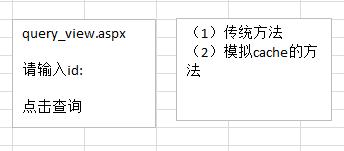
# **第一部分：为什么我的项目中要使用Redis**

我知道有些地方没说到位，希望大神们提出来，我会吸取教训，大家共同进步！

* 注册时邮件激活的部分使用Redis
* 发送邮件时使用Redis的消息队列，减轻网站压力。
* 使用Lucene.Net在进行分词时使用Redis消息队列和多线程来避免界面卡死等性能问题。

* 请大家先思考一个问题：这个问题在大并发、高负载的网站中必须考虑！大家思考如何让速度更快。



三种方法：（1）数据库（2）页面静态化（3）Redis、Memcached

# **第二部分：Redis是什么**

概述：redis是一种nosql数据库,他的数据是保存在内存中，同时redis可以定时把内存数据同步到磁盘，即可以将数据持久化，并且他比memcached支持更多的数据结构(string,list列表[队列和栈],set[集合],sorted set[有序集合] hash(hash表))

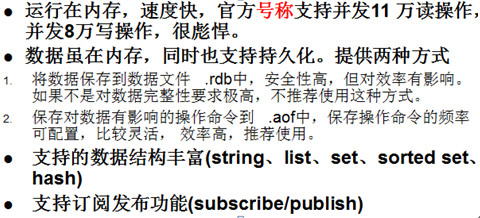
## **2.1介绍：**

* Redis是一个高性能的key-value存储系统。和Memcached类似，它支持存储的value类型相对更多，包括string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。
* Redis 很大程度补偿了memcached这类key/value存储的不足，在部 分场合可以对关系数据库起到很好的补充作用。它提供了Python，Ruby，Erlang，PHP客户端，使用很方便。(注: 摘自百度全科)，1.主要是支持持久化2.支持更多数据结构 3.支持主从同步
* Redis支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。

## **2.2  memcached和redis的比较：**



## **2.3Redis的优势：**



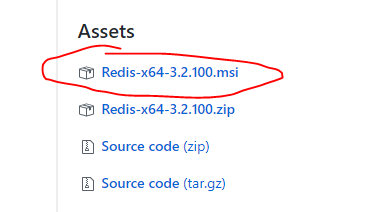
## 

## **2.4Redis在windows下的安装：**

注：关于Redis的安装网上有很多文章，讲的要比我的好，建议大家去看那些大神的文章，这里我只简单介绍一下。

1. 解压Redisbin.zip

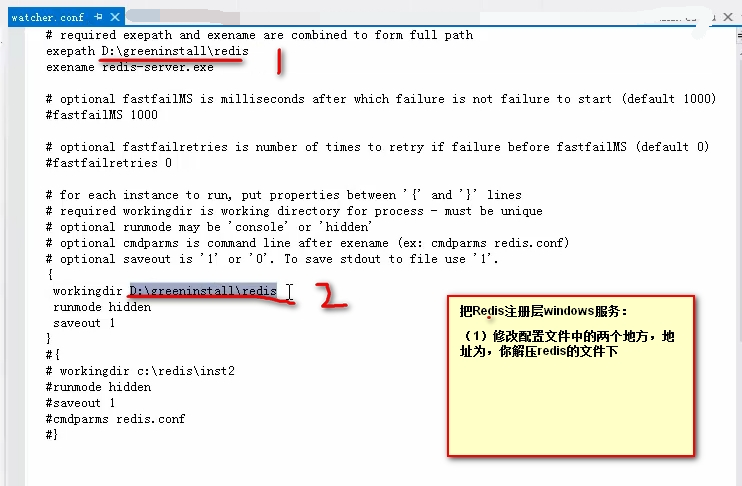
首先下载Redis，下载连接为：https://github.com/MicrosoftArchive/redis/releases，选中,mis文件下载。、



　　下载完成后点击安装。进入安装界面进行安装，所有的选项都选中默认即可。

（2）注册成windows服务

安装RedisWatch,会把Redis注册为一个系统服务，然后到安装RedisWatch的文件夹下找到watcher.config,修改下面的两个地方，如下图：



**特别提醒：作为一个专业的程序员文件放置的路径不要有特殊字符、空格、文件名不要有中文，否则就加班吧。**

当然了，正式的生产环境还是Linux的效率高，因为在Linux上是源码安装。

（3）启动redis服务

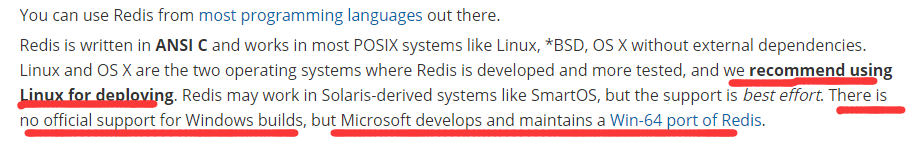
IMG_260

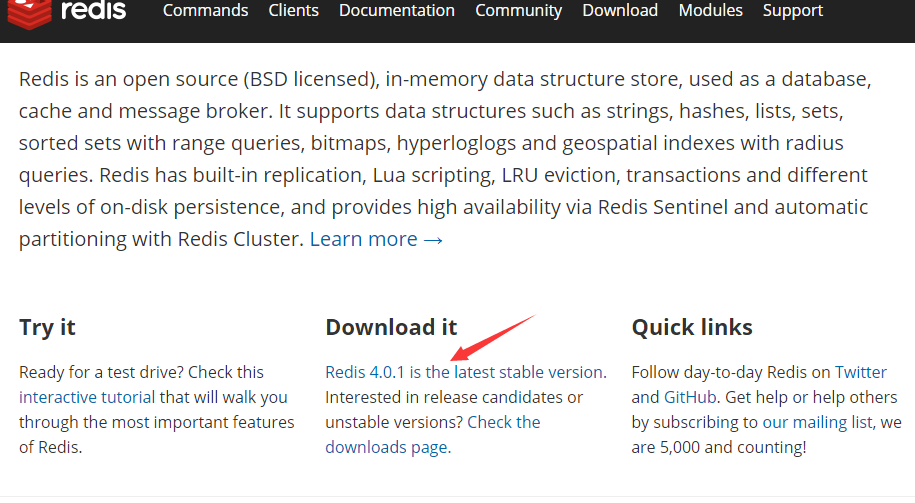
（4）使用netstat -anb | more

如果你看到有6379 在监听，说明ok（默认的端口号时候：6379）

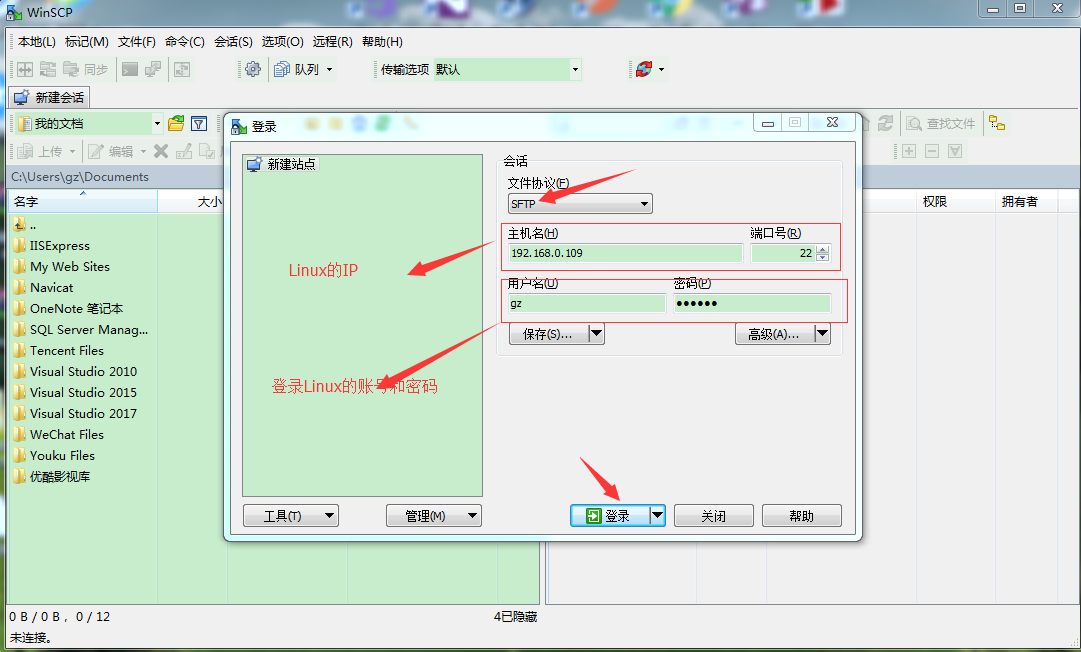
## **正确使用Redis的姿势：Redis在Linux(Ubuntu16.04)下的安装（可以直接忽略上面在Windows上Redis的操作，这里使用的Redis版本为Redis4.0.1稳定版）**

 （1）到官网上下载安装包 redis-stable.tar.gz   https://redis.io/，官网只提供Linux版本，没有Windows版本的，只要Windows版本的都是微软移植过来的，而且官方推荐使用Linux版本。

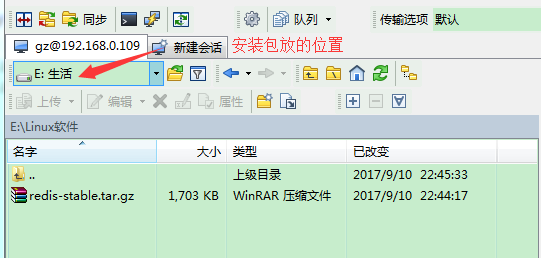


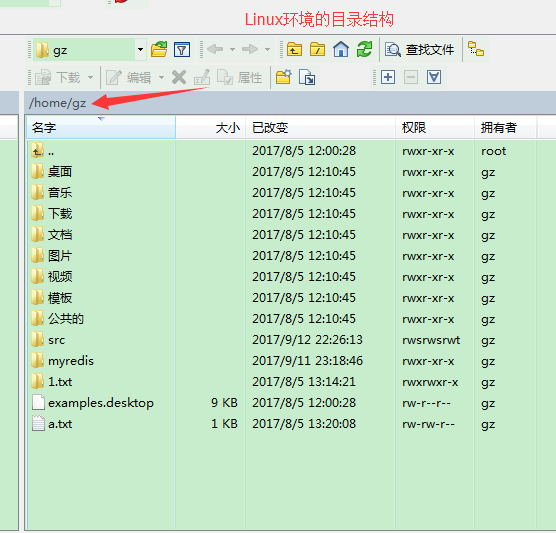


（2）使用WinSCP把下载的安装包，放到Ubuntu中对应的目录中。



如果在登录的过程中有弹窗，不要慌，点击是即可。登录成之后的界面：

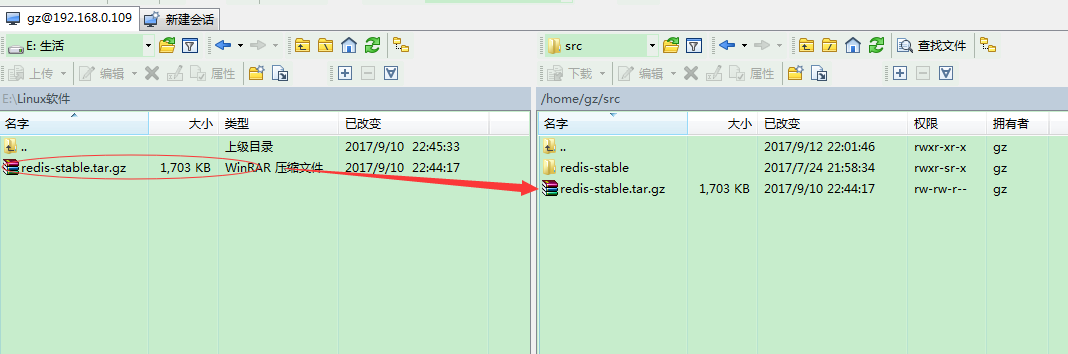




使用Linux的指令（mkdir src）创建一个目录，来放Redis的安装包：

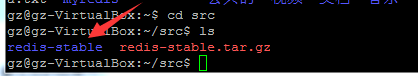


由于之前测试，已经建了src目录，所以在这里我们可以直接把安装包，拖过来即可。



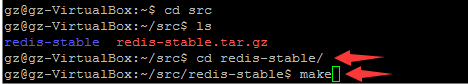
（3）解压

进入到src目录，执行  tar -zxvf  redis-stable.tar.gz 解压,解压的过程就不截图了，解压后的结果为：

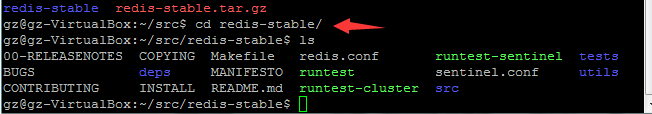


 （4）编译源代码

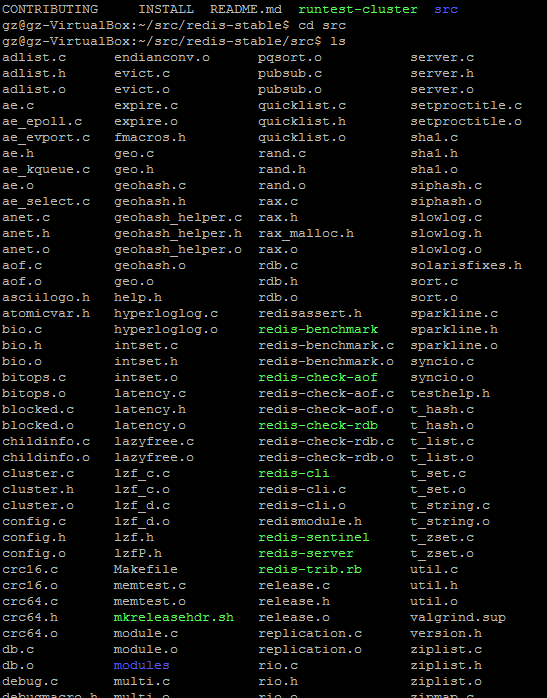
进入到redis-stable目录中，再执行make



（5）使用ls指令，可以看到该目录下所有的文件：

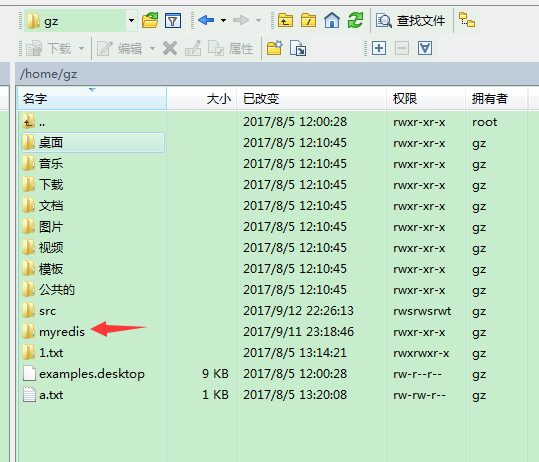


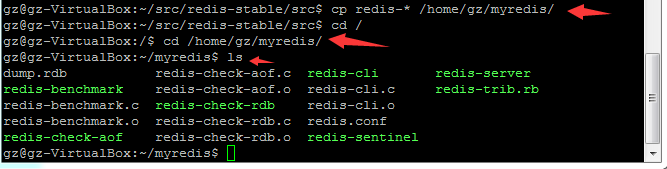
该目录下用一个src的目录，使用cd src进入到该目录，再使用 ls指令



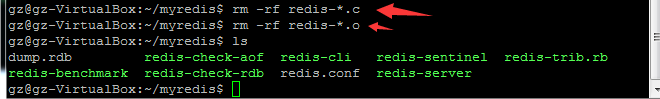
将 redis-benchmark（压力测试工具）、redis-check-aof（检查.aof文件完整性的工具）、redis-check-dump（检查数据文件完整性的工具）、redis-sentinel（监控集群运行状态）、redis-server（服务端）、redis-cli（客户端），**还有一个文件redis.conf 也拷贝到 myredis该文件在src的上级目录**

拷贝到你的工作目录myredis 中：cp redis-\* /home/gz/myredis/





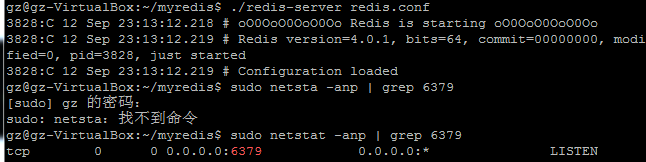
 进入到myredis目录中，发现有多余的文件，然后再使用：



是不是干净多了。

（6）启动Redis

进入到myredis目录中，使用   ./redis-serve redis.conf来启动服务



如果我们的6379端口被监听，说明我们的服务已经成功启动了。

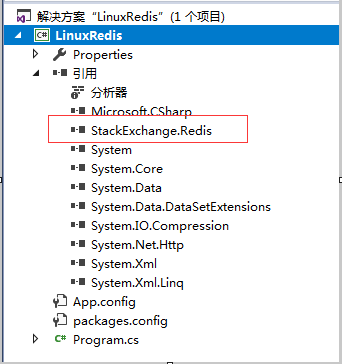
**注意：**

**默认是前端启动，占用你的控制台,我们修改 redis.conf 文件为后台进行，将 daemonize  no 修改成yes。**

 （7）C#连接Redis简单测试一下：

在这里回答一下@Partialsky的问题：用StackExchange.Redis ，而不是ServiceStack.Redis，因为StackExchange.Redis依赖组件少，而且操作更接近原生的redis操作，ServiceStack封装的太厉害，而且之前收费，反正最好还是用StackExchange.Redis。

step1：使用VS2017新建一个控制台程序



step2: Install-Package StackExchange.Redis

 step3:编写代码：

using StackExchange**.**Redis**;**

using System**;**

using System**.**Collections**.**Generic**;**

using System**.**Linq**;**

using System**.**Text**;**

using System**.**Threading**.**Tasks**;**

namespace LinuxRedis

**{**

class Program

**{**

static void Main**(**string**[]** args**)**

**{**

using **(**ConnectionMultiplexer redis **=** ConnectionMultiplexer**.**Connect**(**"192.168.0.109:6379"**))**

**{**

IDatabase db **=** redis**.**GetDatabase**();**

db**.**StringSet**(**"guozheng"**,** "hahaha"**);**

var age **=** db**.**StringGet**(**"guozheng"**);**

Console**.**WriteLine**(**age**);**

**}**

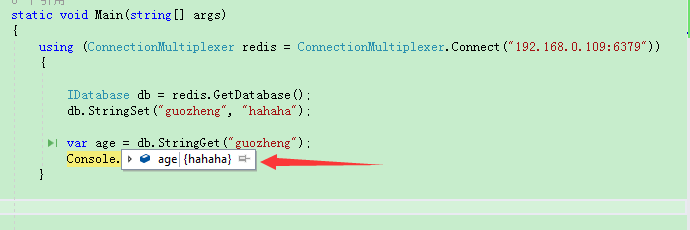
Console**.**ReadKey**();**

**}**

**}**

**}**

测试结果：



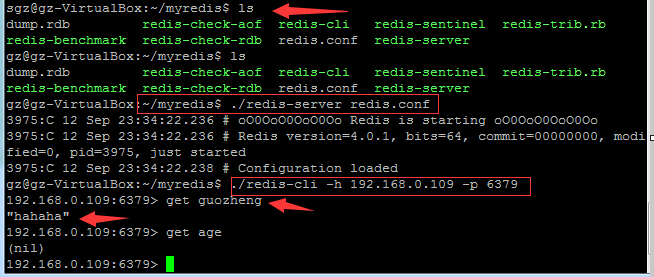
 再到Linux上看看输入是否存入到Redis中：

启动服务：

./redis-server redis.conf

连接到redis

./redis-cli -h ip地址 -p 端口



数据也成功存入到redis中了。好了，C#如何简单操作Redis就讲到这里。如果大家对如何安装Ubuntu和Linux的操作指令不太清楚，可以先看看其他园友的文章，有时间根据大家的反应，再去写篇关于Linux的文章。

## **2.5Redis的数据结构：**

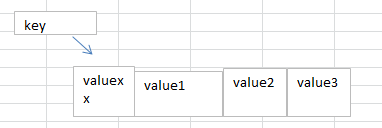
前言：Redis中存储的数据都为字符串格式的。下面来分别介绍Redis中常用的数据结构。

* string数据结构

太简单了，略过。

* list数据结构

概述: 什么是list ,list是一种数据结构，可以当做队列和栈来使用。



当你从左边添加数据，再从左边取数据，就模拟出栈；当你从右边添加数据，再从左边取数据，就模拟出队列。因此Redis真的很强大，看到栈和队列这样的数据结构，你难道就不激动吗？这样的数据结构太TM好了，能帮我们处理很多棘手的问题。这里我先卖个关子，下面会介绍我在项目中是如何使用Redis解决棘手的问题。

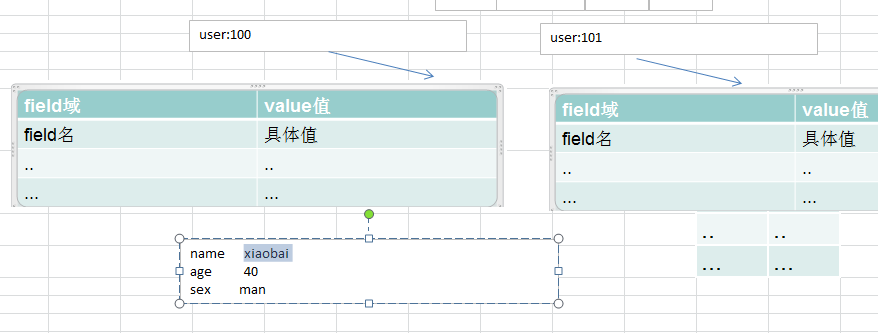
* set集合

和list结构差不多，这里不再啰嗦。

下面就是操作set的一些命令。



* hash数据结构



图中的"user:100"就相当于key,而它所指向的类似于表结构的数据就是value,这样的数据结构有利于存储对象数据。也是非常常用的方法。

强烈推荐：  Redis常用命令文档：http://redisdoc.com/  文档上有详细的操作案例和高级用法。

 注意：

redis指令不区分大小写，但是出于规范考虑，应该使用大写  
**redis中存放的键是区分大小写的.**

# **第三部分：Redis如何使用**

 3.1C#中如何使用Redis来解决邮箱激活的实效性。

首先思考个问题：为什么要进行邮件激活？激活码该存到哪里？（大家先思考，我不直接说，这样通过下面的例子你会体会的更深。）

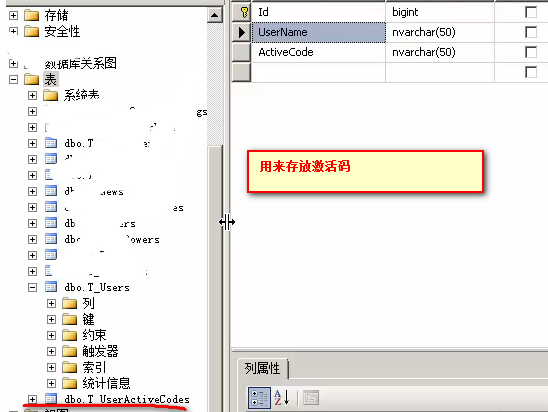
原因：用户在注册的时候，虽然正则表达式能检查邮箱的格式是否正确，但是正则检查不了邮箱是否可用，于是让用户进行激活，就能避免用户填写一个不可用的邮箱。

传统方法的代码实现：

1）数据库表的设计：



在用户注册的表中添加一个字段：IsActive用来判断激活的状态。



该表用来存放激活码。

 代码实现：

BLL层代码：

/// <summary>

/// 2016-08-30

/// 注册的时候看看是否已经存在该用户

/// </summary>

/// <param name="username"></param>

/// <returns></returns>

public T\_Users GetByUserName(string username)

{

T\_UsersDAL userDal = new T\_UsersDAL();

return userDal.GetByUserName(username);

}

/// <summary>

/// 注册的时候看看邮箱是否已经被注册

/// </summary>

/// <returns></returns>

public bool CheckEmailOnReg(string email)

{

T\_UsersDAL userDal = new T\_UsersDAL();

T\_Users user= userDal.CheckEmailOnReg(email);

return user == null;

}

DAL层代码：

/// <summary>

/// 注册时候看看是否已经存在该用户名

/// </summary>

/// <param name="username"></param>

/// <returns></returns>

public T\_Users GetByUserName(string username)

{

string sql = "select \* from T\_Users where UserName=@UserName";

DataTable dt = SqlHelper.ExecuteQuery(sql, new SqlParameter("@UserName", username));

T\_Users userInfo = null;

if (dt.Rows.Count > 0)

{

foreach (DataRow dr in dt.Rows)

{

userInfo = RowToUserInfoByDataRow(dr);

}

}

return userInfo;

}

/// <summary>

/// 注册的时候检查用户的邮箱是否被注册

/// </summary>

/// <param name="email"></param>

/// <returns></returns>

public T\_Users CheckEmailOnReg(string email)

{

string sql = "select \* from T\_Users where Email=@Email ";

DataTable dt = SqlHelper.ExecuteQuery(sql, new SqlParameter("@Email",email));

T\_Users userInfo = null;

if (dt.Rows.Count>0)

{

foreach (DataRow dr in dt.Rows)

{

userInfo=RowToUserInfoByDataRow(dr);

}

}

return userInfo;

}

UI层代码：

<!--#include file="/html/head.html"-->

<title>注册</title>

<!--#include file="/html/linkscript.html"-->

<script type="text/javascript">

function checkPasswordLevel(value) {

if (!value) {

return 1;

}

if (value.length < 6) {

return 1;

}

if (value.length == 6 && (/[0-9]/.test(value) || /[a-z]/.test(value))) {

return 1;

}

if (value.length >= 6 && /[0-9]/.test(value) && /[a-z]/.test(value) && /(?=[\x21-\x7e]+)[^A-Za-z0-9]/.test(value)) {

return 3;

}

return 2;

}

$(function () {

$("#btnReg").click(function () {

var username = $("#username").val();

var password = $("#password").val();

var password2 = $("#password2").val();

var email = $("#email").val();

var phone = $("#PhoneNum").val();

var qq = $("#qq").val();

var school = $("#school").val();

var validCode = $("#validCode").val();

//todo：非空验证。JQuery EasyUI

if (phone == "") {

$("#phoneMsg").text("手机号不能为空！");

return;

}

else {

var reg = "^1(3[0-9]|4[57]|5[0-35-9]|7[01678]|8[0-9])\\d{8}$";

if (!reg.test(phone)) {

$("#phoneMsg").text("手机号不合法！");

return;

}

}

if (qq=="") {

$("#qqMsg").text("QQ号不能为空！");

return;

}

if (school=="") {

$("#schoolMsg").text("学校不能为空！");

}

if (validCode == "") {

$("#validateCodeMsg").text("验证码不能为空!");

return;

}

if (password == "") {

$("#userPasswordMsg").text("密码不能为空!");

return;

}

if (password != password2) {

$("#pwdError").text("两次输入的密码不一致！");

return;

}

$.ajax({

url: "UserController.ashx", type: "post",

dataType: "json",

data: { action: "registerSubmit", username: username, password: password, email: email, validCode: validCode, phone: phone, qq: qq, school: school },

success: function (data) {

if (data.status == "ok") {

alert("注册成功");

window.location.href = "index.shtml";

}

else {

alert("注册失败：" + data.msg);

//只有这句话刷新验证码是不安全的,需要后台也刷新验证码

$("#imgValidCode").attr("src", "UserController.ashx?action=createValideCode&id=" + new Date());

}

},

error: function () {

alert("注册请求失败");

}

});

});

$("#password").keyup(function () {

var level = checkPasswordLevel($("#password").val());

switch (level) {

case 1: {

$("#td1").css("backgroundColor", "#FF8000");

$("#td2").css("backgroundColor", "");

$("#td3").css("backgroundColor", "");

}

break;

case 2: {

$("#td1").css("backgroundColor", "");

$("#td2").css("backgroundColor", "#FF4000");

$("#td3").css("backgroundColor", "");

}

break;

case 3: {

$("#td1").css("backgroundColor", "");

$("#td2").css("backgroundColor", "");

$("#td3").css("backgroundColor", "#5CB85C");

}

break;

}

})

$("#password2").blur(function () {

var password = $("#password").val();

var password2 = $("#password2").val();

if (password != password2) {

$("#pwdError").text("两次输入的密码不一致！");

return;

}

else {

$("#pwdError").text("");

}

});

//todo：焦点离开email的时候，编写正则表达式检查email地址是否正确

//todo:前台用户的注册：邮件发送激活码，一个邮件只能注册一个账号。

$("#email").blur(function () {

var email = $(this).val();

if (email == "") {

$("#userEmailMsg").text("邮箱不能为空!");

return;

}

else {

var re = /^\w+@[a-z0-9]+(\.[a-z]+){1,3}$/;

if (re.test(email)) {

$.ajax({

url: "UserController.ashx", type: "post", dataType: "json",

data: { action: "checkEmail", email: email },

success: function (data) {

if (data.status == "ok") {

$("#userEmailMsg").text(data.msg);

return;

}

else if (data.status == "error") {

$("#userEmailMsg").text(data.msg);

return;

}

},

error: function () {

$("#userEmailMsg").text("检查邮箱是否可用失败");

return;

}

})

}

else {

$("#userEmailMsg").text("邮箱的格式不正确!");

return;

}

}

});

$("#username").keyup(function () {

$("#userNameMsg").text("");

});

$("#email").keyup(function () {

$("#userEmailMsg").text("");

});

//检查用户名是否可用。

$("#username").blur(function () {

var username = $("#username").val();

if (username == "") {

$("#userNameMsg").text("用户名不能为空！");

return;

}

$.ajax({

url: "UserController.ashx", type: "post", dataType: "json",

data: { action: "checkUserName", username: username },

success: function (data) {

if (data.status == "ok") {

$("#userNameMsg").text("此用户名可用");

}

else {

$("#userNameMsg").text("此用户名不可用，请换用其他用户名");

}

},

error: function () {

$("#userNameMsg").text("检查用户名是否可用失败");

}

});

});

});

</script>

<style type="text/css">

#table td

{

width: 70px;

height: 12px;

background-color: lightgray;

border: 1px solid #D0D0D0;

color: #BBBBBB;

line-height: 9px;

color: white;

font-size: 12px;

font-family: 微软雅黑;

}

</style>

<!--#include file="/html/headend.html"-->

<!--#include file="/html/navbar.html"-->

<main id="post-page" class="container mainContent" role="main">

<table>

<tr><td><label for="username">用户名：</label></td><td><input type="text" id="username" /><span id="userNameMsg"></span></td></tr>

<tr>

<td><label for="password">输入密码：</label></td>

<td>

<input type="password" id="password" /><span id="userPasswordMsg"></span>

<table id="table" border="0" cellpadding="0" cellspacing="1" style="display: inline-table;">

<tr>

<td id="td1" style="height: 12px; text-align: center;">弱</td>

<td id="td2" style="height: 12px; text-align: center;">中</td>

<td id="td3" style="height: 12px; text-align: center;">强</td>

</tr>

</table>

</td>

</tr>

<tr><td><label for="password2">再次输入密码：</label></td><td><input type="password" id="password2" /><label id="pwdError"></label></td></tr>

<tr><td><label>邮箱：</label></td><td><input type="text" id="email" /><span id="userEmailMsg"></span></td></tr>

<tr><td><label>手机号：</label></td><td><input type="text" id="PhoneNum" /><span id="phoneMsg"></span></td></tr>

<tr><td><label>QQ:</label></td><td><input type="text" id="qq" /><span id="qqMsg"></span></td></tr>

<tr><td><label>学校：</label></td><td><input type="text" id="school"/><span id="schoolMsg"></span></td></tr>

<tr><td><label>验证码：</label></td><td><input type="text" id="validCode" /><img src="UserController.ashx?action=createValideCode" id="imgValidCode" /><span id="validateCodeMsg"></span></td></tr>

<tr><td><input type="button" id="btnReg" value="注册" /></td><td></td></tr>

</table>

</main>

<!--#include file="/html/foot.html"-->

一般处理程序：

 /// <summary>

/// 用户注册

/// </summary>

/// <param name="context"></param>

public void registerSubmit(HttpContext context)

{

//获取请求报文中从浏览器传过来的数据

string username = context.Request["username"];

string password = context.Request["password"];

string email = context.Request["email"];

string phone = context.Request["phone"];

string qq = context.Request["qq"];

string school = context.Request["school"];

string validCode = context.Request["validCode"];

//注意：通过js进行数据合法性校验，只是为了用户用起来方便而已，在服务器中校验才能保证数据的安全。

if (string.IsNullOrWhiteSpace(phone))

{

AjaxHelper.WriteJson(context.Response, "error", "手机号不能为空！");

return;

}

if (string.IsNullOrWhiteSpace(qq))

{

AjaxHelper.WriteJson(context.Response,"error","QQ号不能为空！");

return;

}

if (string.IsNullOrWhiteSpace(school))

{

AjaxHelper.WriteJson(context.Response,"error","学校不能为空！");

return;

}

if (string.IsNullOrWhiteSpace(username))

{

AjaxHelper.WriteJson(context.Response,"error","用户名不能为空!");

return;

}

if (string.IsNullOrWhiteSpace(password))

{

AjaxHelper.WriteJson(context.Response,"error","密码不能为空!");

return;

}

if (string.IsNullOrWhiteSpace(email))

{

AjaxHelper.WriteJson(context.Response,"error","邮箱不能为空!");

return;

}

if (string.IsNullOrWhiteSpace(validCode))

{

AjaxHelper.WriteJson(context.Response,"error","验证码不能为空!");

return;

}

if (validCode!=CommonHelper.GetValidCode(context))

{

AjaxHelper.WriteJson(context.Response,"error","验证码错误");

CommonHelper.ResetValidCode(context);

return;

}

T\_UsersBLL userBll = new T\_UsersBLL();

if (!userBll.CheckUserNameOnReg(username))

{

AjaxHelper.WriteJson(context.Response,"error","当前用户不可用");

return;

}

if (!userBll.CheckEmailOnReg(email))

{

AjaxHelper.WriteJson(context.Response,"error","该邮箱已被注册!");

return;

}

//插入数据库(T\_Users)

long userId = userBll.AddNewUser(username, password, email,phone,qq,school);

//激活码

Random rand = new Random();

string activeCode = rand.Next(10000,99999).ToString();

//方案一：把激活码存入到数据库(T\_UserActiveCodes)

T\_UserActiveCodes userActiveCode = new T\_UserActiveCodes();

userActiveCode.UserName = username;

userActiveCode.RegDateTime = DateTime.Now;

userActiveCode.ActiveCode = activeCode;

//插入到激活码数据表中

new T\_UserActiveCodesBLL().Add(userActiveCode);

//邮件链接和正文

string activeUrl = "http://localhost:22585/UserController.ashx?action=active&username=" + context.Server.UrlEncode(username) + "&activeCode=" + activeCode;

string emailBody = "尊敬的" + username + "您好，请点击下面的链接激活您的账户"

+ "<a href='" + activeUrl + "'>点击此链接激活您的账号</a>，如果链接打不开，则把下面的地址复制到浏览器中进行激活：" + activeUrl;

//发送邮件

FrontHelper.SendEmail(email,"请激活您的\*\*\*账号",emailBody);

/\*

\* 测试了网易和qq邮箱，能发是能发但是，对所发的邮件标题和内容是有限制的，不能发很容就能识别出来是垃圾邮件的邮件，标题和正文要正式点，负责不会接收到。

\* 在生产环境中：无法使用163、qq等这种免费邮箱发送大量的邮件。

\* Edm专用服务器，掏钱就ok。

SendCloud、Comm100、yiye

\*/

邮件发送代码：

public static void SendEmail(string toEmail, string subject, string body)

{

string smtpServer = ConfigurationManager.AppSettings["SmtpServer"];

string smtpFrom = ConfigurationManager.AppSettings["SmtpFrom"];

string smtpUserName = ConfigurationManager.AppSettings["SmtpUserName"];

string smtpPassword = ConfigurationManager.AppSettings["SmtpPassword"];

MailMessage mailObj = new MailMessage();

mailObj.IsBodyHtml = true;

//from:abc@qq.com

mailObj.From = new MailAddress(smtpFrom); //发送人邮箱地址

mailObj.To.Add(toEmail); //收件人邮箱地址

mailObj.Subject = subject; //主题

mailObj.Body = body; //正文

SmtpClient smtp = new SmtpClient();//通过.Net内置的SmtpClient类和邮件服务器进行通讯，发送邮件。

//是和发邮件方的smtp通讯，由发邮件方的邮件服务器和收邮件方的邮件服务器通讯进行邮件的转接。

smtp.Host = smtpServer; //smtp服务器名称

smtp.UseDefaultCredentials = true;

smtp.Credentials = new NetworkCredential(smtpUserName, smtpPassword); //发送人的登录名和密码

smtp.Send(mailObj);

}

关于邮箱的账号和密码最好配置到配置文件中。为了安全。

好好思考一下这样写的缺陷在哪？不仅有缺陷而且还有安全问题，有哪些安全问题？如果用户量大的话这样设计是否合理？会对什么有压力？如果不合理该如何优化？

首先我们来分析一下：

上面的方法是在用户表的基础上再增加一个字段，用来存激活码。这样合理吗？  
由于激活码只用一次，所以在用户表的基础上再增加一个字段会麻烦一下，之前的功能会有影响。那到底该怎么解决比较好？

这时候Redis的好处就非常明显了，key-value数据库，并且还能设置数据的有效时间，很好的解决了上面遇到的问题，只需要改动上面很少的一部分代码就可以实现想要的功能。

代码如下：

//方案二：把激活码存入的Redis中(最佳)

//Redis代替数据库保存UserName和激活码的字典结构

using **(**var client **=** RedisManager**.**ClientManager**.**GetClient**())**

**{**

client**.**Set**<**string**>(**ACTIVECODE\_PREFIX **+** username**,** activeCode**,** DateTime**.**Now**.**AddMinutes**(**30**));**

**}**

如果到这里真的就OK了吗？大家可以想想为什么我要添加下面的这段代码：

//把注册用户信息，放入消息队列。便于另外一个程序来获取消息队列数据，发送邮件

using (var client = RedisManager.ClientManager.GetClient())

{

string info = username + "|" + email;

client.EnqueueItemOnList("NewRegUsers", info);

}

 今天看看，觉得这篇博客还是有好多没有写的，后续我会重新写这篇博客，整理下今年所有项目中使用redis的案例和自己后续学习过程的总结。