**简介**

ABP提供了后台作业和工作者，用来在后台线程里执行应用里的某些任务。

**后台作业**

后台作业用一种队列且持久稳固的方式安排一些待执行后台任务，你可能有几个理由，需要用到后台作业，例如：

* 为执行长时间运行的任务而用户无需等待，例如：用户按了一下“报告”按钮开始一个长时间运行的报告任务，你把这个任务添加到队列里，它任务完成后，通过电子邮件发送报告结果给你。
* 为创建可重试且持久稳固的任务来保证一个代码将会被完成运行，例如：你可以在后台作业里发送一个电子邮件，为克服临时的失败且保证它最终将会被发送，当然当发送电子邮件时用户也无需等待。

**创建一个后台作业**

我们可以通过继承BackgroundJob<TArgs>类或直接实现IBackgroundJob<TArgs>接口来创建一个后台作业。

下列为一个非常简单的后台作业：

public class TestJob : **BackgroundJob<int>, ITransientDependency**

{

public override void **Execute**(int number)

{

Logger.Debug(number.ToString());

}

}

一个后台作业定义了一个Execute方法，接受一个输入参数，参数类型就是定义泛型类的参数，如上例所示。

一个后台作业必须注册到[依赖注入](http://www.cnblogs.com/kid1412/p/5980068.html" \t "http://www.cnblogs.com/kid1412/p/_blank)，实现ITransientDependency是最简单的方式。

让我们定义一个更真实的作业：在后台队列里发送电子邮件：

public class SimpleSendEmailJob : **BackgroundJob<SimpleSendEmailJobArgs>, ITransientDependency**

{

private readonly IRepository<User, long> \_userRepository;

private readonly IEmailSender \_emailSender;

public SimpleSendEmailJob(IRepository<User, long> userRepository, IEmailSender emailSender)

{

\_userRepository = userRepository;

\_emailSender = emailSender;

}

public override void **Execute**(SimpleSendEmailJobArgs args)

{

var senderUser = \_userRepository.Get(args.SenderUserId);

var targetUser = \_userRepository.Get(args.TargetUserId);

\_emailSender.Send(senderUser.EmailAddress, targetUser.EmailAddress, args.Subject, args.Body);

}

}

我们[注入](http://www.cnblogs.com/kid1412/p/5980068.html" \l "Constructor" \t "http://www.cnblogs.com/kid1412/p/_blank)user[仓储](http://www.cnblogs.com/kid1412/p/5994961.html" \t "http://www.cnblogs.com/kid1412/p/_blank)（可获取用户电子邮件）和邮件发送器（一个发送邮件的服务），并简单的发送邮件，SimpleSendEmailJobArgs是作业的参数，它的定义如下所示：

[Serializable]

public class SimpleSendEmailJobArgs

{

public long SenderUserId { get; set; }

public long TargetUserId { get; set; }

public string Subject { get; set; }

public string Body { get; set; }

}

一个作业的参数应当可序列化，因为它要被序列化后存储到数据库，虽然ABP默认后台作业管理器使用JSOn序列化器（它不需要使用[Serializable]特性），更好还是定义为[Serializable]，因为将来我们可能会替换成另一个作业管理器，可能会使用.net内置的二进制序列化器。

**在队列里添加一个新作业**

在定义完一个后台作业之后，我们可以注入并使用IBackgroundJobManager给队列添加一个作业，看一下使用上面已定义的TestJob的例子：

public class MyService

{

private readonly IBackgroundJobManager \_backgroundJobManager;

public MyService(I**BackgroundJobManager backgroundJobManager**)

{

\_backgroundJobManager = backgroundJobManager;

}

public void Test()

{

**\_backgroundJobManager.Enqueue<TestJob, int>(42);**

}

}

当加入队列时我们发送参数42， IBackgroundManager将实体化并以42为TestJob的参数执行它。

让我们看一下使用上面定义的SimpleSendEmailJob的例子：

[AbpAuthorize]public class MyEmailAppService : ApplicationService, IMyEmailAppService

{

private readonly IBackgroundJobManager \_backgroundJobManager;

public MyEmailAppService(IBackgroundJobManager backgroundJobManager)

{

\_backgroundJobManager = backgroundJobManager;

}

public async Task SendEmail(SendEmailInput input)

{

**await \_backgroundJobManager.EnqueueAsync<SimpleSendEmailJob, SimpleSendEmailJobArgs>(**

**new SimpleSendEmailJobArgs**

**{**

**Subject = input.Subject,**

**Body = input.Body,**

**SenderUserId = AbpSession.GetUserId(),**

**TargetUserId = input.TargetUserId**

**});**

}

}

**配置**

你可以在模块的PreInitialize方法里，使用Configuration.BackgroundJobs配置你的后台作业系统。

**禁用作业执行**

你可能会想为你的应用禁用后台作业执行：

public class MyProjectWebModule : AbpModule

{

public override void PreInitialize()

{

**Configuration.BackgroundJobs.IsJobExecutionEnabled = false;**

}

//...

}

**后台工作者**

后台工作者与后台作业不同，它简单的依赖应用在后台运行的线程，通常地，它定期执行一些任务，例如：

* 一个后台工作者可以定期删除旧日志。
* 一个后台工作者可以定期检测不活跃的用户，然后发邮件给他们，让他们重新使用你的应用。

**创建一个后台工作者**

为创建一个后台工作者，我们应当实现IBackgroundWorker接口，我们还可以选择直接从BackgroundWorkerBase或PeriodicBackgroundWorkerBase基类上继承。

假设我们想把超过30天未登录的用户设置为“消极”的，代码如下：

public class MakeInactiveUsersPassiveWorker : **PeriodicBackgroundWorkerBase, ISingletonDependency**

{

private readonly IRepository<User, long> \_userRepository;

public MakeInactiveUsersPassiveWorker(**AbpTimer timer**, IRepository<User, long> userRepository)

: **base(timer)**

{

\_userRepository = userRepository;

**Timer.Period = 5000;** //5 seconds (good for tests, but normally will be more) }

[UnitOfWork]

**protected override void DoWork()**

{

using (CurrentUnitOfWork.DisableFilter(AbpDataFilters.MayHaveTenant))

{

var oneMonthAgo = Clock.Now.Subtract(TimeSpan.FromDays(30));

var inactiveUsers = \_userRepository.GetAllList(u =>

u.IsActive &&

((u.LastLoginTime < oneMonthAgo && u.LastLoginTime != null) || (u.CreationTime < oneMonthAgo && u.LastLoginTime == null))

);

foreach (var inactiveUser in inactiveUsers)

{

inactiveUser.IsActive = false;

Logger.Info(inactiveUser + " made passive since he/she did not login in last 30 days.");

}

CurrentUnitOfWork.SaveChanges();

}

}

}

**注册后台工作者**

在完成创建后台工作者后，需要把它添加到IBackgroundWorkerManager，非常通用的地方是：你模块的PostInitialize方法里：

public class MyProjectWebModule : AbpModule

{

//...

public override void PostInitialize()

{

var workManager = IocManager.Resolve<IBackgroundWorkerManager>();

**workManager.Add(IocManager.Resolve<MakeInactiveUsersPassiveWorker>());**

}

}