BCSCHOOL

Empowering Digital Skills For The Jobs Of The Future

by

BC Soft
consulenza sistemi informativi

AiRED
Artificial Intelligence Ricerca e Sviluppo

Academy .NET

Classificazione Web Services

Sommario

- Servizi Web SOAP
- Servizi Web REST
- Request Methods
- Codici Stato HTTP
- ASP.NET Core
- I controller

Servizi Web SOAP

Il protocollo **SOAP** (**Simple Object Access Protocol**) consente sia messaggi unidirezionali sia messaggi di chiamata/risposta.

La comunicazione può essere sia **sincrona** che **asincrona.**

Se il protocollo sottostante è sincrono, come nel caso di HTTP, il mittente riceve un riconoscimento:

- avvisa che il messaggio è stato ricevuto (ma non necessariamente elaborato).

Servizi Web SOAP

- Quando viene utilizzata la comunicazione asincrona il mittente deve essere in ascolto di comunicazioni in arrivo.

- Spesso la comunicazione asincrona viene implementata con modello **subscriber/publisher**

- I messaggi sono rappresentati come **documenti XML** chiamati **envelopes**

Servizi Web SOAP

Ogni **envelope** contiene:

header, body, Elemento fault

- **header**: contiene informazioni ausiliarie(token, firma...)
- **body**: contiene il messaggio
- **fault**: contiene possibili errori

Servizi Web SOAP

- Il protocollo utilizzato per inviare le envelopes XML è solitamente HTTP

- La specifica **SOAP** consente qualsiasi protocollo, quindi è possibile utilizzare direttamente TCP/IP o SMTP.

Servizi Web SOAP

Negli ultimi anni, la difficoltà nel gestire le modifiche, insieme alla complessità del gestire la configurazione di tutte le specifiche e problemi di prestazioni, hanno causato un passaggio graduale verso i servizi **REST**.

Servizi Web REST

I servizi **REST** utilizzano **HTTP** per lo scambio dati in formato **JSON**, o meno comunemente, in formato **XML**.

Sostituiscono il **SOAP body** con **body HTTP**, SOAP header con **header HTTP** e **HTTP response** sostituisce l'elemento fault e fornisce ulteriori informazioni.

REST - Esempi

```
{
Name: string,
Surname: string,
Address: string
}
```

```
{
Name: string,
Surname: string,
}
```

REST - Esempi

Supporre un servizio che gestisce un database Persons sostituendo la stringa Address con un oggetto complesso:

```
{
Name: string,
Surname: string,
Address:
{
Country: string,
Town: string
Location: string
}
}
```

REST

- REST utilizza le **funzionalità HTTP native** per implementare tutte le caratteristiche del servizio richieste.

- Non esiste un supporto nativo per la comunicazione asincrona poiché HTTP è un **protocollo sincrono**.

- Non esiste alcun supporto nativo per il pattern Publisher/Subscriber ma due servizi possono interagire con il Publisher/Subscriber esponendo ciascuno l' endpoint all'altro.

Request Methods

REST è l' acronimo di **Representational State Transfer**, il che significa che ogni URL è la rappresentazione di una sorta di oggetto.

Request Methods

Ogni tipo di richiesta deve adottare un metodo di richiesta appropriato:

- **GET** (operazione di lettura): l'URL rappresenta la risorsa restituita

- **POST** (operazione di creazione): l'oggetto JSON/XML contenuto nella richiesta body viene aggiunto come nuova risorsa all'oggetto rappresentato dall'URL dell'operazione.

Request Methods

- **PUT**: l'oggetto JSON/XML contenuto nel corpo della richiesta sostituisce l'oggetto referenziato dall'URL della richiesta

- **PATCH**: l'oggetto JSON/XML contenuto nel corpo della richiesta contiene istruzioni su come modificare l'oggetto a cui fa riferimento l'URL della richiesta.

- **DELETE**: la risorsa a cui fa riferimento l'URL della richiesta viene rimossa.

Codici Stato HTTP

Una volta eseguita l'operazione viene restituito un codice di stato:

- **1xx informational response**: la richiesta è stata ricevuta, processo in corso.
- **2xx successful**: la richiesta è stata ricevuta, compresa e accettata con successo.
- **3xx redirection**: è necessario intraprendere ulteriori azioni per completare la richiesta.
- **4xx client error**: la richiesta contiene una sintassi errata o non può essere soddisfatta.
- **5xx server error**: il server non è riuscito a soddisfare una richiesta apparentemente valida.

ASP.NET Core

Le applicazioni **ASP.NET Core** sono applicazioni **.NET Core** basate sul concetto di **Host**.

ASP.NET Core

Il file **program.cs** di ogni applicazione **ASP.NET** crea un **host**, lo compila e lo esegue con il codice seguente:

```
public class Program
{
public static void Main(string[] args)
{
CreateHostBuilder(args).Build().Run();
}
public static IHostBuilder CreateHostBuilder(string[] args) =>
Host.CreateDefaultBuilder(args)
.ConfigureWebHostDefaults(webBuilder =>
{
webBuilder.UseStartup<Startup>();
});
}
```

ASP.NET Core

- **CreateDefaultBuilder** imposta un host standard, mentre **ConfigureWebHostDefaults** lo configura in modo che possa gestire una **pipeline HTTP**.

I Controller

- Nel framework MVC le richieste HTTP vengono elaborate da classi chiamate **Controller**.
- Ogni richiesta è mappata alla chiamata di un metodo pubblico del controller.
- Il controller selezionato e i metodi del controller dipendono dal **request path** e sono definiti da regole di **routing**.
- Le regole di routing solitamente vengono fornite tramite attributi associati sia con la classe **Controller** che con i suoi metodi.

I Controller

- I metodi del controller che elaborano le richieste HTTP sono chiamati **action methods**.

- Quando **controller** e **action methods** sono selezionati il framework MVC crea un'istanza del controller per la richiesta.

- Tutti i parametri dei costruttori del controller vengono risolti attraverso **dependency injection** con tipi definiti nel metodo **ConfigureServices** della classe **Startup.cs**.

I Controller

Controller **API REST** e le relative definizioni del metodo controller:

```
[Route("api/[controller]")]
[ApiController]
public class ValuesController : ControllerBase
{
// GET api/values/5
[HttpGet("{id}")]
public ActionResult<string> Get(int id)
{
...
```

I Controller

- L'attributo [**ApiController**] dichiara che il controller è un controller API REST.

- **[Route ("api / [controller]")]** dichiara che il controller deve essere selezionato sui percorsi che iniziano **con api/<nome controller>**.

- **[HttpGet ("{id}")]** dichiara che il metodo deve essere invocato sulle richieste **GET** di **api/values/<id>** type

I Controller

- Per impostazione predefinita i parametri vengono passati ai metodi dei controller API in modo differente:

- **Tipi semplici**(int, float e DateTimes) vengono presi dal request path se le **routing rules** li specificano come parametri.

- **Tipi complessi** vengono estratti dal corpo della richiesta dai **formatter**. Se non viene specificata alcuna intestazione Content-Type, viene utilizzato il formatter **JSON**.

I Controller

È possibile personalizzare l'origine utilizzata per i parametri degli **action methods** aggiungendo un prefisso al parametro con un attributo adeguato:

```
...MyAcrionMethod(....[FromHeader] string myHeader....)
// x is taken from a request header named myHeader
...MyAcrionMethod(....[FromServices] MyType x....)
// x is filled with an istance of MyType through dependency injec
tion
```

I Controller

- Il tipo restituito da un metodo Action deve essere un'interfaccia **IAsyncResult** o un tipo che la implementa.

- **IAsyncResult** presenta il seguente metodo:

```
public Task ExecuteResultAsync (ActionContext context)
```

I Controller

- **IAsyncResult** viene chiamato dal framework MVC al momento giusto per creare la risposta effettiva e i **response headers**.

- L'oggetto **ActionContext** contiene l'intero contesto della richiesta HTTP.

I Controller

Per impostazione predefinita gli oggetti vengono serializzati in **JSON** nel **body** della risposta.

I Controller

Gli oggetti complessi che vengono passati come input agli **action methods** possono essere convalidati con attributi di convalida:

```
public class MyType
{
[Required]
public string Name{get; set;}
...
[MaxLength(64)]
public string Description{get; set;}
}
```

I Controller

Se il controller presenta l'attributo [**ApiController**] e la convalida fallisce, il framework MVC crea automaticamente una **BadRequest** contenente un dizionario con tutti gli errori di convalida.

In questo caso l'**action method** non viene eseguito.

I Controller

Gli **action methods** possono anche essere dichiarati come metodi asincroni:

```
public async Task<IActionResult> MyMethod(......)
{
await MyBusinessObject.MyBusinessMethod();
...
}
public async Task<ActionResult<MyType>> MyMethod(......)
{
...
```

Domande & approfondimenti

Academy .NET