Gin

Introduction

Gin is a framework for building web applications in Go. It is a web microframework that is inspired by Martini and Sinatra. It features a Martini-like API with much better performance.

Features

- Designed for high performance
- Zero memory allocation in hot path

Installation

To install Gin, simply run:

```
$ go get github.com/gin-gonic/gin
```

Getting Started

Hello World

```
package main

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

func main() {
    r := gin.Default()
    r.GET("/", func(c *gin.Context) {
        c.String(200, "Hello, world!")
    })
    r.Run(":8080")
}
```

Routing

Gin provides a group-based routing mechanism. Each route is attached to a group, and the group is attached to the engine. The engine is the root group.

```
package main

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

func main() {
    r := gin.Default()
    r.GET("/ping", func(c *gin.Context) {
        c.String(200, "pong")
    })
    r.Run(":8080")
}
```

Grouping

Gin provides a group-based routing mechanism. Each route is attached to a group, and the group is attached to the engine. The engine is the root group.

```
package main

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

func main() {
    r := gin.Default()
    v1 := r.Group("/v1")
    {
        v1.GET("/login", loginEndpoint)
        v1.GET("/submit", submitEndpoint)
        v1.GET("/read", readEndpoint)
    }
    v2 := r.Group("/v2")
    {
        v2.POST("/login", loginEndpoint)
        v2.POST("/submit", submitEndpoint)
        v2.POST("/read", readEndpoint)
    }
    r.Run(":8080")
}
```

Parameters

There are two types of parameters:

- path parameters: parameters are used to capture the values of wildcards in the path.
- query parameters: are used to pass additional information to the server.

Path Parameters

Gin supports named parameters in the path. They are defined with a colon and the name of the parameter. The value of the parameter can be retrieved from the `Context.Params` field.

```
package main

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

func main() {
    r := gin.Default()
    r.GET("/user/:name", func(c *gin.Context) {
        name := c.Params.ByName("name")
        c.String(200, "Hello %s", name)
    })
    r.Run(":8080")
}
```

Query Parameters

Query parameters are parsed using the `Request.URL.Query()` method. The parsed query parameters are stored in a map, `c.Request.URL.Query()`.

```
package main

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

func main() {
    r := gin.Default()
    r.GET("/welcome", func(c *gin.Context) {
        first := c.DefaultQuery("first", "Guest")
        last := c.Query("last") // shortcut for c.Request.URL.Query().Get("last")
        c.String(200, "Hello %s %s", first, last)
    })
    r.Run(":8080")
}
```

Multipart/Urlencoded Form

Gin supports multipart and urlencoded forms. The request body is parsed using the `Request.ParseMultipartForm` method. The parsed form is stored in a map, `c.Request.Form`.

```
package main

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

func main() {
    r := gin.Default()
    r.POST("/form_post", func(c *gin.Context) {
        message := c.PostForm("message")
        nick := c.DefaultPostForm("nick", "anonymous")
        c.JSON(200, gin.H{
            "status": "posted",
            "message": message,
            "nick": nick,
        })
    })
    r.Run(":8080")
}
```

Static Files

Gin provides a built-in static file server. The `Static` method takes two parameters: the route and the local directory to serve files from.

```
package main

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

func main() {
    r := gin.Default()
    r.Static("/assets", "./assets")
    r.Run(":8080")
}
```

Custom 404

Gin provides a built-in custom 404 handler. The `NoRoute` method takes a handler function as a parameter.

```
package main

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

func main() {
    r := gin.Default()
    r.NoRoute(func(c *gin.Context) {
        c.String(404, "The incorrect API route")
    })
    r.Run(":8080")
}
```

Methods

Gin supports the following HTTP methods: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS.

```
package main

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

func main() {
    r := gin.Default()
    r.GET("/someGet", getting)
    r.POST("/somePost", posting)
    r.PUT("/somePut", putting)
    r.DELETE("/someDelete", deleting)
    r.PATCH("/somePatch", patching)
    r.HEAD("/someHead", head)
    r.OPTIONS("/someOptions", options)
    r.Run(":8080")
}
```

Context

The `Context` is the most important part of Gin. It carries the request and response information.

```
package main

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

func main() {
    r := gin.Default()
    r.GET("/someGet", func(c *gin.Context) {
        c.JSON(200, gin.H{
            "message": "hey",
            "status":  200,
        })
    })
    r.Run(":8080")
}
```

The main uses of `Context` *are:*

- Get the request information
- Send the response information
- Set the next handler to be called
- Abort the chain of handlers

From context, we can retrieve the following information:

-

`c.Params`: path parameters

- `c.Query`: query parameters
- `c.PostForm`: form parameters
- `c.Request`: the `*http.Request`
- `c.Writer`: the `http.ResponseWriter`
- `c.FullPath`: the full path matched, including the query string
- `c.Errors`: a list of errors
- `c.Keys`: a map of keys and values stored in the `Context`

We can use context to send response to the client:

- `c.String()`: send a string response
- `c.JSON()`: send a JSON response
- `c.XML()`: send a XML response
- `c.YAML()`: send a YAML response
- `c.ProtoBuf()`: send a ProtoBuf response
- `c.Data()`: send a response with data and content type
- `c.HTML()`: render a template
- `c.Redirect()`: redirect the request
- `c.File()`: send a file as an octet stream
- `c.Attachment()`: send a file as an attachment
- `c.Inline()`: send a file as an inline file

.

Context Keys

Gin provides a mechanism to store and retrieve values from the `Context`. The `Set` method stores a value in the `Context`. The `Get` method retrieves a value from the `Context`.

Get the value from the `Context`

```
package main

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

func main() {
    r := gin.Default()
    r.GET("/someGet", func(c *gin.Context) {
        c.Set("example", "12345")
        value, exists := c.Get("example")
        if exists {
            log.Println("value: %s", value)
        }
        c.JSON(200, gin.H{
            "message": "hey",
            "status":  200,
        })
    })
    r.Run(":8080")
}
```

Set the value in the `Context`

```
package main

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

func main() {
    r := gin.Default()
    r.GET("/someGet", func(c *gin.Context) {
        c.Set("example", "12345")
        value, exists := c.Get("example")
        if exists {
            log.Println("value: %s", value)
        }
        c.JSON(200, gin.H{
            "message": "hey",
            "status":  200,
        })
    })
    r.Run(":8080")
}
```

Context Params

Gin provides a mechanism to retrieve route parameters from the `Context`. The `Param` method retrieves a route parameter from the `Context`.

```
package main

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

func main() {
    r := gin.Default()
    r.GET("/user/:name/*action", func(c *gin.Context) {
        name := c.Param("name")
        action := c.Param("action")
        message := name + " is " + action
        c.String(200, message)
    })
    r.Run(":8080")
}
```

Context Query

Gin provides a mechanism to retrieve query parameters from the `Context`. The `Query` method retrieves a query parameter from the `Context`.

```
package main

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

func main() {
    r := gin.Default()
    r.GET("/welcome", func(c *gin.Context) {
        first := c.DefaultQuery("first", "Guest")
        last := c.Query("last") // shortcut for c.Request.URL.Query().Get("last")
        c.String(200, "Hello %s %s", first, last)
    })
    r.Run(":8080")
}
```

Context PostForm

Gin provides a mechanism to retrieve post form parameters from the `Context`. The `PostForm` method retrieves a post form parameter from the `Context`.

```
package main

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

func main() {
    r := gin.Default()
    r.POST("/form_post", func(c *gin.Context) {
        message := c.PostForm("message")
        nick := c.DefaultPostForm("nick", "anonymous")
        c.JSON(200, gin.H{
            "status": "posted",
            "message": message,
            "nick": nick,
        })
    })
    r.Run(":8080")
}
```

Context Body

Gin provides a mechanism to retrieve the request body from the `Context`. The `ShouldBind` method retrieves the request body from the `Context`.

```
package main

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

type Login struct {
    User     string `form:"user" json:"user" binding:"required"`
    Password string `form:"password" json:"password" binding:"required"`
}

func main() {
    r := gin.Default()
    r.POST("/loginJSON", func(c *gin.Context) {
        var json Login
        if c.ShouldBindJSON(&json) == nil {
            if json.User == "manu" && json.Password == "123" {
                c.JSON(200, gin.H{"status": "you are logged in"})
                return
            }
        }
        c.JSON(401, gin.H{"status": "unauthorized"})
    })
    r.Run(":8080")
}
```