**memcached**是一套[分布式](http://zh.wikipedia.org/w/index.php?title=%E5%88%86%E5%B8%83%E5%BC%8F&action=edit&redlink=1)的高速缓存系统，由[LiveJournal](http://zh.wikipedia.org/wiki/LiveJournal)的Brad Fitzpatrick开发，但目前被许多网站使用。这是一套[开放源代码](http://zh.wikipedia.org/wiki/%E9%96%8B%E6%94%BE%E5%8E%9F%E5%A7%8B%E7%A2%BC)[软件](http://zh.wikipedia.org/wiki/%E8%BB%9F%E9%AB%94)，以[BSD license](http://zh.wikipedia.org/wiki/BSD_license)授权发布。

memcached缺乏[认证](http://zh.wikipedia.org/wiki/%E8%AA%8D%E8%AD%89)以及[安全](http://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%AE%89%E5%85%A8)管制，这代表应该将memcached服务器放置在[防火墙](http://zh.wikipedia.org/wiki/%E9%98%B2%E7%81%AB%E7%89%86)后

<http://www.zhihu.com/question/19645807>

和memcached更为接近的是redis。它们都是内存型数据库，数据保存在内存中，通过tcp直接存取，优势是速度快，并发高，缺点是数据类型有限，查询功能不强，一般用作缓存。

安装好memcached之后，启动memcached，可以指定监听的TCP和UDP端口号。

## 如何查看memcached状态

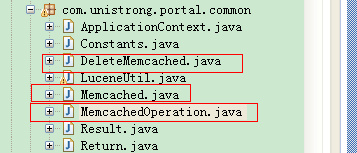
本以为会用配套的memcached客户端连接工具，网上找了下只能用telnet登陆上去，用shell命令查看状态。

telnet 192.168.108.13 11211

参考：

<http://www.blogjava.net/hao446tian/archive/2012/04/11/373773.html>

## Mygou项目是如何使用memcached的



主要是这三个类。

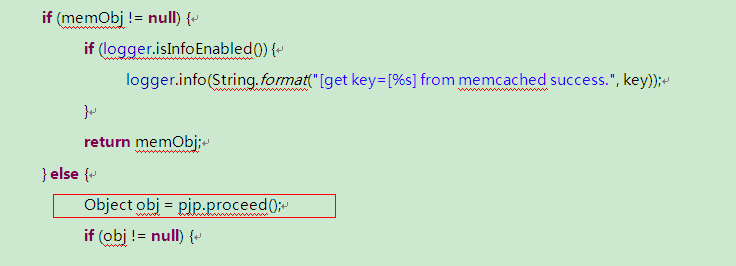
使用了annotation技术，在service层查询数据库之前，先从MemCached查询是否存在！

|  |
| --- |
| **package** com.unistrong.portal.common;  **import** java.lang.annotation.ElementType;  **import** java.lang.annotation.Retention;  **import** java.lang.annotation.RetentionPolicy;  **import** java.lang.annotation.Target;  @Target(ElementType.*METHOD*)  @Retention(RetentionPolicy.*RUNTIME*)  **public** **@interface** Memcached {  String name() **default** "";  String expiryDate() **default** "";  **int** timeout() **default** 0; // 毫秒 0永久缓存  **boolean** hasArgs2Key() **default** **false**;  String description() **default** "memcached";  } |

|  |
| --- |
| **package** com.unistrong.portal.common;  **import** java.lang.annotation.ElementType;  **import** java.lang.annotation.Retention;  **import** java.lang.annotation.RetentionPolicy;  **import** java.lang.annotation.Target;  @Target(ElementType.*METHOD*)  @Retention(RetentionPolicy.*RUNTIME*)  **public** **@interface** DeleteMemcached {  String name() **default** "";  **boolean** hasArgs2Key() **default** **false**;    String description() **default** "delete memcached";  } |

关键代码：

@Around("execution(\* com.unistrong.portal.service.impl.\*.\*(..)) && @annotation(memcached)")



**pjp.proceed() 直接可控制service method是否被执行！！！**

|  |
| --- |
| @Aspect  @Component  **public** **class** MemcachedOperation {  Logger logger = Logger.*getLogger*(**this**.getClass());  @Resource(name = "memcachedClient")  MemCachedClient memCachedClient;  /\*\*  \* **@Description**: 先Memcached中查询，如果有就返回没有就先查询数据库，查完之后插入到Memcached中,方法执行前后操作  \* **@param** joinPoint  \* **@param** deleteMemcached  \* **@throws** Throwable  \* **@author** hy.xu  \* **@version** Apr 11, 2013 4:03:39 PM  \*/  @Around("execution(\* com.unistrong.portal.service.impl.\*.\*(..)) && @annotation(memcached)")  **public** Object AroundMemcached(ProceedingJoinPoint pjp, Memcached memcached) **throws** Throwable {  String key = memcached.name();  **if** (key == **null** || "".equals(key)) {  **throw** **new** Exception("memcached name property is must.");  }  key = getKey(pjp, key, memcached.hasArgs2Key());  Object memObj = memCachedClient.get(key);  **if** (memObj != **null**) {  **if** (logger.isInfoEnabled()) {  logger.info(String.*format*("[get key=[%s] from memcached success.", key));  }  **return** memObj;  } **else** {  Object obj = pjp.proceed();  **if** (obj != **null**) {  Return ret = (Return) obj;  Object dataObj = ret.getData();  **if** (dataObj **instanceof** Track) {  **if** (dataObj != **null**) {  ((Track) dataObj).setTrk\_seg(**null**);  }  }  **if** (dataObj **instanceof** RouterTrack) {  **if** (dataObj != **null**) {  ((RouterTrack) dataObj).setTrk\_seg(**null**);  }  }  **boolean** isCached = **false**;  **if** (memcached.expiryDate() != **null** && !"".equals(memcached.expiryDate())) {  Date expiryDate = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse(memcached.expiryDate());  isCached = memCachedClient.set(key, obj, expiryDate, memcached.timeout());  } **else** {  isCached = memCachedClient.set(key, obj, memcached.timeout());  }  **if** (!isCached) {  logger.warn("[" + key + "] has not memcached.");  } **else** {  **if** (logger.isInfoEnabled()) {  logger.info(String.*format*("[put key=[%s] in memcached success.", key));  }  }  }  **return** obj;  }  }  /\*\*  \* **@Description**: 删除Memcached,方法执行后操作  \* **@param** joinPoint  \* **@param** deleteMemcached  \* **@throws** Throwable  \* **@author** hy.xu  \* **@version** Apr 11, 2013 4:03:39 PM  \*/  @After("execution(\* com.unistrong.portal.service.impl.\*.\*(..)) && @annotation(deleteMemcached)")  **public** **void** deleteMemcached(JoinPoint joinPoint, DeleteMemcached deleteMemcached) **throws** Throwable {  String name = deleteMemcached.name();  **if** (name == **null** || "".equals(name)) {  **throw** **new** Exception("deleteMemcached name property is must.");  }  String key = "";  **boolean** isDeleted = **false**;  **if** (deleteMemcached.hasArgs2Key()) {  Object[] args = joinPoint.getArgs();  **if** (args != **null** && args.length >= 1) {  **if** (args[0] **instanceof** Long) {// Long类型参数 方法体删除操作  // 规则：name\_uuid\_userid  key = name + "\_" + args[0].toString() + "\_" + args[1].toString();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  } **else** **if** (args[0] **instanceof** String) {// String类型参数 方法体删除操作  // 规则：name\_code  **if** (args.length > 1 && args[1] != **null**) {  key = name + "\_" + args[0].toString() + "\_" + args[1].toString();  } **else** {  key = name + "\_" + args[0].toString();  }  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  } **else** **if** (args[0] **instanceof** BaseInfo) {// 基础信息 方法体添加或编辑操作  // 规则：name\_code  key = name + "\_" + ((BaseInfo) args[0]).getCode();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  } **else** **if** (args[0] **instanceof** Waypoint) {// 航点 方法体添加或编辑操作  // 规则：name\_uuid\_userid  key = name + "\_" + ((Waypoint) args[0]).getUuid() + "\_" + ((Waypoint) args[0]).getUser\_id();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  } **else** **if** (args[0] **instanceof** Waypoint[]) { // 航点 批量添加操作  // 规则：name\_uuid\_userid  **for** (Waypoint waypoint : (Waypoint[]) args[0]) {  key = name + "\_" + waypoint.getUuid() + "\_" + waypoint.getUser\_id();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  key = "";  }  } **else** **if** (args[0] **instanceof** Line) {// 航线 方法体添加或编辑操作  // 规则：name\_uuid\_userid  key = name + "\_" + ((Line) args[0]).getUuid() + "\_" + ((Line) args[0]).getUser\_id();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  } **else** **if** (args[0] **instanceof** Line[]) { // 航线 批量添加操作  // 规则：name\_uuid\_userid  **for** (Line line : (Line[]) args[0]) {  key = name + "\_" + line.getUuid() + "\_" + line.getUser\_id();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  key = "";  }  } **else** **if** (args[0] **instanceof** Track) {// 航迹 方法体添加或编辑操作  // 规则：name\_uuid\_userid  key = name + "\_" + ((Track) args[0]).getUuid() + "\_" + ((Track) args[0]).getUser\_id();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  } **else** **if** (args[0] **instanceof** Track[]) { // 航迹 批量添加操作  // 规则：name\_uuid\_userid  **for** (Track track : (Track[]) args[0]) {  key = name + "\_" + track.getUuid() + "\_" + track.getUser\_id();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  key = "";  }  } **else** **if** (args[0] **instanceof** RouterTrack) {// 路书航迹 方法体添加或编辑操作  // 规则：name\_uuid\_userid  key = name + "\_" + ((RouterTrack) args[0]).getUuid() + "\_" + ((RouterTrack) args[0]).getUser\_id();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  } **else** **if** (args[0] **instanceof** RouterTrack[]) { // 路书航迹 批量添加操作  // 规则：name\_uuid\_userid  **for** (RouterTrack routerTrack : (RouterTrack[]) args[0]) {  key = name + "\_" + routerTrack.getUuid() + "\_" + routerTrack.getUser\_id();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  key = "";  }  } **else** **if** (args[0] **instanceof** Router) {// 路书 方法体添加或编辑操作  // 规则：name\_uuid\_userid  key = name + "\_" + ((Router) args[0]).getUuid() + "\_" + ((Router) args[0]).getUser\_id();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  } **else** **if** (args[0] **instanceof** Router[]) { // 路书 批量添加操作  // 规则：name\_uuid\_userid  **for** (Router router : (Router[]) args[0]) {  key = name + "\_" + router.getUuid() + "\_" + router.getUser\_id();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  key = "";  }  } **else** **if** (args[0] **instanceof** String[]) { // 航线 批量分享操作  // 规则：name\_uuid\_userid  **for** (String uuid : (String[]) args[0]) {  key = name + "\_" + uuid + "\_" + args[1];  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  key = "";  }  } **else** **if** (args[0] **instanceof** Long[]) {// Long数组类型参数 批量删除操作  // 规则：name\_uuid\_userid(航点航线使用)  **for** (Long k : (Long[]) args[0]) {  key = name + "\_" + k + "\_" + args[1].toString();  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  key = "";  }  }  }  } **else** {  key = name;  isDeleted = memCachedClient.delete(key);  logDeleted(key, isDeleted);  }  }  **private** String getKey(JoinPoint joinPoint, String key, **boolean** hasArgs2Key) **throws** Exception {  **if** (hasArgs2Key) {  Object[] args = joinPoint.getArgs();  **if** (args != **null** && args.length >= 1) {  **if** (args[0] **instanceof** Long) {// Long类型参数 查找操作  // 规则：name\_uuid\_userid  key += "\_" + args[0].toString() + "\_" + args[1].toString();  } **else** **if** (args[0] **instanceof** String) {// String类型参数 查找操作  // 规则：name\_code  **if** (args.length > 1 && args[1] != **null**) {  key += "\_" + args[0].toString() + "\_" + args[1].toString();  } **else** {  key += "\_" + args[0].toString();  }  } **else** **if** (args[0] **instanceof** BaseInfo) {// 基础信息 查找操作  key += "\_" + ((BaseInfo) args[0]).getCode();  } **else** **if** (args[0] **instanceof** Waypoint) {  key += "\_" + ((Waypoint) args[0]).getUuid() + "\_" + ((Waypoint) args[0]).getUser\_id();  } **else** **if** (args[0] **instanceof** Line) {  key += "\_" + ((Line) args[0]).getUuid() + "\_" + ((Line) args[0]).getUser\_id();  } **else** **if** (args[0] **instanceof** Track) {  key += "\_" + ((Track) args[0]).getUuid() + "\_" + ((Track) args[0]).getUser\_id();  } **else** **if** (args[0] **instanceof** RouterTrack) {  key += "\_" + ((RouterTrack) args[0]).getUuid() + "\_" + ((RouterTrack) args[0]).getUser\_id();  } **else** **if** (args[0] **instanceof** Router) {  key += "\_" + ((Router) args[0]).getUuid() + "\_" + ((Router) args[0]).getUser\_id();  } **else** {  key += "\_" + args[0].toString();  }  }  }  **return** key;  }  **private** **void** logDeleted(String key, **boolean** isDeleted) {  **if** (!isDeleted) {  logger.warn("[" + key + "] has not been deleted in memcached success.");  } **else** {  **if** (logger.isInfoEnabled()) {  logger.info(String.*format*("[deleted key=[%s] from memcached success.", key));  }  }  }  **public** Object getValue(String key) {  **return** memCachedClient.get(key);  }  **public** **void** setValue(String key, Object value) {  memCachedClient.set(key, value);  }  } |

## Memcached java客户端库

Mygou使用**com.danga.MemCached.MemCachedClient**

<https://github.com/gwhalin/Memcached-Java-Client/downloads>