

Lección 3 - Crear un servidor web

Tabla de contenido

- [Introducción](#)
- [Objetivos](#)
 - [Comprender los Conceptos Básicos de Gin y su Estructura de Proyecto](#)
 - [Aprender a Configurar un Servidor Web Básico Utilizando Gin y Gestionar los Puertos de Conexión](#)
 - [Explorar Cómo Diferenciar entre el Tipo de Servidor Nuevo y por Defecto en Gin](#)
- [Desarrollo](#)
 - [Salida por medio de JSON](#)
 - [Cambiar puerto](#)
 - [Crear documentación del algoritmo](#)
- [Conclusión](#)
- [Conoce más del autor](#)

Introducción

En esta clase aprenderemos cómo utilizar el framework Gin de Go para crear un servidor web. Gin es un potente framework web que permite construir rápidamente aplicaciones web eficientes y escalables en Go. A lo largo de esta clase, exploraremos los pasos para configurar un servidor web básico utilizando Gin y cómo manejar los puertos de conexión y diferenciar el tipo de servidor nuevo y por defecto.

Objetivos

Comprender los Conceptos Básicos de Gin y su Estructura de Proyecto

Este objetivo se centra en proporcionar una comprensión sólida de los fundamentos de Gin, incluyendo su papel como framework web en Go y la estructura típica de un proyecto Gin. Los estudiantes aprenderán sobre las características principales de Gin y cómo se organizan los archivos y directorios en un proyecto típico.

Aprender a Configurar un Servidor Web Básico Utilizando Gin y Gestionar los Puertos de Conexión

Vamos a trabajar en el proceso de configuración de un servidor web básico utilizando Gin. Se abordarán temas como la importación del paquete Gin, la creación de un enrutador y la configuración de las rutas. Además, se explorará cómo gestionar los puertos de conexión para garantizar un funcionamiento óptimo del servidor.

Explorar Cómo Diferenciar entre el Tipo de Servidor Nuevo y por Defecto en Gin

Este objetivo se enfoca en la comprensión de las diferencias entre `gin.New()` y `gin.Default()` en Gin. Aprenderemos sobre las características únicas de cada tipo de servidor y se proporcionarán ejemplos prácticos para ilustrar cuándo es apropiado utilizar cada uno en función de las necesidades del proyecto y las preferencias del desarrollador.

Desarrollo

Nos ubicamos en el fichero `main.go`.

```
package main

import "github.com/gin-gonic/gin"

func main() {
    // Creamos una variable "r" que represente una ruta e
    // Inicializamos Gin con su configuración por defecto.
    r := gin.Default()

    // Arrancamos el servidor en el puerto 8080
    r.Run(":8080")
}
```

Miramos la terminal, hacemos algunas consultas al servidor desde el navegador y analizamos la solicitud.

```
[GIN-debug] Listening and serving HTTP on :8080
[GIN] 2024/03/24 - 21:23:59 | 404 |           0s |      127.0.0.1 | GET
"/"
[GIN] 2024/03/24 - 21:24:00 | 404 |           0s |      127.0.0.1 | GET
"/"
[GIN] 2024/03/24 - 21:24:02 | 404 |           0s |      127.0.0.1 | GET
"/"
[GIN] 2024/03/24 - 21:24:02 | 404 |           0s |      127.0.0.1 | GET
```

```
"/"  
[GIN] 2024/03/24 - 21:24:07 | 404 | 0s | 127.0.0.1 | GET  
"/test"
```

Pero con esto solo obtenemos un *404 page not found* y para mejorar esto, necesitamos crear rutas, para ello, vamos a utilizar la página principal, la raíz "/".

```
package main  
  
import "github.com/gin-gonic/gin"  
  
func main() {  
    // Crea una nueva instancia de Gin  
    router := gin.Default()  
  
    // Define una ruta para la raíz  
    router.GET("/", func(c *gin.Context) {  
        c.String(200, "¡Hola, desde Gin!")  
    })  
  
    // Escucha en el puerto 8080  
    router.Run(":8080")  
}
```

```
[GIN] 2024/03/24 - 21:38:45 | 200 | 0s | 127.0.0.1 | GET  
"/"  
[GIN] 2024/03/24 - 21:38:58 | 404 | 0s | 127.0.0.1 | GET  
"/test"
```

También podemos utilizar `.New()` en vez de `.Default()` en Gin.

Notan que no hay diferencia, pero si las hay: `.New()`:

- Este método crea un nuevo enrutador Gin sin ninguna configuración predeterminada aplicada.
- Es útil cuando deseas una mayor flexibilidad para configurar el enrutador según tus propias necesidades, sin que se aplique ninguna configuración predeterminada.
- • Puedes usar las funciones `SetMode()`, `Use()` y `SetRenderer()` para configurar el router.

`.Default()` : Este método crea un enrutador Gin con la configuración predeterminada, que incluye:

- Modo de depuración activado.
- Middleware de recuperación de errores activado.
- Middleware de registro activado.
- Renderizador de HTML activado.
- Soporte para CORS activado.

Mostrar que con `.New()` no es posible ver el log de las solicitudes desde la terminal, mientras que con `.Default()` si es posible.

Salida por medio de JSON

También es posible retornar un JSON directamente al servidor. Por ejemplo, actualizamos el string por un formato JSON:

```
//En vez de:
c.String(200, "Hola, mundo!")

//Poner:
c.JSON(200, gin.H{
    "message": "¡Hola desde Gin!",
})
```

Cambiar puerto

```
// Es posible escribir 127.0.0.1 o localhost
// Agrega un puerto distinto
router.Run("localhost:8181")
```

Hacemos la prueba en el navegador.

Crear documentación del algoritmo

1. Crear el directorio `/docs` dentro del directorio padre `Gin`.
2. Crear un fichero llamado `Servidor Web Básico en Go.md`.
3. Agregar el algoritmo con el título:

```
# Servidor Web Básico en Go

package main

import "github.com/gin-gonic/gin"

func main() {
    router := gin.Default()

    router.GET("/", func(c *gin.Context) {
        c.String(200, "Hola, mundo!")
    })

    router.Run(":8080")
}
```

Subir los cambios a GitHub:

```
$ git add .
$ git commit -m "Servidor web en Go"
$ git push
```








Revisar en la web de GitHub que las actualizaciones estén disponibles. Detallaré mayor contenido sobre cada lección.

Conclusión

Hemos explorado los fundamentos del framework Gin de Go y hemos aprendido cómo crear un servidor web básico utilizando esta potente herramienta. Luego, nos sumergimos en la configuración de un servidor web, donde aprendimos a gestionar los puertos de conexión y a definir las rutas de nuestro servidor. Además, exploramos las diferencias entre los tipos de servidores `gin.New()` y `gin.Default()`, lo que nos proporcionó una comprensión más profunda de las capacidades y usos de Gin.

Conoce más del autor

¡Encuétrame en las siguientes redes sociales para estar al tanto de mis proyectos y actividades!

 Red Social	 Enlace
 Página web	jersonmartinez.com
 LinkedIn	Jerson Martínez - DevOps Engineer
 Canales de YouTube	DevOpsea Side Master
 GitHub	Perfil en GitHub
 Twitter (X)	@antoniomorenosm