[**nodejs＋express的session使用**](http://spartan1.iteye.com/blog/1729148)

**博客分类：**

* [javascript](http://spartan1.iteye.com/category/255588)
* [node.js](http://spartan1.iteye.com/category/210696)

今天终于使用nodejs＋redis把网页版即时聊天框架zenkim搭了起来，非常简陋，不过登录／用户管理／实时消息推送／消息缓存／用户单实例登录控制deng基本能力都有，待完成的功能包括：消息持久化及活动用户队列周期扫描／加强事务能力／消息排序／界面美化／移动版本支持。再优化一下后放到github上。

今天花了一下午时间，都用在了处理session上。nodejs本身不管session，因为用了express框架，express基于connect，connect中有session管理的能力。connect是插件式架构，它的插件称之为“中间件”，其中有个中间件就是叫作session。

使用express命令搭建应用框架时，加上－s选项，就可以添加对session的支持，之后在url映射函数中，直接使用req.session就可以访问和添加session信息。

问题是这样的：为了浏览器兼容，zenkim使用long polling方式实现实时消息推送。这就要求除了正常的浏览器请求链接外，还有一个后台链接用作推送消息，该链接的response对象能够长期（几秒到几分钟）保存在后台。开始时将response对象保存在了session中（当然，后来发现这么做不合适，并且有错），但发现后台链接session中的对象，在正常浏览器请求的session中看不到。不是同一个浏览器看到的session数据应该是同一份吗？为什么出现这种情况？

后来还是看了session这个connect中间件的代码才了解原委。session保存在称为sessionStore的数据仓库中。默认使用MemoryStore，就是所有session信息都保存在内存中。每来一个请求后，在路由分发前，首先使用cookieParser中间件将cookie中的sessionID解析出来，然后根据sessionID去sessionStore中进行查找，如果找到一份session后，就使用sessionStore中的数据构建一个新的session对象，把这个session对象放到req.session中，这就是session的由来。

另外，session中间件还改写了res.end方法，在将应答发送回浏览器之前，先将session数据写回sessionStore。

从实现过程可以看出：

1. 每个session对象都是针对某一个http请求的，每个请求的session对象相互独立，互不影响。

2. 在没有调用res.end之前，session对象不会回写到sessionStore中，除非主动调用了session.save方法。

3. session的解析依赖cookieparser，因此cookieparser中间件应该放到session之前。路由处理函数依赖session数据，因此app.router中间件应该放到session之后

4. 因每个请求有独立的session对象，因此对于同一个客户端有多个链接的情况，需要考虑session回写的并发问题。

所以在将res对象放到后台请求的session中后，同一时刻前台请求的session中是看不到这个res对象的。可以在放置了res之后，立刻调用session.save方法将信息保存回sessionStore，此时前台请求才可能看到。

不过这里要是调用session.save的话会报错，说session数据有循环依赖，无法保存回sessionStore。实际上是session数据如果有循环依赖的话，无法使用JSON.stringify转化为字符串，而memorystore使用字符串保存session数据，所以会出错。

解决办法是将res保存到内存中的一个临时数据结构中。这个res中有大量链接／socket／应用／请求信息，的确不适合保存到session中，另外res只是一个临时的链接数据，缓存在内存即可，也的确不应该放到session中。而将res放到内存中，所有请求都可以看到，这时会有并发问题。谢天谢地，node.js是单线程的，虽然一堆回调让代码不知什么时间会运行，但至少同一个函数中代码的前后顺序还是可以保证的，中间不会插入执行其他代码，函数内部保证数据访问的一致性就好了。

2015-11-7 update:

刚才搜索一个nodejs session管理的问题，没想到搜到自己三年前写的这篇文章。其中的很多内容都过时了，比如express现在和connect合在一起了，session管理也不依赖cookieParser中间件了，不过其中有些关于express和nodejs处理的内容还是有部分价值的。

另外zenkim项目后来改为[webim](https://github.com/zenkj/webim)，已经停止开发了。当时对nodejs的callback hell深恶痛绝，最后切换到使用erlang实现了相同的IM功能，web部分使用mochiweb，消息队列部分使用rabbitmq直接管理消息队列，整个IM功能做成了rabbitmq的一个插件，改名为rabbitmq-webim，已经在线运行两年了，基本没什么问题。源码在github上（https://github.com/zenkj/rabbitmq-webim）。

最近nodejs越来越红火，拿过来重新写了几个应用，发现使用async.js时callback hell也没什么不可忍受的，并且与erlang的变量不可变一样，可以逼着你写很多小函数，实际上能让代码更好维护，所以nodejs还是不错的。