Aula 6

Programação I

Prof. Alan Matheus Pinheiro Araya

1

Conversa Inicial

2

Interações com Web APIs

- Consumindo serviços Web
  - SOAP
  - Rest
- O padrão JSON

3

Introdução a Web *services* e Web APIs

4

O conceito de Web *services* e Web APIs

- Web *services* e Web APIs
  - API (*application programming interface*)
  - Web *services*
    - SOAP
    - Rest
    - XML-RPC

5

- O termo *Web services* não é novo começou a ser utilizado pelos primeiros serviços *web*
  - Iniciando com o SOAP (*simple object access protocol*) baseado em XML
- O termo *Web API* começou a ser empregado quando os serviços *web* passaram a usar o protocolo Rest
  - *Rest* vem de *representational state transfer*
  - Baseado principalmente na combinação de verbos com o estado dos objetos

6

- É comum vermos o termo *Web services* associado a sistemas mais antigos, baseados em SOAP (XML)
- Enquanto *Web API* é um termo que está associado a sistemas que utilizam Rest
  - Verbos HTTP são largamente utilizados para representar uma ação (*get*, *post*, *delete*...)

7

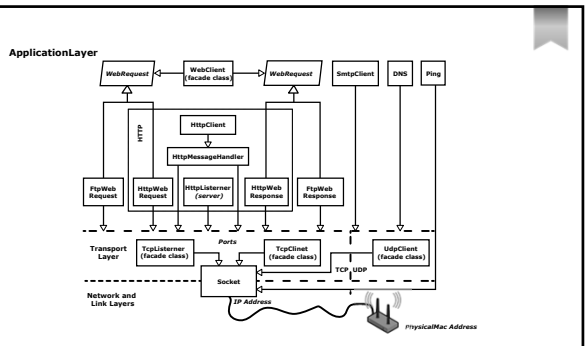
Arquitetura de rede do .NET

- A base da maior parte da comunicação Web são serviços baseados no protocolo HTTP
- O .NET oferece uma série de APIs internas para se trabalhar com a camada de transporte de aplicação

8

- A camada de transporte:
  - TCP e UDP
- A camada de aplicação:
  - HTTP
  - FTP
  - SMTP
  - DNS

9



10

Trabalhando com Web *services* SOAP

11

Web *services* SOAP

- O protocolo SOAP define uma série de metadados que "explicam" os dados de entrada e saída (*request*/*respose*)
- Essas definições ficam dentro de um arquivo conhecido como WSDL (*Web services description language*)
- No início dos Web Services, era comum desenvolvedores ficarem inseguros quanto aos dados trafegados
  - Problemas de formato e conversão dos dados (*encoding*)

12

- **Vamos criar um Web *service* SOAP juntos?**

13

Executando o projeto

- **O projeto abre em localhost**
- **Se acrescentarmos /WebServiceSoapAula.asmx no endereço, o *browser* exibirá uma tela como está ao lado**

14

- **A partir de então, podemos testar nossos métodos diretamente pela interface gerada pelo .NET**
- **O servidor Web (IIS Express) faz a leitura do WSDL e gera essa tela e "atalhos" para os métodos de nosso Web *service***
  - **Útil para testar nosso Web *service***

15

Consumindo um Web *service* SOAP

- **Para consumir um Web Service SOAP, devemos gerar um "client" a partir do WSDL (Web Service Description Language)**

Para visualizar o WSDL, basta acrescentar: "?wsdl" ao final de nossa rota do browser

16

- **Vamos consumir nosso Web *service* em um cliente Console App?**

17

Trabalhando com Web APIs

18

O conceito de Web APIs

- **WebAPIs são serviços Web assim como os Web *services***
  - **São menos restritivos**
  - **Utilizam o protocolo HTTP com seus verbos apenas**
- **Podem receber e retornar dados em vários formatos**
  - **JSON / XML / "string pura" (*raw content*)**

19

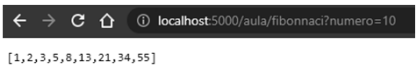
- **O que caracteriza uma Web API é sua capacidade de operar utilizando apenas o HTTP e seus verbos**
- **Não precisa de uma série descritores das mensagens e métodos, como o WSDL do SOAP**
  - **Em compensação necessita de uma boa documentação (para outros desenvolvedores)**

20

- **Vamos construir uma Web API em C#?**

21

- **Para chamar um método da Web API, precisamos completar a url base com o nome do controller/recurso e com a rota do método que desejamos. Depois com os parâmetros, dessa forma:**
- **http://localhost:5000/aula/fibonnaci?numero=10**



22

Consumindo uma Web API

- **WebAPIs preferencialmente devem seguir o padrão OAS (*open API specification*)**
  - **Mas isso não é uma regra que impeça uma API de funcionar**
- **Seguir um padrão facilita o consumo de sua API por outros sistemas**

23

- **Para consumir uma API no C#, precisamos utilizar o HttpClient**
  - **Já o utilizamos em anteriormente**
  - **Abstrair a camada de transporte (TCP/UDP)**

24

- Vamos consumir nossa Web API?

25

Serialização de dados

26

O conceito de *serialização*

- A serialização é o ato de transformar um objeto ou uma relação de objetos da memória e "achatá-los" em um fluxo de *bytes* (*stream*) de tal forma que possam ser armazenados ou transmitidos para um destino
- A "desserialização" funciona ao contrário, obtendo um fluxo de dados e reconstruindo-o em um objeto na memória (Albahari, 2020, p. 743)

27

- A serialização e a "desserialização" são normalmente usadas para:
  - Transmitir objetos por meio da rede ou entre aplicativos
  - Armazenar representações de objetos em um arquivo ou banco de dados

28

- Existem quatro mecanismos "nativos" de serialização no .NET (alguns deles podem ser entregues via pacotes à parte):
  - Data Contract Serializer
  - Binary Serializer
  - XML Serializer
  - JSON Serializer

29

- Serialização e deserialização fazem parte de toda transferência de dados pela rede
  - Não é uma exclusividade do .NET
- Ao transferir um objeto de uma máquina para outra via rede, precisamos:
  - Converter um objeto de memória para outra representação para que o destinatário possa lê-lo e "reconstruí-lo" novamente

30

- **A serialização nunca propaga comportamentos (métodos)**
  - **Apenas estado, ou seja, dados (Albahari, 2020)**
- **Isso significa que qualquer tipo de serialização não poderá propagar "lógica de negócio"**

31

Serialização para JSON

- **A raiz de um documento JSON é sempre um array ou um objeto**
  - **Sob essa raiz estão as propriedades, que podem ser: objetos, arrays, strings, números, "verdadeiro", "falso" ou "nulo"**

```
public class Autor
{
    public int Codigo { get; set; }
    public string Nome { get; set; }
    public string Editora { get; set; }
    public DateTime DataNascimento { get; set; }
}
```

```
{
  "codigo": 2,
  "nome": "LUCAS",
  "editora": "Intersaberes",
  "dataNascimento": "1962-06-25T00:00:00"
}
```

32

- **Para serializar um objeto, basta uma simples instrução para o JsonSerializer e assim ele transforma todos os dados de seu objeto e objetos relacionados em uma *string* JSON**
- **O serializador JSON mapeia diretamente:**

Nomes de propriedade da classe ⇨ **Nomes de propriedade em JSON**

33

- **Vamos ver um exemplo prático de serialização com JSON?**

34

Tolerância a mudanças

- **JSON é um formato muito tolerante a mudanças no modelo**
  - **Pode-se serializar um objeto com mais ou menos dados (propriedades) na origem, sem necessariamente "quebrar" o modelo no destino**
  - **A ordem dos campos não é importante**
- **O formato e *encoding* são MUITO importantes**

35

Pacote Refit

36

O Refit

- O Refit é um pacote Nuget, *open-source*, desenvolvido por terceiros, inspirado no pacote Retrofit para Android, que cria uma abstração de serviços Web Restfull em C#
- Similar ao que o "client" do serviço SOAP faz, conseguindo ser ainda mais simples e leve

37

Usando o Refit

- Interface do C# com os métodos que representem os *endpoints* da API Rest que será consumida
- Os verbos HTTP e rotas viram "Atributos" C#

```
interface IWebApiAulaRefit
{
   [Get("/fibonnaci")]
   public Task<List<int>> CalculaFibonnacciAsync(int numero);

   [Get("/autores")]
   public Task<IEnumerable<Autor>> GetAutoresAsync([AliasAs("editoras")] string editoras
SeparadoPorVirgula);
}
```

38

- Vamos ver um exemplo prático de utilização do Refit?

39

- O Refit suporta a deserialização para dois tipos especiais:
  - *String* – qualquer método HTTP pode ter o conteúdo de seu retorno deserializado para *string*
    - Isso por que o "body" do HTTP suporta essa funcionalidade
    - Independente de trafegar binário, XML, texto livre ou JSON, o Refit poderá "ler o conteúdo" do "body" como *string* sem conversão adicional

40

- HttpResponseMessage – este é o retorno-padrão do HttpClient
  - Toda chamada HTTP retornará esse objeto (internamente no Refit)
  - Basta passá-lo dentro do Type (<T>) esperado em sua interface

41