## **概述**

本教程将创建以下 API：

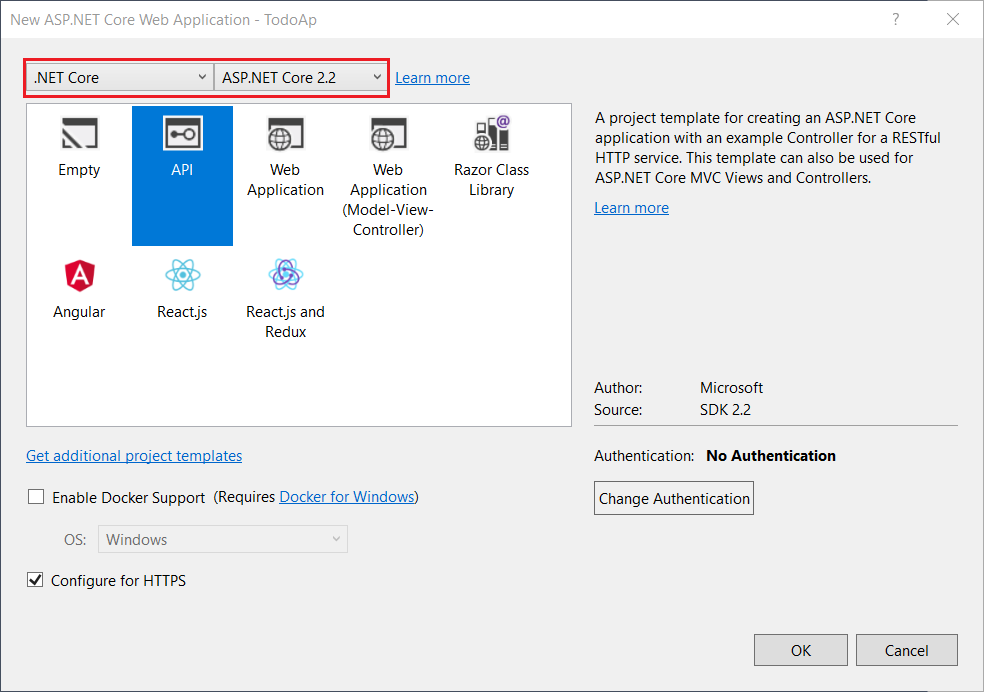
| **API** | **说明** | **请求正文** | **响应正文** |
| --- | --- | --- | --- |
| GET /api/todo | 获取所有待办事项 | 无 | 待办事项的数组 |
| GET /api/todo/{id} | 按 ID 获取项 | 无 | 待办事项 |
| POST /api/todo | 添加新项 | 待办事项 | 待办事项 |
| PUT /api/todo/{id} | 更新现有项 | 待办事项 | 无 |
| DELETE /api/todo/{id} | 删除项 | 无 | 无 |

下图显示了应用的设计。



## **创建 Web 项目**

* 从“文件”菜单中选择“新建” > “项目”。
* 选择“ASP.NET Core Web 应用程序”模板。 将项目命名为 TodoApi，然后单击“确定”。
* 在“新建 ASP.NET Core Web 应用程序 - TodoApi”对话框中，选择 ASP.NET Core 版本。 选择“API”模板，然后单击“确定”。 请不要选择“启用 Docker 支持”。



## **添加模型类**

namespace TodoApi.Models

{

public class TodoItem

{

public long Id { get; set; }

public string Name { get; set; }

public bool IsComplete { get; set; }

}

}

## **添加数据库上下文**

using Microsoft.EntityFrameworkCore;

namespace TodoApi.Models

{

public class TodoContext : DbContext

{

public TodoContext(DbContextOptions<TodoContext> options)

: base(options)

{

}

public DbSet<TodoItem> TodoItems { get; set; }

}

}

## **注册数据库上下文**

// Unused usings removed

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using TodoApi.Models;

namespace TodoApi

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<TodoContext>(opt => opt.UseInMemoryDatabase("TodoList"));

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_2);

}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

// The default HSTS value is 30 days. You may want to change this for

// production scenarios, see https://aka.ms/aspnetcore-hsts.

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseMvc();

}

}

}

## **添加控制器**

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using TodoApi.Models;

namespace TodoApi.Controllers

{

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

[ApiController]

public class TodoController : ControllerBase

{

private readonly TodoContext \_context;

public TodoController(TodoContext context)

{

\_context = context;

if (\_context.TodoItems.Count() == 0)

{

\_context.TodoItems.Add(new TodoItem { Name = "Item1" });

\_context.SaveChanges();

}

}

}

}

## **添加 Get 方法**

若要提供检索待办事项的 API，请将以下方法添加到 TodoController 类中：

// GET: api/Todo

[HttpGet]

public async Task<ActionResult<IEnumerable<TodoItem>>> GetTodoItems()

{

return await \_context.TodoItems.ToListAsync();

}

// GET: api/Todo/5

[HttpGet("{id}")]

public async Task<ActionResult<TodoItem>> GetTodoItem(long id)

{

var todoItem = await \_context.TodoItems.FindAsync(id);

if (todoItem == null)

{

return NotFound();

}

return todoItem;

}

这些方法实现两个 GET 终结点：

* GET /api/todo
* GET /api/todo/{id}

## **路由和 URL 路径**

[[HttpGet]](https://docs.microsoft.com/zh-cn/dotnet/api/microsoft.aspnetcore.mvc.httpgetattribute) 属性表示响应 HTTP GET 请求的方法。 每个方法的 URL 路径构造如下所示：

namespace TodoApi.Controllers

{

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

[ApiController]

public class TodoController : ControllerBase

{

private readonly TodoContext \_context;

将 [controller] 替换为控制器的名称

如果 [HttpGet] 属性具有路由模板（例如 [HttpGet("/products")]），则将它追加到路径。

在下面的 GetTodoItem 方法中，"{id}" 是待办事项的唯一标识符的占位符变量。 调用 GetTodoItem 时，URL 中 "{id}" 的值会在 id 参数中提供给方法。

// GET: api/Todo/5

[HttpGet("{id}")]

public async Task<ActionResult<TodoItem>> GetTodoItem(long id)

{

var todoItem = await \_context.TodoItems.FindAsync(id);

if (todoItem == null)

{

return NotFound();

}

return todoItem;

}

## **返回值**

GetTodoItems 和 GetTodoItem 方法的返回类型是 [ActionResult<T> 类型](https://docs.microsoft.com/zh-cn/aspnet/core/web-api/action-return-types?view=aspnetcore-2.2" \l "actionresultt-type)。 ASP.NET Core 自动将对象序列化为 [JSON](https://www.json.org/)，并将 JSON 写入响应消息的正文中。

ActionResult 返回类型可以表示大范围的 HTTP 状态代码。 例如，GetTodoItem 可以返回两个不同的状态值：

* 如果没有任何项与请求的 ID 匹配，则该方法将返回 404 [NotFound](https://docs.microsoft.com/zh-cn/dotnet/api/microsoft.aspnetcore.mvc.controllerbase.notfound) 错误代码。
* 否则，此方法将返回具有 JSON 响应正文的 200。 返回 item 则产生 HTTP 200 响应。

## **添加创建方法**

添加以下 PostTodoItem 方法：

// POST: api/Todo

[HttpPost]

public async Task<ActionResult<TodoItem>> PostTodoItem(TodoItem todoItem)

{

\_context.TodoItems.Add(todoItem);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetTodoItem", new { id = todoItem.Id }, todoItem);

}

CreatedAtAction 方法：

返回 201 响应。 HTTP 201 是在服务器上创建新资源的 HTTP POST 方法的标准响应。

向响应添加位置标头。 位置标头指定新建的待办事项的 URI。 有关详细信息，请参阅[创建的 10.2.2 201](https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html)。

使用名为 route 的“GetTodo”来创建 URL。 已在 GetTodoItem 中定义名为 route 的“GetTodo”：

// GET: api/Todo/5

[HttpGet("{id}")]

public async Task<ActionResult<TodoItem>> GetTodoItem(long id)

{

var todoItem = await \_context.TodoItems.FindAsync(id);

if (todoItem == null)

{

return NotFound();

}

return todoItem;

}

### **测试 PostTodoItem 方法**

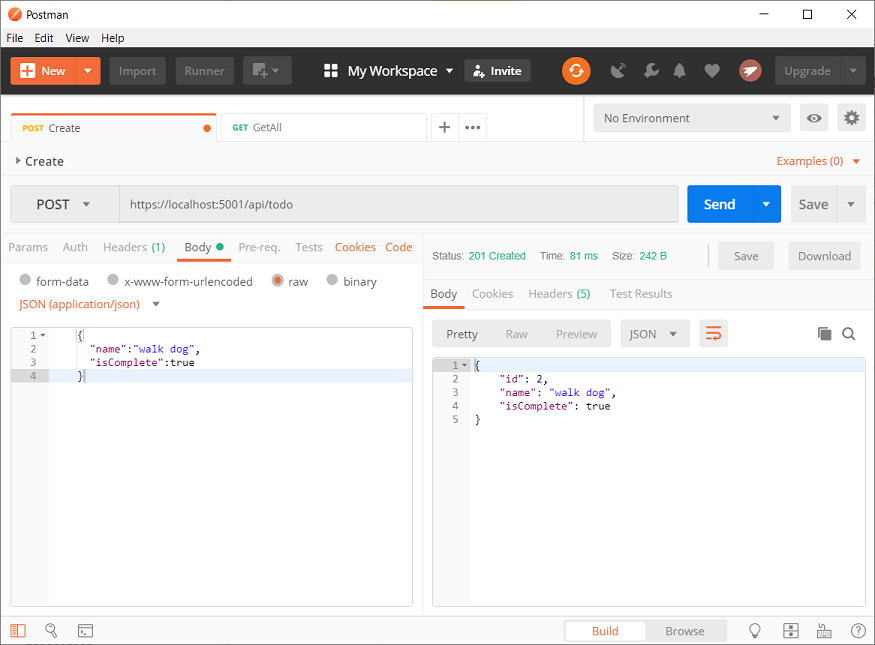
{

"name":"walk dog",

"isComplete":true

}

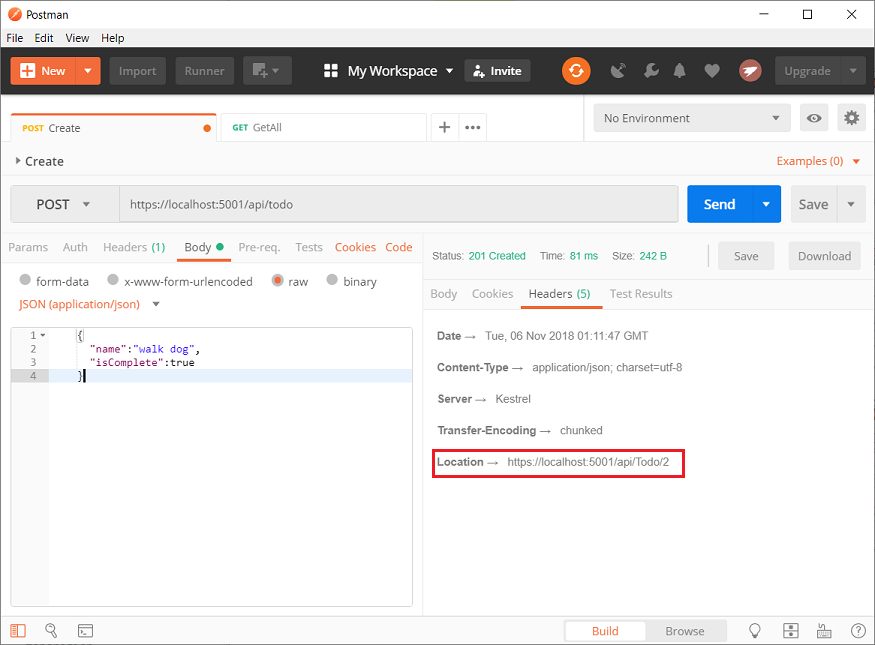
选择“发送”。



### **测试位置标头 URI**

在“响应”窗格中选择“标头”选项卡。

复制“位置”标头值：



将方法设置为“GET”。

粘贴 URI（例如，https://localhost:5001/api/Todo/2）

选择“发送”。

## **添加 PutTodoItem 方法**

添加以下 PutTodoItem 方法：

// PUT: api/Todo/5

[HttpPut("{id}")]

public async Task<IActionResult> PutTodoItem(long id, TodoItem todoItem)

{

if (id != todoItem.Id)

{

return BadRequest();

}

\_context.Entry(todoItem).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return NoContent();

}

PutTodoItem 与 PostTodoItem 类似，但是使用的是 HTTP PUT。 响应是 [204（无内容）](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html)。 根据 HTTP 规范，PUT 请求需要客户端发送整个更新的实体，而不仅仅是更改。 若要支持部分更新，请使用 [HTTP PATCH](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.mvc.httppatchattribute)。

## **添加 DeleteTodoItem 方法**

添加以下 DeleteTodoItem 方法：

// DELETE: api/Todo/5

[HttpDelete("{id}")]

public async Task<ActionResult<TodoItem>> DeleteTodoItem(long id)

{

var todoItem = await \_context.TodoItems.FindAsync(id);

if (todoItem == null)

{

return NotFound();

}

\_context.TodoItems.Remove(todoItem);

await \_context.SaveChangesAsync();

return todoItem;

}

DeleteTodoItem 响应是 [204（无内容）](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html)。