**[.NET Tips](https://www.dntips.ir/)**

[**پیاده سازی RabbitMQ**](https://www.dntips.ir/post/3333/%d9%be%db%8c%d8%a7%d8%af%d9%87-%d8%b3%d8%a7%d8%b2%db%8c-rabbitmq)

اندازه‌ی قلم متن

نمایش کدهای طولانی

 تخمین مدت زمان مطالعه‌ی مطلب: هشت دقیقه

RabbitMq شبیه به یک صف FIFO عمل میکند؛ یعنی داده‌ها به ترتیب وارد queue میشوند و به ترتیب نیز به Consumer‌ها ارسال میشوند. برای شروع، یک سولوشن جدید را به نام RabbitMqExample ایجاد میکنیم و پروژه‌های زیر را به آن اضافه میکنیم.

* یک پروژه از نوع Asp.Net Core Web Application ایجاد میکنیم به نام RabbiMqExample.Producer که همان ارسال کننده (Producer) میباشد.
* یک پروژه از نوع Asp.Net Core Web Application به نام RabbitMqExample.Consumer برای دریافت کننده (Consumer).
* یک پروژه از نوع Class library .Net Core به نام RabbitMqExample.Common که شامل سرویس‌ها و مدل‌های مشترک بین Producer و Consumer میباشد.

ابتدا در لایه Common یک کلاس برای دریافت اطلاعات RabbitMq از appsettings.json ایجاد میکنیم.

public class RabbitMqConfiguration

{

public string HostName { get; set; }

public string Username { get; set; }

public string Password { get; set; }

}

سپس یک سرویس را برای برقراری ارتباط با RabbitMq ایجاد میکنیم

public interface IRabbitMqService

{

IConnection CreateChannel();

}

public class RabbitMqService : IRabbitMqService

{

private readonly RabbitMqConfiguration \_configuration;

public RabbitMqService(IOptions<RabbitMqConfiguration> options)

{

\_configuration = options.Value;

}

public IConnection CreateChannel()

{

ConnectionFactory connection = new ConnectionFactory()

{

UserName = \_configuration.Username,

Password = \_configuration.Password,

HostName = \_configuration.HostName

};

connection.DispatchConsumersAsync = true;

var channel = connection.CreateConnection();

return channel;

}

}

در متد CreateChannel، اطلاعات موردنیاز برای ارتباط با RabbitMq را مانند هاست، نام کاربری و کلمه عبور، وارد میکنیم که از appsettings.json خوانده شده‌اند. مقدار پیش‌فرض نام کاربری و کلمه عبور، guest میباشد.

 اگر بخواهید Consumer شما داده‌های queue‌ها را به صورت async دریافت کند، باید مقدار پراپرتی DispatchConsumersAsync مربوط به ConnectionFactory را برابر true کنید. مقدار پیشفرض آن false است.

در ادامه یک کلاس را برای رجیستر کردن سرویس‌ها ایجاد میکنیم؛ در لایه Common.

public static class StartupExtension

{

public static void AddCommonService(this IServiceCollection services, IConfiguration configuration)

{

services.Configure<RabbitMqConfiguration>(a => configuration.GetSection(nameof(RabbitMqConfiguration)).Bind(a));

services.AddSingleton<IRabbitMqService, RabbitMqService>();

}

}

پکیج‌های مورد نیاز این لایه :

<ItemGroup>

<PackageReference Include="Microsoft.Extensions.Configuration.Abstractions" Version="5.0.0" />

<PackageReference Include="Microsoft.Extensions.Configuration.Binder" Version="5.0.0" />

<PackageReference Include="Microsoft.Extensions.DependencyInjection" Version="5.0.0" />

<PackageReference Include="Microsoft.Extensions.DependencyInjection.Abstractions" Version="5.0.0" />

<PackageReference Include="Microsoft.Extensions.Options" Version="5.0.0" />

<PackageReference Include="RabbitMQ.Client" Version="6.2.1" />

</ItemGroup>

تا به اینجا موارد مربوط به لایه Common تمام شده؛ در ادامه باید یک Consumer و یک Producer را ایجاد کنیم.

در لایه Consumer برای دریافت داده‌ها از RabbitMq؛ یک سرویس را به نام ConsumerService ایجاد میکنیم:

[**Code**](https://www.dntips.ir/post/3333/%D9%BE%DB%8C%D8%A7%D8%AF%D9%87-%D8%B3%D8%A7%D8%B2%DB%8C-rabbitmq#codeItem5)

public interface IConsumerService

{

Task ReadMessgaes();

}

public class ConsumerService : IConsumerService, IDisposable

{

private readonly IModel \_model;

private readonly IConnection \_connection;

public ConsumerService(IRabbitMqService rabbitMqService)

{

\_connection = rabbitMqService.CreateChannel();

\_model = \_connection.CreateModel();

\_model.QueueDeclare(\_queueName, durable: true, exclusive: false, autoDelete: false);

\_model.ExchangeDeclare("UserExchange", ExchangeType.Fanout, durable: true, autoDelete: false);

\_model.QueueBind(\_queueName, "UserExchange", string.Empty);

}

const string \_queueName = "User";

public async Task ReadMessgaes()

{

var consumer = new AsyncEventingBasicConsumer(\_model);

consumer.Received += async (ch, ea) =>

{

var body = ea.Body.ToArray();

var text = System.Text.Encoding.UTF8.GetString(body);

Console.WriteLine(text);

await Task.CompletedTask;

\_model.BasicAck(ea.DeliveryTag, false);

};

\_model.BasicConsume(\_queueName, false, consumer);

await Task.CompletedTask;

}

public void Dispose()

{

if (\_model.IsOpen)

\_model.Close();

if (\_connection.IsOpen)

\_connection.Close();

}

}

ابتدا connection را ایجاد کرده‌ایم؛ توسط متد CreateChannel که آنرا در سرویس قبلی پیاده سازی کردیم.

بعد از ایجاد IModel، باید queue مربوطه را معرفی کنیم که با استفاده از متد QueueDeclare این کار را انجام داده‌ایم.

پارامترهای متد QueueDeclare:

* پارامتر اول، اسم queue میباشد
* پارامتر durable مشخص میکند که داده‌ها به صورت مانا باشند یا نه. اگر برابر true باشد، دیتاهای مربوط به queue‌ها، در دیسک ذخیره میشوند؛ اما اگر برابر false باشد، بر روی حافظه ذخیره میشوند. در محیط‌هایی که مانایی اطلاعات مهم میباشد، باید مقدار این پارامتر را true کنید.
* پارامتر سوم: [اطلاعات بیشتر](https://www.rabbitmq.com/amqp-0-9-1-reference.html" \l "queue.declare.exclusive" \t "_blank)
* پارامتر autoDelete اگر برابر true باشد، زمانی که تمامی Consumer‌ها ارتباطشان با RabbitMq قطع شود، queue هم پاک میشود. اما اگر برابر true باشد، queue باقی میماند؛ حتی اگر هیچ Consumer ای به آن وصل نباشد.

در ادامه باید Exchange مربوط به queue را مشخص کنیم. متد ExchangeDeclare یک Exchange را ایجاد میکند. پارامتر‌های متد ExchangeDeclare:

* نام Exchange
* نوع Exchange که میتواند Headers , Topic ,  Fanout یا Direct باشد. اگر برابر Fanout باشد و اگر داده‌ای وارد Exchange شود، آن‌را به تمامی queue هایی که به آن بایند شده‌است، ارسال میکند. اما اگر نوع آن Direct باشد، داده را به یک queue مشخص ارسال میکند؛ با استفاده از پارامتر routeKey.
* پارامتر‌های بعدی، durable و autoDelete هستند که همانند پارامترهای QueueDeclare عمل میکنند.

سپس در ادامه با استفاده از متد QueueBind میتوانیم queue ایجاد شده را به exchange ایجاد شده، بایند کنیم. پارامتر اول، اسم queue و پارامتر دوم، اسم exchange میباشد و پارامتر سوم، routeKey میباشد و چون نوع Exchange ایجاد شده از نوع Fanout است، آنرا خالی میگذاریم.

چون هنگام تعریف queue مقدار پارامتر DispatchConsumersAsync مربوط به ConnectionFactory را برابر true کردیم، در اینجا نیز باید بجای EventingBasicConsumer، از AsyncEventingBasicConsumer استفاده کنیم. اگر مقدار DispatchConsumersAsync برابر false باشد، باید از EventingBasicConsumer برای ایجاد Consumer استفاده کنید.

سپس باید EventHandler مربوط به دریافت داده‌ها از queue را پیاده سازی کنیم. event مربوط به Received، زمانی اجرا میشود که داده‌ای به queue ارسال شود. زمانیکه داده‌ای ارسال میشود، وارد event مربوطه میشود و ابتدا آنرا به صورت byte دریافت میکنیم. سپس رشته‌ی ارسالی آن‌را توسط متد GetString، بدست می‌آوریم و داده‌ی ارسال شده را در صفحه‌ی کنسول نمایش میدهیم.

 در ادامه به RabbitMq اطلاع میدهیم که داده‌ای که ارسال شده برای queue، توسط Consumer دریافت شده؛ با استفاده از متد BasicAck. این کار یک delivery  به RabbitMq ارسال میکند تا دیتای ارسال شده را را پاک کند. اگر این متد را فراخوانی نکنیم، هربار که برنامه اجرا میشود، تمامی دیتاهای قبلی را مجددا دریافت میکنیم و تا زمانیکه delivery را به RabbitMq نفرستیم، داده‌ها را پاک نمیکند.

نکته آخر در Consumer، متد BasicConsume است که عملا Consumer ایجاد شده را به RabbitMq معرفی میکند. برای دریافت داده‌ها و ثبت Consumer، نیازمند آن هستیم تا یکبار متد ReadMessage فراخوانی شود. برای همین یک HostedService ایجاد میکنیم تا یکبار این متد را فراخوانی کند:

public class ConsumerHostedService : BackgroundService

{

private readonly IConsumerService \_consumerService;

public ConsumerHostedService(IConsumerService consumerService)

{

\_consumerService = consumerService;

}

protected override async Task ExecuteAsync(CancellationToken stoppingToken)

{

await \_consumerService.ReadMessgaes();

}

}

در نهایت سرویس‌های ایجاد شده را رجیستر میکنیم؛ در Startup لایه Consumer

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; set; }

public void ConfigureServices(IServiceCollection services)

{

services.AddCommonService(Configuration);

services.AddSingleton<IConsumerService, ConsumerService>();

services.AddHostedService<ConsumerHostedService>();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

}

}

تا به اینجا کارهای مربوط به Consumer تمام شده و باید قسمت Producer آنرا پیاده سازی کنیم.

در لایه Producer یک کنترلر به نام RabbitController را ایجاد میکنیم که شامل یک متد میباشد که داده‌ها را به Queue ارسال میکند:

[Route("api/[controller]/[action]")]

[ApiController]

public class RabbitController : ControllerBase

{

private readonly IRabbitMqService \_rabbitMqService;

public HomeController(IRabbitMqService rabbitMqService)

{

\_rabbitMqService = rabbitMqService;

}

[HttpPost]

public IActionResult SendMessage()

{

using var connection = \_rabbitMqService.CreateChannel();

using var model = connection.CreateModel();

var body = Encoding.UTF8.GetBytes("Hi");

model.BasicPublish("UserExchange",

string.Empty,

basicProperties: null,

body: body);

return Ok();

}

}

در متد SendMessage، ابتدا ارتباط خود را با RabbitMq برقرار میکنیم و سپس دیتای "Hi" را به صورت byte، به RabbitMq ارسال میکنیم؛ توسط متد BasicPublish.

پارامتر اول، اسم Exchange است و پارامتر دوم، routeKey و body هم دیتای ارسالی میباشد. در نهایت سرویس‌های مربوط به لایه Producer را رجیستر میکنیم؛ در Startup لایه Consumer:

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; set; }

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

services.AddCommonService(Configuration);

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapDefaultControllerRoute();

});

}

}

اکنون اگر هر دو پروژه را اجرا کنید و متد SendMessage مربوط به کنترلر Rabbit را فراخوانی کنید، بعد از آنکه پیام شما ارسال شد، در صفحه کنسول مربوط به Consumer، رشته ارسال شده را مشاهده میکنید.

فایل appsetting.json مربوط به پروژه‌های Consumer و Producer:

{

"RabbitMqConfiguration": {

"HostName": "localhost",

"Username": "guest",

"Password": "guest"

}

}

فایل docker-compose.yml برای اجرای RabbitMq بر روی داکر:

version: "3.2"

services:

rabbitmq:

image: rabbitmq:3-management-alpine

container\_name: 'rabbitmq'

ports:

- 5672:5672

- 15672:15672

کدهای این مقاله را میتوانید از [گیت‌هاب](https://github.com/farhadzm/RabbitMq" \t "_blank)دانلود کنید.

 توسط  [زمانی فرهاد](https://www.dntips.ir/user/%d8%b2%d9%85%d8%a7%d9%86%db%8c%20%d9%81%d8%b1%d9%87%d8%a7%d8%af)  در

 گروه(ها): [ASP.NET Core](https://www.dntips.ir/search/label/asp.net%20core) [Microservices](https://www.dntips.ir/search/label/microservices) [RabbitMQ](https://www.dntips.ir/search/label/rabbitmq)

 امتیاز ۵,۰۰ از ۵ توسط ۲۱ نفر info

پیاده سازی RabbitMQ

Rated 5.00/5 based on 21 readers reviews

زمانی فرهاد more items from: [مطالب ارسالی زمانی فرهاد](https://www.dntips.ir/writer/%d8%b2%d9%85%d8%a7%d9%86%db%8c%20%d9%81%d8%b1%d9%87%d8%a7%d8%af)

**مطالب مرتبط**

* [سیستم‌های توزیع شده در NET. - بخش هفتم- معرفی Apache Kafka](https://www.dntips.ir/post/2822)
* [سیستم‌های توزیع شده در NET. - بخش هشتم- راه اندازی Apache Kafka](https://www.dntips.ir/post/2823)
* [فراخوانی GraphQL API در یک کلاینت ASP.NET Core](https://www.dntips.ir/post/3084)
* [آپلود فایل‌ها در یک برنامه‌ی Angular به کمک کامپوننت ng2-file-upload](https://www.dntips.ir/post/2705)
* [عمومی سازی الگوریتم‌ها با استفاده از Reflection](https://www.dntips.ir/post/2082)
* [سیستم‌های توزیع شده در NET. - بخش ششم- Message Broker](https://www.dntips.ir/post/2614)

[→ روش آپلود فایل‌ها به همراه اطلاعات یک مدل در برنامه‌های Blazor WASM 5x](https://www.dntips.ir/post/3334/%d8%b1%d9%88%d8%b4-%d8%a2%d9%be%d9%84%d9%88%d8%af-%d9%81%d8%a7%db%8c%d9%84%e2%80%8c%d9%87%d8%a7-%d8%a8%d9%87-%d9%87%d9%85%d8%b1%d8%a7%d9%87-%d8%a7%d8%b7%d9%84%d8%a7%d8%b9%d8%a7%d8%aa-%db%8c%da%a9-%d9%85%d8%af%d9%84-%d8%af%d8%b1-%d8%a8%d8%b1%d9%86%d8%a7%d9%85%d9%87%e2%80%8c%d9%87%d8%a7%db%8c-blazor-wasm-5x)[کاربرد Action ها در Github - خودکار سازی فرآیند کامپایل و آپلود فایل در Release گیت‌هاب ←](https://www.dntips.ir/post/3332/%da%a9%d8%a7%d8%b1%d8%a8%d8%b1%d8%af-action-%d9%87%d8%a7-%d8%af%d8%b1-github-%d8%ae%d9%88%d8%af%da%a9%d8%a7%d8%b1-%d8%b3%d8%a7%d8%b2%db%8c-%d9%81%d8%b1%d8%a2%db%8c%d9%86%d8%af-%da%a9%d8%a7%d9%85%d9%be%d8%a7%db%8c%d9%84-%d9%88-%d8%a2%d9%be%d9%84%d9%88%d8%af-%d9%81%d8%a7%db%8c%d9%84-%d8%af%d8%b1-release-%da%af%db%8c%d8%aa%e2%80%8c%d9%87%d8%a7%d8%a8)

**[#](https://www.dntips.ir/post/3333/%D9%BE%DB%8C%D8%A7%D8%AF%D9%87-%D8%B3%D8%A7%D8%B2%DB%8C-rabbitmq" \l "comments) نظرات خوانندگان**

* [#](https://www.dntips.ir/post/3333/%D9%BE%DB%8C%D8%A7%D8%AF%D9%87-%D8%B3%D8%A7%D8%B2%DB%8C-rabbitmq" \l "comment-18509)   [م شاکری](https://www.dntips.ir/user/%d9%85%20%d8%b4%d8%a7%da%a9%d8%b1%db%8c)  در

 امتیاز ۵,۰۰ از ۵ توسط ۲ نفر info

**یک نکته تکمیلی:**

تعیین مقدار durable برابر با true ساعت می‌شود که تعریف صف بصورت مانا باشد نه دیتای داخل صف.

برای اینکه دیتای داخل صف نیز بصورت مانا باشد باید مقدار ویژگی DevliveryMode اینترفیس IBasicProperties را برابر 2 قرار داد:

using var connection = \_rabbitMqService.CreateChannel();

using var model = connection.CreateModel();

IBasicProperties props = model.CreateBasicProperties();

props.DeliveryMode = 2;

var body = Encoding.UTF8.GetBytes("Hi");

model.BasicPublish("UserExchange",

string.Empty,

basicProperties: props,

body: body);

* [#](https://www.dntips.ir/post/3333/%D9%BE%DB%8C%D8%A7%D8%AF%D9%87-%D8%B3%D8%A7%D8%B2%DB%8C-rabbitmq" \l "comment-18510)   [زمانی فرهاد](https://www.dntips.ir/user/%d8%b2%d9%85%d8%a7%d9%86%db%8c%20%d9%81%d8%b1%d9%87%d8%a7%d8%af)  در

 امتیاز ۵,۰۰ از ۵ توسط ۱ نفر info

**نکته تکمیلی :**

میتوانید با استفاده از متد BasicQos حداکثر تعداد داده‌های ارسالی به Consumer هارا مشخص کنید.

 برای مثال اگر از طرف Consumer متد BasicAck رو صدا نزنید و به RabbitMq اعلام نکنید که داده به Consumer رسیده است, داده‌ها در RabbitMq باقی خواهند ماند.( در این حالت هر داده جدیدی در RabbitMq وارد شود به Consumer ارسال میشود اما هیچ داده ای پاک نمیشود).

 اما میتوانیم با استفاده از متد BasicQos تعداد داده‌های دریافتی را مشخص کنیم.

برای مثال در کد زیر حداکثر 16 داده به Consumer ارسال میشود. یعنی اگر Consumer که  16 داده را دریافت کرده, ack را به RabbitMq ارسال نکنند, هیچ داده دیگری به Consumer ارسال نمیشود تا زمانی ack مربوط به داده‌های قبلی به RabbitMq ارسال شود.

\_model.BasicQos(0, 16, true);

این متد را میتوانید بعد از متد QueueBind در هنگام تعریف Consumer مربوط به Queue اضافه کنید.

* [#](https://www.dntips.ir/post/3333/%D9%BE%DB%8C%D8%A7%D8%AF%D9%87-%D8%B3%D8%A7%D8%B2%DB%8C-rabbitmq" \l "comment-18513)   [اختیاری](https://www.dntips.ir/user/%d8%a7%d8%ae%d8%aa%db%8c%d8%a7%d8%b1%db%8c)  در

با تشکر از مطالب بسیار خوبتون.

در صورتی که نیاز باشه از IIS یه عنوان Revers Proxy استفاده کنیم راهکار به چه صورت هست ؟ ( سرور مربوطه ویندوزی می‌باشد و پشت پراکسی کلودفلیر قرار گرفته است )

* + [#](https://www.dntips.ir/post/3333/%D9%BE%DB%8C%D8%A7%D8%AF%D9%87-%D8%B3%D8%A7%D8%B2%DB%8C-rabbitmq" \l "comment-18514)   [زمانی فرهاد](https://www.dntips.ir/user/%d8%b2%d9%85%d8%a7%d9%86%db%8c%20%d9%81%d8%b1%d9%87%d8%a7%d8%af)  در

خواهش میکنم.  
مطلب درباره RabbitMq بود و در مورد Reverse proxy روی IIS  اطلاعی ندارم.

© .NET Tips, ۱۳۸۷-۱۴۰۱