一．Memcache

a) memcache 应用说明

i. 内存缓存，服务器内存相对较大

ii. 内存是用来存储数据的设备，它的存储数据介于寄存器和硬件之间。

iii. 内存条是 CPU 唯一可以访问的大容量的存储设备！所有硬盘中的程序和数据必须调入内存之后方可被 CPU 执行！切记：CPU 不能直处理硬盘中的数据。

iv. Memcache 就是一个软件，可以管理内存的软件（内存是存储临时数据）

b) memcache 工作原理

i. 服务器软件，启动才能使用，遵守一定的 HASH 算法

ii. 发送指定的命令对应的操作，只要数据就能存储

iii. 消耗 CPU，Apache 消耗进程，MySQL 消耗硬盘

iv. memcache 起到中间层作用

v. 分布式应用

vi. 应用于会话控制机制

c) memcache 的安装

i. wget http://memcached.org/files/memcached-1.4.25.tar.gz

ii. 首先安装依赖包 libevent　　yum -y install libevent *

iii. 主包已经安装，别忘记安装 libevent-devel*，不然./configure 过不去

1. tar -xzf memcached-1.4.25.tar.gz
2. cd memcached-1.4.25
3. ./configure --prefix=/Data/apps/memcache
4. make && make install

5. useradd memcache
6. 因为系统不能用 root 运行 memcache 软件
    a) /Data/apps/memcache/bin/memcached -umemcache &
    b) netstat -tunpl | grep 11211 //查看端口
    c) Telnet 192.168.1.137 11211 //连接
    d) 进入连接 stats //查看当前状态
7. 写入自动
    a) vim /etc/rc.local
    b) /Data/apps/memcache/bin/memcached -umemcache &
8. 杀死后台进程 pkill memcached
9. 连接 memcached 服务器
    a) 连接命令 telnet 127.0.0.1 11211
    b) 使用 quit 退出
10. 基本的 memecached 客户端命令
    a) 5 个常用的命令
        i. stats: 当前所有 memcached 服务器运行的状态信息
        ii. add: 添加一个数据到服务器
        iii. set: 替换已经存在的数据，如数据不存在，则和 add 命令相同。
        iv. get: 从服务器端提取指定的数据。
        v. delete: 删除指定的单个数据，如果要清除所有数据，可以使用 flush_all 指令。
    b) Memcache 的协议的错误部分主要是三个错误提示之提示指令：

i. ERROR -- 普通错误信息，比如指令错误之类的

ii. CLIENT_ERROR <错误信息> -- 客户端错误

iii. SERVER_ERROR <错误信息> --服务器端错误

c) 查看当前 memcached 服务器的运行状态信息

i. STAT pid 1552 服务进程的进程 ID

ii. STAT uptime 379 服务从启动到当前所经过的时间，单位是秒。

iii. STAT time 1262517674 服务器当前系统的时间，单位是秒。

iv. STAT version 1.2.6 组件的版本。这里是我当前使用的 1.2.6。

v. STAT pointer_size 32 服务器系统的指针大小一般为 32 或 64.

vi. STAT curr_items 1 表示存放当前的所有缓存对象的数量。不包括已经从缓存中删除的对象。

vii. STAT total_items 2 表示从启动到当前，系统存储过的所有对象数量，包括已经删除的对象。

viii. STAT bytes 593 表示系统存储缓存对象所使用的存储空间，单位为字节。

ix. STAT curr_connections 2 表示当前系统打开的连接数。

x. STAT total_connections 28 表示从 memcached 服务启动到当前时间，系统打开过的连接的总数。

xi. STAT connection_structures 9表示从 memcached 服务启动到当前时间，被服务器分配的连接结构的数量，这个解释是协议文档给的，具体什么意思，我目前还没搞明白。

xii. STAT cmd_get 3 累积获取数据的数量，这里是 3，因为我测试过 3 次，第一次因为没有序列化对象，所以获取数据失败，是 null，后边有 2 次是

我用不同对象测试了 2 次。

xiii. STAT cmd_set 2 累积保存数据的树立数量 ,这里是 2.虽然我存储了 3 次，但是第一次因为没有序列化，所以没有保存到缓存，也就没有记录。

xiv. STAT get_hits 2 表示获取数据成功的次数。

xv. STAT get_misses 1 表示获取数据失败的次数。

xvi. STAT evictions 0 为了给新的数据项目释放空间，从缓存移除的缓存对象的数目。比如超过缓存大小时根据 LRU 算法移除的对象 ,以及过期的对象。

xvii. STAT bytes_read 1284 memcached 服务器从网络读取的总的字节数。

xviii. STAT bytes_written 5362 memcached 服务器发送到网络的总的字节数。

xix. STAT limit_maxbytes 67108864 memcached 服务缓存允许使用的最大字节数。这里为 67108864 字节，也就是是 64M.与我们启动 memcached 服务设置的大小一致。

xx. STAT threads 1 被请求的工作线程的总数量。

d) 数据管理命令

格式：<命令> <键> <标记> <有效期> <数据长度>

其中：命令：add(添加)、set(修改)、delete(删除)、get(获取)

i. <键> -key，是发送过来指令的 key 内容

ii. <标记> - flags，是调用 set 指令保存数据时候的 flags 标记

iii. 有效期：是数据在服务器上的有效期限，如果是 0，则数据永远有效，单位是秒

iv. 数据的长度，block data 块数据的长度，一般在这个个长度结束以后下一行跟着 block data 数据内容，发送完数据以后，客户端一般等待服务器端的返回，服务器端的返回：STORED 数据保存成功，NOT_STORED 数据保存失败，是因为服务器端这个数据 key 已经存在。

v. stats items 看选项号

vi. stats cachedump 选项号 (0|n) 只能看到变量， 值使用 get 获取

d) PHP 的 memcached 管理接口

i. 安装 PHP 中的 MemCached 应用程序扩展接口

1. tar zxvf memcache-2.2.5.tgz 解压软件包
2. cd memcache-2.2.5 进入目录
3. /usr/local/php/bin/phpize 生成配置环境
4. ./configure --with-php-config=/Data/apps/php/bin/php-config
5. make && make install 编译和安装
6. 修改 php.ini
   a) extension_dir"Data/apps/php/lib/php/extensions/no-debug-non-zts-20060613/ “
   b) extension="memcache.so";

ii. MemCache 应用程序扩展接口

1. Memcache — Memcache 类
   a) Memcache::add — 增加一个条目到缓存服务器
   b) Memcache::addServer — 向连接池中添加 memcache 服务器
   c) Memcache::close — 关闭 memcache 连接

d) Memcache::connect — 打开一个 memcached 服务端连接

e) Memcache::decrement — 减小元素的值

f) Memcache::delete — 从服务端删除一个元素

g) Memcache::flush — 清洗（删除）已经存储的所有的元素

h) Memcache::get — 从服务端检回一个元素

i) Memcache::getExtendedStats — 缓存池中所有服务器统计信息

j) Memcache::getServerStatus — 获取服务器的在线/离线状态

k) Memcache::getStats — 获取服务器统计信息

l) Memcache::getVersion — 返回服务器版本信息

m) Memcache::increment — 增加一个元素的值

n) Memcache::pconnect — 打开一个到服务器的持久化连接

o) Memcache::replace — 替换已经存在的元素的值

p) Memcache::set — Store data at the server

q) Memcache::setCompressThreshold — 开启大值自动压缩

r) Memcache::setServerParams — 运行时修改服务器参数和状态

s) Memcache 函数 memcache_debug — 转换调试输出的开/关

2. MemCache 的实例应用

a) $mem = new Memcache(); //1. 创建对象

b) $mem->addServer("192.168.150.250",11211); //2. 添加服务

c) $mem->addServer("192.168.150.138",11211);

d) $mem->addServer("192.168.112.128",11211);

e) //3. 放置信息

f) $mem->add("mystr","hello memcache!",MEMCACHE_COMPRESSED,0);
//字符串

g) $mem->add("myarray",array(10,20,30,40),MEMCACHE_COMPRESSED,0)
; //数组

h) $mem->add("myob",new Stu(),MEMCACHE_COMPRESSED,0);

i) echo $mem->get("mystr"); //4. 获取信息

j) var_dump($mem->get('myarray'));

k) $mem->get("myob")->getinfo();

iii. memcached 服务器的安全防护

1. 内网访问

a) memcached -d -m 1024 -u root –l 192.168.0.10 –p 11211 –c 1024 start

2. 设置防火墙

a) iptables –F

b) iptables –P INPUT DROP

c) iptables –A INPUT –p tcp –s 192.168.0.10 –-dport 11211 –j ACCEPT

d) iptables –A INPUT –p udp –s 192.168.0.10 –-dport 11211 –j ACCEPT

e) 在 PHP 中设置分布式 memcache 存取

i. 分布式，根据内部 hash 算法，每台服务安装 memcache 并开启

ii. 使用 $mem->addServer("127.0.0.1", 11211);//添加到连接池中

iii. 怎么存储，怎么取，怎么根据服务器权重都是 memcache 内存实现

iv. 只要添加服务器到连接池即可，存和取都是内存实现

v. 实例：

```
<?php
$mm=new Memcache(); //创建对象
//添加 memcache 服务器（分布式，如何按照权重存取是由 memcache 处理）
$mm->addServer("192.168.150.250",11211);
$mm->addServer("192.168.150.138",11211);
$sql = "select * from stu"; //查询学生  建议使用 SQL 语句充当键
//先尝试从 memcahe 获取学生信息
$stulist = $mm->get(md5($sql)); //加密使用
if(empty($stulist)){
try{
$pdo = new PDO("mysql:host=localhost;dbname=lamp55","root","root");
}catch(PDOException $e){
echo $e->getMessage();
}
//执行查询
$stmt = $pdo->query($sql);
$stulist = $stmt->fetchAll(PDO::FETCH_ASSOC);
echo $sql."<br/>";
//缓存
$mm->set(md5($sql),$stulist,MEMCACHE_COMPRESSED,3600);
}
$mm->close();
```

```
//输出信息
foreach($stulist as $stu){
    echo $stu['name'].":".$stu['sex']."<br/>";
}
```

二．http 协议和 web 本质

a) 何为 HTTP

i. HTTP 全称是 HyperText Transfer Protocal，即：超文本传输协议，从 1990 年开始就在 WWW 上广泛应用，是现今在 WWW 上应用最多的协议， Http 是应用层协议，当你上网浏览网页的时候，浏览器和 Web 服务器之间就会通过 HTTP 在 Internet 上进行数据的发送和接收。Http 是一个基于请求/响应模式的、无状态的协议。即我们通常所说的 Request/Response。

ii. 通信协议，也就是通信时所遵守的规则，只有双方按照这个规则“说话”，对方才能理解或为之服务。

b) 既然 TCP/UDP 是广泛使用的网络通信协议，那为啥有多出个 http 协议来呢？

i. UDP 协议具有不可靠性和不安全性，显然这很难满足 web 应用的需要。而 TCP 协议是基于连接和三次握手的，虽然具有可靠性，但人具有一定的缺陷。但试想一下，普通的 C/S 架构软件，顶多上千个 Client 同时连接，而 B/S 架构的网站，十万人同时在线也是很平常的事儿，如果十万个客户端和服务器一直保持连续状态，那么服务器如何满足承载呢？

ii. 这就衍生出了 http 协议。基于 TCP 的可靠性连接。通俗点说，就是在请求之后，服务器端立即关闭连接，释放资源。这样既保证了资源可用，也吸引了 TCP 的可靠性的优点。

iii. 正因为这点，所以大家通常说协议是“无状态”的，也就是“服务器不知道你客户端干了啥”，其实很大程度上是基于性能考虑的。导致一个用户在同一个网站浏览不同的页面时无法识别是不是同一个用户，用户通过超链接请求服务器时，无法记录上次操作的状态。

iv. WEB 浏览器与 WEB 服务器之间的一问一答的交互过程必须遵守一定的规则，这个规则就是 HTTP 协议。

v. 它是 TCP/IP 协议的一个应用层协议，用于定义 WEB 浏览器与 WEB 服务器之间交换数据的过程及数据本身的格式。

vi. 详细学习地址：http://www.imooc.com/video/6712

c) TCP、UDP 和 HTTP 关系

i. TCP（Transmission Control Protocol，传输控制协议）是基于连接的协议，也就是说，在正式收发数据前，必须和对方建立可靠的连接。一个 TCP 连接必须要经过三次“对话”才能建立起来，这三次对话的简单过程：主机 A 向主机 B 发出连接请求数据包：“我想给你发数据，可以吗？”，这是第一次对话；主机 B 向主机 A 发送同意连接和要求同步（同步就是两台主机一个在发送，一个在接收，协调工作）的数据包：“可以，你什么时候发？”，这是第二次对话；主机 A 再发出一个数据包确认主机 B 的要求同步：“我现在就发，你接着吧！”，这是第三次对话。三次“对话”的目的是使数据包的发送和接收同步，经过三次“对话”之后，主机 A 才向主机 B 正式发送数据。对话过程中传送的包里不包含数据，三次对话完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP 连接一旦建立，在通信双方中的任何一方主动关闭连 接之前，TCP 连接都将被一直保持下去。断开连接时服务器和客户端均可以主动发起断开 TCP 连接的请求，断开过程需要经过“四次握手”。

ii. UDP（User Data Protocol，用户数据报协议）是与 TCP 相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去！ UDP 适用于一次只传送少量数据、对可靠性要求不高的应用环境。比如，我们经常使用“ping”命令来测试两台主机之间 TCP/IP 通信是否正常，其实“ping”命令的原理就是向对方主机发送 UDP 数据包，然后对方主机确认收到数据包，如果数据包是否到达的消息及时反馈回来，那么网络就是通的。例如，在默认状态下，一次“ping”操作发送 4 个数据包（如图 2 所示）。大家可以看到，发送的数据包数量是 4 包，收到的也是 4 包（因为对方主机收到后会发回一个确认收到的数据包）。这充分说明了 UDP 协议是面向非连接的协议，没有建立连接的过程。正因为 UDP 协议没有连接的过程，所以它的通信效果高；但也正因为如此，它的可靠性不如 TCP 协议高。QQ 就使用 UDP 发消息，因此有时会出现收不到消息的情况。

d) 请求与响应

i. 一个 http 请求代表客户端浏览器向服务器发送的数据，一个完整的 http 请求消息，包含一个请求行，若干个消息头（请求头），换行，实体内容。

ii. 请求行：描述客户端的请求方式，请求资源的名称，http 协议的版本号

iii. 请求头：包含客户端请求服务器主机名，客户端的环境信息

iv. 格式如下：Method Request-URI HTTP-Version CRLF

1. 其中 Method 表示请求方法；Request-URI 是一个统一资源标识符；HTTP-Version 表示请求的 HTTP 协议版本；CRLF 表示回车和换行（除了作为结尾的 CRLF 外，不允许出现单独的 CR 或 LF 字符）。

2. 请求方法（所有方法全为大写）有多种，各个方法的解释如下：

a) GET 请求获取 Request-URI 所标识的资源

b) POST 在 Request-URI 所标识的资源后附加新的数据

c) HEAD 请求获取由 Request-URI 所标识的资源的响应消息报头

d) PUT 请求服务器存储一个资源，并用 Request-URI 作为其标识

e) DELETE 请求服务器删除 Request-URI 所标识的资源

f) TRACE 请求服务器回送收到的请求信息，主要用于测试或诊断

g) CONNECT 保留将来使用

h) OPTIONS 请求查询服务器的性能，或查询与资源相关的选项和需求

3. 应用举例：

a) GET 方法：在浏览器的地址栏中输入网址的方式访问网页时，浏览器采用 GET 方法向服务器获取资源，eg:GET /form.html HTTP/1.1 (CRLF)

b) POST 方法要求被请求服务器接受附在请求后面的数据，常用于提交表单。

v. 请求头： 请求报头允许客户端向服务器端传递请求的附加信息以及客户端自身的信息。

1. 常用的请求报头

a) Accept：用于告诉服务器，客户机支持的数据类型。

b) Accept-Charset：用于告诉服务器，指定客户端接受的字符集。

c) Accept-Encoding：用于告诉服务器，用于指定可接受的内容编码。

d) Accept-Language：服务器假定客户端对各种语言都可以接受。

e) Authorization：用于证明客户端有权查看某个资源。

f) Host：（发送请求时，该报头域是必需的）Host 请求报头域主要用于指定被请求资源的 Internet 主机和端口号

g) User-Agent：请求报头域允许客户端将它的操作系统、浏览器和其它属性告诉服务器。

h) Cookie :客户机通过这个头，将 cookie 信息带给服务器

i) Connection：告诉服务器，请求完成后，是否保存连接

j) Date：告诉服务器，当前请求的时间

k) 请求报头举例：

i. GET /form.html HTTP/1.1 (CRLF)

ii. Accept:image/gif,image/x-xbitmap,image/jpeg,application/x-shockwave-flash,application/vnd.ms-excel,application/vnd.ms-powerpoint,application/msword,*/* (CRLF)

iii. Accept-Language:zh-cn (CRLF)

iv. Accept-Encoding:gzip,deflate (CRLF)

v. If-Modified-Since:Wed,05 Jan 2007 11:21:25 GMT (CRLF)

vi. If-None-Match:W/"80b1a4c018f3c41:8317" (CRLF)

vii. User-Agent:Mozilla/4.0(compatible;MSIE6.0;Windows N5.0) (CRLF)

viii. Host:www.guet.edu.cn (CRLF)

ix. Connection:Keep-Alive (CRLF)

2. 换行

3. 实体内容：就是浏览器端通过 http 协议发送给服务器的实体数据，例如？name=zhangsan&status=1（get 请求时，通过 URL 方式传给服务器的值，POST 请求方式，通过表单向服务器提交的数据）

vi. 响应：

1. 一个 http 响应代表服务器向客户端浏览器发送的数据，一个完整的 http 响应消息，包含一个响应行，若干个消息头（响应头），换行，实体内容。

2. HTTP 响应三个部分组成，分别是：状态行、消息报头、响应正文
    a) 状态行用于描述服务器对请求的处理结果
    b) 消息头用于描述服务器的基本信息，以及数据的描述，服务器哦通过这个数据的描述信息，可以通过客户端如何处理后的它回送的数据。
3. 状态行格式如下：
    a) HTTP-Version Status-Code Reason-Phrase CRLF
    b) 其中，HTTP-Version 表示服务器 HTTP 协议的版本；Status-Code 表示服务器发回的响应状态代码；Reason-Phrase 表示状态代码的文本描述。
    c) 状态代码有三位数字组成，且有五种可能取值：
        i. 1xx：指示信息--表示请求已接收，继续处理
        ii. 2xx：成功--表示请求已被成功接收、理解、接受
        iii. 3xx：重定向--要完成请求必须进行更进一步的操作
        iv. 4xx：客户端错误--请求有语法错误或请求无法实现
        v. 5xx：服务器端错误--服务器未能实现合法的请求
    d) 常见状态代码、状态描述、说明：
        i. 200 OK //客户端请求成功
        ii. 400 Bad Request //客户端请求有语法错误，不能被服务器所理解
        iii. 401 Unauthorized //请求未经授权，这个状态代码必须和 WWW-Authenticate 报头域一起使用
        iv. 403 Forbidden //服务器收到请求，但是拒绝提供服务
        v. 404 Not Found //请求资源不存在，eg：输入了错误的 URL
        vi. 500 Internal Server Error //服务器发生不可预期的错误

vii. 503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

4. 响应报头

a) 响应报头允许服务器传递不能放在状态行中的附加响应信息，以及关于服务器的信息和对 Request-URI 所标识的资源进行下一步访问的信息。

b) 常用的响应报头

i. Location：这个头配合 302 状态码，用于告诉客户端找谁

ii. Server :告诉浏览器服务器的类型

iii. Content-encoding：服务器的数据压缩格式

iv. Content-length：返回的数据长度

v. Content-type：返回数据类型

vi. Last-Modified：告诉浏览器当前资源缓存时间

vii. Refresh：告诉浏览器，隔多长时间刷新

viii. Content-disposition：告诉浏览器以下载的方式打开数据

ix. Transfer-encoding：告诉浏览器，传输数据的编码格式

x. Etag ： 缓存相关头（可以做到实时更新）

xi. Expries：告诉浏览器回送的资源缓存多长时间

xii. Cache-control 告诉所有的缓存机制是否可以缓存及哪种类型

xiii. Pragma： 控制浏览器不要缓存数据

5. 响应正文就是服务器返回的资源的内容

a) 理解：header("Content-type: text/html; charset=utf-8"); 这一句前不能向页面输出任何内容。

e) HTTP 1.0 的基本运行方式

i. 基于 HTTP 协议的客户/服务器模式的信息交换过程，包含四个过程：建立连接，发送请求信息，发送响应信息，关闭连接

ii. 浏览器与 WEB 服务器的连接过程是短暂的，每次连接只能处理一个请求和响应，对每个一个页面的访问，浏览器与 WEB 服务器都要建立一次单独的连接。

iii. 浏览器到 WEB 服务器之间的所有通讯都是完全独立分开的请求和响应

iv. 无状态（每次请求都是立即断开，释放资源）

v. 例如：浏览器访问多图网页

1. 在一个 HTML 页面中如果包含<img>标记的话，当浏览器解析到这些标记时，还会向服务器请求访问标记中指定的文件，即再次建立连接并发出 HTTP 请求。

2. 如果 HML 页面中有一个超级链接，当单击时也是向服务器发起一个请求。

三．会话控制

a) 会话控制概述

i. 理解：一个用户和网站服务器的交流

ii. HTTP 协议是无状态，每次的请求都是独立的

iii. 目的：让用户登录一次，请求不同的页面都能标识当前的用户

b) 涉及变量之间的传递

i. 当前页面的变量（page 级：只能在声明处以后使用，脚本执行完后释放）

ii. 两个页面之间传变量（get 方式：URL）

iii. 会话级别 （同一个用户，在一个网站上共享自己的变量）

iv. 全局（写入数据或文件中的值，任何人都可以访问，但是很难标识唯一性，假如一个用

户访问上千个网站，一个网站被上千万个人访问，这样太难保存信息量）

c) 为什么使用会话技术

i. HTTP 协议是无状态

1. HTTP 协议不能告诉我们多请求时不是来自同一个人

ii. 会话控制的思想就是允许服务器跟踪同一客户端连续做出的请求

d) 会话跟踪的方式

i. HTTP 是无状态的协议，所以不能维护两个事物间的状态

ii. 但是一个用户在请求一个页面以后再请求另一个页面时，需要让服务器知道这是一个用户。总共有 3 种数据传递方式。

1. 超链接或者 header（ ）函数等重定向方式

2. 使用 cookie 将用户的信息状态，存放在客户端的计算机文件中

3. 使用 session 将用户的信息状态，存放在服务器中

四．会话控制之 Cookie

a) Cookie 概述

i. 单用户登录访问网站时，服务器脚本会向浏览器设置 cookie，客户端会以文件的形式保存在指定的文件夹目录下，内容会记录你访问的相关信息，当再次访问时浏览器会读取文件的内容，携带这些信息（在请求头中会携带）给服务器中的不同页面（变量都是共享整个网站）

ii. 设置 cookie 是通过响应头，携带 cookie 是通过请求头（慢半拍）

b) Cookie 的操作

i. 使用 setcookie ，可以将一个或者多个变量存放在客户端的 cookie 中

ii. 所有客户端访问同时自动将所有这个网站的 cookie 都携带过来（全局数组）$_COOKIE

iii. Setcookie 类型

1. 字符串 setcookie(“username”,$username，time()+24*60*60)

2. 关联数组 setcookie(“lx[phone]”,$username，time()+24*60*60)

3. 索引数组 setcookie(“lx[0]”,$username，time()+24*60*60) 手动添加索引

iv. 慢半拍原理

1. <?Php

a) setcookie(“username”,$username，time()+24*60*60)

b) Var_dump($_COOKIE[‘username’]) //第一刷新是设置，第二才能获取

v. 删除 cookie (只要设置时间过期即可)

1. Setcookie(“username” , “” ，time()-1)；

c) 登录应用

五．会话控制之 session

a) Session 的工作机制

i. 用户访问网站时服务器设置 Session 时会生成对应 SessionID 并将此 ID 以 cookie 方式传给客户端保存，同时会在服务器端以 SessionID 名称的文本文件保存用户的信息，用户再次访问时只要携带 SessionID 即可查找出对应在服务器端保存信息的文件。

ii. 每个用户对应都有一个 Session 文件保存信息

iii. 关闭 cookie 可以通过 URL 保存 SessionID

b) session 的应用

i. session 跟踪用户变量

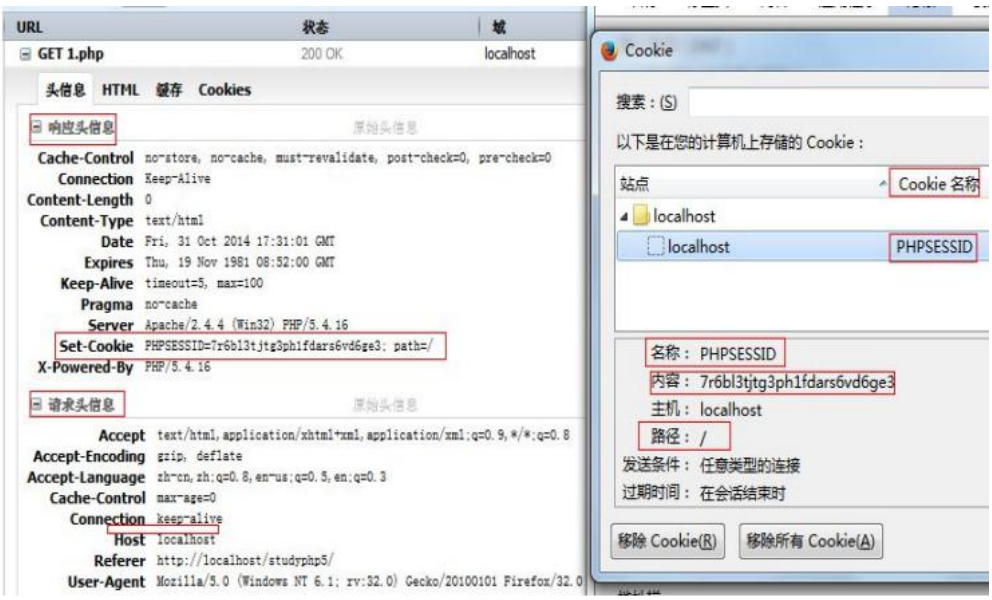
ii. Session start（） 作用

1. 只有开启 session start（） 后才能告诉 PHP 调用服务器的 session 机制

2. session start（ ）不能有任何输出，所以头函数中的一员。

3. 第一时间会向客户端发送 session id，并且会在服务器端创建 session id 同名文件

4. 以后使用时，根据 http 协议和 cookie 机制（请求头信息）将 session id 传过来，并找服务器端找到对应的文件信息。



iii. 存储，读取 session

1. $_SESSION[‘下标’] = 值　//设置值

2. Var_dump($_SESSION[‘下标’]); //$_SESSION 全局数组

3. 注意：

   a) 在任何使用 session 机制时一定要开启 session start

   b) 关闭浏览器时 session id 立即消失，但是 session 的信息继续会在服务器端保留，直到 session 自动回收机制清空。

   c) 判断客户端是否有 session id ，变量名为 PHPSESSID，直接使用这个 session 开启 session 不会新创建文件，直接找到这个同名 session id 的 session 文件。

4. 销毁 session

   a) 开启 session_start()

b) 清空 session 数组$_SESSION=array() 即清空对应 sessionid 文件中的内容

c) 使用 session_destroy（ ），来销毁 session 的文件

d) 销毁客户端的 session id setcookie (session_name( )，‘’ ,time()-1，‘/’)

c) session 的配置选项

i. 可以通过 phpinfo()查看 session 扩展配置信息

ii. session.auto_start = 0；是否自动启动 session，默认为 0，不需要修改，不建议使用

iii. Session.name =PHPSESSID： 是 SessionID 储存的变量名称，默认值"PHPSESSID"

iv. session.cookie_lifetime：SessionID 在客户端 Cookie 储存的时间，默认是 0，代表浏览器一关闭 SessionID 就作废，以秒为单位（应用在一周免登录）

v. session.gc_maxlifetime：Session 数据在服务器端储存的时间，如果超过这个时间，那么 Session 数据就自动删除！

vi. session.cookie_path = /；sessionid 的 cookie 路径，不需要修改

vii. session.save_path = /tmp；是 session 的保存路径，比如默认就是/tmp

viii. session.save_handler = files；这个是 session 的方式，默认的 files 就可以了，代表用文件储存，还有两种方式，user 和 memcache。user 方式指的是你自己（也就是用户）定义 session 的句柄，用于 session 的存取等，这个可以把 session 扩展存到数据库里，memcache 方式，需要你配置好 memcache,还要配置 session.save_path。

d) sessoion 自动垃圾回收机制

i. 原因：因为用户一般操作都喜欢关闭浏览器，当 session.cookie_lifetime 使用默认为 0 时，浏览器一关闭 SessionID 就作废，但是在服务器还是会保留对应的 session id 为名称的文本文件，当用户访问量过大时，也会对服务器性能的损耗。

ii. 如何定义垃圾回收机制

1. 设置 session.gc_maxlifetime，表示是 Session 数据在服务器端储存的时间，如果超过这个时间，那么 Session 数据就自动删除！ 默认是 1440 秒，24 分钟
2. session.gc_probability=1 ，session.gc_divisor =1000 ，session.gc_maxlifetime =1440//过期时间默认 24 //概率是 session.gc_probability/session.gc_divisor 结果 1/1000, //不建议设置过小，因为 session 的垃圾回收，是需要检查每个文件是否过期的。

e) 基于 url 传递 session 的 id

i. 为啥要使用 URL 传递 session id

1. 采用 session 会话机制跟踪用户是最优选择，但是 session id 还是基于 cookie
2. 但有些用户会在浏览器中禁用 cookie，导致 session 机制无法跟踪用户（每次刷新请求都会产生一个 session id，无法保证 session id 相同）.
3. 通过 URL 参数

ii. 采取方式（场景一：关闭 cookie）

1. One.php

```
<?php
$sid = !empty($_GET[session_name()])?$_GET[session_name()]:false;
if($sid){
    session_id($sid);//关闭 cookie，每次开启都会生成 session_id()
}
session_start(); //以设置 session_id 开启会话机制

if(true){
    $username = 'yuncopy';
    $_SESSION['username'] = $username;
}
echo session_name().'='.session_id();
?>
<br/><a href="two.php<?php echo '?'.session_name().'='.session_id()?>"> two.php</a>
<br/><a href="three.php<?php echo '?'.session_name().'='.session_id()?>"> three.php</a>
<br/><a href="exit.php<?php echo '?'.session_name().'='.session_id()?>"> exit.php</a>
```

2. Two.php

```
<?php
$sid = !empty($_GET[session_name()])?$_GET[session_name()]:false;
if($sid){
    session_id($sid);//关闭 cookie，每次开启都会生成 session_id()
}
session_start(); //以设置 session_id 开启会话机制

if(true){
    echo !empty($_SESSION['username'])?$_SESSION['username']:''.'<br>';
}
echo session_name().'='.session_id();
?>
<br/><a href="one.php<?php echo '?'.session_name().'='.session_id()?>"> one.php</a>
<br/><a href="exit.php<?php echo '?'.session_name().'='.session_id()?>"> exit.php</a>
```

总结：以这种方式页面中的所有 URL 都要填写，是否 cookie 开启，比较麻烦，不理想。

f) 参数规则

i. Session_name() 和 session_id() 应用

ii. 常量 SID （开启 cookie 时为空，关闭时等效于 session_name().'='.session_id();）

1. One.php

```
<?php
$sid = !empty($_GET[session_name()])?$_GET[session_name()]:false;
if($sid){
    session_id($sid);//关闭 cookie，每次开启都会生成 session_id()
}
session_start(); //以设置 session_id 开启会话机制

if(true){
    $username = 'yuncopy';
    $_SESSION['username'] = $username;
}
echo session_name().'='.session_id();
?>
<br/><a href="one.php<?php echo '?'.SID ?>"> one.php</a>
<br/><a href="exit.php<?php echo '?'.SID ?>"> exit.php</a>
```

2. Two.php

```
<?php
```

```
$sid = !empty($_GET[session_name()])?$_GET[session_name()]:false;
if($sid){
    session_id($sid);//关闭 cookie，每次开启都会生成 session_id()
}
session_start(); //以设置 session_id 开启会话机制

if(true){
    echo !empty($_SESSION['username'])?$_SESSION['username']:''.'<br>';
}
echo session_name().'='.session_id();
?>
<br/><a href="one.php<?php echo '?'.SID ?>"> one.php</a>
<br/><a href="exit.php<?php echo '?'.SID ?>"> exit.php</a>
```

总结：以这种方式页面中的所有 URL 还是得填写，比较麻烦，还是不理想。

g) 最优配置 session id

i. 在使用 linux 系统做服务器时 ,在编辑 PHP 时 ,如果使用了--enable-trans-sid 配置项。和运行时选项 session.use_trans_sid 都被激活，在客户端禁用 cookie 时，相对于 URL 将被自动修改为包含 sessionid ，如果没有配置，或使用 windows 系统作为服务器时，可以使用 SID 常量。

h) 我们来抛开 cookie 使用 session，主要途径有三条：

i. 设置 php.ini 中 session.use_trans_sid = 1 或者编译时打开了--enable-trans-sid 选项，

ii. 让 PHP 自动跨页传递 session id。

iii. 手动通过 URL 传值、隐藏表单传递 session id。

iv. 用文件、数据库等形式保存 session_id,在跨页过程中手动调用。

i) 删除 session

下面是 PHP 官方关于删除 session 的案例：

```
<?php
// 初始化 session.
session_start();
/*** 删除所有的 session 变量..也可用 unset($_SESSION[xxx])逐个删除。 ****/
```

```
$_SESSION = array();
/***删除 sessin id.由于 session 默认是基于 cookie 的，所以使用 setcookie 删除包含 session id
的 cookie.***/
if (isset($_COOKIE[session_name()])) {
    setcookie(session_name(), '', time()-42000, '/');
}
// 最后彻底销毁 session.
session_destroy();
?>
```

六．自定义 session 的存储机制

a) 瓶颈：

i. Session 默认的存储方式文件的形式（session.save_handler = files）默认路径 /tmp

ii. 同时登录用户上千万，生成的文件很多，查找文件效率很低，垃圾回收机制

iii. 在大网站实现负载均衡时，使用文件存储方式无法共享用户会话信息（网站代码在每台机器上同步，但是 session 会话保存文件还是单独存在各自的服务器硬盘中，无法跟踪用户）

b) 解决

i. 使用一个单独的服务器存储 session 信息，共享到每台负载均衡的服务器

1. 采用文件共享方式（NFS，Samba）,还是找文件，效率很是很低

2. 数据库存储

ii. 使用各个负载服务器内存，使用分布式 NoSQL 数据库，存储内存

c) 如何自定义 session 的处理方式

1. 保存 session 方式

a) Registered save handlers files user memcache

2. 修改 memcache 保存配置

   a) session.save_handler = memcache

   b) session.save_path = "tcp://host:11211;tcp://host1:11211;"

3. 自定义 session

   a) session.save_handler = user

   b) Session 生命周期：开启，关闭，读取，写入，销毁，回收

d) 自定义 session 以文件形式保存 （filesession.php 函数形式）

```
<?php
session_module_name("user"); //指定 session 的处理模式
//指定 session 的处理方式
//临时设置 session 的记录方式（采用用户自定义）
session_set_save_handler("open","close","read","write","destroy","gc");
//注意回调函数位置顺序（查阅手册）
//告诉 PHP 系统，SESSION 使用自定义形式，将对应的函数注册进入，不使用系统默认的了
$path=null;
function open($save_path){ //记录保存路径
    global $path; //每个操作都需要相同位置
    $path=$save_path;
    //echo "open...<br/>";
}

function close(){
    return true;
    //echo "close...<br/>";
}
// 当使用 echo $_SESSION['username'] 触发执行此函数
function read($sid){
    global $path;
    return file_get_contents($path."/my_".$sid.".txt");
```

```
}
// 当使用 $_SESSION['username'] = 'yuncopy'; 触发执行此函数
function write($sid,$data){

    global $path;

    file_put_contents($path."/my_".$sid.".txt",$data); //字符串格式不要求转化，会自动处理

}
// 当使用 session_destroy()，触发执行此函数
function destroy($sid){
    global $path;
    if(file_exists($path."/my_".$sid.".txt")){
        unlink($path."/my_".$sid.".txt");
    }
}

function gc($maxtime){
    global $path;

    $list = glob($path."/my_*"); //获取以 my_开始的 session 文件

    foreach($list as $name){
        if((filemtime($name)+$maxtime)<time()){
            unlink($name);
        }
    }
    return true;
}
```

使用：

```
    //自定义文件式 session，读操作

    require("filesession.php"); //开启之前使用


    session_start();
```

e) 自定义 session 以文件形式保存 （filesession.class.php 类型形式）

i. 使用静态方法定义，效率高，不用 new 直接使用

```
<?php
class FileSession{
    static $path;
```

```
static function start(){
    //session_set_save_handler("open","close","read","write","destroy","gc");
    //以上解释:调用回调函数时回调的是类外面的函数，并不是回调本类的方法,所不行
    session_set_save_handler(
        array(__CLASS__,"open"),
        array(__CLASS__,"close"),
        array(__CLASS__,"read"),
        array(__CLASS__,"write"),
        array(__CLASS__,"destroy"),
        array(__CLASS__,"gc")
    );//回调函数调用类中方法格式要求 session_set_save_handler(类名[对象],"open")
    session_start();
}

static function open($save_path){
    self::$path=$save_path;//使用自身成员属性(self::属性)
}

static function close(){
    return true;
}

//读取
static function read($sid){
    $sessfile=self::$path.'/bro_'.$sid;
    return @file_get_contents($sessfile);
}

//写入
static function write($sid, $data){
    $sessfile=self::$path.'/bro_'.$sid;
    return @file_put_contents($sessfile, $data);
}

static function destroy($sid){
    $sessfile=self::$path.'/bro_'.$sid;
    if(file_exists($sessfile))
        unlink($sessfile);
```

```
    }

    static function gc($maxtime){
        foreach(glob(self::$path.'/bro_*') as $filename){
            if(filemtime($filename)+$maxtime < time())
                unlink($filename);
        }
        return true;
    }

}
FileSession::start();
```

f) 自定义 session 以数据库形式保存 （dbsession.class.php 类型形式）

i. 记录信息表 （统计在线用户等）

1. Session_id ，session 数据，修改时间（回收用到），[用户名，ip，浏览器]

2. 创建内存表（频繁操作）

```
<?php
session_module_name("user"); //指定 session 的处理模式

//使用数据库管理 session
class DbSession{
    public static $pdo;                 //pdo 对象

    public static $ctime;               //当前时间

    public static $maxlifetime;         //最大生存时间

    public static $uip;                 //用户 IP

    public static $uagent;              //用户使用浏览器

    //PDO $pdo 限制$pdo 为 PDO 类
    public static function start(PDO $pdo){

        self::$pdo = $pdo;
        self::$ctime = time();
        self::$maxlifetime = ini_get('session.gc_mxlifetime');
```

```
        self::$uip = !empty($_SERVER['HTTP_CLIENT_IP'])?$_SERVER['HTTP_CLIENT_IP']:(
!empty($_SERVER['HTTP_X_FORWARDED_FOR'])?$_SERVER['HTTP_X_FORWARDED_FOR']:(
        !empty($_SERVER['REMOTE_ADDR'])?$_SERVER['REMOTE_ADDR']:'')
        );

        //filter_var(self::$uip,FILTER_VALIDATE_IP) && self::$uip = ''; //短路

        // IP    REMOTE_ADDR    HTTP_CLIENT_IP    HTTP_X_FORWARDED_FOR

        self::$uagent = !empty($_SERVER['HTTP_USER_AGENT'])?
        $_SERVER['HTTP_USER_AGENT']:null;

        session_set_save_handler(
            array(__CLASS__,"open"),
            array(__CLASS__,"close"),
            array(__CLASS__,"read"),
            array(__CLASS__,"write"),
            array(__CLASS__,"destroy"),
            array(__CLASS__,"gc")
        );
        session_start();
    }

    public static function open($save_path){
        return true;
    }

    public static function close(){
        return true;
    }

    public static function read($sid){
        $sql="select sdata from session where sid=?";
        $stmt=self::$pdo->prepare($sql);
        $stmt->execute(array($sid));
        $row=$stmt->fetch(PDO::FETCH_ASSOC);

        //没有会话信息

        if(!$row){
            return '';
        }

        //如果超出时间，销毁 session

        if($row["utime"] + self::$maxlifetime < self::$ctime){
```

```
            self::destroy($sid);return '';
        }
        //换了 IP 或者浏览器
        if($row['uip'] != self::$uip || $row['uagent'] != self::$uagent){
            self::destroy($sid);return '';
        }
        return $row["sdata"];
    }

    public static function write($sid, $data){
        if(!empty($data)) {
            $sql="select * from session where sid=?";
            $stmt=self::$pdo->prepare($sql);
            $stmt->execute(array($sid));
            $row=$stmt->fetch(PDO::FETCH_ASSOC);
            if($stmt->rowCount() > 0 ){
                if($row['utime'] + self::$maxlifetime < self::$ctime || $row['sdata'] != $data){
                    $sql="update session set sdata=?, utime=? where sid=?";
                    $stmt=self::$pdo->prepare($sql);
                    return $stmt->execute(array($data, self::$ctime, $sid));
                }
            }else{
                $sql="insert into session(sid, sdata, utime, uip, uagent) values(?,?,?,?,?)";
                $stmt=self::$pdo->prepare($sql);
                return $stmt->execute(array($sid, $data, self::$ctime,self::$uip,self::$agent));
            }
        }

    }

    public static function destroy($sid){
        $sql="delete from session where sid=?";
        $stmt=self::$pdo->prepare($sql);
        return $stmt->execute(array($sid));
    }

    public static function gc($maxlifetime){
        $sql="delete from session where utime < ?";
        $stmt=self::$pdo->prepare($sql);
        return $stmt->execute(array(self::$ctime - self::$maxlifetime));
    }

}
```

```
try{
    $driver_opts=array(PDO::ATTR_AUTOCOMMIT=>0);
    $pdo=new PDO("mysql:host=localhost;dbname=mysqldb","root","root",$driver_opts);
    DbSession::start($pdo);

    //默认是 0,什么也不提示；PDO::ERRMODE_WARNING 也可以用 1 代替；

    //PDO::ERRMODE_EXCEPTION 也可以        //用 2 代替


    $pdo->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
  }catch(PDOException $e){

    echo "连接数据库失败：".$e->getMessage();

}


//可以分离开写，主要传递 PDO  对象即可

try{
    $driver_opts=array(PDO::ATTR_AUTOCOMMIT=>0);
    $pdo=new PDO("mysql:host=localhost;dbname=mysqldb","root","root",$driver_opts);
    DbSession::start($pdo);

    //默认是 0,什么也不提示；PDO::ERRMODE_WARNING 也可以用 1 代替；

    //PDO::ERRMODE_EXCEPTION 也可以        //用 2 代替

    $pdo->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
  }catch(PDOException $e){

    echo "连接数据库失败：".$e->getMessage();

}
```

g) 自定义 session 以内存形式保存 （memsession.class.php 类型形式）

```
<?php
class MemSession{
    static $mem;
    static $maxtime;
    static function start(Memcache $mem){
        self::$maxtime=ini_get("session.gc_maxlifetime");
        self::$mem=$mem;
        session_set_save_handler(
            array(__CLASS__,"open"),
            array(__CLASS__,"close"),
            array(__CLASS__,"read"),
            array(__CLASS__,"write"),
```

```
                array(__CLASS__,"destroy"),
                array(__CLASS__,"gc")
            );
            session_start();       //开始之前使用 session_set_save_handler
        }

        static function open($save_path){

            return true;
        }

        static function close(){
            return true;
        }

        static function read($sid){

            return self::$mem->get($sid);
        }

        static function write($sid, $data){

            self::$mem->set($sid, $data, MEMCACHE_COMPRESSED, self::$maxtime);
        }

        static function destroy($sid){

            self::$mem->delete($sid, 0);

        }

        static function gc($maxtime){
            return true;

        }

}


$mem=new Memcache;

$mem->addServer("localhost",11211);
$mem->addServer("192.168.50.7", 11211);
```

```
$mem->addServer("192.168.50.119", 11211);
$mem->addServer("192.168.50.26", 11211);

MemSession::start($mem);
```

七．总结采用用户自定义 session 的步骤

1.session_module_name('user');

session 文件保存方式，这个是必须的！除非在 Php.ini 文件中设置了

2. session_set_save_handler ("open", "close", "read", "write","destroy", "gc");

open:　在运行 session_start()时执行

close: 在脚本执行完成或调用 session_write_close() 或 session_destroy()时 被执行,即在所有 session 操作完后被执行

read:　在运行 session_start()时执行,因为在 session_start 时,会去 read 当前 session 数据

write: 此方法在脚本结束和使用 session_write_close()强制提交 SESSION 数据时 执行

destroy: 在运行 session_destroy()时执行

gc: 执行概率由 session.gc_probability 和 session.gc_divisor 的值决定,时机是 在 open,read 之后,session_start 会相继执行 open,read 和 gc

3. session_start(); 这也是必须的，打开 session，必须在 session_set_save_handler 后面执行

4. 注意： session 和 cookie 是不能跨域或跨服务器,让两台服务器访问同一个 memcache 内存，去访问同一个 session 通过$_COOKIE['PHPSESSID']

a) vi php.ini
b) session.save_handler = memcache
c) session.save_path="tcp://192.168.100.254:11211";
d) //重启 apache,httpd -k restart