工作：暂停和恢复抓取

有时，对于大型网站，最好暂停抓取并稍后恢复。

Scrapy通过提供以下设施支持此功能：

- 一个调度程序，用于在磁盘上保留计划的请求
- 重复过滤器，用于保留磁盘上的访问请求
- 一个扩展，使一些蜘蛛状态（键/值对）在批次之间保持不变

工作目录

要启用持久性支持，您只需通过该设置定义*作业目录* `JOBDIR` 。该目录将用于存储所有必需的数据以保持单个作业的状态（即蜘蛛运行）。重要的是要注意，这个目录不能由不同的蜘蛛共享，甚至不能由同一个蜘蛛的不同作业/运行共享，因为它意味着用于存储*单个*作业的状态。

如何使用

要启用支持持久性支持的蜘蛛，请按以下方式运行：

```
scrapy crawl somespider -s JOBDIR=crawls/somespider-1
```

然后，您可以随时安全地停止蜘蛛（通过按Ctrl-C或发送信号），然后通过发出相同的命令恢复它：

```
scrapy crawl somespider -s JOBDIR=crawls/somespider-1
```

在批次之间保持持久状态

有时你会想要在暂停/恢复批次之间保留一些持久的蜘蛛状态。您可以使用该 `spider.state` 属性，该属性应该是一个字典。有一个内置的扩展，负责在蜘蛛启动和停止时从作业目录中序列化，存储和加载该属性。

这是一个使用蜘蛛状态的回调示例（为简洁起见省略了其他代码）：

```
def parse_item(self, response):
    # parse item here
    self.state['items_count'] = self.state.get('items_count', 0) + 1
```

持久性问题

如果您希望能够使用Scrapy持久性支持，请记住以下几点：

Cookie到期

Cookie可能会过期。因此，如果您没有快速恢复蜘蛛，则预定的请求可能不再有效。如果您的蜘蛛不依赖cookie，这将不会成为问题。

请求序列化

请求必须由*pickle*模块序列化，以便持久性工作，因此您应确保您的请求是可序列化的。

这里最常见的问题是使用`lambda`无法持久化的请求回调函数。

因此，例如，这不起作用：

```
def some_callback(self, response):
    somearg = 'test'
    return scrapy.Request('http://www.example.com', callback=lambda r: self.other_callback(r,
somearg))

def other_callback(self, response, somearg):
    print "the argument passed is:", somearg
```

但这会：

```
def some_callback(self, response):
    somearg = 'test'
    return scrapy.Request('http://www.example.com', callback=self.other_callback, meta=
{'somearg': somearg})

def other_callback(self, response):
    somearg = response.meta['somearg']
    print "the argument passed is:", somearg
```

如果要记录无法序列化的请求，可以在项目的设置页面中将`SCHEDULER_DEBUG`设置设置为`True`。这是`False`默认的。