**第四次作业总结报告**

**Nodejs关于cheerio模块**

1.首先需要对html页面进行加载，可以通过以下三种方式：

①首选方式

var cheerio = require(‘cheerio’);

$=cheerio.load(‘<ul id=”fruits”>…</ul>’);

②通过将字符串作为内容或者作为根

$= require(‘cheerio’);

$ (‘ul’, ‘<ul id=”fruits”>…</ul>’); 或者

$ (‘li’,‘ul’, ‘<ul id=”fruits”>…</ul>’);

③如果你需要修改默认的解析选项，你也可以通过传入额外的对象到load中

$= cheerio.load(‘<ul id=”fruits”>…</ul>’,{

normalizeWhitespace:true,

xmlMode:true

});

2.关于选择器selector

$( selector, [context], [root] )

selector在context范围内进行搜索，而context在root范围内进行搜索,selector和context可以是字符串，DOM元素，DOM元素数组，或者cheerio对象。root是典型的DOM文档字符串。选择器方法是遍历和操作文档的起点。 像jQuery一样，它是在文档中选择元素的主要方法，但不同于jQuery，它建立在CSSSelect库之上，该库实现了大部分Sizzle选择器。？

例：

$(‘li .clearfix’)//返回值为li下所有包含clearfix类的元素

关于Attributes

.attr( name, value )

获得以及设置attributes的方法。仅获取匹配集中第一个元素的属性值，如果将属性值设为null，代表移除了这个属性值。

例：

$(‘li’).attr(‘class’,’first’) 代表获取class为first的li

3.关于text()

.text( [textString] )

返回每一个匹配集中元素对应的文本内容，包括他们的子元素文本内容，如果指定了textString，则用其替换对应文本内容。

例：

$(‘li’).attr(‘class’,’clearfix’).text()将类为clearfix的li下的所有文字内容全部显示

**Nodejs中关于crypto模块**

该模块为加密模块，它提供加密功能，并实现了对OpenSSL的哈希，HMAC，密码，解密，签名和验证功能等的封装

1.关于createHash函数

函数原型：crypto.createHash(algorithm)

创建并且返回一个Hash对象，这个对象可以根据传入的参数algorithm生成哈希摘要（Hash digest）？这个参数依赖于平台上OpenSSL的版本可以支持的算法，例如’sha256’,’sha512’‘sha1’‘md5’等。OpenSSL是一个安全套接字层密码库

SHA1的全称是Secure Hash Algorithm(安全哈希算法)。加密哈希函数将任意长度的二进制字符串映射为固定长度的小型二进制字符串。加密哈希函数有这样一个属性：在计算上不大可能找到散列为相同的值的两个不同的输入；也就是说，两组数据的哈希值仅在对应的数据也匹配时才会匹配。数据的少量更改会在哈希值中产生不可预知的大量更改。所以你很难从加密后的文字中找到蛛丝马迹。

2.关于Hash类

Hash类的效用在于创建数据的哈希描述，它可以通过两种方式实现：

作为一个可读写的流，数据在可读侧被写入用于计算相应的哈希描述，

使用hash.update()以及hash.digest()方法产生计算过的哈希值

对于hash.update()方法是有记忆功能的

**hash.digest([encoding])**

计算所有要经过哈希处理的数据的描述值，encoding参数可以是’hex’,’latin1’或者’base64’。如果传入encoding则返回一个string类型数据，否则返回buffer。在使用过该函数后，hash对象不能再被使用。

hash.update()方法就是将字符串相加，然后在hash.digest()将字符串加密返回

**关于URL模块**

Nodejs提供了url模块，该模块提供把一个url字符串转换成一个url对象的功能，

url.parse(urlStr,[parseQueryString],[slashesDenoteHost])

第一个参数为url字符串，第二个参数parseQueryString是一个bool类型的参数，如果为真，把url的查询字符串部分解析为对象字面量，默认为假，第三个参数slashesDenoteHost也是bool类型的参数，若为真，把格式为//host/path的url解析为{host:’host’,pathname:’/path’}而不是{pathname:’//host/path’}。它的默认值为假。

**关于querystring模块**

querystring模块提供了用于解析和格式化url查询字符串的功能，

querystring.parse(str,[sep],[ eq],[options])

str参数是查询或参数字符串，sep允许指定使用的分隔符，默认为&，eq参数允许指定分析时使用的赋值运算符，默认为=。Options指定要解析的最大密钥数，默认为1000，若指定为0，则表示没有任何限制。decodeURIComponent <Function>解码查询字符串中百分号编码字符时使用的功能。 默认为querystring.unescape()。

**关于Later**

1.创建Schedule

使用Later模块首先需要定义一个新的时间表schedule，创建schedule的方式有很多种，作业过程中通过parse方式实现。

later.parse.recur()

该方法提供了创建schedule的简单的可链接的API。

on(vals)

指定一个时间段内的一个或多个值，当使用特殊时间时，可以使用表明24小时时间的字符串。

例：

later.parse.recur().on(2).minute();在每个小时的第二分钟启动。

later.parse.recur().on(8,20).hour();在每天的早8点以及晚上8点启动。

first()

on(min)的缩写，指定一个时间段内的最小值。

例：

later.parse.recur().first().minute(); 在每个小时的第0分钟启动。

every(val)

指定时间间隔值，默认情况下，间隔开始于时间段中的最小值，并且到时间段中的最大值。

例：

later.parse.recur().every(10).minute();//启动于每一个小时的0分钟，10分钟，20分钟…50分钟。

2.Occurrences

一旦定义了时间表，它可以用于计算该日程表的未来和过去的事件。 一个事件是满足日程表施加的所有限制的日期。

计算实例

实例是满足schedule的所有条件的个别日期。 实例可以按照任何数量，在可选范围内向前和向后计算。 在计算多个实例时，实例之间的最小时间基于范围时间段。

later.schedule(schedule).next(count, start, end)

计算接下来的时间中count个在start以及end之间满足schedule条件的日期。

later.schedule(schedule).prev(count, start, end)

计算之前的时间中count个在start以及end之间满足schedule条件的日期。

3.setInterval函数

该函数需要两个参数分别是回调函数名称以及延时，功能是每经历一个延时长度的时间段就调用一次回调函数。

later.setInterval(test,sched)

代表没次按照sched启动时就调用一次test函数。

4.setTimeout函数

该函数需要两个参数分别是回调函数名称以及延时，功能是经历一个延时长度的时间段只调用一次回调函数。

setTimeout(test,2000);2s时调用一次test函数