爬虫分类

·通用爬虫一般关注三个主要的问题：下载、排序、索引。

·主题爬虫一般关注的是：下载、内容抽取、灵活的业务逻辑处理。

### GeccoEngine

GeccoEngine是爬虫引擎，每个爬虫引擎最好是一个独立进程，在分布式爬虫场景下，建议每台爬虫服务器（物理机或者虚机）运行一个GeccoEngine。爬虫引擎包括主要Scheduler、Downloader、Spider、SpiderBeanFactory、PipelineFactory5个主要模块。

### Scheduler

Scheduler负责下载地址的管理。FIFO 广度优先

### Downloader

Downloader负责从Scheduler中获取需要下载的请求。httpclient4.x，实现Downloader接口自定义

### SpiderBeanFactory

Gecco将下载下来的内容渲染为SpiderBean，所有爬虫渲染的JavaBean都统一继承SpiderBean，SpiderBean又分为HtmlBean和JsonBean分别对应html页面的渲染和json数据的渲染。SpiderBeanFactroy会根据请求的url地址，匹配相应的SpiderBean，同时生成该SpiderBean的上下文SpiderBeanContext。上下文SpiderBeanContext会告知这个SpiderBean采用什么渲染器，采用那个下载器，渲染完成后采用哪些pipeline处理等相关上下文信息。

### PipelineFactory

pipeline是SpiderBean渲染完成的后续业务处理单元，PipelineFactory是pipeline的工厂类，负责pipeline实例化。**通过扩展PipelineFactory就可以实现和Spring等业务处理框架的整合。**

### Spider

Gecco框架最核心的类应该是Spider线程，一个爬虫引擎可以同时运行多个Spider线程。Spider描绘了这个框架运行的基本骨架，先从Scheduler获取请求，再通过SpiderBeanFactory匹配SpiderBeanClass，再通过SpiderBeanClass找到SpiderBean的上下文，下载网页并对SpiderBean做渲染，将渲染后的SpiderBean交个pipeline处理。

1. 从downloader 开始写

采用HttpClient

CloseableHttpClient

httpclient = HttpClients.createDefault();

this.httpClient = HttpClientBuilder.*create*()  
 .setDefaultRequestConfig(clientConfig)//分配默认RequestConfig实例，如果未在客户端执行上下文中显式设置，该实例将用于请求执行  
 .setConnectionManager(syncConnectionManager)//分配 HttpClientConnectionManager实例

setDefaultRequestConfig(RequestConfig config) 注意禁止重定向

setConnectionManager(HttpClientConnectionManager connManager)

HttpClientConnectionManager接口

两个实现类：

BasicHttpClientConnectionManager，**PoolingHttpClientConnectionManager**

**PoolingHttpClientConnectionManager提供多线程支持**

//提供多线程请求，先采用默认Registry *TODO*//socketFactoryRegistry = RegistryBuilder.create().register("http", PlainConnectionSocketFactory.getSocketFactory()).register("https", sslsf).build();  
PoolingHttpClientConnectionManager syncConnectionManager = new PoolingHttpClientConnectionManager();

private static Registry<ConnectionSocketFactory> getDefaultRegistry() {  
 return RegistryBuilder.create().register("http", PlainConnectionSocketFactory.getSocketFactory()).register("https", SSLConnectionSocketFactory.getSocketFactory()).build();  
}

Gecco中实现

try {  
 SSLContext sslContext = SSLContexts.custom().loadTrustMaterial((KeyStore)null, new TrustStrategy() {  
 public boolean isTrusted(X509Certificate[] chain, String authType) throws CertificateException {  
 return true;  
 }  
 }).build();  
 SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(sslContext);  
 socketFactoryRegistry = RegistryBuilder.create().register("http", PlainConnectionSocketFactory.getSocketFactory()).register("https", sslsf).build();  
} catch (Exception var4) {  
 socketFactoryRegistry = RegistryBuilder.create().register("http", PlainConnectionSocketFactory.getSocketFactory()).register("https", SSLConnectionSocketFactory.getSocketFactory()).build();  
}

添加日志配置commons-logging.properties

#添加这个的作用之后再说  
org.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog

HttpRequest HttpResponse

Gecco 实现

public class HttpResponse {  
 private ByteArrayInputStream raw;//字节数组输入流  
 private String content;//响应正文  
 private String contentType;  
 private String charset;//字符编码  
 private int status;//HTTP状态

//getter setter

}

要加"http","https"否则会抛出org.apache.http.client.ClientProtocolException

响应实体

response = this.httpClient.execute(request, this.cookieContext);  
//获取响应实体  
HttpEntity httpEntity = response.getEntity();  
*log*.info(httpEntity.getContent());  
if (httpEntity != null) {  
 *log*.info("长度：\t" + httpEntity.getContentLength());  
 System.*out*.println("内容：\t" + EntityUtils.*toString*(httpEntity,"UTF-8"));  
}