~]# composer
```

看下是否输出帮助信息

安装sonata-admin

先看一下工程目录下的composer.json这个文件，这里面记录了当前工程都依赖哪些扩展包及其版本信息，这个文件一般我们不去编辑，当用composer更新扩展包时会自动更新这个文件

执行：

```
[root@centos7vm mywebsite]# composer require sonata-project/admin-bundle "2.3.*"
```

安装sonata的admin-bundle，因为admin-bundle会依赖一些其他bundle，所以安装过程可能会比较慢，需要

耐心等待，当输出有：

```
[OK] All assets were successfully installed.
```

说明安装完成

为了让admin-bundle能操作我们的数据库，还需要安装doctrine-orm-admin-bundle，执行：

```
[root@centos7vm mywebsite]# composer require sonata-project/doctrine-orm-admin-bundle "2.3.*"
```

安装了sonata扩展，相当于往vendor目录里拷贝了一批文件，但实际上我们还没有真正使用上，如果要使用起来，需要修改app/AppKernel.php把要使用的组件注册进来才能生效（所有的symfony2组件的安装都需要这个过程），修改app/AppKernel.php中的registerBundles()函数，在$bundles = array(...最后添加：

```
            new Sonata\CoreBundle\SonataCoreBundle(),
            new Sonata\BlockBundle\SonataBlockBundle(),
            new Knp\Bundle\MenuBundle\KnpMenuBundle(),
            new Sonata\DoctrineORMAdminBundle\SonataDoctrineORMAdminBundle(),
            new Sonata\AdminBundle\SonataAdminBundle(),
```

注：如果某个bundle已经注册过，则不需要重复注册

配置sonata-admin

sonata-admin的接口都是基于SonataBlockBundle，按block组织在一起的，所以必须先告诉blockbundle，sonata-admin这个block的存在，所以修改*app/config/config.yml*

添加如下配置(注意缩进)：

```
sonata_block:
    default_contexts: [cms]
    blocks:
        # enable the SonataAdminBundle block
        sonata.admin.block.admin_list:
            contexts: [admin]
```

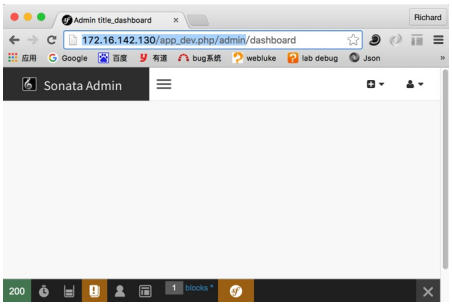
block配置完还需要给admin-bundle指定路由，这样才能通过url访问，admin-bundle是有自己的一套路由配置的，我们只需要加载进来即可，修改app/config/routing.yml，添加如下内容：

```
admin_area:
    resource: "@SonataAdminBundle/Resources/config/routing/sonata_admin.xml"
    prefix: /admin
```

OK，清缓存：

```
[root@centos7vm mywebsite]# php app/console cache:clear
[root@centos7vm mywebsite]# php app/console assets:install
```

访问http://172.16.142.130/app_dev.php/admin看看（这里的172.16.142.130是我的虚拟机ip，需要换成你的ip，另外如果访问不了可以尝试手工清缓存rm -rf app/cache/*），效果如下：



创建BlogPost的后台管理

创建src/AppBundle/Admin/BlogPostAdmin.php文件，内容如下：

```
<?php

namespace AppBundle\Admin;

use Sonata\AdminBundle\Admin\Admin;
use Sonata\AdminBundle\Datagrid\ListMapper;
use Sonata\AdminBundle\Form\FormMapper;
use Sonata\AdminBundle\Datagrid\DatagridMapper;

class BlogPostAdmin extends Admin
{
    protected function configureFormFields(FormMapper $formMapper)
    {
        $formMapper
            ->add('title', 'text')
            ->add('body', 'text')
            ->add('create_time', 'datetime');
    }

    protected function configureListFields(ListMapper $listMapper)
    {
        $listMapper
            ->addIdentifier('title')
            ->add('createTime')
            ;
    }
}
```

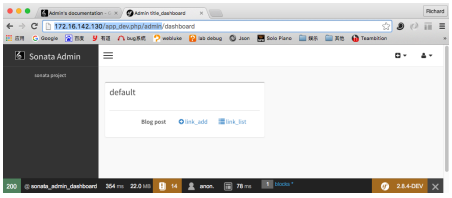
创建了BlogPostAdmin还不能让symfony2知道，所以还需要注册一下，修改app/config/services.yml，在services组中添加服务：

```
    admin.blog_post:
        class: AppBundle\Admin\BlogPostAdmin
        arguments: [~, AppBundle\Entity\BlogPost, ~]
        tags:
            - { name: sonata.admin, manager_type: orm, label: Blog post }
```

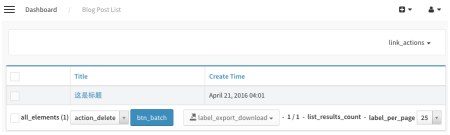
这样还不够，还需要配置好有关自定义Admin类的路由，修改app/config/routing.yml，添加：

```
_sonata_admin:
    resource: .
    type: sonata_admin
    prefix: /admin
```

OK，重新打开http://172.16.142.130/app_dev.php/admin看到如下：

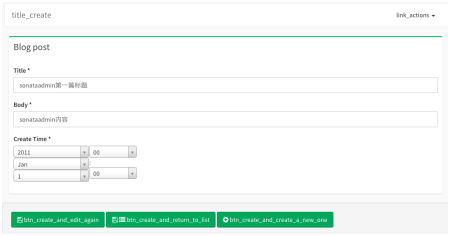


点开"link_list"看到了什么？

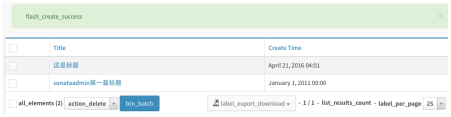


(*@ o @*) 哇～，是我们手工在数据库里添加进去的那一行哎！

点击“link_add"进去添加一条记录试试，我们添加如下一行：



点击”btn_create_adn_return_to_list“后就成功写到了数据库里啦



到此，我们已经实现了对数据库表的增删改查的管理功能

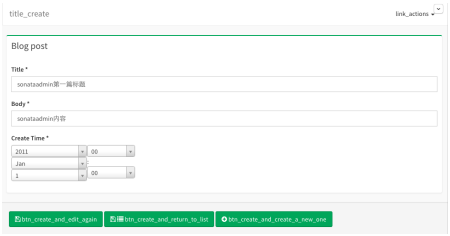
教你成为全栈工程师(Full Stack Developer) 十二-SonataAdmin管理后台轻松配置

发表于 2016-04-26 08:34

sonata-admin管理后台不只是简单地把数据库表的管理映射到网页上，它有非常多的支持对各种数据类型的编辑组件，本节来举几个例子，包括文本编辑插件ckeditor的支持，它能让你像编辑word文档一样所见即所得，还包括时间选择器，让你方便填写时间类型。

特殊字段的困扰

上节中我们编辑一个blogpost时是这样的：



这里的body本应该有图文并茂的内容，但是我们只能输入一些纯文本，createTime的填写也非常不方便，下面我们来解决这个问题

ckeditor插件支持

首先安装MediaBundle扩展和SonataFormatterBundle扩展，执行：

```
[root@centos7vm mywebsite]# composer require sonata-project/media-bundle "2.3.*"
[root@centos7vm mywebsite]# composer require sonata-project/formatter-bundle "2.3.*"
```

注册bundle，修改app/AppKernel.php，添加注册如下：

```
            new Ivory\CKEditorBundle\IvoryCKEditorBundle(),
            new Sonata\FormatterBundle\SonataFormatterBundle(),
            new Knp\Bundle\MarkdownBundle\KnpMarkdownBundle(),
            new Sonata\MediaBundle\SonataMediaBundle(),
            new JMS\SerializerBundle\JMSSerializerBundle(),
            new Sonata\EasyExtendsBundle\SonataEasyExtendsBundle(),
            new Application\Sonata\MediaBundle\ApplicationSonataMediaBundle(),
```

修改app/config/config.yml，把doctrine配置组改成如下的样子：

```
doctrine:
    dbal:
        driver:   pdo_mysql
        host:     "%database_host%"
        port:     "%database_port%"
        dbname:   "%database_name%"
        user:     "%database_user%"
        password: "%database_password%"
        charset:  UTF8
        types:
            json: Sonata\Doctrine\Types\JsonType

    orm:
        auto_generate_proxy_classes: "%kernel.debug%"
        entity_managers:
            default:
                mappings:
                    AppBundle:
                        type:       ~
                        dir:        "Entity"
                        prefix:     "AppBundle\Entity"
                        is_bundle: ~
```

并添加如下配置：

```
sonata_formatter:
    formatters:
        markdown:
            service: sonata.formatter.text.markdown
            extensions:
                - sonata.formatter.twig.control_flow
                - sonata.formatter.twig.gist
                - sonata.media.formatter.twig

        text:
            service: sonata.formatter.text.text
            extensions:
                - sonata.formatter.twig.control_flow
                - sonata.formatter.twig.gist
                - sonata.media.formatter.twig

        rawhtml:
            service: sonata.formatter.text.raw
            extensions:
                - sonata.formatter.twig.control_flow
                - sonata.formatter.twig.gist
                - sonata.media.formatter.twig

        richhtml:
            service: sonata.formatter.text.raw
            extensions:
                - sonata.formatter.twig.control_flow
                - sonata.formatter.twig.gist
                - sonata.media.formatter.twig

        twig:
            service: sonata.formatter.text.twigengine
            extensions: [] # Twig formatter cannot have extensions
    ckeditor:
        templates:
            browser: 'SonataFormatterBundle:Ckeditor:browser.html.twig'
            upload: 'SonataFormatterBundle:Ckeditor:upload.html.twig'

sonata_media:
    default_context: default
```

```
    default_context: default
    db_driver: doctrine_orm # or doctrine_mongodb, doctrine_phpcr
    contexts:
        default:  # the default context is mandatory
            providers:
                - sonata.media.provider.dailymotion
                - sonata.media.provider.youtube
                - sonata.media.provider.image
                - sonata.media.provider.file

            formats:
                small: { width: 100 , quality: 70}
                big:   { width: 500 , quality: 70}

    cdn:
        server:
            path: /uploads/media # http://media.sonata-project.org/

    filesystem:
        local:
            directory:  %kernel.root_dir%/../web/uploads/media
            create:     false

ivory_ck_editor:
    default_config: default
    configs:
        default:
            filebrowserBrowseRoute: admin_sonata_media_media_ckeditor_browser
            filebrowserImageBrowseRoute: admin_sonata_media_media_ckeditor_browser
                # Display images by default when clicking the image dialog browse button
            filebrowserImageBrowseRouteParameters:
                provider: sonata.media.provider.image
            filebrowserUploadRoute: admin_sonata_media_media_ckeditor_upload
            filebrowserUploadRouteParameters:
                provider: sonata.media.provider.file
                    # Upload file as image when sending a file from the image dialog
            filebrowserImageUploadRoute: admin_sonata_media_media_ckeditor_upload
            filebrowserImageUploadRouteParameters:
                provider: sonata.media.provider.image
                context: my-context # Optional, to upload in a custom context
```

修改app/config/routing.yml，添加如下内容：

```
gallery:
    resource: '@SonataMediaBundle/Resources/config/routing/gallery.xml'
    prefix: /media/gallery

media:
    resource: '@SonataMediaBundle/Resources/config/routing/media.xml'
    prefix: /media
```

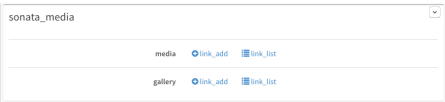
OK, 清缓存(不行就rm -rf app/cache/*)

```
[root@centos7vm mywebsite]# php app/console cache:clear
```

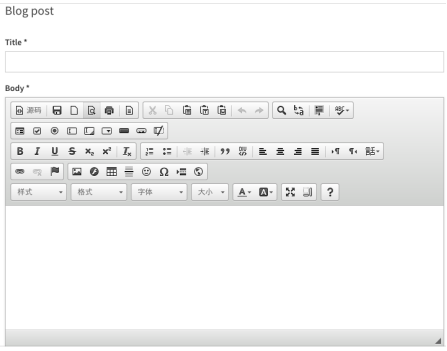
安装ckeditor静态文件到web目录：

```
[root@centos7vm mywebsite]# php app/console assets:install web
```

这时重新打开http://172.16.142.130/app_dev.php/admin，你应该会看到多出了下面的板块



这是用来管理图片视频等信息的板块，但尚未配置好还不能用，暂时也不需要，我们继续来看我们需要的内容，还是点BlogPost的link_add，看效果：



这回body可以用来编辑图文并茂的内容啦

但虽然这里可以编辑文本格式，但是如果要上传图片还不能用，我们再来准备些东西

执行：

```
[root@centos7vm mywebsite]# php app/console sonata:easy-extends:generate --dest=src SonataMediaBundle
```

如果执行成功，会自动生成src/Application/Sonata/MediaBundle，说明我们成功生成了我们自定义的MediaBundle，但是我们一行代码都不需要写

在app/AppKernel.php中注册

```
            new Sonata\IntlBundle\SonataIntlBundle(),
            new Application\Sonata\MediaBundle\ApplicationSonataMediaBundle(),
```

修改app/config/config.yml，把doctrine的orm组改成如下：

```
    orm:
        auto_generate_proxy_classes: "%kernel.debug%"
        entity_managers:
            default:
                mappings:
                    ApplicationSonataMediaBundle: ~
                    SonataMediaBundle: ~
                    AppBundle:
                        type:       ~
                        dir:        "Entity"
                        prefix:     "AppBundle\Entity"
                        is_bundle: ~
```

更新数据库表：

```
[root@centos7vm mywebsite]# php app/console doctrine:schema:update --force
```

创建图片视频等上传目录：

```
[root@centos7vm mywebsite]# mkdir web/uploads
[root@centos7vm mywebsite]# mkdir web/uploads/media
[root@centos7vm mywebsite]# chmod -R 0777 web/uploads
```

好，大功告成，我们试一下上传图片吧，点击ckeditor（上面的编辑器）的图片按钮

点击上传选择文件

选好文件，点上传到服务器后，显示

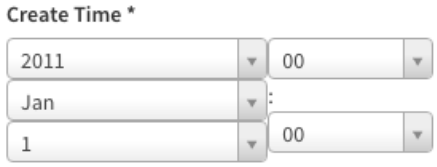
这时已经把图片上传到服务器的web/uploads/media目录下了，点确定就可以插入到我们要编辑的内容里了

ckeditor还有很多丰富的内容可以配置，具体可以参考官方文档，比如代码高亮显示，像下面的样子：

```
public function showAction(Request $request)
{
    return $this->render('blog/show.html.twig');
}
```

漂亮的时间选择器

下面我们来解决时间类型数据的填写问题，我们现在的时间选择是这样的难看难用：



我们来介绍一种sonata_type_date_picker类型的时间选择器，它是SonataCore中自带的组件

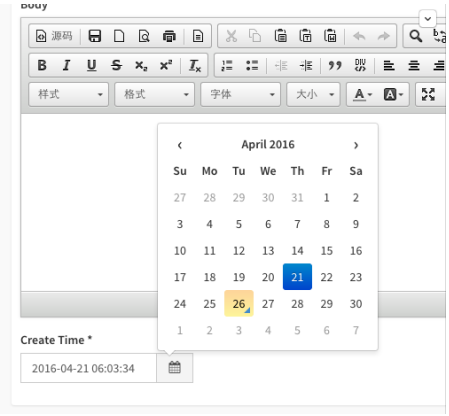
在app/config/config.yml的twig组下添加模板配置：

```
form:
    resources:
        - 'SonataCoreBundle:Form:datepicker.html.twig'
```

修改src/AppBundle/Admin/BlogPostAdmin.php的configureFormFields方法，如下：

```
    $formMapper
        ->add('title', 'text')
        ->add('body', 'ckeditor', array('autoload' => true))
        ->add('create_time', 'sonata_type_date_picker', array(
            'format'=>'yyyy-MM-dd HH:mm:ss',
            'dp_default_date'         => date('Y-m-d H:i:s'),));
```

在管理后台重新编辑一个blog看效果：



教你成为全栈工程师(Full Stack Developer) 十三-用表关联结构实现类目管理

发表于 2016-04-27 08:27

一切事物都是有联系的，联系在一起的事物必定成结构性。比如博客多了就需要归类，映射到数据库表上就是关联关系，映射到model上就是多对一关系的一个成员变量。本节就介绍怎么配置sonata-admin实现博客的分类

以终为始

想象一下我们希望有怎么样的分类，比如你是一个音乐爱好者，那么你的博客可能会分成：摇滚、流行、古典……；如果你是一个体育爱好者，那么可能会分成：篮球赛事、户外运动、健身……；如果是一个IT爱好者，可能会分成：大数据、数学知识、信息检索……。那么首先我们需要的是一个类目表。

创建src/AppBundle/Entity/Subject.php这个model，如下：

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use Doctrine\Common\Collections\ArrayCollection;

/**
 * Subject
 *
 * @ORM\Table()
 * @ORM\Entity
 */
class Subject
{
    /**
     * @ORM\OneToMany(targetEntity="BlogPost", mappedBy="subject")
     */
    private $blogPosts;

    public function __construct()
    {
        $this->blogPosts = new ArrayCollection();
    }

    public function getBlogPosts()
    {
        return $this->blogPosts;
    }


    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="name", type="string", length=255)
     */
    private $name;
```

```
    /**
     * @var string
     *
     * @ORM\Column(name="photo", type="string", length=255)
     */
    private $photo;
    /**
     * @var string
     *
     * @ORM\Column(name="introduce", type="string", length=255)
     */
    private $introduce;

    /**
     * @return string
     */
    public function getIntroduce()
    {
        return $this->introduce;
    }

    /**
     * @param string $introduce
     */
    public function setIntroduce($introduce)
    {
        $this->introduce = $introduce;
    }

    /**
     * Get id
     *
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * Set name
     *
     * @param string $name
     * @return Subject
     */
    public function setName($name)
    {
        $this->name = $name;

        return $this;
    }

    /**
     * Get name
     *
     * @return string
     */
    public function getName()
    {
        return $this->name;
    }

    /**
     * Set photo
     *
```

```
     * @param string $photo
     * @return Subject
     */
    public function setPhoto($photo)
    {
        $this->photo = $photo;

        return $this;
    }

    /**
     * Get photo
     *
     * @return string
     */
    public function getPhoto()
    {
        return $this->photo;
    }
}
```

这里的private $blogPosts;是一个一对多（OneToMany）的关系类型，@ORM声明了targetEntity="BlogPost", mappedBy="subject"，表示$blogPosts只能存储BlogPost类型的数据，并且这些数据都是以BlogPost::subject来做hash的，那么就需要BlogPost必须有subject成员变量，下面我们来添加，修改src/AppBundle/Entity/BlogPost.php，添加：

```
    /**
     * @ORM\ManyToOne(targetEntity="Subject", inversedBy="blogPosts")
     */
    private $subject;
```

并添加get和set方法：

```
    public function setSubject(Subject $subject)
    {
        $this->subject = $subject;
    }

    public function getSubject()
    {
        return $this->subject;
    }
```

这里的private $subject;是一个多对一(ManyToOne)的关系类型，@ORM声明了targetEntity="Subject", inversedBy="blogPosts"，表示$subject存储的是Subject类型的数据，并和Subject::blogPosts有对应关系

添加SubjectAdmin管理类，添加src/AppBundle/Admin/SubjectAdmin.php文件，内容如下：

```
<?php

namespace AppBundle\Admin;

use Sonata\AdminBundle\Admin\Admin;
use Sonata\AdminBundle\Datagrid\ListMapper;
use Sonata\AdminBundle\Datagrid\DatagridMapper;
use Sonata\AdminBundle\Form\FormMapper;

class SubjectAdmin extends Admin
{
    protected function configureFormFields(FormMapper $formMapper)
    {
        $formMapper->add('name', 'text')
            ->add('introduce', 'text')
            ->add('photo', 'text');
    }

    protected function configureDatagridFilters(DatagridMapper $datagridMapper)
    {
        $datagridMapper->add('name');
    }

    protected function configureListFields(ListMapper $listMapper)
    {
        $listMapper->addIdentifier('name')
            ->add('introduce')
            ->add('photo');
    }
}
```

执行如下语句来创建subject表和更新blogpost表：

```
[root@centos7vm mywebsite]# php app/console doctrine:schema:update --force
```


修改app/config/services.yml注册admin.subject服务，在services组添加：

```
    admin.subject:
        class: AppBundle\Admin\SubjectAdmin
        arguments: [~, AppBundle\Entity\Subject, ~]
        tags:
            - { name: sonata.admin, manager_type: orm, label: Subject }
```

这时打开http://172.16.142.130/app_dev.php/admin看到多出了subject的管理界面：

我们添加一个subject，如下：

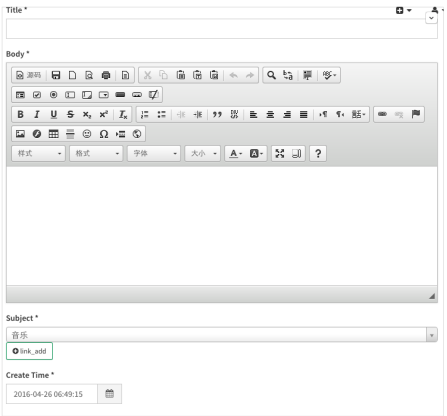
保存后，看到我们生成了一个subject数据：

这里的photo先随便填一个，以后再用到这个字段

如果你打算新建一个blogpost来指定一个subject，那么会发现，在blogpost的编辑界面没有选择科目的地方，这时怎么办呢？我们还需要修改一下BlogPostAdmin管理类，修改src/AppBundle/Admin/BlogPostAdmin.php，如下：

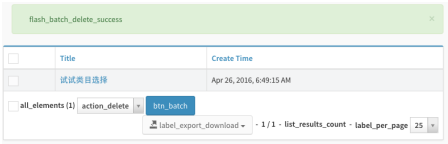
```
$formMapper
    ->add('title', 'text')
    ->add('body', 'ckeditor', array('autoload' => true))
    ->add('subject', 'sonata_type_model', array(
        'class' => 'AppBundle\Entity\Subject',
        'property' => 'name',
    ))
    ->add('create_time', 'sonata_type_date_picker', array(
        'format'=>'yyyy-MM-dd HH:mm:ss',
        'dp_default_date'         => date('Y-m-d H:i:s'),));
```

这时打开blogpost编辑界面看到：



多出了subject选择框

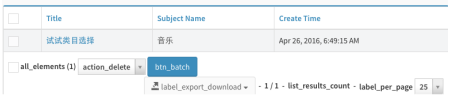
我们新建一篇看下：



列表中列出了这篇blog，但是没有看到类目呢？这是因为我们没有配置显示subject字段，继续修改src/AppBundle/Admin/BlogPostAdmin.php的configureListFields方法，内容如下：

```
$listMapper
    ->addIdentifier('title')
    ->add('subject.name')
    ->add('createTime')
    ;
```

这时候再看blog列表：



成功啦

下面的任务就留给你自己：用同样的方式来添加category，用来表示另一个层级的分类，比如：原创、转载、学习笔记……，这样你的文章可以是：“大数据”类别下的“原创”文章。

提示一下：

第一步添加如下文件：src/AppBundle/Entity/Category.php、src/AppBundle/Admin/CategoryAdmin.php

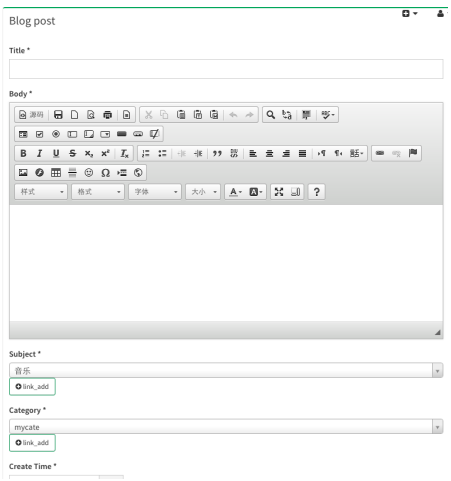
第二步修改如下文件：src/AppBundle/Entity/BlogPost.php、src/AppBundle/Admin/BlogPostAdmin.php

第三步修改如下配置：app/config/services.yml

第四步更新数据库：php app/console doctrine:schema:update --force

最终效果如下：

添加博客页面



博客列表页面

教你成为全栈工程师(Full Stack Developer) 十四-实现漂亮的网站管理后台

发表于 2016-04-27 08:58

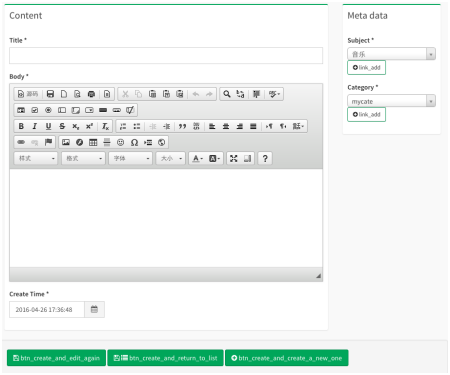
如果你按照前几章节的内容一步一步建设管理后台，你会发现在编辑BlogPost的界面中每一项都是堆叠在一起的，假如增加非常多的管理项，那么这个页面会拉的很长，又难看又不好管理，这时你是否想到可不可以把界面重新组织一下呢？这一节为你提供一个简单的方法重构界面

重组BlogPost编辑界面

打开src/AppBundle/Admin/BlogPostAdmin.php文件，修改configureFormFields方法，增加几行，像下面的样子：

```
$formMapper
    ->with('Content', array('class' => 'col-md-9'))
    ->add('title', 'text')
    ->add('body', 'ckeditor', array('autoload' => true))
    ->add('create_time', 'sonata_type_date_picker', array(
        'format'=>'yyyy-MM-dd HH:mm:ss',
        'dp_default_date'         => date('Y-m-d H:i:s'),))
    ->end()

    ->with('Meta data', array('class' => 'col-md-3'))
    ->add('subject', 'sonata_type_model', array(
        'class' => 'AppBundle\Entity\Subject',
        'property' => 'name',
    ))
    ->add('category', 'sonata_type_model', array
        'class' => 'AppBundle\Entity\Category',
        'property' => 'name',
    ))
    ->end();
```


看下BlogPost编辑界面：



我们把有关分类的项目单独作为一列放到右侧，并且占据相对狭窄的宽度，是不是看起来井井有条了？

原理讲解

我们其实只增加了这样几行：

```
        ->with('Content', array('class' => 'col-md-9'))
    ......
        ->end()
    ......
        ->with('Meta data', array('class' => 'col-md-3'))
    ......
        ->end()
```

这是一种布局方式，with和end成对出现，它表示把其间的项目组织成一个组布局在一起，Content和Meta data都是组显示的名称，而col-md-9和col-md-3是bootstrap框架的内容，表示两个组放到一行里，列的宽度比例是9:3，有关bootstrap的详细内容后面再详细讨论

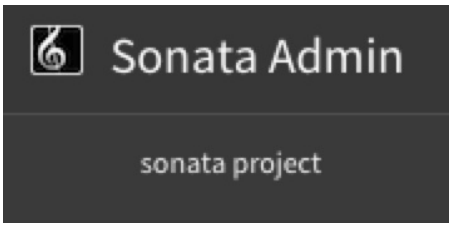
教你成为全栈工程师(Full Stack Developer) 十五-做一个完美的管理后台侧边栏

发表于 2016-04-28 09:21

在sonata管理后台除了我们常用的部分之外，还有侧栏的内容我们没有涉及过，这一节讲几个简单的配置来让我们管理后台内容更丰富更易用

把管理后台据为己有

我们的管理后台左上角logo部分显示的内容是这样的：



是不是有点low了？我想改个名字叫做“后台管理系统”怎么办呢？非常简单，打开app/config/parameters.yml，添加如下配置：

```
sonata_admin:
    title: 后台管理系统
```

重新打开后台界面看到什么了？

简直太简单了

侧栏快捷入口

sonata-admin都是通过block组织的，所以侧栏也可以通过配置block修改，我们修改app/config/config.yml中的sonata_block配置组，改成：

```
sonata_block:
    default_contexts: [cms]
    blocks:
        # enable the SonataAdminBundle block
        sonata.admin.block.admin_list:
            contexts: [admin]
        sonata.admin.block.search_result:
            contexts: [admin]
        sonata.user.block.menu:    # used to display the menu in profile pages
        sonata.user.block.account: # used to display menu option (login option)
```


并添加如下：

```
sonata_admin:
    templates:
        layout: SonataAdminBundle::my_layout.html.twig
```

我们创建我们自己的模板文件app/Resources/SonataAdminBundle/views/my_layout.html.twig，内容如下：

```
{% extends 'SonataAdminBundle::standard_layout.html.twig' %}

{% block side_bar_after_nav %}
    <br/>
    <p class="text-center"><a href="{{ path('homepage') }}">前往首页</a></p>
{% endblock %}
```

看下效果：

在这里你可以任意定制你自己的菜单

教你成为全栈工程师(Full Stack Developer) 十六-网站主页设计

发表于 2016-04-29 07:51

前面几个章节我们做了充分的准备工作，从本节开始，我们开始从幕后走出来，研究给用户看到的部分，也就是网站页面。这里除了技术上的能力外，还需要你的设计天赋和审美能力喽！

找到默认主页入口

我们知道页面入口都是配置在路由中的，我们来看下app/config/routing.yml发现没有“/”的路由，但是我们发现了这么几句：

```
app:
    resource: "@AppBundle/Controller/"
    type:     annotation
```

annotation的意思是“注解”，也就是说这一部分路由配置放在了注释里面，而资源在@AppBundle/Controller/，那么我看下src/AppBundle/Controller/目录

```
[root@centos7vm mywebsite]# ls src/AppBundle/Controller/
BlogController.php  DefaultController.php
```

这里面的BlogController.php是我们自己写的，没有什么annotation的东东。那我们直接看下src/AppBundle/Controller/DefaultController.php，这个文件是在创建工程时自动生成的

```
<?php

namespace AppBundle\Controller;

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

class DefaultController extends Controller
{
    /**
     * @Route("/", name="homepage")
     */
    public function indexAction(Request $request)
    {
        // replace this example code with whatever you need
        return $this->render('default/index.html.twig', array(
            'base_dir' => realpath($this->container->getParameter('kernel.root_dir').'/..'),
        ));
    }
}
```

这里面的

```
    /**
     * @Route("/", name="homepage")
     */
```

意思是说声明了一个路由配置，路由的名字叫做homepage，路由的路径是"/"，当被访问时要执行DefaultController::indexAction方法，至此我们又学会了一种路由配置方法annotation，但个人不推荐这种配置，因为配置分散，不方便统一管理

我们把这段代码改的单纯一点：

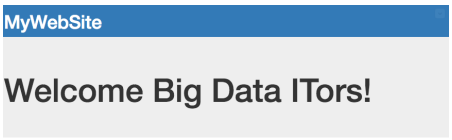
```
    public function indexAction(Request $request)
    {
        return $this->render('default/index.html.twig');
    }
```

主页模板

清掉app/Resources/views/default/index.html.twig，并改成如下：

```
{% extends "base.html.twig" %}

{% block body %}

<div class="row jumbotron">
    <div class="col-md-1 col-xs-1"></div>
    <div class="col-md-10 col-xs-10"><h1>Welcome Big Data ITors!</h1></div>
    <div class="col-md-1 col-xs-1"></div>
</div>

{% endblock body %}
```

浏览下主页



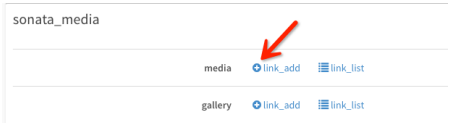
我们熟悉的内容又回来啦

在主页中展示科目列表

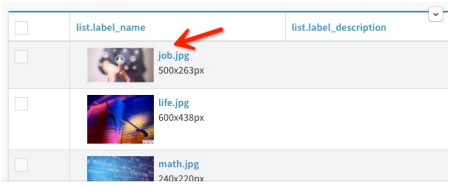
下面我们把Subject的分类展示在主页中，首先我们在管理后台创建这样几个Subject：

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

其中的Photo是你上传的封面图片的地址，举个例子，在



上传一个image后，在图片列表中点击



找到

这个地址，并把这个地址填写到Subject的Photo中就行了

修改src/AppBundle/Controller/DefaultController.php，改成：

```
public function indexAction(Request $request)
{
    $this->subjectRepository = $this->getDoctrine()->getRepository('AppBundle:Subject');
    $subjects = $this->subjectRepository->findAll();

    return $this->render('default/index.html.twig', array(
        'subjects' => $subjects
    ));
}
```

修改app/Resources/views/default/index.html.twig，如下：

```
{% extends "base.html.twig" %}

{% block body %}

<div class="row jumbotron">
    <div class="col-md-1 col-xs-1"></div>
    <div class="col-md-10 col-xs-10"><h1>Welcome Big Data ITors!</h1></div>
    <div class="col-md-1 col-xs-1"></div>
</div>
<div class="row">
    <div class="col-sm-1 col-xs-1"></div>

    {% for subject in subjects %}
    <div class="col-sm-2 col-xs-12">
        <a href="{{ path('blog_list', {'subjectId':subject.id}) }}" class="thumbnail">
            <img src="{{ subject.photo }}" alt="{{ subject.name }}">

            <div class="caption">
                <h3>{{ subject.name }}</h3>

                <p>{{ subject.introduce }}</p>
            </div>
        </a>
    </div>
    {% endfor %}

    <div class="col-sm-1 col-xs-1"></div>
</div>

<br />
<br />


{% endblock body %}
```

见证奇异的时刻到了，打开http://172.16.142.130/app_dev.php/，你会不禁哇塞一下，高大上的内容出来啦：

是不是很漂亮的说

下一节我们继续改进我们的主页内容，让他更专业一些

教你成为全栈工程师(Full Stack Developer) 十七-网站分类列表页面设计

发表于 2016-04-29 08:47

上一节我们完成了高大上的主页，并展示了每个分类，那么按照常理，用户点击分类项怎么能展示出这一分类的博客列表呢？我们这一节来制作分类列表页

创建一些文章

首先，为了能展示我们的效果，我们先在后台创建几篇文章，例如如下：

安装分页插件

列表性质的页面一般都需要做分页处理，这个工作要是让我们自己处理是十分繁琐的，所以我们利用symfony2的扩展knp-paginator-bundle。

修改app/AppKernel.php文件，增加如下注册语句：

```
new Knp\Bundle\PaginatorBundle\KnpPaginatorBundle(),
```

修改app/config/config.yml，增加如下配置：

```
knp_paginator:
    page_range: 5                      # default page range used in pagination control
    default_options:
        page_name: page                # page query parameter name
        sort_field_name: sort          # sort field query parameter name
        sort_direction_name: direction # sort direction query parameter name
        distinct: true                 # ensure distinct results, useful when ORM queries are using GROUP BY stateme
nts
    template:
        pagination: KnpPaginatorBundle:Pagination:twitter_bootstrap_v3_pagination.html.twig     # sliding paginatio
n controls template
        sortable: KnpPaginatorBundle:Pagination:sortable_link.html.twig # sort link template
```

解释一下，KnpPaginator是一个分页插件，配置项中page_range是默认的每一页的条目数，其他几个配置项可以不用详细了解

pagination: KnpPaginatorBundle:Pagination:twitter_bootstrap_v3_pagination.html.twig这个是配置分页中底部页码部分的样式模板，有几种可以选择：

- KnpPaginatorBundle:Pagination:sliding.html.twig (by default)
- KnpPaginatorBundle:Pagination:twitter_bootstrap_v3_pagination.html.twig
- KnpPaginatorBundle:Pagination:twitter_bootstrap_pagination.html.twig

- KnpPaginatorBundle:Pagination:foundation_v5_pagination.html.twig

修改controller

修改src/AppBundle/Controller/BlogController.php，把其中的listAction函数内容改成：

```
    public function listAction(Request $request, $subjectId)
    {
        $em    = $this->get('doctrine.orm.entity_manager');
        $dql   = "SELECT a FROM AppBundle:BlogPost a WHERE a.subject=" . $subjectId . " ORDER BY a.creat
eTime DESC";
        $query = $em->createQuery($dql);

        $paginator  = $this->get('knp_paginator');
        $pagination = $paginator->paginate(
            $query,
            $request->query->get('page', 1)/*page number*/,
            10/*limit per page*/
        );

        return $this->render('blog/list.html.twig', array('pagination' => $pagination));
    }
```

解释一下，注意这次listAction参数里除了$request之外，多出了一个$subjectId，这是和后面要配置的路由相对应的，表示会在url里传过来一个subjectId的数值。

看$em这一行，在symfony2中对model的操作都需要先获取到EntityManager进行操作

第二行$dql赋值是一条sql语句，这里面根据$subjectId来从数据库里取BlogPost，并且按照createTime来排序，这里面需要注意的是：sql语句中写的变量名不是数据库表的字段名（小写），而是model类里定义的成员名

$pagination这一行则是调用分页插件的函数，默认取第一页，最后一个参数10表示每页展示多少条，这个会冲掉app/config/config.yml配置的5，这里面的$pagination是一个对象数组，每个成员都是一个BlogPost这个model的对象

修改模板

修改app/Resources/views/blog/list.html.twig，把内容改成：

```
{% extends "base.html.twig" %}

{% block body %}

<div class="row">
    <div class="col-sm-3 col-xs-1"></div>
    <div class="col-sm-6 col-xs-10">

        <br />
        {% for article in pagination %}
        <div style="background-color: whitesmoke; padding-left: 30px;padding-right: 30px;padding-top: 10px;">
            <div class="row">
                <div class="col-sm-2 col-xs-2"></div>
                <div class="col-sm-8 col-xs-12">
                    <div class="row">
                        <h3>{{ article.title }}</h3>
                    </div>
                </div>
                <div class="col-sm-2"></div>
            </div>
            <br />
        </div>
        <br />
        {% endfor %}
        <div class="navigation">
            {{ knp_pagination_render(pagination) }}
        </div>

    </div>
    <div class="col-sm-3 col-xs-1"></div>
</div>

{% endblock body %}
```

解释一下，pagination是一个对象数组，for循环遍历每一个对象，通过{{ article.title }}获取它的title属性展示出来

下面的{{ knp_pagination_render(pagination) }}是分页插件特有的功能，就是底部页码的按钮部分

配置路由

修改app/config/routing.yml，把曾经添加过的：

```
blog_list:
    path:     /bloglist/
    defaults: { _controller: AppBundle:Blog:list }
```

改成：

```
blog_list:
    path:     /bloglist/{subjectId}
    defaults: { _controller: AppBundle:Blog:list }
```

解释一下，这里的{subjectId}会自动传到controller中，并以参数形式传到listAction里，见上面listAction函数的实现

好，现在打开http://172.16.142.130/app_dev.php/bloglist/4，（这里的4是Subject的id，你可以查看你的数据库看你的subject的id都有哪些值）

从首页点进来

我们先打开首页http://172.16.142.130/app_dev.php，点击一个图标，就能直接进到博客列表页，这是怎么做到的呢？其实我们在上一节已经做好了相关准备，看下app/Resources/views/default/index.html.twig里的这几句：

```
<a href="{{ path('blog_list', {'subjectId':subject.id}) }}" class="thumbnail">
    <img src="{{ subject.photo }}" alt="{{ subject.name }}">

    <div class="caption">
        <h3>{{ subject.name }}</h3>

        <p>{{ subject.introduce }}</p>
    </div>
</a>
```

这里面的a标签已经把跳转链接写成了{{ path('blog_list', {'subjectId':subject.id}) }}，它会自动从路由配置里找到名字为blog_list的路由，并把subject.id的值传递到subjectId变量中

下一节我们继续对博客列表也做一些美化，然后开发博客的展示页

教你成为全栈工程师(Full Stack Developer) 十八-让网站分类列表页变得更漂亮

发表于 2016-04-29 09:32

爱美之心人皆有之，对于一个追求完美的你来说，丑陋的列表页实在是看不下去，这一节对列表也做一些美化，让它不那么丑陋，或者说有一点点的美感

添加图片属性

有图有真相！一篇好的博客怎么能没有漂亮的图片来搭配呢？我们来在BlogPost中增加图片属性，之后展示到列表页和博客内容页中。

修改src/AppBundle/Entity/BlogPost.php，增加如下成员变量：

```
/**
 * @var Media
 *
 * @ORM\ManyToOne(targetEntity="Application\Sonata\MediaBundle\Entity\Media")
 */
private $image;
```

同时增加如下get和set方法：

```
/**
 * @return Media
 */
public function getImage()
{
    return $this->image;
}

/**
 * @param Media $image
 */
public function setImage($image)
{
    $this->image = $image;
}
```

执行

```
[root@centos7vm mywebsite]# php app/console doctrine:schema:update --force
```

更新数据库

这里你不必担心已经写入的数据库被冲掉，它其实调用的是mysql的alter语句，加一列不会删数据

添加图片属性的管理项

修改src/AppBundle/Admin/BlogPostAdmin.php中的configureFormFields方法，在

```
        ->add('title', 'text')
```

下面加入一句：

```
        ->add('image', 'sonata_type_model', array(
            'property' => 'name'
        ))
```

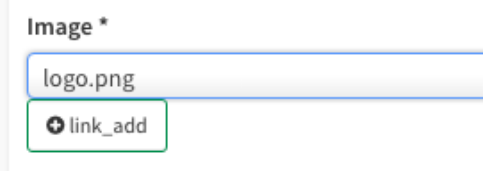
并修改configureListFields方法，在

```
        ->add('category.name')
```

后面加入一句：

```
            ->add('image.name')
```

OK，重新进入管理后台，编辑一个Blog看到多处了这样的编辑项：



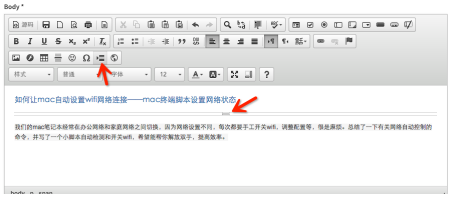
这里图片可以选择，也可以添加，我们把已有的每一篇文章都添加一张不同的图片，最终像下面的样子：

文章摘要生成

每篇博客可能会很长，但是希望它能在列表页中展示一个摘要文本，通过文章生成摘要是一个long story啊，太难了，所以我们干脆在每篇文章撰写时先写一段摘要放到篇头，然后用一个分割线表明下面才是真正内容。好，我们修改src/AppBundle/Entity/BlogPost.php文件里的BlogPost类，添加一个getAbstract方法，如下：

```
/**
 * body去标签然后截取前多少个字,在twig中使用方法:{{ article.abstract }}
 * @return string
 */
public function getAbstract()
{
    $pos = strpos($this->body, '<div style="page-break-after');
    if ($pos > 0) {
        return strip_tags(substr($this->body, 0, $pos));
    } else {
        return strip_tags(mb_substr($this->body, 0, 100, "utf-8")) . '...';
    }
}
```

讲解一下，这里的'<div style="page-break-after'是经过实践得出的一个分割线的标识，也就是我们在博客编辑框（ckeditor）里面输入这个东西的效果：



strip_tags是php自带的函数，用来去掉html标签，保留文本内容。

总结起来这个getAbstract的功能就是把BlogPost的body内容在分割线前面的部分提出来，并提出其中的纯文本内容，如果body中没有找到分割线，那么就截取前100个字符。

文章创建时间规范化

我们数据库中文章创建时间是DateTime类型，也就是年月日时分秒，我们在页面展示时并不需要这么详细，所以对这个字段做一些简化处理，同样在BlogPost类中添加如下方法：

```
/**
 * 创建时间转成字符串才能展示
 * @return string
 */
public function getCreateTimeStr()
{
    $newDate = $this->createTime->format('Y-m-d H:i');
    return $newDate;
}
```

修改模板文件

重新修改app/Resources/views/blog/list.html.twig，内容改成如下：

```
{% extends "base.html.twig" %}

{% block body %}

<div class="row">
    <div class="col-sm-3 col-xs-1"></div>
    <div class="col-sm-6 col-xs-10">

        <br />
        {% for article in pagination %}
            <div style="background-color: whitesmoke; padding-left: 30px;padding-right: 30px;padding-top: 10px;">
                <div class="row">
                    <div class="col-sm-3 col-xs-10">
                        <a href="{{ path('blog_show', {'blogId':article.id}) }}">
                            <img style="width: 100%; height: 100%; margin:10%;" src="/uploads/media/default/0001/01/t
humb_{{ article.image.id }}_default_small.{{ article.image.extension }}">
                        </a>
                    </div>
                    <div class="col-sm-1 col-xs-2"></div>
                    <div class="col-sm-7 col-xs-12">
                        <div class="row">
                            <h3><a href="{{ path('blog_show', {'blogId':article.id}) }}">{{ article.title }}</a></h3>
                        </div>
                        <div class="row">
                            <a class="btn btn-info btn-xs" href="{{ path('blog_list', {'subjectId':article.subject.id}) }}">
                                {{ article.subject.name }}
                            </a>
                            <a class="btn btn-success btn-xs" href="{{ path('blog_list', {'subjectId':article.subject.id}) }}">
                                {{ article.category.name }}
                            </a>
                            <small>发表于 {{ article.createTimeStr }}</small>
                        </div>
                        <br />
                        <div class="row">
                            <small>{{ article.abstract }}</small>
                        </div>
                    </div>
                    <div class="col-sm-1"></div>
                </div>
                <br />
            </div>
            <br />
        {% endfor %}
        <div class="navigation">
            {{ knp_pagination_render(pagination) }}
        </div>

    </div>
    <div class="col-sm-3 col-xs-1"></div>
</div>

{% endblock body %}
```

编辑我们的文章，都添加上分割线，重新打开我们的网站，进到博客列表页，看下效果还不错

教你成为全栈工程师(Full Stack Developer) 十九-文章内容展示页面设计

发表于 2016-05-01 07:08

前面已经对整个网站的布局做了周密的准备，下面就是最关键内容的展示了，完成了这一部分网站也就基本搞成了。本节我们把文章内容展示部分完成。

修改controller

修改src/AppBundle/Controller/BlogController.php，把BlogController类的showAction改成：

```
public function showAction($blogId)
{
    $this->blogPostRepository = $this->getDoctrine()->getRepository('AppBundle:BlogPost');
    return $this->render('blog/show.html.twig', array('blogpost' => $this->blogPostRepository->find($blogId)));
}
```

解释一下

$blogId是BlogPost的id号，用来在url中传入参数，后面会通过路由配置来对应。

render函数通过参数传入的$this->blogPostRepository->find($blogId)是通过BlogPost这个model获取id为$blogId的文章的BlogPost实例

修改模板

修改app/Resources/views/blog/show.html.twig，把内容清掉并改成：

```
{% extends "base.html.twig" %}

{% block body %}

<div class="container-fluid">
    <div class="row">
        <div class="col-sm-3 col-xs-1"></div>
        <div class="col-sm-6 col-xs-10">
            <div class="row">
                <h1>{{ blogpost.title }}</h1>
            </div>
            <div class="row">
                <a class="btn btn-info btn-xs" href="{{ path('blog_list', {'subjectId':blogpost.subject.id}) }}">
                    {{ blogpost.subject.name }}
                </a>
                <a class="btn btn-success btn-xs" href="{{ path('blog_list', {'subjectId':blogpost.subject.id}) }}">
                    {{ blogpost.category.name }}
                </a>
                <small>发表于 {{ blogpost.createTimeStr }}</small>
            </div>

            <div class="row">
                <hr/>
            </div>
            <div class="row">
                <div class="row">
                    <div class="col-sm-12 col-xs-12">
                        <img style="width: 100%; height: 100%;" src="/uploads/media/default/0001/01/thumb_{{ blogpo
st.image.id }}_default_big.{{ blogpost.image.extension }}">
                    </div>
                </div>
                <br/>
                {{ blogpost.body|raw }}
            </div>
        </div>
        <div class="col-sm-3 col-xs-1"></div>
    </div>
</div>

{% endblock %}
```

配置路由

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

修改app/config/routing.yml，把之前写过的：

```
blog_show:
    path:     /blogshow/
    defaults: { _controller: AppBundle:Blog:show }
```

改成：

```
blog_show:
    path:     /blogshow/{blogId}
    defaults: { _controller: AppBundle:Blog:show }
```

打开博客列表页，点击一篇文章标题，看下效果吧

总结

至此，我们的网站内容已经达到可以发布上线的状态了，方法可以参考前面章节讲过的服务器那些事，你可以申请一个阿里云服务器，注册个域名，部署上去，就像我的网站www.shareditor.com一样，以后只需要在后台管理管理文章就行了

教你成为全栈工程师(Full Stack Developer) 二十-管理后台的权限控制

发表于 2016-05-05 08:04

如果你按照前面的章节一步一步实现到现在，应该可以在管理后台编辑和发布你的博客文章了，但是如果别人知道你的链接，也可以编辑、发布甚至删除你的文章，这时就需要你做权限控制了，也就是能够支持输入用户名密码登陆的用户管理部分

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

SonataUserBundle

SonataUserBundle是sonata项目中有关用户管理的部分，它其实是集成了FOS/UserBundle组件（感兴趣可以去git上找，但个人觉得直接用SonataUserBundle就够了）并增添了一些功能，使用SonataUserBundle需要安装如下扩展，执行：

```
[root@centos7vm mywebsite]# composer require sonata-project/user-bundle
```

并修改app/AppKernel.php，增加如下组件的注册：

```
            new FOS\UserBundle\FOSUserBundle(),
            new Sonata\UserBundle\SonataUserBundle('FOSUserBundle'),
```

修改配置

修改app/config/config.yml，增加如下配置：

```
fos_user:

    db_driver:      orm
    firewall_name:  main
    user_class:     Sonata\UserBundle\Entity\BaseUser

    group:
        group_class:   Sonata\UserBundle\Entity\BaseGroup
```

并找到对应配置组添加如下内容：

```
doctrine:
    orm:
        entity_managers:
            default:
                mappings:
                    SonataUserBundle: ~
```

修改app/config/security.yml，改成如下样子：

```
security:

    role_hierarchy:
        ROLE_ADMIN:       [ROLE_USER, ROLE_SONATA_ADMIN]
        ROLE_SUPER_ADMIN: [ROLE_ADMIN, ROLE_ALLOWED_TO_SWITCH]
        SONATA:
            - ROLE_SONATA_PAGE_ADMIN_PAGE_EDIT  # if you are using acl then this line must be commente
d


    # http://symfony.com/doc/current/book/security.html#where-do-users-come-from-user-providers
    providers:
        fos_userbundle:
            id: fos_user.user_manager


    firewalls:
        # disables authentication for assets and the profiler, adapt it according to your needs
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false

        # -> custom firewall for the admin area of the URL
        admin:
            pattern:            /admin(.*)
            context:            user
            form_login:
                provider:       fos_userbundle
                login_path:     /admin/login
                use_forward:    false
                check_path:     /admin/login_check
                failure_path:   null
            logout:
                path:           /admin/logout
            anonymous:          true

        # -> end custom configuration

        main:
```

```
            pattern:              .*
            context:              user
            form_login:
                provider:       fos_userbundle
                login_path:     /login
                use_forward:    false
                check_path:     /login_check
                failure_path:   null
            logout:             true
            anonymous:          true

    access_control:
        # URL of FOSUserBundle which need to be available to anonymous users
        - { path: ^/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/register, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/resetting, role: IS_AUTHENTICATED_ANONYMOUSLY }

        # Admin login page needs to be access without credential
        - { path: ^/admin/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/admin/logout$, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/admin/login_check$, role: IS_AUTHENTICATED_ANONYMOUSLY }

        # Secured part of the site
        # This config requires being logged for the whole site and having the admin role for the admin part.
        # Change these rules to adapt them to your needs
        - { path: ^/admin/, role: [ROLE_ADMIN, ROLE_SONATA_ADMIN] }
        - { path: ^/.*, role: IS_AUTHENTICATED_ANONYMOUSLY }

    encoders:
        FOS\UserBundle\Model\UserInterface: sha512

    acl:
        connection: default
```


修改app/config/routing.yml，添加如下内容：

```
sonata_user_security:
    resource: "@SonataUserBundle/Resources/config/routing/sonata_security_1.xml"

sonata_user_resetting:
    resource: "@SonataUserBundle/Resources/config/routing/sonata_resetting_1.xml"
    prefix: /resetting

sonata_user_profile:
    resource: "@SonataUserBundle/Resources/config/routing/sonata_profile_1.xml"
    prefix: /profile

sonata_user_register:
    resource: "@SonataUserBundle/Resources/config/routing/sonata_registration_1.xml"
    prefix: /register

sonata_user_change_password:
    resource: "@SonataUserBundle/Resources/config/routing/sonata_change_password_1.xml"
    prefix: /profile

sonata_user:
    resource: '@SonataUserBundle/Resources/config/routing/admin_security.xml'
    prefix: /admin
```

生成自定义用户类

执行：

```
[root@centos7vm mywebsite]# php app/console sonata:easy-extends:generate SonataUserBundle -d src
```

可以自动在src/Application/Sonata/UserBundle/下生成有关用户的自定义类

注册自定义用户类，修改app/AppKernel.php，增加：

```
            new Application\Sonata\UserBundle\ApplicationSonataUserBundle(),
```

重新修改配置

这时重新修改app/config/config.yml，并找到对应配置组添加如下内容：

```
doctrine:
    orm:
        entity_managers:
            default:
                mappings:
                    FOSUserBundle: ~
                    ApplicationSonataUserBundle: ~
```

把fos_user配置组改成如下的样子：

```
fos_user:
    db_driver:      orm
    firewall_name:  main
    user_class:     Application\Sonata\UserBundle\Entity\User

    group:
        group_class:   Application\Sonata\UserBundle\Entity\Group
        group_manager: sonata.user.orm.group_manager

    profile:
        # Authentication Form
        form:
            type:               fos_user_profile
            handler:            fos_user.profile.form.handler.default
            name:               fos_user_profile_form
            validation_groups:  [Authentication] # Please note : this is not the default value

    service:
        user_manager: sonata.user.orm.user_manager
```

生效

更新数据库，执行

```
[root@centos7vm mywebsite]# php app/console doctrine:schema:update --force
```

创建一个管理员账户，执行：

```
[root@centos7vm mywebsite]# php app/console fos:user:create yourname youemail yourpasswd --super-admin
```

请cache后重新打开http://172.16.142.134/app_dev.php/admin，会看到提示登录啦，输入刚才创建的管理员用户名和密码就可以登录啦

本章节的内容是和官方文档有所不同的，经过我的尝试以及网上的一些说法也都表示sonata官方文档里的方法是有问题的，达不到想要的目的而且会报错，按照我上面试验过的方法是可行的

至此，你的管理后台就有权限控制了，不会被其他人篡改，可以尽情发布了

教你成为全栈工程师(Full Stack Developer) 二十一-网站开发资源汇总

发表于 2016-05-05 09:24

总结了一些不错的网上的资源，如果希望更系统的学习网站开发，可以按需选择资源进行学习

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

学习资源

入门教程

- 教你成为全栈工程师(Full Stack Developer)：http://www.shareditor.com/bloglist/2
  - 适合于初学者入门学习，内容涉及到前后端以及周边技术的方方面面，每部分内容虽讲的不深，但知识涉及范围比较广，看了之后可以知道自己要继续学些什么
- HTML 30分钟入门教程：http://www.jb51.net/shouce/html/html.htm
  - html教程网上有很多，随便找一个看看就行了，深入的知识不必一次性学会，后面先用现查就行了
- php菜鸟教程：http://www.runoob.com/php/php-tutorial.html
  - php语言上手比较简单，但是真正学精还是比较有难度的，这个菜鸟教程由浅入深教会你php
- symfony2中文教程：http://symfony.newlifeclan.com/
  - symfony2这种框架的教程基本上就是更多讲操作步骤，很少讲原理，所以如果想快速学习，就看这个中文教程

系统教程

- html教程：http://www.w3school.com.cn/h.asp
- css教程：http://www.w3school.com.cn/css/index.asp
- javascript教程：http://www.w3school.com.cn/js/index.asp
- php教程：http://www.w3school.com.cn/php/index.asp
- symfony2官方文档：http://symfony.cn/docs/

视频教程

- HTML零基础快速入门视频课程：http://edu.51cto.com/course/course_id-1134.html

- php+mysql网站开发视频教程：http://tech.163.com/special/000915SN/stphpcaopeng.html
- symfony2：http://www.imooc.com/learn/244

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

素材资源

模板素材

- 优站精选：http://expo.bootcss.com/
    - 这里是使用了bootstrap的优秀网站，有bootstrap做基础，界面设计还是相当赞的
- 优秀博客网站：http://www.shareditor.com
    - 这个网站是一个典型的博客网站，界面设计简约，个人比较喜欢

图片素材

- http://image.baidu.com/

社区资源

- https://stackoverflow.com/
    - 国外的问答社区，这里面有关symfony2的问题很多，解答也很赞
- http://www.oschina.net/question/tag/symfony
    - 国内的问答社区

教你成为全栈工程师(Full Stack Developer) 二十二-番外篇之网站开发不能放过的小细节

发表于 2016-05-06 08:41

对之前开发的博客网站中一些小细节做一些补充，让他变得更完美，更像一个产品

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

logo超链接

网站左上角的logo作为网站的一个标记，通常情况下点击会回到主页，刚好我们这部分是通过模板继承实现的，修改app/Resources/views/base.html.twig，找到logo文字的部分，加上超链接到主页，如下：

```
    <div class="col-sm-11 col-xs-11"><h1><a href="{{ path('homepage') }}" style="text-decoration: none;color: whi
te;">MyWebSite</a></h1></div>
```

版权说明

每一个网站底部都会有一个版权说明以及备案文字，这个是我们天朝特殊要求的，如果没有在网站底部写明备案号，你的网站发布以后会在几天之内被封掉，所以还是修改app/Resources/views/base.html.twig，在</body>前增加以下内容

```
<div class="row navbar navbar-inverse" style="margin-bottom: 0;">
    <div class="row">
        <div class="col-sm-12 col-xs-12 text-center" style="color: #959595;margin-bottom: 10px;">
            Copyright © <a href="{{ path('homepage') }}">MyWebSite.com</a> | 京ICP备*****号
        </div>
    </div>
</div>
```

效果如下：

自定义标题

点开每一个网页的标题（浏览器tab页上的文字标题）应该是不同的，模板里面已经有了

```
<title>{% block title %}自定义标题{% endblock title %}</title>
```

来为我们扩展用

改变首页的标题，修改app/Resources/views/default/index.html.twig，在{% extends "base.html.twig" %}下增加：

```
{% block title %}MyWebSite - 我的网站{% endblock title %}
```

同样分别在app/Resources/views/blog/list.html.twig和app/Resources/views/blog/show.html.twig中也增加如下两条：

```
{% block title %}{{ subject.name }} - MyWebSite - 我的网站{% endblock title %}
```

和

```
{% block title %}{{ blogpost.title }} - SharEDITor - 关注大数据技术{% endblock title %}
```


因为在BlogController的listAction中没有传递subject变量，所以还需要增加这个变量的传递，修改src/AppBundle/Controller/BlogController.php的listAction方法，增加：

```
        $this->subjectRepository = $this->getDoctrine()->getRepository('AppBundle:Subject');
        $subject = $this->subjectRepository->find($subjectId);
```

来获取$subject，然后把render语句改成：

```
        return $this->render('blog/list.html.twig', array('pagination' => $pagination,'subject' => $subject));
```

管理后台本地化

管理后台的一些显示文字不是很友好，比如按钮上的文字都是btn_create_and_****，这就需要我们做本地化的处理，本地化的意思就是本土化，也就是编程我们国家的语言，但是因为symfony2对中文支持不是很好，我们直接开启英文配置，修改app/config/config.yml，把

```
    #translator:      { fallbacks: ["%locale%"] }
```

前面的注释符号"#"去掉即可，按钮会变成：Create、Create and return list等

增加文章数目展示

首页里的类别展示中如果能展示出这一类里已经有了多少篇文章，会让用户体验更好，修改app/Resources/views/default/index.html.twig，修改显示类别名称一行为：

```
                <h3>{{ subject.name }}({{ blogcounts[subject.id] }})</h3>
```

这里的blogcounts在DefaultController.php的indexAction中没有传递，因此修改src/AppBundle/Controller/DefaultController.php，为DefaultController类增加如下方法：

```
    public function getSubjectBlogCountMap($subjects)
    {
        $this->blogPostRepository = $this->getDoctrine()->getRepository('AppBundle:BlogPost');
        $map = array();
        for ($i = 0; $i < sizeof($subjects); $i = $i + 1)
        {
            $this->subject = $subjects[$i];
            $map[$this->subject->getId()] = sizeof($this->blogPostRepository->findBy(array('subject' => $this->subj
ect->getId())));
        }
        return $map;
    }
```

并把indexAction方法的render语句改成：

```
        return $this->render('default/index.html.twig', array(
            'subjects' => $subjects,
            'blogcounts' => $this->getSubjectBlogCountMap($subjects),
        ));
```

教你成为全栈工程师(Full Stack Developer) 二十三-番外篇之搜索引擎优化（SEO）

发表于 2016-05-06 09:13

网站发布了，怎么能快速被百度等搜索引擎收录呢？怎么能更容易被用户搜到呢？这是SEO范畴的知识，本节来讲述一下，并帮你把你的网站做一些优化

网站结构

搜索引擎分为三部分：抓取、建索引、检索。抓取就是通过爬虫软件自动爬取你的网站内容存储到搜索引擎的网页库中，建索引就是对抓取到的网页内容做分析并建成倒排索引，检索就是让用户在搜索框中能够搜到你的网页。

抓取的过程是通过外部指向你网站的某个链接或者你的主动推送的种子链接开始广度加深度遍历，最终抓取完你的整个网站，当然后续还会不断的重试抓取你的网页，如果发现新的链接还会继续抓取，保持时新性。

为了让搜索引擎抓取我们的网站，就需要把我们的网站结构做的足够简单清晰，建议尽量减少链接深度：爬虫是优先做广度遍历的，链接深度约小，就更容易被快速爬取

内链外链

爬虫是通过超链接来遍历网站的，根据排名算法，指向网站的链接越多是会提升网站排名的，所以尽量建立足够多的优质外链，这里提到优质，不是说外链越多越好，过多会被搜索引擎antispam掉，重要的是优质，有较多转化流量才能带来用户的访问，所以可以优化我们的网站，增加网页里的链接数目，比如可以在网站底部加上“最新文章”，如果你按照前面章节一步一步实现了我们的博客网站的话，那么可以修改app/Resources/views/base.html.twig，在版权说明前加上：

```
 <div class="row">
     <div class="col-sm-1 col-xs-1"></div>
     <div class="col-sm-4 col-xs-4">
         <h4 style="color: #FFFFFF; border-bottom: 1px solid #695d69; padding-bottom: 10px; margin-top: 30px;
">
             最新文章</h4>
         {% for article in latestblogs %}

             <div class="row" style="margin: 10px;margin-left: 0; overflow:hidden;text-overflow:ellipsis;white-spac
e:nowrap;">
                 <a style="color: #959595;"
                    href="{{ path('blog_show', {'blogId':article.id}) }}">{{ article.title }}</a>
             </div>

         {% endfor %}
         <br/>
     </div>
     <div class="col-sm-1 col-xs-1"></div>

     <div class="col-sm-4 col-xs-4">
         <h4 style="color: #FFFFFF; border-bottom: 1px solid #695d69; padding-bottom: 10px; margin-top: 30px;
">
             为你推荐</h4>
         {% for article in recommends %}

             <div class="row" style="margin: 10px;margin-left: 0;overflow:hidden;text-overflow:ellipsis;white-space
:nowrap;">
                 <a style="color: #959595;"
                    href="{{ path('blog_show', {'blogId':article.id}) }}">{{ article.title }}</a>
             </div>

         {% endfor %}
         <br/>
     </div>

     <div class="col-sm-2 col-xs-2"></div>
   </div>
```

并修改src/AppBundle/Controller/BlogController.php，为BlogController增加如下方法：

```
public static function getLatestBlogs($contoller)
{
    $blogPostRepository = $contoller->getDoctrine()->getRepository('AppBundle:BlogPost');
    $blogposts = $blogPostRepository->findBy(array(), array('createTime' => 'DESC'), 5);
    return $blogposts;
}

public static function getRecommends($contoller)
{
    $blogPostRepository = $contoller->getDoctrine()->getRepository('AppBundle:BlogPost');
    $allrecommends = $blogPostRepository->findBy(array(), array('createTime' => 'DESC'), 100);

    $randList = BlogController::genRandList(0, sizeof($allrecommends), 5);
    $recommends = array();
    foreach ($randList as $index => $value) {
        $recommends[] = $allrecommends[$index];
    }
    return $recommends;
}

public static function genRandList($min, $max, $num)
{
    $num = min($num, $max-$min);
    $map = array();
    while (sizeof($map) < $num) {
        $r = rand($min, $max-1);
        $map[$r] = 1;
    }
    return $map;
}
```


并修改listAction方法的render语句如下：

```
    return $this->render('blog/list.html.twig', array('pagination' => $pagination,
        'subject' => $subject,
        'latestblogs' => BlogController::getLatestBlogs($this),
        'recommends' => BlogController::getRecommends($this)));
```

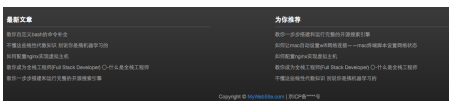
修改showAction方法的render语句如下:

```
public function showAction($blogId)
{
    $this->blogPostRepository = $this->getDoctrine()->getRepository('AppBundle:BlogPost');
    return $this->render('blog/show.html.twig', array('blogpost' => $this->blogPostRepository->find($blogId),
        'latestblogs' => BlogController::getLatestBlogs($this),
        'recommends' => BlogController::getRecommends($this)));
}
```

同时修改src/AppBundle/Controller/DefaultController.php里的indexAction方法的render语句，如下：

```
    return $this->render('default/index.html.twig', array(
        'subjects' => $subjects,
        'blogcounts' => $this->getSubjectBlogCountMap($subjects),
        'latestblogs' => BlogController::getLatestBlogs($this),
        'recommends' => BlogController::getRecommends($this),
    ));
```

看下效果：



解释一下，这里的getLatestBlogs（最新文章）就是获取最近发布的几篇文章，这里的getRecommends（为你推荐）其实是一个伪的推荐，里面就是做了一些随机性，这样的随机性有一些好处，就是可以让搜索引擎每次抓取的结果都有所不同，这样它会认为你的网站时刻有更新，从而缩短抓取周期，更容易发现你的新链接

内容质量

网站的内容质量是搜索引擎优化的关键，搜索引擎是基于文本的，所以你的页面里要尽量包含足够多的文本信息，图片和js对于搜索引擎来说无疑增加了它的分析难度。努力上传优秀的原创文章一定会受搜索引擎的青睐，一味的转载，搜索引擎是有办法知道谁是原创谁是抄袭的，原创一定会被排在前面的

主动提交

百度站长平台提供给了我们主动提交链接的方法，到http://zhanzhang.baidu.com/注册一个账号，获取到自动提交代码嵌入到网页里，可以在你发布文章的第一时间传给百度知晓，这块相信你自己能够搞定，我的实例代码如下，仅供参考：

```
<script>
(function(){
    var bp = document.createElement('script');
    bp.src = '//push.zhanzhang.baidu.com/push.js';
    var s = document.getElementsByTagName("script")[0];
    s.parentNode.insertBefore(bp, s);
})();
</script>
```

教你成为全栈工程师(Full Stack Developer) 二十四-ES(elasticsearch)搜索引擎安装和使用

发表于 2016-05-09 09:09

大数据时代，搜索无处不在。搜索技术是全栈工程师必备技术之一，如今是开源时代，数不尽的资源供我们利用，如果要自己写一套搜索引擎无疑是浪费绳命。本节主要介绍搜索引擎开源项目elasticSearch的安装和使用

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

为什么需要搜索引擎

首先想一下：在一篇文章里找一个关键词怎么找？字符串匹配是最佳答案。

然后再想一下：找到100篇文章里包含某个关键词的文章列表怎么找？依然是关键词匹配

再继续想：找到100000000000（一千亿）篇文章里包含某个关键词的文章怎么找？如果用关键词匹配，以现在的电脑处理速度，从远古时代到人类灭绝这么长时间都处理不完，这时候搜索引擎要发挥作用了

搜索引擎技术有多么高深？

搜索引擎这种技术实际上从古代就有了。想象一个国家存储各类编撰资料的部门，有几个屋子的书，如果想找到某一本书的时候会怎么找呢？对了，有分类目录，先确定要找的书籍是哪个类别的，然后从目录里面找到想要找的书籍位于屋子的什么位置，然后再去拿。搜索引擎其实就是做了生成目录（也就是索引）的事情。那么如今的搜索引擎是怎么生成索引的呢？

要把互联网上的资料生成索引，拢共分三步：第一步，把资料编号；第二步，把每篇资料内容切成词；第三步，把词和资料编号的对应关系处理成“词=》编号列表”的形式

这时候你就可以迅速的找到一千亿篇文章里包含某个关键词的文章了，告诉我关键词是什么，我直接就从索引里找到了这个词对应的文章编号列表了，搞定！把需要数万年才能做完的工作用了不到一秒钟就搞定了，这就是搜索引擎的魅力！

当然，上面说的搜索引擎技术很简单，但百度数万工程师也不都是白吃饭的，如果想做好一个搜索引擎产品需要解决的问题就有很多了：收集网页时要考虑全、快、稳、新、优的问题，建索引时要考虑质量、效率、赋权、周期、时效性、资源消耗等问题，搜索的时候要考虑query分析、排序、筛选、展示、性能、广告、推荐、个性化、统计等问题，整体上要考虑地域性、容灾、国际化、当地法律、反作弊、垂直需求、移动互联网等诸多问题，所以百度大厦彻夜通明也是可以理解的。

开源搜索引擎

既然搜索引擎技术这么复杂，那么我们何必自寻烦恼了，开源社区为我们提供了很多资源，世界很美好。

说到开源搜索引擎一定要用的开源项目就是lucene，它不是搜索引擎产品，而是一个类库，而且是至今开源搜索引擎的最好的类库没有之一，因为只有它一个。lucene是用java语言开发，基本上涵盖了搜索引擎中索引和检索两个核心部分的全部功能，而且抽象的非常好，我后面会单独写数篇文章专门讲lucene的使用。最后强调一遍，它是一个类库，不是搜索引擎，你可以比较容易的基于lucene写一个搜索引擎。

然后要说的一个开源项目是solr，这是一个完整的搜索引擎产品，当然它底层一定是基于lucene的，毫无疑问，因为lucene是最好且唯一的搜索引擎类库。solr使用方法请看我的另一篇文章《教你一步步搭建和运行完整的开源搜索引擎》

最后要说的就是elasticSearch，这个开源项目也可以说是一个产品级别的开源项目，当然它底层一定是基于lucene的，毫无疑问，因为lucene是最好且唯一的搜索引擎类库，我承认我是唐僧。它是一种提供了RESTful api的服务，RESTful就是直接通过HTTP协议收发请求和响应，接口比较清晰简单，是一种架构规则。话不多说，下面我就说下安装方法和简单使用方法，这样更容易理解，之后我会单独讲解怎么让你的网站利用elasticSearch实现搜索功能

elasticSearch安装

从github下载1.7版tag并编译（选择1.7版是因为当前我们的网站的symfony2版本还不支持2.x版本，但请放心的用，1.7版是经过无数人验证过的最稳定版本）

```
wget https://codeload.github.com/elastic/elasticsearch/tar.gz/v1.7.5
```

解压后进入目录执行

```
mvn package -DskipTests
```

这会花费你很长一段时间，你可以去喝喝茶了

编译完成后会在target/releases中生成编译好的压缩包（类似于elasticsearch-1.7.5.zip这样的文件），把这个压缩包解压放到任意目录就安装好了

安装ik插件

ik是一个中文切词插件，elasticSearch自带的中文切词很不专业，ik对中文切词支持的比较好。

在https://github.com/medcl/elasticsearch-analysis-ik上找到我们elasticSearch对应的版本，1.7.5对应的ik版本是1.4.1，所以下载https://github.com/medcl/elasticsearch-analysis-ik/releases/tag/v1.4.1

解压出的目录是：

elasticsearch-analysis-ik-1.4.1

进入目录后执行

```
mvn clean package
```

编译完后依然是在target/releases生成了类似于elasticsearch-analysis-ik-*.zip的压缩包，把里面的内容解压到elasticsearch安装目录的plugins/ik下

再把elasticsearch-analysis-ik-1.4.1/config/ik目录整体拷贝到elasticsearch安装目录的config下

修改elasticsearch安装目录下的config/elasticsearch.yml，添加：

```
index:
  analysis:
    analyzer:
      ik:
        alias: [ik_analyzer]
        type: org.elasticsearch.index.analysis.IkAnalyzerProvider
      ik_max_word:
        type: ik
        use_smart: false
      ik_smart:
        type: ik
        use_smart: true
```

这样ik就安装好了

启动并试用

直接进入elasticsearch安装目录，执行

```
./bin/elasticsearch -d
```

后台启动完成

elasticSearch是通过HTTP协议收发数据的，所以我们用curl命令来给它发命令，elasticSearch默认监听9200端口

写入一篇文章：

```
curl -XPUT 'http://localhost:9200/myappname/myblog/1?pretty' -d '
{
  "title": "我的标题",
  "content": "我的内容"
}'
```

会收到返回信息：

```
{
  "_index" : "myappname",
  "_type" : "myblog",
  "_id" : "1",
  "_version" : 1,
  "created" : true
}
```

这说明我们成功把一篇文章发给了elasticSearch，它底层会利用lucene自动帮我们建好搜索用的索引

再写一篇文章：

```
curl -XPUT 'http://localhost:9200/myappname/myblog/2?pretty' -d '
{
  "title": "这是第二篇标题",
  "content": "这是第二篇内容"
}'
```

会收到返回信息：

```
{
  "_index" : "myappname",
  "_type" : "myblog",
  "_id" : "2",
  "_version" : 1,
  "created" : true
}
```


这时我们找到elasticsearch安装目录的data目录下会生成这样的目录和文件：

```
ls data/nodes/0/indices/myappname/
0  1  2  3  4  _state
```

不同环境会稍有不同，但是都会生成myappname目录就对了

查看所有文章：

```
curl -XGET 'http://localhost:9200/myappname/myblog/_search?pretty=true' -d '
{
    "query" : {
        "match_all" : {}
    }
}'
```

这时会把我们刚才添加的两篇文章都列出来

搜索关键词“我的”：

```
curl -XGET 'http://localhost:9200/myappname/myblog/_search?pretty=true' -d '
{
    "query":{
        "query_string":{"query":"我的"}
    }
}'
```

会返回：

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.191783,
    "hits" : [ {
      "_index" : "myappname",
      "_type" : "myblog",
      "_id" : "1",
      "_score" : 0.191783,
      "_source":
{
  "title": "我的标题",
  "content": "我的内容"
}
    } ]
  }
}
```

搜索关键词“第二篇”：

```
curl -XGET 'http://localhost:9200/myappname/myblog/_search?pretty=true' -d '
{
    "query":{
        "query_string":{"query":"第二篇"}
    }
}'
```

会返回：

```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.1879082,
    "hits" : [ {
      "_index" : "myappname",
      "_type" : "myblog",
      "_id" : "2",
      "_score" : 0.1879082,
      "_source":
{
  "title": "这是第二篇标题",
  "content": "这是第二篇内容"
}
    } ]
  }
}
```

如果想检查ik的切词效果，可以执行：

```
curl 'http://localhost:9200/myappname/_analyze?analyzer=ik_max_word&pretty=true' -d'
{
    "text":"中华人民共和国国歌"
}'
```

通过返回结果可以看出，ik_max_word切词把中华人民共和国国歌切成了“中华人民共和国”、“中华人民”、“中华”、“华人”、“人民共和国”、“人民”、“共和国”、“共和”、“国”、“国歌”

也就是说我们搜索这些词中的任意一个都能把这句话搜到，如果不安装ik插件的话，那只会切成：“中”、“华”、“人”、“民”、“共”、“和”、“国”、“国”、“歌”，不够智能，搜索也不好进行了

讲解一下

上面几条命令都是json形式，elasticSearch就是这么人性化，没治了。

这里面的myappname是你自己可以改成自己应用的名字，这在elasticSearch数据存储中是完全隔离的，而myblog是我们在同一个app中想要维护的不同的数据，就是你的不同数据，比如文章、用户、评论，他们最好都分开，这样搜索的时候也不会混

pretty参数就是让返回的json有换行和缩进，容易阅读，调试时可以加上，开发到程序里就可以去掉了

analyzer就是切词器，我们指定的ik_max_word在前面配置文件里遇到过，表示最大程度切词，各种切，360度切

返回结果里的hits就是“命中”，total是命中了几条，took是花了几毫秒，_score就是相关性程度，可以用来做排序的依据

elasticSearch有什么用

上面都是json的接口，那么我们怎么用呢？其实你想怎么用就怎么用，煎着吃、炒着吃、炖着吃都行。比如我们的博客网站，当你创建一篇博客的时候可以发送“添加”的json命令，然后你开发一个搜索页面，当你输入关

键词点搜索的时候，可以发送查询的命令，这样返回的结果就是你的搜索结果，只不过需要你自己润色一下，让展现更美观。感觉复杂吗？下一节告诉你怎么用symfony2的扩展来实现博客网站的搜索功能

教你成为全栈工程师(Full Stack Developer) 二十五-为你的网站添加强大的搜索功能

发表于 2016-05-10 08:22

当你的网站文章内容变多时，搜索的需求会逐渐显现，mysql的like功能无论从性能还是效果上都是一个山寨的实现，本节帮你集成elasticSearch实现专业的搜索功能

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

elasticSearch安装

elasticSearch服务的安装请见我的上一篇文章《教你成为全栈工程师(Full Stack Developer) 二十四-ES(elasticsearch)搜索引擎安装和使用》

elastic-bundle插件安装

需要在我们的symfony2项目中安装elastic-bundle插件，执行

```
[root@centos7vm mywebsite]# composer require friendsofsymfony/elastica-bundle
```

配置方法

修改app/AppKernel.php添加如下注册：

```
new FOS\ElasticaBundle\FOSElasticaBundle(),
```

并在app/config/config.yml中添加

```
fos_elastica:
    clients:
        default: { host: localhost, port: 9200 }
    indexes:
        app:
            types:
                blogpost:
                    mappings:
                        title:
                            type: string
                            index: analyzed
                            analyzer: ik_max_word
                        body:
                            type: string
                            index: analyzed
                            analyzer: ik_max_word
                    persistence:
                        driver: orm
                        model: AppBundle\Entity\BlogPost
                        provider: ~
                        listener:
                            insert: true
                            update: true
                            delete: true
                        finder: ~
```

解释一下

这里的clients配置的是本地elasticSearch服务的host和port

app是我们应用的名字，你可以随意改成你指定的名字，比如shareditor

blogpost是我们应用里不同数据的名字，每一个都会把索引集中建到一起

title和body是我们要搜索的域，这里我们指定了切词器是ik_max_word

persistence里配置了blogpost的读取orm，也就是BlogPost这个Entity

批量生成索引

执行

```
[root@centos7vm mywebsite]# php app/console fos:elastica:populate
```

ps：这一句只在第一次集成elasticSearch的时候执行，以后每当有新文章发布就会自动写入索引，无需手动更新

下面我们开始修改网站添加搜索功能

添加action

修改src/AppBundle/Controller/BlogController.php，增加如下方法：

```
    public function searchAction(Request $request)
    {
        $query = $request->get('q');
        $finder = $this->container->get('fos_elastica.finder.app.blogpost');
        $paginator = $this->get('knp_paginator');
        $results = $finder->createPaginatorAdapter($query);
        $pagination = $paginator->paginate($results, 1, 10);

        return $this->render('blog/search.html.twig', array('pagination' => $pagination,
            'query' => $query,
            'latestblogs' => BlogController::getLatestBlogs($this),
            'recommends' => BlogController::getRecommends($this)));
    }
```

解释一下

$request->get('q');这是从url参数里获取q这个参数来作为query词，后面网页修改里会通过form表单的input空间传递q的value

$this->container->get('fos_elastica.finder.app.blogpost');这是根据config.yml里的配置获取finder实例来做搜索用

下面利用$paginator做翻页，每页10条，默认展示第一页

添加路由配置

修改app/config/routing.yml，添加

```
blog_search:
    path:     /blogsearch
    defaults: { _controller: AppBundle:Blog:search }
```

修改网页模板

修改app/Resources/views/base.html.twig，增加如下代码：

```
    <div class="col-sm-8 col-xs-10"><h1><a href="{{ path('homepage') }}" style="text-decoration: none;color: whit
e;">SharEDITor</a></h1></div>
    <div class="col-sm-3 col-xs-1">
        <form action="{{ path('blog_search') }}" style="margin-top: 10px;">
            <input type="search" name="q" placeholder="搜文章" maxlength="200" style="background-color: transp
arent;">
        </form>
    </div>
```

这其实就是在网页顶部增加了一个搜索框，这里面的input输入框的name设置成q，刚好对应了上面的$request->get('q');

搜索结果页

为了简单，我们尽量复用博客列表页，拷贝app/Resources/views/blog/list.html.twig到app/Resources/views/blog/search.html.twig，把

```
{% block title %}{{ subject.name }} - SharEDITor - 关注大数据技术{% endblock title %}
```

改成

```
{% block title %}{{ query }}的搜索结果 - SharEDITor - 关注大数据技术{% endblock title %}
```

教你成为全栈工程师(Full Stack Developer) 二十六-storm安装与初识

发表于 2016-05-11 09:21

storm是如今用来做实时数据处理的首选，它的高效以及基于分布式系统的考虑备受技术团队青睐，尤其是实时日志处理，以及基于日志的实时分析，从本章节开始我们来认识一下这个强大的开源工具，并通过这个工具来做一个非常有意思的事情，敬请期待

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

安装zookeeper

zookeeper是storm运行强依赖

注意：

1）用supervision启动zookeeper，保证不能退出

2）用cron定期清理zookeeper的日志数据，不然磁盘会很快占满（以后再研究）

安装步骤：

```
wget http://mirrors.hust.edu.cn/apache/zookeeper/zookeeper-3.4.8/zookeeper-3.4.8.tar.gz
tar zxvf zookeeper-3.4.8.tar.gz
cd zookeeper-3.4.8/
cp conf/zoo_sample.cfg conf/zoo.cfg
```

修改conf/zoo.cfg，把

```
dataDir=/tmp/zookeeper
```

改成

```
dataDir=/var/zookeeper
```

创建/var/zookeeper/myid，内容为数字1

执行

```
./bin/zkServer.sh start
```

成功启动

验证方法：

```
./bin/zkCli.sh
```

输入help命令查看帮助

为了让zookeeper异常退出后能自动重启，需要安装deamontools

```
wget http://cr.yp.to/daemontools/daemontools-0.76.tar.gz
tar zxvf daemontools-0.76.tar.gz
cd admin/daemontools-0.76/
```

vim src/error.h 找到：extern int errno; 改成：#include <errno.h>

执行

```
package/install
```

这时已经安装好了

```
[root@centos7vm daemontools-0.76]# which supervise
/usr/local/bin/supervise
```

创建/data/service/zookeeper/run文件，内容为：

```
#!/bin/bash
exec 2>&1
exec /data/zookeeper-3.4.8/bin/zkServer.sh start
```

增加执行权限

```
chmod +x /data/service/zookeeper/run
```

杀了之前手工启动的zookeeper，然后执行

```
cd /data/service/zookeeper
nohup supervise /data/service/zookeeper &
```

这时zookeeper被supervice启动了，尝试杀一次zookeeper后还会自动起来

参考：http://zookeeper.apache.org/doc/r3.3.3/zookeeperAdmin.html

安装storm

```
wget http://apache.fayea.com/storm/apache-storm-1.0.1/apache-storm-1.0.1.tar.gz
tar apache-storm-1.0.1.tar.gz
cd apache-storm-1.0.1
```

修改conf/storm.yaml，添加如下配置

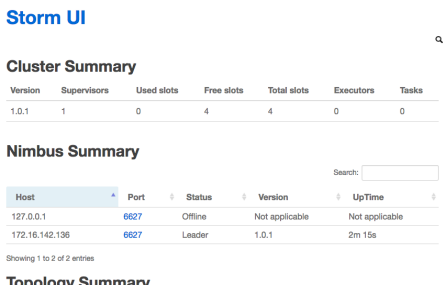
```
storm.zookeeper.servers:
    - "127.0.0.1"
nimbus.seeds: ["127.0.0.1"]
supervisor.slots.ports:
    - 6700
    - 6701
    - 6702
    - 6703
```

启动storm，执行

```
./bin/storm nimbus &
./bin/storm supervisor &
./bin/storm ui &
```

打开web界面，http://127.0.0.1:8080

界面如下：



安装完成

讲讲storm

storm系统由一个nimbus节点和多个supervisor节点组成，上面因为是部署单机版本，所以只启动了一个supervisor。他们之间是通过zookeeper协调运行的，所以必须依赖zookeeper。nimbus负责分配任务和监控任务，本身不做计算，supervisor负责真正的计算任务。

storm上运行的任务和map-reduce的不同在于它运行的是一种topology任务，也就是一种有向无环图形式的任务服务。

上面配置文件中配置的supervisor.slots.ports包含了4个port，也就是这个supervisor可以监听4个端口同时并发的执行4个任务，因此在web界面里我们看到Free slots是4

在map-reduce系统上运行的任务我们叫做mapper和reducer，相对之下，在storm上运行的任务叫做spout（涛涛不绝地喷口）和bolt（螺栓），在拓扑里传递的消息叫做tuple。spout其实就是信息产生的源头，而bolt就是处理逻辑

下一节我们来试验一些简单的用途，来发觉storm可以用来做些什么事情

教你成为全栈工程师(Full Stack Developer) 二十七-开发第一个storm任务

发表于 2016-05-13 09:00

每门语言的学习都要经历最经典的helloworld过程，学习storm框架也一样，本节我们先来运行storm为我们做好的storm-starter例子，来确定storm服务搭建是正常的，然后我们自己来从零开始写一个简单的storm任务

运行storm-starter

首先要下载storm源代码，我们服务部署的是1.0.1版本，那么我就下载同样的版本，在github.com上找到https://github.com/apache/storm/releases/tag/v1.0.1的源代码（注意，这里的源代码和上一节中降到的部署用的包是不同的，这个是未编译的原始代码，上节中的是编译好的直接可运行的包，当然你也可以用本节下载的源代码重新编译来部署），下载：

```
wget https://github.com/apache/storm/archive/v1.0.1.tar.gz
```

解压后编译代码中的样例start-storm（位于源代码的examples/storm-starter），编译方法如下（也可以加载到eclipse，用maven编译）：

```
mvn -D maven.test.skip=true clean package
```

注：如果没有安装maven则先安装，具体方法百度一下

编译好了之后会生成target/storm-starter-1.0.1.jar文件，下面我们来在我们部署好的storm上运行这个jar包里的storm.starter.StatefulTopology类

执行：

```
storm jar examples/storm-starter/storm-starter-topologies-*.jar storm.starter.StatefulTopology statetopology
```

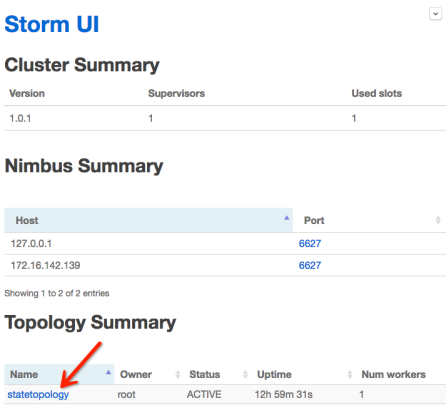
这里的storm命令是在storm部署目录的bin目录下的脚本，如果没有配置PATH环境变量，可以用绝对路径执行，如果我的storm部署在/data/apache-storm-1.0.1/，那么就执行：

```
/data/apache-storm-1.0.1/bin/storm jar storm-starter-1.0.1.jar storm.starter.StatefulTopology statetopology
```

如果运行成功说明你的storm是正常的，输出如下：

```
[root@centos7vm storm-1.0.1]# /data/apache-storm-1.0.1/bin/storm jar storm-starter-1.0.1.jar storm.starter.StatefulTopology statetopology
Running: java -client -Ddaemon.name= -Dstorm.options= -Dstorm.home=/data/apache-storm-1.0.1 -Dstorm.log.dir=/data/apache-storm-1.0.1/logs -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/lib -Dstorm.conf.file= -cp /data/apache-storm-1.0.1/lib/storm-core-1.0.1.jar:/data/apache-storm-1.0.1/lib/kryo-3.0.3.jar:/data/apache-storm-1.0.1/lib/reflectasm-1.10.1.jar:/data/apache-storm-1.0.1/lib/asm-5.0.3.jar:/data/apache-storm-1.0.1/lib/minlog-1.3.0.jar:/data/apache-storm-1.0.1/lib/objenesis-2.1.jar:/data/apache-storm-1.0.1/lib/clojure-1.7.0.jar:/data/apache-storm-1.0.1/lib/disruptor-3.3.2.jar:/data/apache-storm-1.0.1/lib/log4j-api-2.1.jar:/data/apache-storm-1.0.1/lib/log4j-core-2.1.jar:/data/apache-storm-1.0.1/lib/log4j-slf4j-impl-2.1.jar:/data/apache-storm-1.0.1/lib/slf4j-api-1.7.7.jar:/data/apache-storm-1.0.1/lib/log4j-over-slf4j-1.6.6.jar:/data/apache-storm-1.0.1/lib/servlet-api-2.5.jar:/data/apache-storm-1.0.1/lib/storm-rename-hack-1.0.1.jar:storm-starter-1.0.1.jar:/data/apache-storm-1.0.1/conf:/data/apache-storm-1.0.1/bin -Dstorm.jar=storm-starter-1.0.1.jar storm.starter.StatefulTopology statetopology
1329 [main] INFO  o.a.s.StormSubmitter - Generated ZooKeeper secret payload for MD5-digest: -6806752055047040892:-6736308748043757911
1492 [main] INFO  o.a.s.s.a.AuthUtils - Got AutoCreds []
1687 [main] INFO  o.a.s.StormSubmitter - Uploading topology jar storm-starter-1.0.1.jar to assigned location: /data/apache-storm-1.0.1/storm-local/nimbus/inbox/stormjar-89ac13d0-dc05-4aec-ab4c-0c6fc6b7f5e0.jar
Start uploading file 'storm-starter-1.0.1.jar' to '/data/apache-storm-1.0.1/storm-local/nimbus/inbox/stormjar-89ac13d0-dc05-4aec-ab4c-0c6fc6b7f5e0.jar' (62385346 bytes)
[==================================================] 62385346 / 62385346
File 'storm-starter-1.0.1.jar' uploaded to '/data/apache-storm-1.0.1/storm-local/nimbus/inbox/stormjar-89ac13d0-dc05-4aec-ab4c-0c6fc6b7f5e0.jar' (62385346 bytes)
2992 [main] INFO  o.a.s.StormSubmitter - Successfully uploaded topology jar to assigned location: /data/apache-storm-1.0.1/storm-local/nimbus/inbox/stormjar-89ac13d0-dc05-4aec-ab4c-0c6fc6b7f5e0.jar
2992 [main] INFO  o.a.s.StormSubmitter - Submitting topology statetopology in distributed mode with conf {"storm.zookeeper.topology.auth.scheme":"digest","storm.zookeeper.topology.auth.payload":"-6806752055047040892:-6736308748043757911","topology.workers":1,"topology.debug":false}
3682 [main] INFO  o.a.s.StormSubmitter - Finished submitting topology: statetopology
```

web界面如下：



stateopology详细运行情况可以点击进去

我们用eclipse（你可以用其他IDE）导入源代码，找到main函数如下：

```
public static void main(String[] args) throws Exception {
    TopologyBuilder builder = new TopologyBuilder();
    builder.setSpout("spout", new RandomIntegerSpout());
    builder.setBolt("partialsum", new StatefulSumBolt("partial"), 1).shuffleGrouping("spout");
    builder.setBolt("printer", new PrinterBolt(), 2).shuffleGrouping("partialsum");
    builder.setBolt("total", new StatefulSumBolt("total"), 1).shuffleGrouping("printer");
    Config conf = new Config();
    conf.setDebug(false);

    if (args != null && args.length > 0) {
        conf.setNumWorkers(1);
        StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());
    } else {
        LocalCluster cluster = new LocalCluster();
        StormTopology topology = builder.createTopology();
        cluster.submitTopology("test", conf, topology);
        Utils.sleep(40000);
        cluster.killTopology("test");
        cluster.shutdown();
    }
}
```

这个storm任务是由一个Spout（随机生成整数值）和三个Bolt（主要对数字做汇总并打印一些信息）组成，为了能看到运行的效果，我们找到部署目录里的log4j2/worker.xml配置文件，找到appenders配置，如果没有自己指定log目录，那么这里面默认应该是部署目录里的logs/workers-artifacts下

我们进入logs/workers-artifacts目录，这里面就是每个storm任务的日志目录

点网页里的stateopology

Topology summary

Name	Id
statetopology	statetopology-2-1463051043

这里的statetopology-2-1463051043就是我们的任务id，那么在logs/workers-artifacts目录中就会有statetopology-2-1463051043目录

```
cd statetopology-2-1463051043
```

会看到又有多个目录，继续点web里的spout进入到spout的状态页面，看到如下：

Executors (All time)

Id	Uptime	Host	Port
[6-6]	16m 40s	172.16.142.139	6700

Showing 1 to 1 of 1 entries

这是由supervisor启动的executor的主机地址和port，那么我们进入到刚才的statetopology-2-1463051043目录的6700目录就是这个spout的日志啦，执行

```
[root@centos7vm 6700]# tail -f worker.log
```

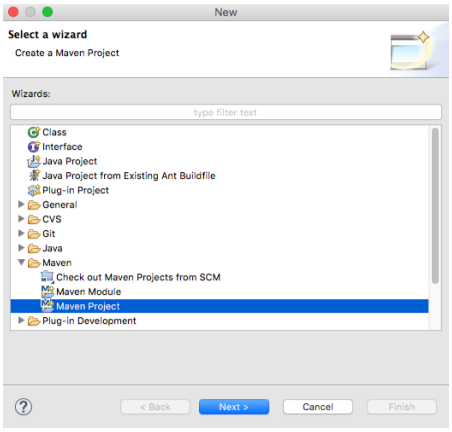
日志在定期输出随机生成的数字

以上就是storm-starter的运行，在storm-starter项目里还有很多功能的演示，在真正使用时会经常参考

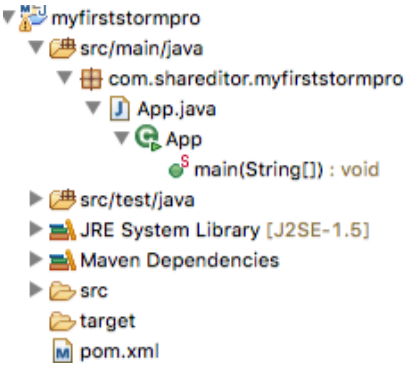
从零开始开发storm任务

下面我们准备不依赖storm的源代码，而是自己从零开始写一个storm任务，来理解一下storm的开发要点

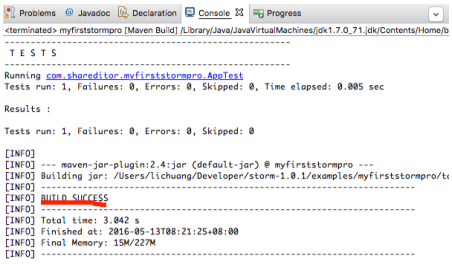
打开eclipse（需要预安装mvn插件，没装的请百度一下），我们新建一个maven工程：



group id设置为：com.shareditor，artifact id设置为：myfirststormpro，其他都默认下一步，最终我们的工程如下：



这是maven为我们自动生成的helloworld，我们用maven编译一下，项目上点右键，选择Run as -> Maven build，在Goals中填clean package，点Run编译，如果编译成功会看到：



以上如果成功，说明maven编译没有问题

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

下一步我们来编辑storm项目依赖，打开pom.xml源文件如下：

```
<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.shareditor</groupId>
  <artifactId>myfirststormpro</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>myfirststormpro</name>
  <url>http://maven.apache.org</url>

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

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

在dependencies标签中添加如下依赖：

```
      <dependency>
          <groupId>org.apache.storm</groupId>
          <artifactId>storm-core</artifactId>
          <version>1.0.1</version>
      </dependency>
```

然后在com.shareditor.myfirststormpro包下创建MySpout.java，如下：

```
package com.shareditor.myfirststormpro;

import java.util.Map;

import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.utils.Utils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MySpout extends BaseRichSpout{

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

    private SpoutOutputCollector collector;

    public void open(Map conf, TopologyContext context,
            SpoutOutputCollector collector) {
        this.collector = collector;
    }

    public void nextTuple() {
        Utils.sleep(1000);
        LOG.info("MySpout nextTuple");
    }


    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("value"));
    }
}
```

创建MyBolt.java，如下：

```
package com.shareditor.myfirststormpro;

import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Tuple;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MyBolt extends BaseBasicBolt {

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

    public void execute(Tuple input, BasicOutputCollector collector) {
        LOG.info("MyBolt execute");
    }

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        // TODO Auto-generated method stub

    }
}
```

修改App.java，如下：

```
package com.shareditor.myfirststormpro;

import org.apache.storm.Config;
import org.apache.storm.StormSubmitter;
import org.apache.storm.topology.TopologyBuilder;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main(String[] args) throws Exception {

        System.out.println("main");

        TopologyBuilder builder = new TopologyBuilder();
        builder.setSpout("myspout", new MySpout());
        builder.setBolt("mybolt", new MyBolt()).shuffleGrouping("myspout");

        Config conf = new Config();
        conf.setDebug(false);
        conf.setNumWorkers(1);
        StormSubmitter.submitTopologyWithProgressBar(args[0], conf, builder.createTopology());
    }
}
```

重新用maven编译项目，没问题会输出[INFO] BUILD SUCCESS，并且在myfirststormpro/target中会生成：myfirststormpro-0.0.1-SNAPSHOT.jar

用storm启动它，执行：

```
/data/apache-storm-1.0.1/bin/storm jar myfirststormpro-0.0.1-SNAPSHOT.jar com.shareditor.myfirststormpro.App myfirststormproname
```

像上面找storm-starter的方法一样找到日志，看到：

```
[root@centos7vm 6700]# tail -f worker.log
2016-05-13 08:48:59.100 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:49:00.112 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:49:02.308 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:49:03.310 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:49:04.312 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:49:05.314 c.s.m.MySpout [INFO] MySpout nextTuple
……
```

每隔一秒钟打印一行MySpout nextTuple

下面我们做一些修改，让他真正射点东西：

```
    public void nextTuple() {
        Utils.sleep(1000);
        LOG.info("MySpout nextTuple");

        collector.emit(new Values(10));
    }
```

重新编译并部署后查看日志如下：

```
[root@centos7vm 6700]# tail -f worker.log
2016-05-13 08:52:13.558 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:52:13.566 c.s.m.MyBolt [INFO] MyBolt execute
2016-05-13 08:52:14.563 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:52:14.565 c.s.m.MyBolt [INFO] MyBolt execute
2016-05-13 08:52:15.564 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:52:15.568 c.s.m.MyBolt [INFO] MyBolt execute
2016-05-13 08:52:16.565 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:52:16.567 c.s.m.MyBolt [INFO] MyBolt execute
2016-05-13 08:52:17.566 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:52:17.569 c.s.m.MyBolt [INFO] MyBolt execute
2016-05-13 08:52:18.567 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:52:18.569 c.s.m.MyBolt [INFO] MyBolt execute
```

我们看到Bolt开始工作了

那么我们怎么知道Bolt有没有收到数据呢，继续修改Bolt如下：

```
    public void execute(Tuple input, BasicOutputCollector collector) {

        int value = (Integer)input.getValueByField("value");
        LOG.info("MyBolt execute receive " + value);
    }
```

重新编译并部署后查看日志如下：

```
[root@centos7vm 6700]# tail -f worker.log
2016-05-13 08:58:02.565 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:58:02.567 c.s.m.MyBolt [INFO] MyBolt execute receive 10
2016-05-13 08:58:03.566 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:58:03.568 c.s.m.MyBolt [INFO] MyBolt execute receive 10
2016-05-13 08:58:04.567 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:58:04.570 c.s.m.MyBolt [INFO] MyBolt execute receive 10
2016-05-13 08:58:05.567 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:58:05.570 c.s.m.MyBolt [INFO] MyBolt execute receive 10
2016-05-13 08:58:06.569 c.s.m.MySpout [INFO] MySpout nextTuple
2016-05-13 08:58:06.573 c.s.m.MyBolt [INFO] MyBolt execute receive 10
```

试验一下storm的ack机制

在我们的Spout中添加：

```
@Override
public void ack(Object msgId) {
 LOG.info("MySpout arc");
 super.ack(msgId);
}
```

重新编译部署后查看log发现没什么变化，这是为什么呢？

因为我们的bolt没有显式地调用ack函数，所以继续修改bolt，但是只有OutputCollector才能调用ack函数，因此我们再尝试一个bolt基类：BaseRichBolt，把整个代码修改成如下：

```
package com.shareditor.myfirststormpro;

import java.util.Map;

import org.apache.storm.task.OutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichBolt;
import org.apache.storm.tuple.Tuple;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MyBolt extends BaseRichBolt {

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

  private OutputCollector collector;

 public void declareOutputFields(OutputFieldsDeclarer declarer) {
  // TODO Auto-generated method stub

 }

 public void prepare(Map stormConf, TopologyContext context,
   OutputCollector collector) {
  this.collector = collector;
 }

 public void execute(Tuple input) {
  int value = (Integer)input.getValueByField("value");
  LOG.info("MyBolt execute receive " + value);
  collector.ack(input);
 }

}
```

再次编译部署查看log，看到了打印ack的日志啦

总结一下：storm就是帮我们管理了流式计算的服务部署和数据流传递，我们只需要实现具体的业务逻辑即可

教你成为全栈工程师(Full Stack Developer) 二十八-在storm上运行python程序

发表于 2016-05-14 09:02

---

storm是原生支持java的，但同时也支持其他语言，python和java一样有非常多且用起来更爽的类库，所以这一节我们尝试一下使用python来运行storm任务

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

安装lein

下载安装脚本

```
wget https://raw.githubusercontent.com/technomancy/leiningen/stable/bin/lein
```

把lein放到~/bin下并chmod +x lein增加可执行权限（默认我们的bash是会自动把~/bin加到PATH环境变量里的）

执行lein，自动下载安装所需的软件包

安装streamparser

如果没有安装pip（python包的管理工具），先安装pip

```
wget https://bootstrap.pypa.io/get-pip.py --no-check-certificate
python get-pip.py
```

这时pip就安装好了，执行

```
pip -h
```

能看到帮助信息

安装virtualenv命令

```
pip install virtualenv
```

安装python-dev，执行

```
yum install python-devel
```

安装streamparser，执行

```
pip install streamparse
```

注意：如果你是在直接使用root账户，那么需要在~/.bash_profile中添加

```
export LEIN_ROOT=1
```

这是因为在root账户下使用lein时会有警告提示并等待你输入回车才能继续，这样的话下面你执行sparse命令时会莫名其妙的被卡住

创建第一个项目，执行

```
sparse quickstart wordcount
```

运行：

```
cd wordcount
sparse run
```

先会有一段较长时间的编译过程，之后会不断输出log如下说明运行正常：

```
51774 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:27202, name:count-bolt elephant: 1327
51776 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:27202, name:count-bolt dog: 1335
51781 [Thread-27] INFO  backtype.storm.task.ShellBolt - ShellLog pid:27227, name:count-bolt cat: 1335
51783 [Thread-27] INFO  backtype.storm.task.ShellBolt - ShellLog pid:27227, name:count-bolt zebra: 1328
51784 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:27202, name:count-bolt elephant: 1328
```

这都是在本地模拟运行，下面讲解如何部署到真正的storm集群运行

部署到storm并运行

修改config.json文件，配置好自己的storm集群，如下：

```
{
    "library": "",
    "topology_specs": "topologies/",
    "virtualenv_specs": "virtualenvs/",
    "envs": {
        "prod": {
            "user": "",
            "nimbus": "centos",
            "workers": ["centos"],
            "log": {
                "path": "",
                "max_bytes": 1000000,
                "backup_count": 10,
                "level": "info"
            },
            "virtualenv_root": "/tmp/virtualenvs/"
        }
    }
}
```

然后执行sparse submit

等待较长时间之后如果出现了：Finished submitting topology: wordcount，说明部署成功

可能存在的问题

问题1：如果你在部署的单机storm上执行sparse submit还有可能报错：

```
IOError: Local port: 6627 already in use, unable to open ssh tunnel to centos:6627
```

这是因为端口冲突，解决方案就是你找到另外一台机器来安装streamparse并创建wordcount提交就可以了

问题2：如果你部署的storm是1.0.1版本，而sparse版本当前还不支持storm 1.0.1，因此即使部署上去了还是会报错不能正常执行，所以只能部署一个0.9.5版本的storm或者等待sparse什么时候支持1.0.1了

修改project.clj，把

```
:dependencies  [[org.apache.storm/storm-core "0.9.5"]
```

改成

```
:dependencies  [[org.apache.storm/storm-core "1.0.1"]
```

教你成为全栈工程师(Full Stack Developer) 二十九-在storm上运行python程序（修正）

发表于 2016-05-16 08:54

上一节部署python程序出现了一些错误，没能正确运行，本节基于稳定版storm做了修正，运行成功

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

部署0.9.5版本的storm集群

streamparse最新稳定版是基于0.9.5版本的storm的，所以我们需要把storm集群的版本回退到0.9.5，方法如下：

```
wget http://apache.fayea.com/storm/apache-storm-0.9.5/apache-storm-0.9.5.tar.gz
```

解压后修改conf/storm.yaml文件，添加如下配置项：

```
storm.zookeeper.servers:
    - "127.0.0.1"
nimbus.seeds: ["127.0.0.1"]
supervisor.slots.ports:
    - 6700
    - 6701
    - 6702
    - 6703
```

启动，执行：

```
[root@centos7vm apache-storm-0.9.5]# ./bin/storm nimbus &
[root@centos7vm apache-storm-0.9.5]# ./bin/storm supervisor &
[root@centos7vm apache-storm-0.9.5]# ./bin/storm ui &
```

启动需要花费数秒钟时间，直到打开http://localhost:8080能正常显示web页说明启动正常

创建wordcount并修改配置

确定安装的streamparse版本是

```
[root@centos7vm tmp]# sparse --version
sparse 2.1.4
```

执行

```
[root@centos7vm tmp]# sparse quickstart wordcount
```

生成了wordcount目录，进入wordcount目录修改config.json文件

把nimbus和workers配置成你的storm机器（填写别名或ip，需要提前建立好ssh host的无密码登陆），比如我的是：

```
{
    "library": "",
    "topology_specs": "topologies/",
    "virtualenv_specs": "virtualenvs/",
    "envs": {
        "prod": {
            "user": "",
            "nimbus": "centos",
            "workers": ["centos"],
            "log": {
                "path": "",
                "max_bytes": 1000000,
                "backup_count": 10,
                "level": "info"
            },
            "virtualenv_root": "/root/tmp/wordcount/virtualenvs/"
        }
    }
}
```

生成jar包

执行

```
[root@centos7vm wordcount]# sparse jar
```

会看到生成了_build/wordcount-0.0.1-SNAPSHOT-standalone.jar文件

部署任务

执行

```
[root@centos7vm wordcount]# storm jar _build/wordcount-0.0.1-SNAPSHOT-standalone.jar streamparse.commands.submit_topology topologies/wordcount.clj
```

查看日志

```
[root@centos7vm wordcount]# tail /data/apache-storm-0.9.5/logs/worker-6700.log
2016-05-16T08:53:38.579+0800 b.s.t.ShellBolt [INFO] ShellLog pid:74811, name:count-bolt elephant: 656
2016-05-16T08:53:38.580+0800 b.s.t.ShellBolt [INFO] ShellLog pid:74811, name:count-bolt dog: 690
2016-05-16T08:53:38.585+0800 b.s.t.ShellBolt [INFO] ShellLog pid:74811, name:count-bolt zebra: 659
2016-05-16T08:53:38.588+0800 b.s.t.ShellBolt [INFO] ShellLog pid:74811, name:count-bolt dog: 691
2016-05-16T08:53:38.595+0800 b.s.t.ShellBolt [INFO] ShellLog pid:74811, name:count-bolt elephant: 657
2016-05-16T08:53:38.596+0800 b.s.t.ShellBolt [INFO] ShellLog pid:74811, name:count-bolt dog: 692
2016-05-16T08:53:38.604+0800 b.s.t.ShellBolt [INFO] ShellLog pid:74811, name:count-bolt elephant: 658
2016-05-16T08:53:38.606+0800 b.s.t.ShellBolt [INFO] ShellLog pid:74811, name:count-bolt dog: 693
2016-05-16T08:53:38.635+0800 b.s.t.ShellBolt [INFO] ShellLog pid:74811, name:count-bolt zebra: 660
2016-05-16T08:53:38.640+0800 b.s.t.ShellBolt [INFO] ShellLog pid:74811, name:count-bolt dog: 694
```

看到如上日志，说明正常运行了

教你成为全栈工程师(Full Stack Developer) 三十-十分钟掌握最强大的python爬虫

发表于 2016-05-16 08:57

如今大数据是互联网技术的热门，应用也很广泛，所以无论是做互联网产品还是学术研究，抓取他人的资源是快速有效的方法，只要不盗取版权就不为过。开源的爬虫软件很多，本节来介绍最流行也是使用最多的python爬虫开源项目scrapy

安装方法

执行

```
yum install libffi-devel
yum install openssl-devel
pip install scrapy
```

scrapy的代码会安装在

```
/usr/local/lib/python2.7/site-packages/scrapy
```

中文文档在

http://scrapy-chs.readthedocs.io/zh_CN/latest/

使用样例

创建文件shareditor.py如下：

```
# -*- coding: utf-8 -*-
import scrapy


class ShareditorSpider(scrapy.Spider):
    name = 'shareditor'
    start_urls = ['http://www.shareditor.com/']

    def parse(self, response):
        for href in response.css('a::attr(href)'):
            full_url = response.urljoin(href.extract())
            yield scrapy.Request(full_url, callback=self.parse_question)

    def parse_question(self, response):
        yield {
            'title': response.css('h1 a::text').extract()[0],
            'link': response.url,
        }
```

然后执行：

```
scrapy runspider shareditor.py
```

会看到抓取打印的debug信息，它爬取了www.shareditor.com网站的全部网页

是不是很容易掌握？

创建网络爬虫常规方法

上面是一个最简单的样例，真正网络爬虫需要有精细的配置和复杂的逻辑，所以介绍一下scrapy的常规创建网络爬虫的方法

执行

```
[root@centos7vm tmp]# scrapy startproject myfirstpro
```

自动创建了myfirstpro目录，进去看下内容：

```
[root@centos7vm tmp]# cd myfirstpro/myfirstpro/
[root@centos7vm myfirstpro]# ls
__init__.py  items.py  pipelines.py  settings.py  spiders
```

讲解一下几个文件

settings.py是爬虫的配置文件，讲解其中几个配置项：

USER_AGENT是ua，也就是发http请求时指明我是谁，因为我们的目的不纯，所以我们伪造成浏览器，改成

```
USER_AGENT = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36'
```

ROBOTSTXT_OBEY表示是否遵守robots协议（被封禁的不抓取），因为我们的目的不纯，所以我们不遵守，改成

```
ROBOTSTXT_OBEY = False
```

DOWNLOAD_DELAY表示对同一个站点抓取延迟，也就是抓一个，歇一会，再抓一个，再歇一会，为了对对方站点冲(yi)击(bei)太(fa)大(xian)，我们调整为1，也就是一秒抓一个

CONCURRENT_REQUESTS_PER_DOMAIN表示对同一个站点并发有多少个线程抓取，同样道理，我们也调整为1

CONCURRENT_REQUESTS_PER_IP同理也调整为1

items.py是抓取结果解析出来的结构体，一会再说

下面我们进入spiders目录，创建我们的第一个spider程序，如下：

```
import scrapy

class ShareditorSpider(scrapy.Spider):
    name = "shareditor"
    allowed_domains = ["shareditor.com"]
    start_urls = [
        "http://www.shareditor.com/"
    ]

    def parse(self, response):
        filename = response.url.split("/")[-2] + '.html'
        with open(filename, 'wb') as f:
            f.write(response.body)
```

这里面start_urls是初始抓取时的种子链接，parse方法在抓取完成后自动调用，会把抓取回来的body内容写到以.html为结尾的文件中

退到上一级目录执行：

```
[root@centos7vm myfirstpro]# scrapy crawl shareditor
```

执行完成后会多出来一个www.shareditor.com.html文件，内容就是http://www.shareditor.com/网页的内容

页面解析

下面说一下页面解析，这是所有的网络爬虫不能忽略的功能，而且是最核心的部分。python库用于页面解析的有BeautifulSoup（速度慢）和lxml（非标准库），但两者都有一些缺点，所以scrapy基于lxml实现了一套页面解析工具，叫做Selectors

Selector的使用方法非常简单，创建TestSelectors.py如下：

```
from scrapy.selector import Selector
from scrapy.http import HtmlResponse
body = '<html><body><span>good</span></body></html>'
span_text=Selector(text=body).xpath('//span/text()').extract()
print span_text
```

执行python TestSelectors.py输出：

```
[root@centos7vm tmp]# python TestSelectors.py
[u'good']
```

如果能直接拿到HtmlResponse对象，也可以直接调用：

```
response.selector.xpath('//span').extract()
```

有关xpath的内容，可以参考https://www.w3.org/TR/xpath/#location-paths

调试页面解析

scrapy提供了非常方便的页面解析方法，直接执行

```
scrapy shell http://www.shareditor.com/
```

进入调试终端，可以直接使用request、response等变量来做各种操作尝试

注意：

1. 如果想获取文本，调用selector的xpath方法后要调用extract()方法

2. 如果想找到第一个匹配xpath的内容，需要调用extract_first()方法

教你成为全栈工程师(Full Stack Developer) 三十一-利用微信搜索抓取公众号文章

发表于 2016-05-18 13:15

我喜欢看微信公众号里的技术文章，但是总是有一些鸡汤文阻碍我的实现，我是怎么让机器帮我自动摆脱鸡汤文的呢？接下来的几个章节讲述我的解决方案，让你感兴趣的文章扑面而来，无关的鸡汤文随风而去。本章节先将怎么利用搜狗微信搜索抓取公众号的文章

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

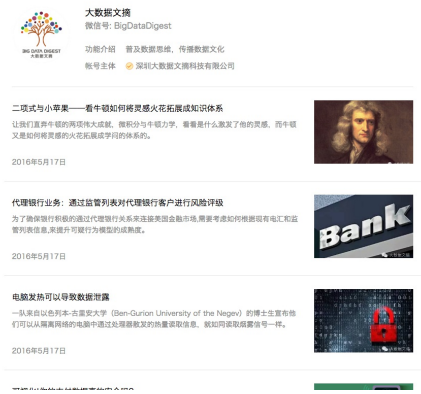
自动收集我关注的微信公众号文章

我的微信里关注了数十个有关大数据的公众号，每天都会出现那个小红点让我点进去看，但是点多了就会觉得烦了，所以我要做的第一步就是自动把公众号里的新文章都收集到一块，怎么做呢？scrapy！

对！scrapy抓取！但是scrapy顺着超链接抓取web网页容易，抓取微信app里的内容就有难度了，暂时还是做不到模拟一个收集app软件。庆幸的是，腾讯和搜狗搜索结婚啦！生出了一个小宝宝：搜狗微信搜索。下面我们就借助搜狗微信搜索来实现我的目的

举个例子，我关注了一个公众号叫：大数据文摘。打开http://weixin.sogou.com/，输入“大数据文摘”，点“搜公众号”，搜索结果如下：

点击这个搜索结果，跳到了新页面



这里面显示的都是最新发布的文章

好！我们就沿着这条路线来追踪公众号的新文章

下面我们来分析一下url

第一个搜索结果页的url是：http://weixin.sogou.com/weixin?type=1&query=%E5%A4%A7%E6%95%B0%E6%8D%AE%E6%96%87%E6%91%98&ie=utf8&_sug_=n&_sug_type_=，我们去掉query以外的参数得到：http://weixin.sogou.com/weixin?query=%E5%A4%A7%E6%95%B0%E6%8D%AE%E6%96%87%E6%91%98，打开之后结果是一样的，ok，这个就作为我们抓取的种子入口，如果搜索其他公众号就把query参数换掉

下面分析搜索结果里怎么提取第二章页面，也就是公众号profile页的链接，我们看下搜索结果页的部分html如下：

<div target="_blank" href="http://mp.weixin.qq.com/profile?src=3&timestamp=1463443372&ver=1&signature=lNY-ZbjfPHr40G-zyUe*Sdc9HIn2IisEo0vwpKEAV*Z*ALBYuYf2HaMUtEP*15rQzs47zSEiORN3BOWPNA2R*A==" class="wx-rb bg-blue wx-rb_v1 _item" uigs_exp_id="" onclick="gotourl('http://mp.weixin.qq.com/profile?src=3&timestamp=1463443372&ver=1&signature=lNY-ZbjfPHr40G-zyUe*Sdc9HIn2IisEo0vwpKEAV*Z*ALBYuYf2HaMUtEP*15rQzs47zSEiORN3BOWPNA2R*A==',event,this);return true;" id="sogou_vr_11002301_box_0" uigs="sogou_vr_11002301_box_0">

<img style="visibility: visible; height: 57px; margin-left: 0px;" src="http://img01.sogoucdn.com/app/a/100520090/oIWsFt58NVJTkYWvPtICKgg8ka60" onload="vrImgLoad(this, 'fit', 57, 57)" onerror="vrImgErr(this, '/wechat/images/account/def56-56.png')" extra="err:'http://img01.sogoucdn.com/net/a/04/link?appid=100520078&url=http://wx.qlogo.cn/mmhead/Q3auHgzwzM46WJlQ8GYRWPhThl25rSKJEYBm408fnEkYS9DUkiaSxGg/0/0'">

<h3><!--red_beg-->大数据文摘<!--red_end--></h3>
<h4>
微信号：<label name="em_weixinhao">BigDataDigest</label>
</h4>

功能介绍：普及<!--red_beg-->数据<!--red_end-->思维,传播<!--red_beg-->数据<!--red_end-->文化

<script>authnamewrite('2')</script>微信认证：深圳大数据文摘科技有限公司

最近文章：二项式与小苹果——看牛顿如何将灵感火花拓展成知识体系<script>vrTimeHandle552write('1463440604')</script>46分钟前

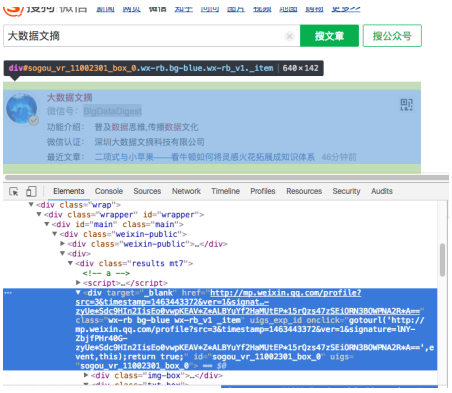
……

看这里关键的href一行：

<div target="_blank" href="http://mp.weixin.qq.com/profile?src=3&timestamp=1463443372&ver=1&signature=lNY-ZbjfPHr40G-zyUe*Sdc9HIn2IisEo0vwpKEAV*Z*ALBYuYf2HaMUtEP*15rQzs47zSEiORN3BOWPNA2R*A==" class="wx-rb bg-blue wx-rb_v1 _item" uigs_exp_id="" onclick="gotourl('http://mp.weixin.qq.com/profile?src=3&timestamp=1463443372&ver=1&signature=lNY-ZbjfPHr40G-zyUe*Sdc9HIn2IisEo0vwpKEAV*Z*ALBYuYf2HaMUtEP*15rQzs47zSEiORN3BOWPNA2R*A==',event,this);return true;" id="sogou_vr_11002301_box_0" uigs="sogou_vr_11002301_box_0">

这就是我们要提取的profile页链接，提取方式可以直接通配成：“url里带http://mp.weixin.qq.com/profile?src=的href属性”

ps：找xpath的方便方法是利用浏览器的开发者工具，比如chrome界面如下：



在Elements的标签处点右键选择：Copy->Copy XPath，就自动把xpath路径拷贝到剪切板了

注意：在这里我突然想到一个问题，每个公众号对应的profile页面是不是永远不变的呢？经过我的实验，这条url里的timestamp参数和signature是有对应关系的，任意一个错了都无法打开，而且每次搜索生成的链接都是不同的，所以我断定在微信搜索内容是动态生成链接的，那么这个动态链接的生命周期就不可预测了，所以为了保险起见，我们每次都从搜索入口追溯，才是万全之策

下面我们分析profile页里的文章链接，我们看profile页的部分 html如下：

```
<h4 class="weui_media_title" hrefs="/s?timestamp=1463443165&amp;src=3&amp;ver=1&amp;signature=dZCo9et5C6nyZfVAQAl416OW-eXJbi0VaS0QPQdvEv1tawqgsjlVYUd0oav0tUHAf38HOGU3Lskd7qqXbFg9D2mP8cv36CZ1dW0bGxbP4YyJcRdy*M*Mow6xD5YWDK8-82r9MX*4WqgbGqo4FAhZeiGTEl27YhIbaIxPiQgMbxc=">代理银行业务：通过监管列表对代理银行客户进行风险评级</h4>
<p class="weui_media_desc">为了确保银行积极的通过代理银行关系来连接美国金融市场,需要考虑如何根据现有电汇和监管列表信息,来提升可疑行为模型的成熟度。</p>
<p class="weui_media_extra_info">2016年5月17日</p>
```

这里面可以找到文章的内容了链接、标题、摘要、发布时间，简直太完美了

链接的提取方式可以直接通配成：h4.weui_media_title hrefs

标题的提取方式可以直接通配成：h4.weui_media_title text

摘要的提取方式可以直接通配成：p.weui_media_desc

发布时间的提取方式可以直接通配成：p.weui_media_extra_info

开发我的scrapy爬虫

如果还没有安装scrapy，请见《教你成为全栈工程师(Full Stack Developer) 三十-十分钟掌握最强大的python爬虫》

创建一个scrapy工程

```
scrapy startproject weixin
```

在weixin/spiders/中创建dashujuwenzhai.py内容如下：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import scrapy

class ShareditorSpider(scrapy.Spider):
    name = "dashujuwenzhai"
    allowed_domains = ["qq.com"]
    start_urls = [
        "http://weixin.sogou.com/weixin?query=大数据文摘"
    ]

    def parse(self, response):
        print response.body
        href = response.selector.xpath('//div[@id="sogou_vr_11002301_box_0"]/@href').extract()[0]
        yield scrapy.Request(href, callback=self.parse_profile)

    def parse_profile(self, response):
        print response.body
```

执行

```
scrapy crawl dashujuwenzhai
```

即可以抓到大数据文摘的profile页面内容

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

接下来来研究profile页，抓回的页面不是普通的html页面，而是通过js渲染出来的，也就是我们看到的每一条文章的标题、摘要等都是通过js计算出来的，代码里有这么一句：

```
var msgList = '{&quot;list&quot;:[{&quot;comm_msg_info&quot;:{&quot;id&quot;:410106318,&quot;type&quot;:49,&quot;datetime&quot;:1463528503,&quot;fakeid&quot;:&quot;2391437564&quot;,&quot;status&quot;:2,&quot;content&quot;:&quot;&quot;},&quot;app_msg_ext_info&quot;:{&quot;title&quot;:&quot;机器人前传：达芬奇的机器狮和日耳曼装甲骑士&quot;,&quot;digest&quot;:&quot;这是一篇描述阿尔法狗和Atlas机器人祖先的文章。远在500多年前的达芬奇时代，已经有了不少关于机器人的探索。这个大天才写了大量关于自动机描述，在他的个人笔记中也充斥着各种机械发明的构思，比如弹簧驱动的汽车和机器狮子。&quot;,&quot;content&quot;:&quot;&quot;,&quot;fileid&quot;:504157567,&quot;content_url&quot;:&quot;\\/s?timestamp=1463529354&amp;amp;src=3&amp;amp;ver=1&amp;amp;signature=cG*R8qc-PGKV-aZ4q9IlJQfIHtGp5I3H63xIK-h5mBO0W2FRAzCddav9cPf*GuwUBI4x0zJzmtcoOU7sQQeMf3CfNzaTEIq4C8YwnsZQGnqnauqr2wQYvEFvAooyecPF3H6bg8OiqpSZsd5LnY*fVrZOMINmQwV8Qup*D9qvUkw=&quot;,&quot;source_url&quot;:&quot;https:\\/\\/mp.weixin.qq.com\\/s?__biz=MzA4OTYwNzk0NA==&amp;amp;mid=401744027&amp;amp;idx=1&amp;amp;sn=43699667dca4438a49db51fb3700af4f&amp;amp;scene=1&amp;amp;srcid=0517MRoAk1EzgC5iSMtvoYC5&amp;amp;pass_ticket=06ybKvJknob%2F5%2B%2FAmkUtnjcyCqWcuNxZTJapLW5QZyk7PWh1jD7ubwb5H1zXzMWB#rd&quot;,&quot;cover&quot;:&quot;http:\\/\\/mmbiz.qpic.cn\\/mmbiz\\/wc7YNPm3YxXiajPXq2Y2PWQsic1SmjCxnTicHKtwItmARwkha1RI1gH1WwTfRvEUzauWJibjuJC9oJ8eibeVIDjRkwg\\/0?wx_fmt=jpeg&quot;,&quot;subtype&quot;:0,&quot;is_multi&quot;:1,&quot;multi_app_msg_item_list&quot;:[{&quot;title&quot;:&quot;清华论坛实录|刘瑞宝:洞见数据内涵，提升公共安全研判能力&quot;,&quot;digest&quot;:&quot;本文为刘瑞宝先生于2016年3月24日在RONG—大数据与公共安全专场上所做的题为《洞见数据内涵，提升公共安全研判能力》的演讲实录。&quot;,&quot;content&quot;:&quot;&quot;,&quot;fileid&quot;:504157565,&quot;content_url&quot;:&quot;\\/s?timestamp=1463529354&amp;amp;src=3&amp;amp;ver=1&amp;amp;signature=cG*R8qc-PGKV-aZ4q9IlJQfIHtGp5I3H63xIK-h5mBO0W2FRAzCddav9cPf*GuwUBI4x0zJzmtcoOU7sQQeMf3CfNzaTEIq4C8YwnsZQGnrmdiX-aBZzJtqDGa76CoHH8gL7PEfN3ZQN5lNa4YgJUeUyE*SIna3B7W*zKWYskkU=&quot;,&quot;source_url&quot;:&quot;https:\\/\\/mp.weixin.qq.com\\/s?__biz=MzAxMzA2MDYxMw==&amp;amp;mid=2651555964&amp;amp;idx=2&amp;amp;sn=479aaf7f3b687b973ffa303d3d3be6b9&amp;amp;scene=1&amp;amp;srcid=0517C5DgLArlrdVAlQ9GIHOl&amp;amp;pass_ticket=06ybKvJknob%2F5%2B
.....
```

当然还没有截取全，这就是文章的全部内容，写到了一个js变量里，这样就无法通过scrapy原生的response.xpath拿到，这怎么办呢？

我们来利用phantomjs来渲染，这是一个强大的工具，它是无界面的浏览器，所以渲染js速度非常快，但是也有一些缺陷，有一些浏览器渲染功能不支持，所以如果再深入可以借助selenium工具，这又是一个强大的工具，它原本是用来做web应用程序自动化测试用的，也就是可以模拟各种点击浏览等动作，那么用他来做爬虫几乎就是一个真人，本节先来研究phantomjs，有关selenium的内容后面有需求了再研究

安装phantomjs

```
wget https://bitbucket.org/ariya/phantomjs/downloads/phantomjs-2.1.1-linux-x86_64.tar.bz2
tar jxvf phantomjs-2.1.1-linux-x86_64.tar.bz2
cd phantomjs-2.1.1-linux-x86_64/
./bin/phantomjs examples/netlog.js http://www.shareditor.com/
```

以上输出了网路通信日志，说明没有问题

为了方便，可以把./bin/phantomjs拷贝到~/bin下

写一个phantomjs渲染脚本

```
var page = require('webpage').create();
var system = require('system');
page.open(system.args[1], function(status) {
    var sc = page.evaluate(function() {
        return document.body.innerHTML;
    });
    window.setTimeout(function() {
        console.log(sc);
        phantom.exit();
    }, 100);
});
```

创建phantomjs渲染脚本getBody.js内容如下：

执行

```
phantomjs getBody.js 'http://mp.weixin.qq.com/profile?src=3&timestamp=1463529344&ver=1&signature=INY-ZbjfPHr40G-zyUe*Sdc9HIn2IisEo0vwpKEAV*Z*ALBYuYf2HaMUtEP*15rQ7TpyhXFL52e8W929D4nd2g==' > profile.html
```

这里的链接可能已经失效，请换成在搜狗微信搜索搜到某个公众号profile页面里的某一篇文章的url

打开profile.html会发现内容已经被渲染完成了，每篇文章的地方变成了：

```
                <div id="WXAPPMSG410106318" class="weui_media_box appmsg" msgid="410106318">
                    <span class="weui_media_hd" style="background-image:url(http://mmbiz.qpic.cn/mmbiz/wc7YNPm3YxXiajPXq2Y2PWQsic1SmjCxnTicHKtwItmARwkha1RI1gH1WwTfRvEUzauWJibjuJC9oJ8eibeVIDjRkwg/0?wx_fmt=jpeg)" data-s="640" data-t="1463528503000" hrefs="/s?timestamp
                    <div class="weui_media_bd">
                        <h4 class="weui_media_title" hrefs="/s?timestamp=1463531541&amp;src=3&amp;ver=1&amp;signature=n187YKNZjqgxyUtJ*yFEQGG7wJOH79RQeRrjQ0RGRdKEiZmR6iM0oNE5P0DPbQEwWTnShlZ4C3JIZr9PYThxbnhuCPl2UTc5NGE0ZkARKXEhTqCe7QvAGFf8vy2QWnPKqA9iSBBgBrocHKLBAuTM

                        机器人前传：达芬奇的机器狮和日耳曼装甲骑士
                        </h4>
                        <p class="weui_media_desc">这是一篇描述阿尔法狗和Atlas机器人祖先的文章。远在500多年前的达芬奇时代，已经有了不少关于机器人的探索。这个大天才写了大量关于自动机描述，在他的个人笔记中也充斥着各种机械发明的构思，比如弹簧驱动的汽车和机器狮子。</p>
                        <p class="weui_media_extra_info">2016年5月18日</p>
                    </div>
                </div>
```

这便可以通过scrapy的request.xpath提取了

重新完善我们的scrapy爬虫脚本

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import scrapy
import subprocess
from scrapy.http import HtmlResponse
from scrapy.selector import Selector

class ShareditorSpider(scrapy.Spider):
    name = "dashujuwenzhai"
    allowed_domains = ["qq.com"]
    start_urls = [
        "http://weixin.sogou.com/weixin?query=算法与数学之美"
    ]

    def parse(self, response):
        href = response.selector.xpath('//div[@id="sogou_vr_11002301_box_0"]/@href').extract()[0]
        cmd="~/bin/phantomjs ./getBody.js '%s'" % href
        stdout, stderr = subprocess.Popen(cmd, shell=True, stdout = subprocess.PIPE, stderr = subprocess.PIPE
).communicate()
        response = HtmlResponse(url=href, body=stdout)

        for selector in Selector(response=response).xpath('//*[@id="history"]/div/div/div/div'):
            hrefs= selector.xpath('h4/@hrefs').extract()[0].strip()
            title = selector.xpath('h4/text()').extract()[0].strip()
            abstract = selector.xpath('//*[contains(@class, "weui_media_desc")]/text()').extract()[0].strip()
            pubtime = selector.xpath('//*[contains(@class, "weui_media_extra_info")]/text()').extract()[0].strip()
            print hrefs
            print title
            print abstract
            print pubtime

    def parse_profile(self, response):
        print response.body
```

这是一段我用了数天精力创造成功的一段代码，耗费了我很多体力值，所以重点讲解一下

```
href = response.selector.xpath('//div[@id="sogou_vr_11002301_box_0"]/@href').extract()[0]
```

从公众号搜索结果页里提取profile页面的链接，这个id我怀疑不久后将失效，所以如果想做完美，还得不断完善，有关xpath的使用技巧可以参考http://ejohn.org/blog/xpath-css-selectors/

```
cmd="~/bin/phantomjs ./getBody.js '%s'" % href
stdout, stderr = subprocess.Popen(cmd, shell=True, stdout = subprocess.PIPE, stderr = subprocess.PIPE).co
mmunicate()
```

加载phantomjs脚本getBody.js来渲染profile页面，把里面的js渲染成html

```
response = HtmlResponse(url=href, body=stdout)
```

用渲染后的html页面来创建一个HtmlResponse，用于 后面继续xpath提信息

```
Selector(response=response).xpath('//*[@id="history"]/div/div/div/div')
```

找到每一条文章模块所在的div

```
            hrefs= selector.xpath('//h4/@hrefs').extract()[0].strip()
            title = selector.xpath('h4/text()').extract()[0].strip()
            abstract = selector.xpath('//*[contains(@class, "weui_media_desc")]/text()').extract()[0].strip()
            pubtime = selector.xpath('//*[contains(@class, "weui_media_extra_info")]/text()').extract()[0].strip()
```

根据这个div结构提取各个字段

基于这个爬虫脚本，想造就怎样的神奇，就看你之后的想象力了，没有做不到，只有想不到！

教你成为全栈工程师(Full Stack Developer) 三十二-scrapy爬虫抓取网页写入mysql数据库

发表于 2016-05-19 07:42

scrapy抓取的网页默认存成了python的dict结构，scrapy提供了pipeline接口来存储数据，为了方便以后使用，我们把抓取的结构化内容存入mysql

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

安装MySQL-python

```
[root@centos7vm ~]# pip install MySQL-python
```

执行如下不报错说明安装成功：

```
[root@centos7vm ~]# python
Python 2.7.5 (default, Nov 20 2015, 02:00:19)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import MySQLdb
>>>
```

创建page表

为了保存网页，在mysql数据库中创建page表，sql语句如下：

```
CREATE TABLE `page` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `post_date` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `post_user` varchar(255) COLLATE utf8_unicode_ci DEFAULT '',
  `body` longtext COLLATE utf8_unicode_ci,
  `content` longtext COLLATE utf8_unicode_ci,
  PRIMARY KEY (`id`),
  UNIQUE KEY `title` (`title`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

其中title是文章标题，post_date是文章发布时间，post_user是发布者（也就是公众号），body是网页原始内容，content是提取出的纯文本格式的正文

创建item结构

在我们的scrapy项目里修改item.py文件，用来保存提取出来的结构化数据，内容如下：

```
import scrapy

class WeixinItem(scrapy.Item):
    # define the fields for your item here like:
    title = scrapy.Field()
    post_date = scrapy.Field()
    post_user = scrapy.Field()
    body = scrapy.Field()
    content = scrapy.Field()
```

生成item结构

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

修改爬虫脚本，在parse函数中增加如下几句：

```
    def parse_profile(self, response):
        title = response.xpath('//title/text()').extract()[0]
        post_date = response.xpath('//em[@id="post-date"]/text()').extract()[0]
        post_user = response.xpath('//a[@id="post-user"]/text()').extract()[0]
        body = response.body
        tag_content = response.xpath('//div[@id="js_content"]').extract()[0]
        content = remove_tags(tag_content).strip()
        item = WeixinItem()
        item['title'] = title
        item['post_date'] = post_date
        item['post_user'] = post_user
        item['body'] = body
        item['content'] = content
        return item
```

注：如果不知道爬虫脚本怎么写，请看上一篇文章《教你成为全栈工程师(Full Stack Developer) 三十一-利用微信搜索抓取公众号文章》

另外：这里的content是去标签后的纯文本，使用了remove_tags，这需要加载库：

```
from w3lib.html import remove_tags
```

创建pipelines

scrapy持久化数据的方式是通过pipeline来实现。开源的各种爬虫软件均会对持久化方式提供各种方法，比如pyspider提供了写入mysql、mongodb、文件等的持久化方法，scrapy作为爬虫老将把接口留给了我们，我们可以自定义各种pipeline，同时可以通过配置灵活选择

pipeline机制通过pipelines.py文件和settings.py文件结合实现

修改scrapy项目里的pipelines.py内容如下：

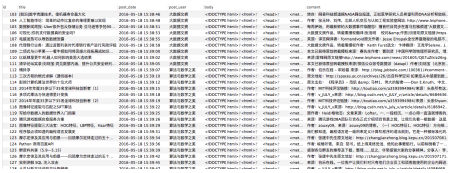
```
# -*- coding: utf-8 -*-
import sys
reload(sys)
sys.setdefaultencoding('utf8')
import MySQLdb

class WeixinPipeline(object):
    def __init__(self):
        self.conn = MySQLdb.connect(host="127.0.0.1",user="myname",passwd="mypasswd",db="mydbname",c
harset="utf8")
        self.cursor = self.conn.cursor()

    def process_item(self, item, spider):
        sql = "insert ignore into page(title, post_user, body, content) values(%s, %s, %s, %s)"
        param = (item['title'], item['post_user'], item['body'], item['content'])
        self.cursor.execute(sql,param)
        self.conn.commit()
```

里面的数据库配置按照自己的修改好，这里的process_item会在抓取时自动调用，并把爬虫脚本返回的item通过参数传递进来，这里通过insert将item结构化数据插入了mysql数据库

下面再看settings.py文件，如下：

```
ITEM_PIPELINES = {
    'weixin.pipelines.WeixinPipeline': 300,
}
```

运行爬虫后看到数据库如下：



相当完美，准备用这些数据来作为训练样本来做机器学习之用，预知后事如何，且听下回分解

教你成为全栈工程师(Full Stack Developer) 三十三-利用scikit-learn计算tf-idf做文本词频分析

发表于 2016-05-20 09:13

为了让机器帮我自动筛出我想看的文章，我利用上一节实现的爬虫抓取了近500篇微信公众号文章，接下来我来讲述我是怎么对这500篇文章做机器训练的，本节先说一下怎么做tf-idf计算

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

TF-IDF

TF-IDF（Term Frequency and Inverse Document Frequency），是一种用于信息检索与数据挖掘的常用加权技术。它的主要思想是：如果某个词或短语在一篇文章中出现的频率（term frequency）高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。

计算公式是TF * IDF

而这里的：

$$\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

$$\mathrm{idf_i} = \log \frac{|D|}{|\{j : t_i \in d_j\}|}$$

scikit-learn

基于python的一种机器学习工具，主要功能包括：分类、回归、聚类、数据降维、模型选择、数据预处理等

安装步骤：

```
pip install numpy
pip install scipy
pip install scikit-learn
```

这里如果报出了MemoryError，则增加--no-cache-dir参数，执行

```
pip --no-cache-dir install numpy
pip --no-cache-dir install scipy
pip --no-cache-dir install scikit-learn
```

词语转矩阵

写一个testscikit.py文件，内容如下：

```
# coding:utf-8
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

corpus=["中华 民族 血汗",
        "人民 血汗",
        "共和 国 数学 中华 英语 物理"]

vectorizer=CountVectorizer()

print type(vectorizer.fit_transform(corpus))
print vectorizer.fit_transform(corpus)
print vectorizer.fit_transform(corpus).todense()
```

这里面CountVectorizer是一个向量计数器

第一个print type(vectorizer.fit_transform(corpus))输出结果如下：

```
<class 'scipy.sparse.csr.csr_matrix'>
```

这说明fit_transform把corpus二维数组转成了一个csr_matrix类型（稀疏矩阵）

第二个print vectorizer.fit_transform(corpus)输出结果如下：

```
  (0, 0)    1
  (0, 4)    1
  (0, 7)    1
  (1, 7)    1
  (1, 1)    1
  (2, 0)    1
  (2, 2)    1
  (2, 3)    1
  (2, 6)    1
  (2, 5)    1
```

这就是稀疏矩阵的表示形式，即把二维数组里的所有词语组成的稀疏矩阵的第几行第几列有值

第三个print vectorizer.fit_transform(corpus).todense()输出如下：

```
[[1 0 0 0 1 0 0 1]
 [0 1 0 0 0 0 0 1]
 [1 0 1 1 0 1 1 0]]
```

这就是把稀疏矩阵输出成真实矩阵

下面我们把代码改成：

```
# coding:utf-8
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

corpus=["中华 民族 血汗",
        "人民 血汗",
        "共和 国 数学 中华 英语 物理"]

vectorizer=CountVectorizer()
csr_mat = vectorizer.fit_transform(corpus)
transformer=TfidfTransformer()
tfidf=transformer.fit_transform(csr_mat)
print type(tfidf)
print tfidf
print tfidf.todense()
```


输出如下：

```
<class 'scipy.sparse.csr.csr_matrix'>
  (0, 7)    0.517856116168
  (0, 4)    0.680918560399
  (0, 0)    0.517856116168
  (1, 1)    0.795960541568
  (1, 7)    0.605348508106
  (2, 5)    0.467350981811
  (2, 6)    0.467350981811
  (2, 3)    0.467350981811
  (2, 2)    0.467350981811
  (2, 0)    0.35543246785
[[ 0.51785612  0.          0.          0.68091856  0.          0.          0.51785612]
 [ 0.          0.79596054  0.          0.          0.          0.          0.60534851]
 [ 0.35543247  0.          0.46735098  0.46735098  0.          0.46735098  0.46735098  0.        ]]
```

注意：scikit-learn算出的tf-idf值和我们用自己的公式计算出来的不一样，因为它做了很多其他的工作，比如去停用词、平滑、正则化等，只要数值都是用scikit-learn算出来的，就有可比性

从数据可以看出，返回的数据依然是稀疏矩阵csr_matrix结构

分析一下这个tf-idf数据

只出现一次的词在不同总词数的文档之间的对比：数学在6个词文档里是0.46735098，民族在三个词文档里是0.68091856，人民在两个词文档里是0.79596054，可见文档总次数越多，tf-idf越小

在所有文档里出现过一次的词和出现过两次的词在同一个文档里的tf-idf对比：数学只出现过1次，在文档3里是0.46735098，中华出现过两次，在文档3里是0.35543247，因此出现频次越多相对来说tf-idf越小

总之tf-idf越大，这个词对于这个文档越重要

我们继续修改代码，添加如下一行：

```
word=vectorizer.get_feature_names()
for w in word:
    print w
```

输出如下：

```
中华
人民
共和
数学
民族
物理
英语
血汗
```

这里为什么没有“国”字呢？

我们做这样一个实验，在corpus里添加一些一个字的词，如：

```
corpus=["中华 民族 血汗",
        "人民 血汗",
        "共和 国 数学 中华 英语 物理 号 弄 门"]
```

重新执行输出依然是：

```
中华
人民
共和
数学
民族
物理
英语
血汗
```

经过搜寻网上的文档了解到：

note that CountVectorizer discards "words" that contain only one character, such as "s"

也就是说因为scikit-learn原生是处理英文的，所以对于单个字母的词（如：a、an、I）会被过滤掉。这个feature对于中文来说其实也试用，所以我们不去解决这个事情

好，我们整理一下，添加如下代码：

```
for i in range(len(tfidf_array)):
    line=""
    for j in range(len(word)):
        line="%s\t\t%s" % (line,word[j])
    print line

print

for i in range(len(tfidf_array)):
    line=""
    for j in range(len(word)):
        line="%10s\t\t%10s" % (line,tfidf_array[i][j])
    print line
```

输出如下：

```
      中华      人民      共和      数学      民族      物理      英语      血汗
      中华      人民      共和      数学      民族      物理      英语      血汗
      中华      人民      共和      数学      民族      物理      英语      血汗

            0.517856116168          0.0           0.0           0.0       0.680918560399          0.0
0.0       0.517856116168
                 0.0       0.795960541568          0.0           0.0           0.0           0.0           0.0
0.605348508106
            0.35543246785           0.0       0.467350981811      0.467350981811          0.0       0.46735
0981811       0.467350981811          0.0
```

可以很清晰的看到词矩阵，以及每个词在每个文档里的tfidf了

教你成为全栈工程师(Full Stack Developer) 三十四-基于python的高效中文文本切词

发表于 2016-05-20 18:55

对中文文本做自然语言处理一定设计到切词的操作，流行的切词工具有很多，本文介绍基于python的结巴中文分词，它是基于文本分析的，不是基于词库的，因此更精确

安装jieba分词工具

在https://pypi.python.org/pypi/jieba/下载jieba-0.38.zip

解压后执行：

```
python setup.py install
```

试验切词效果

创建testjieba.py文件内容如下：

```
# coding:utf-8
#!/usr/local/bin/python

import jieba

seg_list = jieba.cut("www.shareditor.com全栈工程师教程、讲述程序猿自己的故事，shareditor涉及机器学习、深度学习、大数据相关知识的系统学习方法及热文推荐")
for str in seg_list:
    print str
```

输出分词结果如下：

```
www
.
shareditor
.
com
全栈
工程师
教程
、
讲述
程序
猿
自己
的
故事
，
shareditor
涉及
机器
学习
、
深度
学习
、
大
数据
相关
知识
的
系统
学习
方法
及
热文
推荐
```

jieba.cut返回的是python的generator结构，遍历后得到分词结果中包括：中文词汇（单字、多字）、英文单词、标点符号

对批量文档分词

我们之前抓取的微信公众号文章都保存到了mysql的page表里，并且提取纯文本内容保存在了content列中，所以我们从mysql加载content并做切词，之后再利用scikit-learn计算tf-idf值

创建cut_and_cal_tfidf.py，内容如下：

```
# coding:utf-8

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import jieba
import MySQLdb

conn = MySQLdb.connect(host="127.0.0.1",user="myuser",passwd="mypasswd",db="mydb",charset="utf8")
cursor = conn.cursor()

sql = "select content from page"
cursor.execute(sql)
corpus=[]
for content in cursor.fetchall():
    seg_list = jieba.cut(content[0])
    line = ""
    for str in seg_list:
        line = line + " " + str
    corpus.append(line)
conn.commit()
conn.close()

vectorizer=CountVectorizer()
csr_mat = vectorizer.fit_transform(corpus)
transformer=TfidfTransformer()
tfidf=transformer.fit_transform(csr_mat)
word=vectorizer.get_feature_names()
print tfidf.toarray()
```

执行后输出结果如下：

```
Building prefix dict from the default dictionary ...
Loading model from cache /var/folders/tq/c0kp5y857x138x5vf8bzxfc80000gp/T/jieba.cache
Loading model cost 0.447 seconds.
Prefix dict has been built succesfully.
[[ 0.          0.00567306  0.         ...,  0.          0.          0.        ]
 [ 0.          0.          0.         ...,  0.          0.          0.        ]
 [ 0.          0.          0.         ...,  0.          0.          0.        ]
 ...,
 [ 0.          0.          0.         ...,  0.          0.          0.        ]
 [ 0.          0.          0.         ...,  0.          0.          0.        ]
 [ 0.          0.          0.         ...,  0.          0.          0.        ]]
```

讲解一下

先说下代码逻辑：首先加载数据库的content数据，如果对于page数据库表不熟请回过头来看《教你成为全栈工程师(Full Stack Developer) 三十二-scrapy爬虫抓取网页写入mysql数据库》，然后用jieba.cut对每一条content做分词，把分词按空格分隔存到二维数组corpus中，最后用scikit-learn计算tfidf，如果对这部分不熟请回过头来看《教你成为全栈工程师(Full Stack Developer) 三十三-利用scikit-learn计算tf-idf做文本词频分析》

再说下输出的内容，前面几行文字是jieba分词器自动打印的加载词典的信息，下面是整个tf-idf矩阵，因为维度太高，scikit-learn库自动帮我们用...做了省略

教你成为全栈工程师(Full Stack Developer) 三十五-利用sonataadmin做样

本标注

发表于 2016-05-21 11:04

要想实现对新发现的公众号文章做自动分类，需要对样本做训练，而样本属于哪一类别是需要我们做人工标注的，直接操作数据库非常不方便，我想到了我们的网站后台管理系统，直接把样本数据整合进来就可以方便标注了

创建CrawlPage实体

在sonataadmin中每一张表都要对应一个Entity实体，也就是MVC里的model，因此我们在网站目录中创建src/AppBundle/Entity/CrawlPage.php，内容如下：

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * CrawlPage
 *
 * @ORM\Table()
 * @ORM\Entity
 */
class CrawlPage
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="title", type="string", length=255, unique=true)
     */
    private $title;

    /**
     * @var string
     *
     * @ORM\Column(name="body", type="text")
     */
    private $body;

    /**
     * @var string
     *
     * @ORM\Column(name="content", type="text", nullable=true)
     */
    private $content;

    /**
```

```
    /**
     * @var \DateTime
     *
     * @ORM\Column(name="create_time", type="datetime")
     */
    private $createTime;

    /**
     * @var string
     *
     * @ORM\Column(name="source", type="string", length=255)
     */
    private $source;

    /**
     * @var string
     * @ORM\Column(name="isTec", type="string", length=8, options={"default": "0"})
     */
    private $isTec;

    /**
     * @var string
     * @ORM\Column(name="isSoup", type="string", length=8, options={"default": "0"})
     */
    private $isSoup;

    /**
     * @var string
     * @ORM\Column(name="isMR", type="string", length=8, options={"default": "0"})
     */
    private $isMR;

    /**
     * @var string
     * @ORM\Column(name="isMath", type="string", length=8, options={"default": "0"})
     */
    private $isMath;
……
```

省略的部分是getter和setter

这里面除了网页表本身的几个字段（title、body、content、createTime、source）之外，创建几个0-1标注字段：isTec（是否纯技术）、isSoup（是否鸡汤文）、isMR（是否机器学习）、isMath（是否数学），这个用来做分类标注

另外，我们为title添加了unique=true属性，为了保证表里的title是唯一key，避免重复写入数据。同时为几个标注字段添加了options={"default": "0"}选项，默认都是非选中

创建CrawlPage管理类

为了在sonataadmin后台页面中方便标注数据，创建CrawlPage的管理类，创建src/AppBundle/Admin/CrawlPageAdmin.php文件，内容如下：

```php
<?php

namespace AppBundle\Admin;

use Sonata\AdminBundle\Admin\Admin;
use Sonata\AdminBundle\Datagrid\ListMapper;
use Sonata\AdminBundle\Form\FormMapper;
use Sonata\AdminBundle\Datagrid\DatagridMapper;

class CrawlPageAdmin extends Admin
{
    protected function configureFormFields(FormMapper $formMapper)
    {
        $formMapper
            //->tab('Post')
            ->with('Content', array('class' => 'col-md-9'))
            ->add('title', 'text')
            ->add('source', 'text')
            ->add('body', 'ckeditor', array('autoload' => true))
            ->add('create_time', 'sonata_type_date_picker', array(
                'format'=>'yyyy-MM-dd HH:mm:ss',
                'dp_default_date'         => date('Y-m-d H:i:s'),))
            ->end()

            ->with('Meta data', array('class' => 'col-md-3'))
            ->end();
    }

    protected function configureListFields(ListMapper $listMapper)
    {
        $listMapper
            ->addIdentifier('title')
            ->add('isTec', 'boolean', array('editable' => 'Yes'))
            ->add('isSoup', 'boolean', array('editable' => 'Yes'))
            ->add('isMR', 'boolean', array('editable' => 'Yes'))
            ->add('isMath', 'boolean', array('editable' => 'Yes'))
            ->add('source')
            ->add('createTime')
            ;
    }

    public function toString($object)
    {
        return $object instanceof BlogPost
            ? $object->getTitle()
            : 'Crawl Page'; // shown in the breadcrumb on the create view
    }

    protected function configureDatagridFilters(DatagridMapper $datagridMapper)
    {
        $datagridMapper
            ->add('title')
            ;
    }
}
```

注意这里的configureListFields方法中的

```
            ->add('isTec', 'boolean', array('editable' => 'Yes'))
            ->add('isSoup', 'boolean', array('editable' => 'Yes'))
            ->add('isMR', 'boolean', array('editable' => 'Yes'))
            ->add('isMath', 'boolean', array('editable' => 'Yes'))
```

使用boolean类型可以在列表页中通过ajax修改数据，这样就不必点开文章、修改、保存那么麻烦了

修改配置并生效

修改app/config/services.yml添加：

```
    admin.crawl_page:
        class: AppBundle\Admin\CrawlPageAdmin
        arguments: [~, AppBundle\Entity\CrawlPage, ~]
        tags:
            - { name: sonata.admin, manager_type: orm, label: Crawl page }
```

执行

```
php app/console doctrine:schema:update --force
```

来生效数据库表

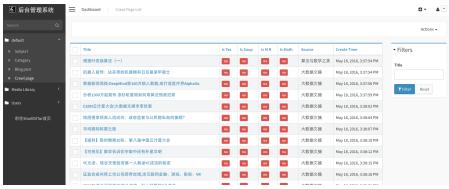
灌入微信公众号网页数据

如果还没有抓取公众号文章，请见《教你成为全栈工程师(Full Stack Developer) 三十二-scrapy爬虫抓取网页写入mysql数据库》

因为之前写入了一个单独的数据库，我们现在把这部分网页导入进来，执行sql语句：

```
insert into CrawlPage(title, create_time, source, body, content) select title, post_date, post_user, body, content f
rom test.page;
```

打开后台管理页面看下效果：



现在可以直接点击“no”标签来标注数据啦

教你成为全栈工程师(Full Stack Developer) 三十六-对微信公众号文章做样本标注与特征提取

发表于 2016-05-23 09:10

基于上一节实现的web界面的样本标注系统做人工标注，然后详细讲解如何对标注好的样本做挖掘和分析，并根据分析结果提取出最优代表性的特征，用于后面的训练

多类分类问题解法

解法一：通过一系列两类分类问器并将它们组合到一起形成多类分类器

解法二：将多个分类面的参数求解合并到一个最优化问题中

我们利用解法一，通过多个两类分类问题分别计算

人工标注

这部分工作完全是基于个人的判断，逐个文章进行标注，如果判断文章属于纯技术类，则把isTec标记为yes，如果判断为鸡汤文，则把isSoup标记为yes，其他两类也一样

经过我耗时近一小时的纯手工标注，最终每类文章数为：

```
select sum(isTec), sum(isSoup), sum(isMR), sum(isNews) from CrawlPage;


sum(isTec)    sum(isSoup)    sum(isMR)    sum(isNews)
31    98    69    240
```

切词并保存

下面我要把这四个类别的所有文章做切词，为了调试需要，我们把切词之后的中间结果保存在数据库中，以便重复调试不用每次都做切词操作，所以我们在php的CrawlPage实体中增加如下变量：

```
/**
 * @var text
 * @ORM\Column(name="segment", type="text", nullable=true)
 */
private $segment;
```

执行

```
php app/console doctrine:schema:update --force
```

后数据库会多处一列

```
  `segment` longtext COLLATE utf8_unicode_ci,
```

创建我们的feature_extract.py，内容如下：

```
# coding:utf-8

import sys
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import jieba
from jieba import analyse
import MySQLdb

conn = MySQLdb.connect(host="127.0.0.1",user="myuser",passwd="mypasswd",db="mydatabase",charset="utf8")

def get_segment():
    cursor = conn.cursor()
    sql = "select id, content from CrawlPage"
    cursor.execute(sql)
    jieba.analyse.set_stop_words("stopwords.txt")
    for result in cursor.fetchall():
        id = result[0]
        content = result[1]
        seg_list = jieba.cut(content)
        line = ""
        for str in seg_list:
            line = line + " " + str
        line = line.replace('\'', ' ')
        sql = "update CrawlPage set segment='%s' where id=%d" % (line, id)
        try:
            cursor.execute(sql)
            conn.commit()
        except Exception,e:
            print line
            print e
            sys.exit(-1)
    conn.close()

if __name__ == '__main__':
    get_segment();
```


这里我们对每一篇文章做切词，并且把切词后的结果存储到segment列中

注意：为了避免sql的语法问题，需要把文章里的单引号'\''去掉，这里我替换成了空格，方便切词识别

计算tf-idf

继续编辑feature_extract.py，增加如下内容：

```
def feature_extract():
    cursor = conn.cursor()
    category={}
    category[0] = 'isTec'
    category[1] = 'isSoup'
    category[2] = 'isMR'
    category[3] = 'isMath'
    category[4] = 'isNews'

    corpus=[]
    for index in range(0, 5):
        sql = "select segment from CrawlPage where " + category[index] + "=1"
        cursor.execute(sql)
        line = ""
        for result in cursor.fetchall():
            segment = result[0]
            line = line + " " + segment
        corpus.append(line)

    conn.commit()
    conn.close()

    vectorizer=CountVectorizer()
    csr_mat = vectorizer.fit_transform(corpus)
    transformer=TfidfTransformer()
    tfidf=transformer.fit_transform(csr_mat)
    word=vectorizer.get_feature_names()
    print tfidf.toarray()

if __name__ == '__main__':
    #get_segment();
    feature_extract();
```

执行后输出：

```
[[ 0.          0.          0.         ...,  0.          0.          0.        ]
 [ 0.          0.          0.         ...,  0.          0.          0.        ]
 [ 0.00670495  0.00101195  0.00453306 ...,  0.          0.          0.        ]
 [ 0.          0.00164081  0.         ...,  0.          0.          0.        ]
 [ 0.01350698  0.0035783   0.         ...,  0.0003562   0.0003562   0.00071241]]
```

特征提取

我们采取分别对每一类看做一个两类分类问题来求解，所以对这5大类别分别做特征提取，提取的方式就是提取每一类中tf-idf最大的n个特征，首先我们先把全部特征输出出来

```
    for index in range(0, 5):
        f = file("tfidf_%d" % index, "wb+")
        for i in np.argsort(-tfidf.toarray()[index]):
            if tfidf.toarray()[index][i] > 0:
                f.write("%f %s\n" % (tfidf.toarray()[index][i], word[i]))
        f.close()
```

这已经按照tf-idf从大到小排序了，所以从生成的5个文件里前n行就能拿到我们需要的n个特征啦

下一节我们将通过提取出来的特征来对测试样本进行测试

教你成为全栈工程师(Full Stack Developer) 三十七-利用支持向量机做文本分类

发表于 2016-05-25 08:47

从上一节提取出的全部特征中选取出关键的特征，并利用支持向量机对测试样本做回归计算，判断准确率

选取出关键特征

通过tf-idf计算出来的数值是某个特征（词）对于这篇文档的权重，不代表这个特征（词）在文本分类中的权重。这很容易理解，比如某一个特征（词）在多个分类中的tf-idf是不一样的，但是这个特征对于这个分类问题的权重肯定是一个定值。

选取重要的特征的方法可以是：1.）按tf-idf排序从大到小选topN；2）按特征的普遍性选取（在多个类别中出现过）；3）按特征在不同文档中tf-idf的差距选择；

我们这次采取结合的形式：在至少2个类别中tf-idf大于0，同时在多个类别中第一名高于第二名10%以上。这么选择的原因是：我的总类别一共5种比较少，所以2个类别以上就说明具有普遍性了，你可以根据你的类别数目调整，第一名高于第二名10%表示这个特征具有一定的区分度。

修改我们的feature_extract.py如下：

```
def feature_dump():
    cursor = conn.cursor()
    category={}
    category[0] = 'isTec'
    category[1] = 'isSoup'
    category[2] = 'isMR'
    category[3] = 'isMath'
    category[4] = 'isNews'

    corpus=[]
    for index in range(0, 5):
        sql = "select segment from CrawlPage where " + category[index] + "=1"
        print sql
        cursor.execute(sql)
        line = ""
        for result in cursor.fetchall():
            segment = result[0]
            line = line + " " + segment
        corpus.append(line)

    conn.commit()
    conn.close()

    vectorizer=CountVectorizer()
    csr_mat = vectorizer.fit_transform(corpus)
    transformer=TfidfTransformer()
    tfidf=transformer.fit_transform(csr_mat)
    word=vectorizer.get_feature_names()
    print tfidf.toarray()

    for index in range(0, 5):
        f = file("tfidf_%d" % index, "wb")
        for i in np.argsort(-tfidf.toarray()[index]):
            if tfidf.toarray()[index][i] > 0:
                f.write("%f %s\n" % (tfidf.toarray()[index][i], word[i]))
        f.close()
def feature_extraction():
    d = {}
    for index in range(0, 5):
        f = file("tfidf_%d" % index, "r")
        lines = f.readlines()
        for line in lines:
            word = line.split(' ')[1][:-1]
            tfidf = line.split(' ')[0]
            if d.has_key(word):
                d[word] = np.append(d[word], tfidf)
            else:
                d[word] = np.array(tfidf)

        f.close();
    f = file("features.txt", "wb")
    for word in d:
        if d[word].size >= 2:
            index = np.argsort(d[word])
            if float(d[word][index[d[word].size-0-1]]) - float(d[word][index[d[word].size-1-1]]) > 0.01:
                f.write("%s %s\n" % (word, d[word]))
    f.close()

if __name__ == '__main__':
    #get_segment();
    feature_dump();
    feature_extraction();
```

最终输出的features.txt中有809个特征供我们使用，如下：

```
集群 ['0.027741' '0.014016' '0.010606']
分类器 ['0.002870' '0.045052' '0.000943']
中心 ['0.008167' '0.001647' '0.004274' '0.006954' '0.031360']
首席 ['0.003015' '0.017036']
fit ['0.016885' '0.000284']
懂得 ['0.035888' '0.001629' '0.003064']
密度 ['0.002414' '0.002106' '0.015073' '0.001586']
master ['0.021045' '0.002002' '0.000471']
对方 ['0.002414' '0.020451' '0.001684' '0.003569']
物品 ['0.020088' '0.001158' '0.001414']
……
```

研究scikit-learn SVM的使用

利用支持向量机模型可以直接使用scikit-learn，下一节我们再来研究一下scikit-learn的支持向量机怎么用

教你成为全栈工程师(Full Stack Developer) 三十八-100行python代码实现机器学习自动分类

发表于 2016-05-25 09:17

现在朋友圈、公众号、微博信息应接不暇，以微信公众号举例，看技术极客是怎么自动筛出自己想看的文章的，100行机器学习代码就能自动帮你归好类，要想找出想看的和不想看的，你再也不用刷朋友圈了

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

准备工作

1. 准备一张mysql数据库表，至少包含这些列：id、title(文章标题)、content(文章内容)、segment(中文切词)、isTec(技术类)、isSoup(鸡汤类)、isMR(机器学习类)、isMath(数学类)、isNews(新闻类)

2. 根据你关注的微信公众号，把更新的文章的title和content自动写入数据库中，具体方法见《教你成为全栈工程师(Full Stack Developer) 三十一-利用微信搜索抓取公众号文章》

3. 收集一部分文章做人工标注，按照你自己的判断找几百篇文章标记出属于那种类别

100行python代码

代码如下：

```
# coding:utf-8

import sys
reload(sys)
sys.setdefaultencoding( "utf-8" )

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import jieba
```

```
import jieba
from jieba import analyse
import MySQLdb
import numpy as np
from sklearn import metrics
from sklearn.svm import SVC

conn = MySQLdb.connect(host="127.0.0.1",user="myuser",passwd="mypasswd",db="mydatabase",charset="ut
f8")

def get_segment(all=False):
    cursor = conn.cursor()

    if True == all:
        # 找到全部文章的标题和内容
        sql = "select id, title, content from CrawlPage"
    else:
        # 找到尚未切词的文章的标题和内容
        sql = "select id, title, content from CrawlPage where segment is null"
    cursor.execute(sql)

    for row in cursor.fetchall():
        print "cutting %s" % row[1]
        # 对标题和内容切词
        seg_list = jieba.cut(row[1]+' '+row[2])
        line = ""
        for str in seg_list:
            line = line + " " + str
        line = line.replace('\", ' ')

        # 把切词按空格分隔并去特殊字符后重新写到数据库的segment字段里
        sql = "update CrawlPage set segment='%s' where id=%d" % (line, row[0])
        try:
            cursor.execute(sql)
            conn.commit()
        except Exception,e:
            print line, e
            sys.exit(-1)

def classify():
    cursor = conn.cursor()

    # 一共分成5类，并且类别的标识定为0，1，3，4，5
    category_ids = range(0, 5)
    category={}
    category[0] = 'isTec'
    category[1] = 'isSoup'
    category[2] = 'isMR'
    category[3] = 'isMath'
    category[4] = 'isNews'

    corpus = []
    for category_id in category_ids:
        # 找到这一类的所有已分类的文章并把所有切词拼成一行，加到语料库中
        sql = "select segment from CrawlPage where " + category[category_id] + "=1"
        cursor.execute(sql)
        line = ""
        for result in cursor.fetchall():
            segment = result[0]
            line = line + " " + segment
        corpus.append(line)

    # 把所有未分类的文章追加到语料库末尾行
    sql = "select id, title, segment from CrawlPage where isTec=0 and isSoup=0 and isMR=0 and isMath=0 and i
sNews=0"
    cursor.execute(sql)
```

```
    line = ""
    update_ids = []
    update_titles = []
    need_predict = False
    for result in cursor.fetchall():
        id = result[0]
        update_ids.append(id)
        title = result[1]
        update_titles.append(title)
        segment = result[2]
        corpus.append(segment)
        need_predict = True

    if False == need_predict:
        return

    # 计算tf-idf
    vectorizer=CountVectorizer()
    csr_mat = vectorizer.fit_transform(corpus)
    transformer=TfidfTransformer()
    tfidf=transformer.fit_transform(csr_mat)
    y = np.array(category_ids)

    # 用前5行已标分类的数据做模型训练
    model = SVC()
    model.fit(tfidf[0:5], y)

    # 对5行以后未标注分类的数据做分类预测
    predicted = model.predict(tfidf[5:])

    # 把机器学习得出的分类信息写到数据库
    for index in range(0, len(update_ids)):
        update_id = update_ids[index]
        predicted_category = category[predicted[index]]
        print "predict title: %s <===============> category: %s" % (update_titles[index], predicted_category)
        sql = "update CrawlPage set %s=1 where id=%d" % (predicted_category, update_id)
        cursor.execute(sql)

    conn.commit()

if __name__ == '__main__':
    # 切词
    get_segment()
    # 分类
    classify()
    conn.close()
```


原理说明

python代码利用了scikit-learn机器学习库的svm（支持向量机）算法，先计算出tf-idf矩阵，这个矩阵包括已经标注好分类的前5行，和未进行分类5行以后的行，以前5行为样本交给svm做训练，然后用训练出来的模型对5行以后的部分做预测，并写回数据库

代码说明

get_segment()利用了jieba中文分词器对文章内容做分词，并保存到数据库的segment列中，如果已经做过分词，则不再重复做

classify()分五部分：

第一部分先取出同一类文章的segment数据拼成一行作为特征向量

第二部分取出未分类的文章的segment数据追加到后面，每篇文章一行

第三部分以第一部分为输入做SVM模型训练

第四部分用训练好的SVM模型对第二部分的行做预测

第五部分把预测出来的分类信息写到数据库中

效果

我人工标注了600多篇文章，然后对未标注的测试样本执行上面代码做预测，准确率在90%以上，以后就再也不用“刷”公众号文章了，我感兴趣的是机器学习和数学部分，不喜欢看的就是鸡汤文，以后就只看机器学习和数学类别的文章就行喽！看看我刚才随便抓了几篇文章的自动分类效果吧

```
predict title: 刷屏朋友圈的北京彩虹图，你看到的是它真实的颜色吗？ <==============> category: isNews
predict title: 玩儿游戏也能学编程？12个学习编程的游戏化平台 <==============> category: isNews
predict title: 互联网教父凯文·凯利：大数据时代没有旁观者 <==============> category: isNews
predict title: 对话周涛：抓住大数据最性感的方向 <==============> category: isNews
predict title: 视频 | MIT研发了一款可食用机器人，能移除胃部异物 <==============> category: isNews
predict title: 京津冀大数据峰会八月亮相北京大数据展 <==============> category: isNews
predict title: 世界这么大，一起去看看（大数据观察） <==============> category: isNews
predict title: 京筑合作共建大数据战略重点实验室 <==============> category: isNews
predict title: 细数IT巨头们那些年十大悔断肠的错误决定 <==============> category: isNews
predict title: 从BAT看企业构建大数据体系的六层级 <==============> category: isNews
predict title: 关于数学的恐怖故事：从前有棵树,叫高数,树上挂了很多人 <==============> category: isMath
predict title: 最前沿的数字内容大会上，他们这么创作 VR 内容 <==============> category: isNews
predict title: TED|数学告诉你 完美伴侣如何选择 <==============> category: isNews
predict title: 德鲁克：自我管理的七个维度 <==============> category: isSoup
predict title: 福利|听课免费,还送礼品:亚马逊云计算AWS在线技术干货52小时 <==============> category: isN
ews
predict title: 【重磅长文】技术和科学是不同的源流，一万字读懂人类技术史 <==============> category: isN
ews
predict title: Slideshare上火爆的PPT | 互联网世界的60秒 <==============> category: isNews
predict title: 【数据新闻周报】小米谷歌共同研发VR设备，阿里旅行发布未来酒店2.0战略 <==============>
category: isNews
predict title: 手把手：使用OpenCV进行面部合成— C++ / Python <==============> category: isTec
predict title: “云南省经济社会大数据研究院”揭牌仪式正式举行 <==============> category: isNews
predict title: [50页报告下载]《中英开放数据发展和合作》报告新鲜发布 <==============> category: isNews
```

教你成为全栈工程师(Full Stack Developer) 三十九-简单几步做到外链自动生成，从此妈妈再也不用担心我的SEO

发表于 2016-05-30 08:12

搜索引擎排名最关键的数据是高质量外链，无法被被人收录，我们就自己做一个收录自己的网站，同时用最佳的相关性生成高质量外链，百度也无法辨别真伪

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

具体方案

用抓取回来的所有网页做一个简单的好文站，用来做外链，引流量。然后写一个complete_link.py，遍历所有CrawlPage，找到content没有标记过<a>的，拿它的title去查elasticSearch，得到的第一篇文章，提出这篇文

章的标题做成anchor添加到content尾部

为什么做高质量外链？

搜索引擎判断一个网站质量的好坏，主要会看有没有优秀的网页指向了它，就好比“你自己说你好，你不一定真的好，很多人说你好，你才是真的好”。强调高质量，就是说不是什么外链都起到加分的作用，如果在新浪头条引用你的网站，那会加分，一个垃圾站提到你，没有什么用。

怎么样的外链是高质量的？

首先引用你的网站比较优秀，本身打分比较高，那么你就会被抬高。其次，需要有优秀描述的锚文本，优秀体现在：简短的话，准确表述。百度对网页的标题和锚文本是最高优先级做相关性匹配的，所以标题和锚文本非常重要。然后，重复的引用不要太多，以免被判为垃圾。

为什么用elasticSearch？

这不是重点，重点是要找到一个最相关的链接，这样用户转化才够精准，找到这样的链接不求多，一个就够了。

为什么用标题作为锚文本

因为标题是对这篇文章最好的描述。锚文本内容是用来做检索召回的，用户搜了这个词，才会返回这个结果。

详细步骤

安装elasticsearch的python扩展

```
pip install elasticsearch
```

创建complete_link.py，内容如下：

```
# coding:utf-8

import sys
reload(sys)
sys.setdefaultencoding( "utf-8" )

import MySQLdb
import re
from conn import Conn
from elasticsearch import Elasticsearch

es = Elasticsearch()

conn = Conn().getConnection()
cursor = conn.cursor()
upcursor = conn.cursor()
sql = "select id, title, substring_index(content,'相关原创文章,敬请关注',1) from CrawlPage where content not like '
%</a>'"
cursor.execute(sql)
for row in cursor.fetchall():
    id = row[0]
    title = row[1]
    content = row[2]
    title = re.sub('\[|\]|\/|\'|\"|\(|\)|\!|\?|\~',",title)

    try:
        res = es.search(index="app", body={"fields":["title"],"size":1,"query": {"query_string": {"query":title}}})
        for hit in res['hits']['hits']:
            print "process:", title
            update_sql = "update CrawlPage set content=CONCAT(content,\"%s%s%s%s%s\") where id=%d" % (
                    " 相关原创文章,敬请关注：<a href='http://www.shareditor.com/blogshow/?blogId=",
                    hit['_id'],
                    "'>",
                    hit['fields']['title'][0],
                    "</a>",
                    id)
            upcursor.execute(update_sql)
            conn.commit()

    except Exception,e:
        print "Error:"
        print title
        print e
        sys.exit(-1)
```

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

讲解一下

```
update_sql = "update CrawlPage set content=CONCAT(content,\"%s%s%s%s%s\") where id=%d" % (
  " 相关原创文章,敬请关注：<a href='http://www.shareditor.com/blogshow/?blogId=",
  hit['_id'],
  "'>",
  hit['fields']['title'][0],
  "</a>",
  id)
```

这是生成链接的样子，锚文本是文章标题，链接指向文章页面

```
res = es.search(index="app", body={"fields":["title"],"size":1,"query": {"query_string": {"query":title}}})
```

是查询elasticSearch搜博客，取第一条，因为第一条是相关性最高的一篇

有关elasticSearch的python接口详细可以参考：http://elasticsearch-py.readthedocs.io/en/latest/

最终效果

教你成为全栈工程师(Full Stack Developer) 四十-pdf文档自动生成方法

发表于 2016-06-07 19:20

积累了很多博客，希望能产出一个pdf电子书来分发给可能感兴趣的人，但是编辑pdf是比较繁琐的，怎么能直接利用我们的html页面生成一篇pdf呢？本节讲解pdf的自动生成方法

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

安装配置KnpSnappyBundle扩展

先安装扩展，执行：

```
composer require knplabs/knp-snappy-bundle
```

再安装wkhtmltopdf工具

```
wget http://download.gna.org/wkhtmltopdf/0.12/0.12.3/wkhtmltox-0.12.3_linux-generic-amd64.tar.xz
xz -d wkhtmltox-0.12.3_linux-generic-amd64.tar.xz
```

拷贝二进制到/usr/local/bin

因为linux存在无法转中文的问题，所以下载一个simsun.ttc拷贝到/usr/share/fonts即可

创建pdf链接

配置路由如下：

```
pdf_latest:
    path:     /pdf/{title}_{year}_{month}_{day}.pdf
    requirements:
        "year": "201[6-9]"
        "month": "[01]\d"
        "day": "[0123]\d"
    defaults: { _controller: AppBundle:Pdf:generate }
```

这里的title后面会作为查数据的关键词来形成一类pdf（这样做的是为了按不同类别把文章生成多个pdf）

这里的年月日后面会用来限定博客的最后更新日志，这样这个pdf会看起来定期有更新

配置里的**requirements**通过正则表达式约束**url**的值

创建Pdf**Contoller**

如下：

```
<?php

namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\BinaryFileResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpKernel\Kernel;
use Doctrine\ORM\EntityManager;
use Doctrine\ORM\Query;
use Doctrine\Common\Persistence\ObjectRepository;

class PdfController extends Controller
{

    /**
     * @var ObjectRepository
     */
    protected $blogPostRepository;
    /**
     * @var Kernel
     */
    protected $kernel;

    /**
     * @var EntityManager
     */
    protected $em;

    public function generateAction(Request $request, $title, $year, $month, $day)
    {
        $time = mktime(0, 0, 0, $month, $day, $year);
        $now = time();
        if ($time > $now) {
            return new Response('file not exist');
        }
        $this->kernel = $this->get('kernel');
        $rootDir = $this->kernel->getRootDir();
        $pdfFilePath = $rootDir . '/../web/pdf/' . $title . '/' . $year . "-" .  $month . "-" .  $day . '.pdf';

        $this->em     = $this->get('doctrine.orm.entity_manager');

        $qb = $this->em->createQueryBuilder();
        $q = $qb->select(array('blogpost'))->from('AppBundle:BlogPost', 'blogpost')
            ->where($qb->expr()
                ->lt('blogpost.createTime', '\'' . date("Y-m-d 00:00:00", $time) . '\'')
            )
            ->andWhere($qb->expr()
                ->like('blogpost.title', '\'%' . $title . '%\'')
            )
            ->getQuery();

        if (file_exists($pdfFilePath)) {
            unlink($pdfFilePath);
        }
        $this->get('knp_snappy.pdf')->generateFromHtml(
```

```
            $this->renderView(
                'pdf/generate.html.twig',
                array('blogposts' => $q->getresult())
            ),
            $pdfFilePath
        );


        return new BinaryFileResponse($pdfFilePath);
    }

}
```


讲解一下：

首先要判断指定的年月日是不是超过当前日期，如果超过会返回错误

接下来确定生成本地的pdf文件路径

然后用url获取到的年月日和title来查数据库，获取博客集合

再利用**knp_snappy.pdf服务把网页渲染好并转成pdf**

最后返回pdf文件作为Response

创建渲染模板generate.html.twig

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="utf-8">
    <!-- 新 Bootstrap 核心 CSS 文件 -->
    <link rel="stylesheet" href="https://cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap.min.css">

    <!-- 可选的Bootstrap主题文件（一般不用引入） -->
    <link rel="stylesheet" href="https://cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap-theme.min.css">

    <!-- jQuery文件。务必在bootstrap.min.js 之前引入 -->
    <script src="https://cdn.bootcss.com/jquery/1.11.3/jquery.min.js"></script>

    <!-- 最新的 Bootstrap 核心 JavaScript 文件 -->
    <script src="https://cdn.bootcss.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>

</head>
<body>
{% for blogpost in blogposts %}

    <div class="container-fluid">
        <div class="row">
            <div class="col-sm-3 col-xs-1"></div>
            <div class="col-sm-6 col-xs-10">
                <div class="row">
                    <h1><a href="http://{{ app.request.httphost }}{{ path('blog_show', {'blogId':blogpost.id}) }}"
                           target="_blank">{{ blogpost.title }}</a></h1>
                </div>
                <div class="row">
                    <small>发表于 {{ blogpost.createTimeStr }}</small>
                </div>

                <div class="row">
                    <hr/>
                </div>
                <div class="row">
                    <br/>
                    {{ blogpost.body|replace({'img src="http://www.shareditor.com':"img src=\"http://#{app.request.httph
ost}"})|raw }}
                </div>
            </div>
        </div>
    </div>

    <br/>
    <br/>

{% endfor %}
</body>
</html>
```

讲解一下：

这依然是按照html的方式加载bootstrap框架，遍历所有的博客并展示他们的title、createTimeStr、body等

注意：这里的a和img里的链接都是用完整的url，这是因为如果用相对的url则wkhtmltopdf无法生成超链接和图片

直接访问http://localhost/app_dev.php/pdf/全栈工程师/2016-06-07.pdf就可以直接下载到2016-06-07以前title包含“全栈工程师”的所有博客拼成的一个pdf文件啦

教你成为全栈工程师(Full Stack Developer) 四十一-如何准确的统计文章浏览量(pv)

发表于 2016-06-13 20:50

自己写的原创文章很想知道都有多少网友浏览过，单纯的显示次数并不能准确地反应真实浏览量，还需要解决一些意想不到的问题，本节我把刚刚完成的浏览量统计方案分享出来

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

数据库表建立

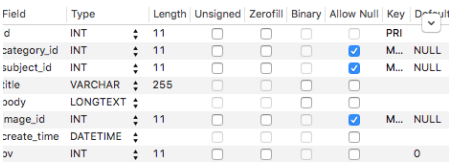
pv（Page View，网页浏览量）是针对blog文章的，因此数据库表直接在BlogPost表中扩充，修改BlogPost.php，增加如下内容：

```
/**
 * @var int
 *
 * @ORM\Column(name="pv", type="integer", options={"default": "0"})
 */
private $pv;

/**
 * @return mixed
 */
public function getPv()
{
    return $this->pv;
}

/**
 * @param mixed $pv
 */
public function setPv($pv)
{
    $this->pv = $pv;
}
```

并执行：

```
php app/console doctrine:schema:update --force
```

生效数据库，我们可以看到数据库多了一列pv，默认值为0：



方案

浏览量统计的关键点在于“排除干扰因素”，所谓的干扰因素主要是非真实用户浏览：爬虫、恶意刷网页等

因此我们不采取实时浏览写数据的方案，我们采取基于浏览日志做清洗和统计

首先，分析了一下我们的日志，通过不断的grep一些关键词（spider、bot）总结出如下爬虫关键，后面我们把这部分过滤掉：

```
YunGuanCe|Baiduspider|Sogou web spider|spiderman|pyspider|bingbot|Googlebot|MJ12bot|Slurp|Uptimebot|SafeDNSBot|AhrefsBot|YandexBot|GimmeUSAbot|Alibaba|YisouSpider|360Spider|semanticbot|TeeRaidBot|Cliqzbot|pycurl|SurveyBot|yandex.com/bots|linkdexbot|xml-sitemaps|Slackbot-LinkExpanding|Applebot|rogerbot|org_bot|TweetmemeBot|Twitterbot|Scrapy
```

然后，继续通过统计来源ip的方式发现存在一类无http_referer、无http_user_agent、无http_x_forwarded_for的三无日志，类似这样：

```
180.153.163.191 - - [11/Jun/2016:18:30:31 +0800] "GET /blogshow/?blogId=30 HTTP/1.1" 200 18399 "-" "-" "-"
```

这种一定不是正常的浏览器，因此我们也要过滤掉

日志清洗

按照上述方案，我们的日志清洗脚本新鲜出炉：

```
#!/bin/bash

date

LOG_PATH="/data/httpdir/logs/shareditor2.0.log"

SPIDER_UA="YunGuanCe|Baiduspider|Sogou web spider|spiderman|pyspider|bingbot|Googlebot|MJ12bot|Slurp|Uptimebot|SafeDNSBot|AhrefsBot|YandexBot|GimmeUSAbot|Alibaba|YisouSpider|360Spider|semanticbot|TeeRaidBot|Cliqzbot|pycurl|SurveyBot|yandex.com/bots|linkdexbot|xml-sitemaps|Slackbot-LinkExpanding|Applebot|rogerbot|org_bot|TweetmemeBot|Twitterbot|Scrapy"

egrep -v "$SPIDER_UA" $LOG_PATH | grep 'GET /blogshow' | grep -v '"-" "-" "-"' > ./clean.log
```

生成的clean.log就是清洗后的日志

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

pv统计

下面就是基于清洗后的日志做pv统计了，创建gen_pv.py如下：

```
#!/usr/bin/python
# coding:utf-8

import sys
reload(sys)
sys.setdefaultencoding( "utf-8" )

import MySQLdb
from conn import Conn
import re

pattern1 = re.compile('/blogshow/\?blogId=([0-9][0-9]*)$')
pattern2 = re.compile('/blogshow/([0-9][0-9]*)$')

pv={}

f = open('./clean.log')
try:
    while True:
        line = f.readline()
        if line:
            fields = line.split(' ')
            url_path = fields[6] # /blogshow/3 or /blogshow/?blogId=3
            match1 = pattern1.match(url_path)
            match2 = pattern2.match(url_path)
            blog_id = -1
            if match1:
                blog_id = int(match1.group(1))
            if match2:
                blog_id = int(match2.group(1))

            if blog_id != -1:
                if pv.has_key(blog_id):
                    pv[blog_id] = pv[blog_id] + 1
                else:
                    pv[blog_id] = 1

        else:
            break
finally:
    f.close( )

conn = Conn().getConnection()
cursor = conn.cursor()

for id in pv:
    sql = "update BlogPost set pv=%d where id=%d" % (pv[id], id)
    cursor.execute(sql)
    conn.commit()

conn.close()
```

讲解一下：

因为网站做过改版，blogshow网页的url存在过两种形式，因此要把这两部分都考虑进来，所以做了两部分正则匹配

获取到全部blog的pv数据最后刷入数据库

crontab配置

为了能比较及时又不损失服务器性能，我们配置5分钟执行一次统计，配置crontab如下：

```
*/5 * * * * cd /data/gen_pv && sh run.sh &>> run.sh.log
```

教你成为全栈工程师(Full Stack Developer) 四十二-利用n-n表关联实现文章标签

发表于 2016-06-20 08:59

当文章积累了比较多时，总是希望能够从某一个方面把整理到一起，但一篇文章可能同时属于大数据类和机器学习类，所以这会是一个多对多的关系，那么如何实现这种关系呢？本节我们通过一种标签功能来说明

创建标签Entity

首先我们要创建标签实体类，创建src/AppBundle/Entity/Tag.php文件，内容为：

```
<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use Doctrine\Common\Collections\ArrayCollection;

/**
 * Tag
 *
 * @ORM\Table()
 * @ORM\Entity
 */
class Tag
{
    /**
     * @ORM\ManyToMany(targetEntity="BlogPost", mappedBy="tag")
     */
    private $blogPosts;

    public function __construct()
    {
        $this->blogPosts = new ArrayCollection();
    }

    public function getBlogPosts()
    {
        return $this->blogPosts;
    }


    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
```

```
    @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="name", type="string", length=255)
     */
    private $name;

    /**
     * Get id
     *
     * @return integer
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * Set name
     *
     * @param string $name
     * @return Tag
     */
    public function setName($name)
    {
        $this->name = $name;

        return $this;
    }

    /**
     * Get name
     *
     * @return string
     */
    public function getName()
    {
        return $this->name;
    }
}
```

创建关联关系

有了标签实体Tag，我们要把BlogPost和Tag关联起来，这是一个多对多的关系，因此在src/AppBundle/Entity/BlogPost.php中添加：

```
/**
 * @ORM\ManyToMany(targetEntity="Tag")
 * @ORM\JoinTable(name="blogpost_tag",
 *   joinColumns={@ORM\JoinColumn(name="blogpost_id", referencedColumnName="id")},
 *   inverseJoinColumns={@ORM\JoinColumn(name="tag_id", referencedColumnName="id")}
 * )
 */
private $tags;
```

同时生成它的get和set方法

```
/**
 * @return mixed
 */
public function getTags()
{
    return $this->tags;
}

/**
 * @param mixed $tags
 */
public function setTags($tags)
{
    $this->tags = $tags;
}
```

创建mysql表

执行

```
php app/console doctrine:schema:update --force
```

会自动生成两个表：

```
Tag(id, name)
blogpost_tag(blogpost_id, tag_id)
```

增加Tag的Admin类

为了能在sonataadmin管理后台对Tag进行管理，我们创建src/AppBundle/Admin/TagAdmin.php内容如下：

```
<?php

namespace AppBundle\Admin;

use Sonata\AdminBundle\Admin\Admin;
use Sonata\AdminBundle\Datagrid\ListMapper;
use Sonata\AdminBundle\Datagrid\DatagridMapper;
use Sonata\AdminBundle\Form\FormMapper;

class TagAdmin extends Admin
{
    protected function configureFormFields(FormMapper $formMapper)
    {
        $formMapper->add('name', 'text');
    }

    protected function configureDatagridFilters(DatagridMapper $datagridMapper)
    {
        $datagridMapper->add('name');
    }

    protected function configureListFields(ListMapper $listMapper)
    {
        $listMapper->addIdentifier('name');
    }
}
```

增加tag的service配置

修改app/config/services.yml，增加：

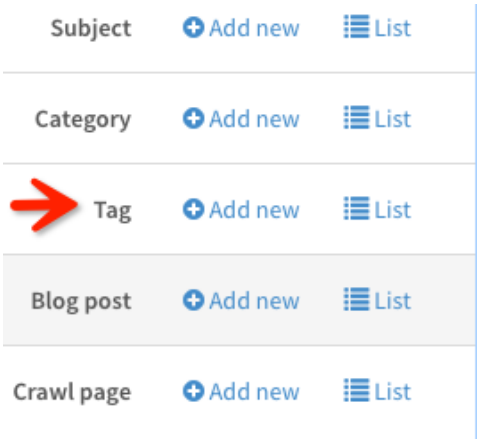
```
admin.tag:
    class: AppBundle\Admin\TagAdmin
    arguments: [~, AppBundle\Entity\Tag, ~]
    tags:
        - { name: sonata.admin, manager_type: orm, label: Tag }
```

增加BlogPost的tag管理控件

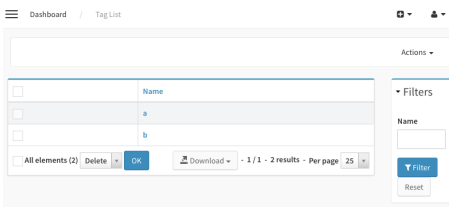
修改src/AppBundle/Admin/BlogPostAdmin.php，在configureFormFields中增加

```
->with('Meta data', array('class' => 'col-md-3'))
->add('tags', null, array(
    'class' => 'AppBundle\Entity\Tag',
    'property' => 'name',
))
->end()
```

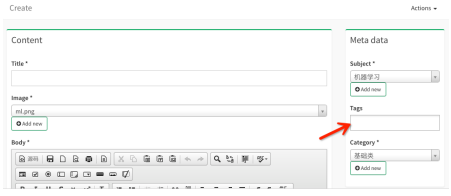

好，现在看一下效果，打开后台管理页面，我们看到多了Tag的一组管理：



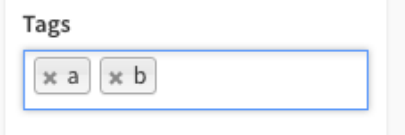
点开List可以看到我刚刚添加的两个标签



新建一个BlogPost，也会发现在右侧多了Tag的填写框：



因为BlogPost和Tag是多对多的关系，所以这里是可以填写多个值的：



Tag的展示

Tag的管理功能已经完成了，那么怎么才能把tag展示出来呢？我们希望有两处展示，一处是单独一个页面，列出指定tag的所有文章列表，另一处是在文章展示页面展示出它的所有标签，同时链接到tag文章列表

为BlogPost创建listbytagAction，修改src/AppBundle/Controller/BlogController.php，在适当位置增加如下内容（如果你看了前面的文章，你就知道放到什么位置了）：

```
use Doctrine\ORM\EntityManager;
use Doctrine\ORM\QueryBuilder;



/**
 * @var EntityManager
 */
protected $em;
/**
 * @var QueryBuilder
 */
protected $builder;




public function listbytagAction(Request $request)
{
    $tagName = $request->get('tagname');
    $this->em = $this->get('doctrine.orm.entity_manager');
    $this->builder = $this->em->createQueryBuilder();
    $query = $this->builder->select('b')
        ->add('from', 'AppBundle:BlogPost b INNER JOIN b.tags t')
        ->where('t.name=:tag_name')
        ->setParameter('tag_name', $tagName)
        ->getQuery();

    $paginator = $this->get('knp_paginator');
    $pagination = $paginator->paginate(
        $query,
        $request->query->get('page', 1)/*page number*/,
        100/*limit per page*/
    );

    return $this->render('blog/listbytag.html.twig', array('pagination' => $pagination,
        'tagname' => $tagName,
        'latestblogs' => BlogController::getLatestBlogs($this),
        'tophotblogs' => BlogController::getTopHotBlogs($this)));
}
```

创建app/Resources/views/blog/listbytag.html.twig，内容如下：

```
{% extends "base.html.twig" %}

{% block title %}{{ tagname }} - SharEDITor - 关注大数据技术{% endblock title %}

{% block body %}

<div class="row">
    <div class="col-sm-3 col-xs-1"></div>
    <div class="col-sm-6 col-xs-10">
        <h1>{{ tagname }}</h1>
    </div>
    <div class="col-sm-3 col-xs-1"></div>
</div>
<div class="row">
    <div class="col-sm-3 col-xs-1"></div>
    <div class="col-sm-6 col-xs-10">
        <br />
        {% for article in pagination %}
            <h4><a href="{{ path('blog_show', {'blogId':article.id}) }}">{{ article.title }}</a>({{ article.createDate }})</h4
>
        {% endfor %}
        <div class="navigation">
            {{ knp_pagination_render(pagination) }}
        </div>

    </div>
    <div class="col-sm-3 col-xs-1"></div>
</div>

{% endblock body %}
```

为这个action创建路由，修改app/config/routing.yml，增加如下内容：

```
blog_listbytag:
    path:     /bloglistbytag/
    defaults: { _controller: AppBundle:Blog:listbytag }
```

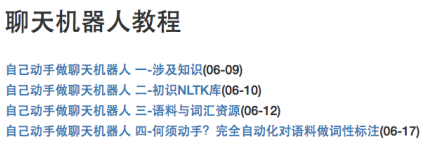
修改app/Resources/views/blog/show.html.twig，在展示subject和category两个标签的后面添加：

```
{% for tag in blogpost.tags %}
<a class="btn btn-warning btn-xs" href="{{ path('blog_listbytag', {'tagname':tag.name}) }}">
    {{ tag.name }}
</a>
{% endfor %}
```

在app/Resources/views/blog/list.html.twig也同样加入如下内容：

```
{% for tag in article.tags %}
    <a class="btn btn-warning btn-xs" href="{{ path('blog_listbytag', {'tagname':tag.name}) }}">
        {{ tag.name }}
    </a>
{% endfor %}
```

下面欣赏一下最终效果：



教你成为全栈工程师(Full Stack Developer) 四十三-网站优化必备良器：点击事件的上报与统计

发表于 2016-07-11 08:41

有这样一个问题：怎么样对网站页面做布局是最优的？光凭站长YY显然是不够的，用户也许对你精心设计的按钮或链接不感兴趣，甚至都没注意到那里竟然还可以点。为了探究用户的行为，有效的做法是在你想了解的地方埋下点击事件触发的上报，本节来介绍如何用简短的几行代码实现点击事件上报和统计

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

为什么要做网站优化

先来看一下我的网站当前的页面：

在这个页面中除了正文部分，还有一些链接是希望能够带来用户点击的，比如顶部的github和weibo链接，还有底部的最新文章和最热文章，但是我设计在顶部和底部，而不是设计在正文左侧和右侧，完全是因为我觉得这样好看，然而用户会不会点击我并不清楚，所以我需要埋下点击事件来监控用户的行为，来判断我这样的设计是不是最优的

为什么要通过事件上报而不是访问日志

访问日志我们是不清楚用户是点击哪里的链接，即使有referrer字段也无法判断细节，因此我们通过js的点击事件来上报。这样做还有一个好处，就是可以排除掉爬虫的足迹，因为爬虫为了降低成本，一般不会渲染js，也就不会触发点击上报事件。

利用cnzz做事件上报和统计

自己实现上报事件需要做的事情包括：定接口、埋js、配置http服务、开发自动统计和可视化等。这一系列工作cnzz已经帮我们实现了大部分了，智者当借力而行，推荐直接用cnzz，当然也有很多其他类似的工具可以选择

cnzz的事件上报文档见http://open.cnzz.com/a/new/trackevent/

首先注册账号添加站点，然后在你要添加事件上报的页面中(可以统一放到模板中)的<head>标签里的地方添加如下代码(**这里**和官方文档的描述有些不同，因为按照官方文档的方法你页面上总会出现一个跳到cnzz的一个链接)：

```
<script type="text/javascript">
    var cnzz_protocol = (("https:" == document.location.protocol) ? " https://" : " http://");
    var _czc = _czc || [];
    _czc.push(["_setAccount", "********"]);
</script>
```

这里的******替换成你在siteid(可以在你的页面统计代码里找到)

然后在你想上报点击事件的标签里添加如下属性：

```
onclick="_czc.push(['_trackEvent', category,action,label,value,nodeid);"
```

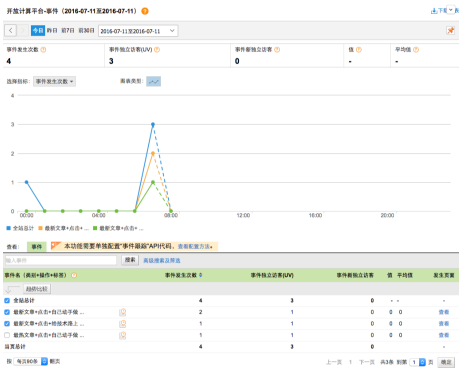
比如我埋在github和微博点击事件的代码如下：

```
<div class="fa fa-github" style="margin-top: 40px; margin-left: 20px;">
    <a href="https://github.com/warmheartli" style="color: white;"
       onclick="_czc.push(['_trackEvent', 'github链接', '点击']);"
    >
        github
    </a>
</div>

<div class="fa fa-weibo" style="margin-top: 40px; margin-left: 20px;">
    <a href="http://weibo.com/chuangwanglaile" style="color: white;"
       onclick="_czc.push(['_trackEvent', 'weibo链接', '点击']);"
    >
        weibo
    </a>
</div>
```


我在底部“最新文章”和“最热文章”中也埋下了：

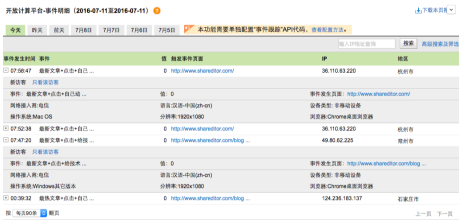
```
<div class="row" style="margin: 10px;margin-left: 0; overflow:hidden;text-overflow:ellipsis;white-space:nowrap;"
>
    <span style="color: #959595;">({{ article.createDate }})</span>
    <a title="{{ article.title }}" style="color: #959595;"
       href="{{ path('blog_show', {'blogId':article.id}) }}"
       onclick="_czc.push(['_trackEvent', '最新文章', '点击', '{{ article.title }}']);"
    >{{ article.title }}</a>
</div>



<div class="row" style="margin: 10px;margin-left: 0;overflow:hidden;text-overflow:ellipsis;white-space:nowrap;">
    <span style="color: #FFFFFF; background-color: #AAAAAA; border-radius: 12px;padding-left: 5px;padding-rig
ht: 5px;">{{ article.pv }}</span>
    <a title="{{ article.title }}" style="color: #959595;"
       href="{{ path('blog_show', {'blogId':article.id}) }}"
       onclick="_czc.push(['_trackEvent', '最热文章', '点击', '{{ article.title }}']);"
    >{{ article.title }}</a>
</div>
```

以上完成后可以在cnzz后台看到页面上的点击事件统计效果：



我们还可以看点击事件发生的url：

还可以看每次点击发生在什么操作系统下：



总之，无论是什么点击上报的第三方服务，基本都能覆盖我们想做的各种基础性统计，当然如果想要做更深层次的分析，还需要你自己按照自己的业务逻辑来做进一步挖掘分析

教你成为全栈工程师(Full Stack Developer) 四十四-如何基于数据统计做网站页面布局的优化和适配

发表于 2016-07-13 08:51

当今是移动互联的时代，网站针对移动设备的适配是必须的，那么网站页面怎样布局是最合理的，又如何针对pc和移动端做不同的布局设计呢？本节我们基于数据统计来试验一下不同页面布局的效果

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址

移动适配的必要性

pc的页面布局的特点是宽度比较大，一般可以分成两栏或三栏，除了正文部分，左侧和右侧都可以放置一些相关的组件，比如：相关链接、推广等。而移动端宽度一般比较窄，只能做一栏，因此这些相关组件只能放在底部或以其他方式展现。因此特殊针对移动端做适配是有必要的。

怎么做移动适配

移动适配属于表现层的范畴，因此最好在前端代码中实现，bootstrap已经为我们抽象出了方便的适配方法，在div布局中可以通过区分class是**col-sm-*和col-xs-*来分别表示pc端和移动端的视图，而hidden-xs表示移动端这个div隐藏，这为移动适配提供了极大的方便**

我是怎么做移动适配的

搜索框的移动适配

文章搜索需求是为了满足用户针对特殊关键词做搜索提供的，放置在了我们网站右上角，如下：

```
<div class="col-sm-3 hidden-xs">
    <form action="{{ path('blog_search') }}" style="margin-top: 10px;">
        <input type="search" name="q" placeholder="搜文章" maxlength="200" style="background-color: transpare
nt;">
    </form>
</div>
```

这里面声明了hidden-xs，表示在移动端隐藏，因为在移动端搜索是一个高成本的功能，用户需要手工输入文字，一般情况下，用户在手机主要是阅览，因此这个在手机端做了隐藏

右侧系列链接的移动适配

为了方便用户浏览全部相关文章，也为了提升网站的pv，在右侧加入了本文章所属tag的相关链接

首先修改src/AppBundle/Controller/BlogController.php在showAction()方法中添加如下代码：

```
        $tags = $this->blogPost->getTags();
        $pagination = null;
        if (!empty($tags) && sizeof($tags) > 0) {
            $tag = $tags[0];
            $tagName = $tag->getName();
            $this->em = $this->get('doctrine.orm.entity_manager');
            $this->builder = $this->em->createQueryBuilder();
            $query = $this->builder->select('b')
                ->add('from', 'AppBundle:BlogPost b INNER JOIN b.tags t')
                ->where('t.name=:tag_name')
                ->setParameter('tag_name', $tagName)
                ->getQuery();

            $paginator = $this->get('knp_paginator');
            $pagination = $paginator->paginate(
                $query,
                $request->query->get('page', 1)/*page number*/,
                100/*limit per page*/
            );
        }
    ......
    return $this->render('blog/show.html.twig', array('blogpost' => $this->blogPost,
                'latestblogs' => BlogController::getLatestBlogs($this),
                'tophotblogs' => BlogController::getTopHotBlogs($this),
                'is_original' => true,
                'lastblog' => $this->findLastBlog($blogId),
                'nextblog' => $this->findNextBlog($blogId),
                'pagination' => $pagination,
                'tags' => $this->getAllTagNames()
            ));
```

其中的**getAllTagNames()**方法如下：

```
private function getAllTagNames()
{
    $blogPostRepository = $this->getDoctrine()->getRepository('AppBundle:Tag');
    $tags = $blogPostRepository->findAll();
    return $tags;
}
```

再修改**app/Resources/views/blog/show.html.twig**，先把原来的中部文章标题、内容、评论布局原封不动提出来放到重新设计的一个四列网格中的第二列，然后第三列加入如下内容：

```
    <div class="col-sm-2 hidden-xs">

        <br/>

        <h4>
            {% for tag in blogpost.tags %}
                系列:{{ tag.name }}
            {% endfor %}
        </h4>

        <div style="overflow:hidden;text-overflow:ellipsis;white-space:nowrap;">
            {% for article in pagination %}
                {% if blogpost.id == article.id %}
                    <h6 style="color: red;">{{ article.simpleTitle }}</h6>
                {% else %}
                    <h6><a href="{{ path('blog_show', {'blogId':article.id}) }}"
                           onclick="_czc.push(['_trackEvent', '右侧链接', '点击', '{{ article.title }}']);"
                        >{{ article.simpleTitle }}</a></h6>
                {% endif %}
            {% endfor %}
        </div>

        <h4>
            全部系列
        </h4>
        {% for tag in tags %}
            <h6><a href="{{ path('blog_listbytag', {'tagname':tag.name}) }}"
                   onclick="_czc.push(['_trackEvent', '全部系列', '点击', '{{ tag.name }}']);"
                >{{ tag.name }}</a></h6>
        {% endfor %}
    </div>
```

讲解一下，根据当前文章所属的**tag**，找到这个**tag**下所有的链接并展示，同时再展示全部**tag**，这都是为了方便用户点击，同时你会发现网格的第三列声明了**hidden-xs**，也就是移动端不展示，因为移动端屏幕无法容纳两列内容，因此这部分隐藏。另外在右侧链接中我们也加入了点击事件，通过**cnzz**来统计事件触发情况

看下最终的效果：

数据统计

我们观察了一段时间的点击统计如下：

我们发现右侧链接的点击次数比页面底部的“最新文章”、“最热文章”点击次数多很多，这说明我们可以完善右侧来替代底部的链接，这是我们基于数据统计来做的页面布局优化的过程

教你成为全栈工程师(Full Stack Developer) 四十五-一文读懂hadoop、hbase、hive、spark分布式系统架构

发表于 2016-08-09 09:06

机器学习、数据挖掘等各种大数据处理都离不开各种开源分布式系统，hadoop用于分布式存储和map-reduce计算，spark用于分布式机器学习，hive是分布式数据库，hbase是分布式kv系统，看似互不相关的他们却都是基于相同的hdfs存储和yarn资源管理，本文通过全套部署方法来让大家深入系统内部以充分理解分布式系统架构和他们之间的关系

请尊重原创，转载请注明来源网站www.shareditor.com以及原始链接地址。本文较长，精华在最后

本文结构

首先，我们来分别部署一套hadoop、hbase、hive、spark，在讲解部署方法过程中会特殊说明一些重要配置，以及一些架构图以帮我们理解，目的是为后面讲解系统架构和关系打基础。

之后，我们会通过运行一些程序来分析一下这些系统的功能

最后，我们会总结这些系统之间的关系

分布式hadoop部署

首先，在http://hadoop.apache.org/releases.html找到最新稳定版tar包，我选择的是

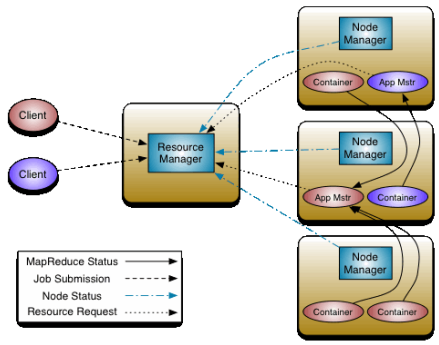
http://apache.fayea.com/hadoop/common/hadoop-2.7.2/hadoop-2.7.2.tar.gz

下载到/data/apache并解压

在真正部署之前，我们先了解一下hadoop的架构

hadoop分为几大部分：yarn负责资源和任务管理、hdfs负责分布式存储、map-reduce负责分布式计算

先来了解一下yarn的架构：



yarn的两个部分：资源管理、任务调度。

资源管理需要一个全局的ResourceManager(RM)和分布在每台机器上的NodeManager协同工作，RM负责资源的仲裁，NodeManager负责每个节点的资源监控、状态汇报和Container的管理

任务调度也需要ResourceManager负责任务的接受和调度，在任务调度中，在Container中启动的ApplicationMaster(AM)负责这个任务的管理，当任务需要资源时，会向RM申请，分配到的Container用来起任务，然后AM和这些Container做通信，AM和具体执行的任务都是在Container中执行的

yarn区别于第一代hadoop的部署(namenode、jobtracker、tasktracker)

然后再看一下hdfs的架构：hdfs部分由NameNode、SecondaryNameNode和DataNode组成。DataNode是真正的在每个存储节点上管理数据的模块，NameNode是对全局数据的名字信息做管理的模块，SecondaryNameNode是它的从节点，以防挂掉。

最后再说map-reduce：Map-reduce依赖于yarn和hdfs，另外还有一个JobHistoryServer用来看任务运行历史

hadoop虽然有多个模块分别部署，但是所需要的程序都在同一个tar包中，所以不同模块用到的配置文件都在一起，让我们来看几个最重要的配置文件：

各种默认配置：core-default.xml, hdfs-default.xml, yarn-default.xml, mapred-default.xml

各种web页面配置：core-site.xml, hdfs-site.xml, yarn-site.xml, mapred-site.xml

从这些配置文件也可以看出hadoop的几大部分是分开配置的。

除上面这些之外还有一些重要的配置：hadoop-env.sh、mapred-env.sh、yarn-env.sh，他们用来配置程序运行时的java虚拟机参数以及一些二进制、配置、日志等的目录配置

下面我们真正的来修改必须修改的配置文件。

修改etc/hadoop/core-site.xml，把配置改成：

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://127.0.0.1:8000</value>
    </property>
    <property>
        <name>io.file.buffer.size</name>
        <value>131072</value>
    </property>
</configuration>
```

这里面配置的是hdfs的文件系统地址：本机的9001端口

修改etc/hadoop/hdfs-site.xml，把配置改成：

```
<configuration>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/data/apache/dfs/name</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/data/apache/dfs/data</value>
    </property>
    <property>
        <name>dfs.datanode.fsdataset.volume.choosing.policy</name>
        <value>org.apache.hadoop.hdfs.server.datanode.fsdataset.AvailableSpaceVolumeChoosingPolicy</value>
    </property>
    <property>
        <name>dfs.namenode.http-address</name>
        <value>127.0.0.1:50070</value>
    </property>
    <property>
        <name>dfs.namenode.secondary.http-address</name>
        <value>127.0.0.1:8001</value>
    </property>
</configuration>
```

这里面配置的是hdfs文件存储在本地的哪里以及secondary namenode的地址

修改etc/hadoop/yarn-site.xml，把配置改成：

```
<configuration>
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>127.0.0.1</value>
    </property>
    <property>
        <name>yarn.resourcemanager.webapp.address</name>
        <value>127.0.0.1:8088</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
    </property>
    <property>
        <name>yarn.log-aggregation-enable</name>
        <value>true</value>
    </property>
    <property>
        <name>yarn.log-aggregation.retain-seconds</name>
        <value>864000</value>
    </property>
    <property>
        <name>yarn.log-aggregation.retain-check-interval-seconds</name>
        <value>86400</value>
    </property>
    <property>
        <name>yarn.nodemanager.remote-app-log-dir</name>
        <value>/YarnApp/Logs</value>
    </property>
    <property>
        <name>yarn.log.server.url</name>
        <value>http://127.0.0.1:19888/jobhistory/logs/</value>
    </property>
    <property>
        <name>yarn.nodemanager.local-dirs</name>
        <value>/data/apache/tmp/</value>
    </property>
    <property>
        <name>yarn.scheduler.maximum-allocation-mb</name>
        <value>5000</value>
    </property>
    <property>
        <name>yarn.scheduler.minimum-allocation-mb</name>
        <value>1024</value>
    </property>
    <property>
        <name>yarn.nodemanager.vmem-pmem-ratio</name>
        <value>4.1</value>
    </property>
    <property>
        <name>yarn.nodemanager.vmem-check-enabled</name>
        <value>false</value>
    </property>
</configuration>
```

这里面配置的是yarn的日志地址以及一些参数配置

通过cp etc/hadoop/mapred-site.xml.template etc/hadoop/mapred-site.xml创建etc/hadoop/mapred-site.xml，内容改为如下：

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
        <description>Execution framework set to Hadoop YARN.</description>
    </property>
    <property>
        <name>yarn.app.mapreduce.am.staging-dir</name>
        <value>/tmp/hadoop-yarn/staging</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.address</name>
        <value>127.0.0.1:10020</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.webapp.address</name>
        <value>127.0.0.1:19888</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.done-dir</name>
        <value>${yarn.app.mapreduce.am.staging-dir}/history/done</value>
    </property>

    <property>
        <name>mapreduce.jobhistory.intermediate-done-dir</name>
        <value>${yarn.app.mapreduce.am.staging-dir}/history/done_intermediate</value>
    </property>
    <property>
        <name>mapreduce.jobhistory.joblist.cache.size</name>
        <value>1000</value>
    </property>

    <property>
        <name>mapreduce.tasktracker.map.tasks.maximum</name>
        <value>8</value>
    </property>
    <property>
        <name>mapreduce.tasktracker.reduce.tasks.maximum</name>
        <value>8</value>
    </property>
    <property>
        <name>mapreduce.jobtracker.maxtasks.perjob</name>
        <value>5</value>
        <description>The maximum number of tasks for a single job.
            A value of -1 indicates that there is no maximum.
        </description>
    </property>
</configuration>
```

这里面配置的是mapred的任务历史相关配置

如果你的hadoop部署在多台机器，那么需要修改etc/hadoop/slaves，把其他slave机器ip加到里面，如果只部署在这一台，那么就留一个localhost即可

下面我们启动hadoop，启动之前我们配置好必要的环境变量：

```
export JAVA_HOME="你的java安装地址"
```

先启动hdfs，在此之前要格式化分布式文件系统，执行：

```
./bin/hdfs namenode -format myclustername
```

如果格式化正常可以看到/data/apache/dfs下生成了name目录

然后启动namenode，执行：

```
./sbin/hadoop-daemon.sh --script hdfs start namenode
```

如果正常启动，可以看到启动了相应的进程，并且logs目录下生成了相应的日志

然后启动datanode，执行：

```
./sbin/hadoop-daemon.sh --script hdfs start datanode
```

如果考虑启动secondary namenode，可以用同样的方法启动

下面我们启动yarn，先启动resourcemanager，执行：

```
./sbin/yarn-daemon.sh start resourcemanager
```

如果正常启动，可以看到启动了相应的进程，并且logs目录下生成了相应的日志

然后启动nodemanager，执行：

```
./sbin/yarn-daemon.sh start nodemanager
```

如果正常启动，可以看到启动了相应的进程，并且logs目录下生成了相应的日志

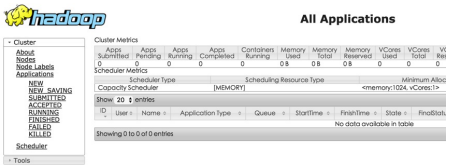
然后启动MapReduce JobHistory Server，执行：

```
./sbin/mr-jobhistory-daemon.sh start historyserver
```

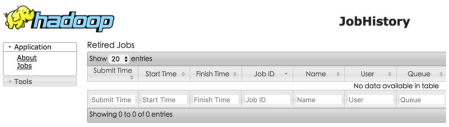
如果正常启动，可以看到启动了相应的进程，并且logs目录下生成了相应的日志

下面我们看下web界面

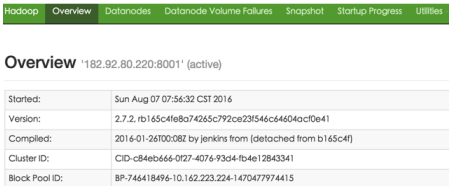
打开http://127.0.0.1:8088/cluster看下yarn管理的集群资源情况(因为在yarn-site.xml中我们配置了yarn.resourcemanager.webapp.address是127.0.0.1:8088)



打开http://127.0.0.1:19888/jobhistory看下map-reduce任务的执行历史情况(因为在mapred-site.xml中我们配置了mapreduce.jobhistory.webapp.address是127.0.0.1:19888)



打开http://127.0.0.1:50070/dfshealth.html看下namenode的存储系统情况(因为在hdfs-site.xml中我们配置了dfs.namenode.http-address是127.0.0.1:50070)



到此为止我们对hadoop的部署完成。下面试验一下hadoop的功能

先验证一下hdfs分布式文件系统，执行以下命令看是否有输出：

```
[root@MYAY hadoop]# ./bin/hadoop fs -mkdir /input
[root@MYAY hadoop]# cat data
1
2
3
4
[root@MYAY hadoop]# ./bin/hadoop fs -put input /input
[root@MYAY hadoop]# ./bin/hadoop fs -ls /input
Found 1 items
-rw-r--r--   3 root supergroup          8 2016-08-07 15:04 /input/data
```

这时通过http://127.0.0.1:50070/dfshealth.html可以看到存储系统的一些变化

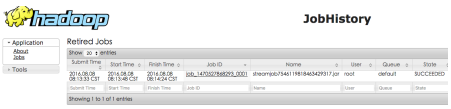
下面我们以input为输入启动一个mapreduce任务

```
[root@MYAY hadoop]# ./bin/hadoop jar ./share/hadoop/tools/lib/hadoop-streaming-2.7.2.jar -input /input -output /output -mapper cat -reducer wc
```

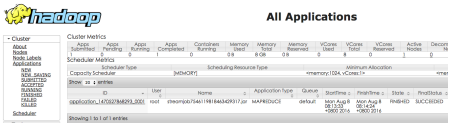
之后看是否产生了/output的输出：

```
[root@MYAY hadoop]# ./bin/hadoop fs -ls /output
Found 2 items
-rw-r--r--   3 root supergroup          0 2016-08-07 15:11 /output/_SUCCESS
-rw-r--r--   3 root supergroup         25 2016-08-07 15:11 /output/part-00000
[root@MYAY hadoop]# ./bin/hadoop fs -cat /output/part-00000
      4       4       12
```

这时通过http://127.0.0.1:19888/jobhistory可以看到mapreduce任务历史：



也可以通过http://127.0.0.1:8088/cluster看到任务历史



为什么两处都有历史呢？他们的区别是什么呢？

我们看到cluster显示的其实是每一个application的历史信息，他是yarn(ResourceManager)的管理页面，也就是不管是mapreduce还是其他类似mapreduce这样的任务，都会在这里显示，mapreduce任务的Application Type是MAPREDUCE，其他任务的类型就是其他了，但是jobhistory是专门显示mapreduce任务的

hbase的部署

首先从http://www.apache.org/dyn/closer.cgi/hbase/下载稳定版安装包，我下的是https://mirrors.tuna.tsinghua.edu.cn/apache/hbase/stable/hbase-1.2.2-bin.tar.gz

解压后修改conf/hbase-site.xml，改成：

```
<configuration>
    <property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
    </property>
    <property>
        <name>hbase.rootdir</name>
        <value>hdfs://127.0.0.1:8001/hbase</value>
    </property>
    <property>
        <name>hbase.zookeeper.quorum</name>
        <value>127.0.0.1</value>
    </property>
</configuration>
```

其中hbase.rootdir配置的是hdfs地址，ip:port要和hadoop/core-site.xml中的fs.defaultFS保持一致

其中hbase.zookeeper.quorum是zookeeper的地址，可以配多个，我们试验用就先配一个

启动hbase，执行：

```
./bin/start-hbase.sh
```

这时有可能会让你输入本地机器的密码

启动成功后可以看到几个进程起来，包括zookeeper的HQuorumPeer和hbase的HMaster、HRegionServer

下面我们试验一下hbase的使用，执行：

```
hbase(main):001:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 3.0000 average load
```

创建一张表

```
hbase(main):004:0> create 'table1','field1'
0 row(s) in 1.3430 seconds

=> Hbase::Table - table1
```

获取一张表

```
hbase(main):005:0> t1 = get_table('table1')
0 row(s) in 0.0010 seconds

=> Hbase::Table - table1
```

添加一行

```
hbase(main):008:0> t1.put 'row1', 'field1:qualifier1', 'value1'
0 row(s) in 0.4160 seconds
```

读取全部

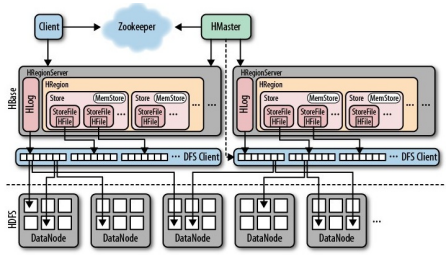
```
hbase(main):009:0> t1.scan
ROW                                                   COLUMN+CELL
 row1                                                 column=field1:qualifier1, timestamp=1470621285068, value=val
ue1
1 row(s) in 0.1000 seconds
```

我们同时也看到hdfs中多出了hbase存储的目录：

```
[root@MYAY hbase]# ./hadoop/bin/hadoop fs -ls /hbase
Found 7 items
drwxr-xr-x   - root supergroup          0 2016-08-08 09:05 /hbase/.tmp
drwxr-xr-x   - root supergroup          0 2016-08-08 09:58 /hbase/MasterProcWALs
drwxr-xr-x   - root supergroup          0 2016-08-08 09:05 /hbase/WALs
drwxr-xr-x   - root supergroup          0 2016-08-08 09:05 /hbase/data
-rw-r--r--   3 root supergroup         42 2016-08-08 09:05 /hbase/hbase.id
-rw-r--r--   3 root supergroup          7 2016-08-08 09:05 /hbase/hbase.version
drwxr-xr-x   - root supergroup          0 2016-08-08 09:24 /hbase/oldWALs
```

这说明hbase是以hdfs为存储介质的，因此它具有分布式存储拥有的所有优点

hbase的架构如下：



其中HMaster负责管理HRegionServer以实现负载均衡，负责管理和分配HRegion(数据分片)，还负责管理命名空间和table元数据，以及权限控制

HRegionServer负责管理本地的HRegion、管理数据以及和hdfs交互。

Zookeeper负责集群的协调(如HMaster主从的failover)以及集群状态信息的存储

客户端传输数据直接和HRegionServer通信

hive的部署

从http://mirrors.hust.edu.cn/apache/hive下载安装包，我下的是
http://mirrors.hust.edu.cn/apache/hive/stable-2/apache-hive-2.1.0-bin.tar.gz

解压后，我们先准备hdfs，执行：

```
[root@MYAY hadoop]# ./hadoop/bin/hadoop fs -mkdir /tmp
[root@MYAY hadoop]# ./hadoop/bin/hadoop fs -mkdir /user
[root@MYAY hadoop]# ./hadoop/bin/hadoop fs -mkdir /user/hive
[root@MYAY hadoop]# ./hadoop/bin/hadoop fs -mkdir /user/hive/warehourse
[root@MYAY hadoop]# ./hadoop/bin/hadoop fs -chmod g+w /tmp
[root@MYAY hadoop]# ./hadoop/bin/hadoop fs -chmod g+w /user/hive/warehourse
```

使用hive必须提前设置好HADOOP_HOME环境变量，这样它可以自动找到我们的hdfs作为存储，不妨我们把各种HOME和各种PATH都配置好，如：

```
HADOOP_HOME=/data/apache/hadoop
export HADOOP_HOME
HBASE_HOME=/data/apache/hbase
export HBASE_HOME
HIVE_HOME=/data/apache/hive
export HIVE_HOME
PATH=$PATH:$HOME/bin
PATH=$PATH:$HBASE_HOME/bin
PATH=$PATH:$HIVE_HOME/bin
PATH=$PATH:$HADOOP_HOME/bin
export PATH
```

拷贝创建hive-site.xml、hive-log4j2.properties、hive-exec-log4j2.properties，执行

```
[root@MYAY hive]# cp conf/hive-default.xml.template conf/hive-site.xml
[root@MYAY hive]# cp conf/hive-log4j2.properties.template conf/hive-log4j2.properties
[root@MYAY hive]# cp conf/hive-exec-log4j2.properties.template conf/hive-exec-log4j2.properties
```

修改hive-site.xml，把其中的${system:java.io.tmpdir}都修改成/data/apache/tmp，你也可以自己设置成自己的tmp目录，把${system:user.name}都换成用户名

```
:%s/${system:java.io.tmpdir}/\/data\/apache\/tmp/g
:%s/${system:user.name}/myself/g
```

初始化元数据数据库(默认保存在本地的derby数据库，也可以配置成mysql)，注意，不要先执行hive命令，否则这一步会出错，具体见http://stackoverflow.com/questions/35655306/hive-installation-issues-hive-metastore-database-is-not-initialized，下面执行：

```
[root@MYAY hive]# schematool -dbType derby -initSchema
```

成功之后我们可以以客户端形式直接启动hive，如：

```
[root@MYAY hive]# hive
hive> show databases;
OK
default
Time taken: 1.886 seconds, Fetched: 1 row(s)
hive>
```

试着创建个数据库是否可以：

```
hive> create database mydatabase;
OK
Time taken: 0.721 seconds
hive> show databases;
OK
default
mydatabase
Time taken: 0.051 seconds, Fetched: 2 row(s)
hive>
```

这样我们还是单机的hive，不能在其他机器登陆，所以我们要以server形式启动：

```
nohup hiveserver2 &> hive.log &
```

默认会监听10000端口，这时可以通过jdbc客户端连接这个服务访问hive

hive的具体使用在这里不赘述

spark部署

首先在http://spark.apache.org/downloads.html下载指定hadoop版本的安装包，我下载的是http://d3kbcqa49mib13.cloudfront.net/spark-2.0.0-bin-hadoop2.7.tgz

spark有多种部署方式，首先支持单机直接跑，如执行样例程序：

```
./bin/spark-submit examples/src/main/python/pi.py 10
```

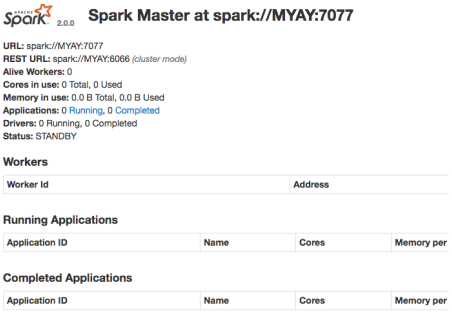
它可以直接运行得出结果

下面我们说下spark集群部署方法：

解压安装包后直接执行：

```
[root@MYAY spark-2.0.0-bin-hadoop2.7]# sbin/start-master.sh
```

这时可以打开http://127.0.0.1:8080/看到web界面如下：



根据上面的url：spark://MYAY:7077，我们再启动slave：

```
[root@MYAY spark-2.0.0-bin-hadoop2.7]# ./sbin/start-slave.sh spark://MYAY:7077
```

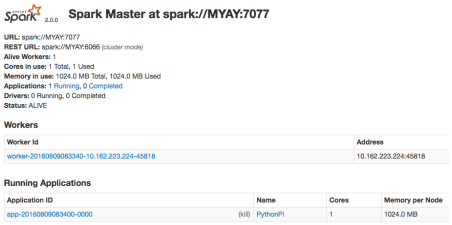
刷新web界面如下：

出现了一个worker，我们可以根据需要启动多个worker

下面我们把上面执行过的任务部署到spark集群上执行：

```
./bin/spark-submit --master spark://MYAY:7077 examples/src/main/python/pi.py 10
```

web界面如下：



spark程序也可以部署到yarn集群上执行，也就是我们部署hadoop时启动的yarn

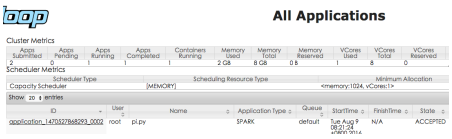
我们需要提前配置好HADOOP_CONF_DIR，如下：

```
HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop/
export HADOOP_CONF_DIR
```

下面我们把任务部署到yarn集群上去：

```
./bin/spark-submit --master yarn --deploy-mode cluster examples/src/main/python/pi.py 10
```

看http://127.0.0.1:8088/cluster效果如下：



总结一下

hdfs是所有hadoop生态的底层存储架构，它主要完成了分布式存储系统的逻辑，凡是需要存储的都基于其上构建

yarn是负责集群资源管理的部分，这个资源主要指计算资源，因此它支撑了各种计算模块

map-reduce组件主要完成了map-reduce任务的调度逻辑，它依赖于hdfs作为输入输出及中间过程的存储，因此在hdfs之上，它也依赖yarn为它分配资源，因此也在yarn之上

hbase基于hdfs存储，通过独立的服务管理起来，因此仅在hdfs之上

hive基于hdfs存储，通过独立的服务管理起来，因此仅在hdfs之上

spark基于hdfs存储，即可以依赖yarn做资源分配计算资源也可以通过独立的服务管理，因此在hdfs之上也在yarn之上，从结构上看它和mapreduce一层比较像

总之，每一个系统负责了自己擅长的一部分，同时相互依托，形成了整个hadoop生态。