go-restful

package for building REST-style Web Services using Google Go

- Code examples

REST asks developers to use HTTP methods explicitly and in a way that's consistent with the protocol definition. This basic REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods. According to this mapping:

- GET = Retrieve a representation of a resource
- POST = Create if you are sending content to the server to create a subordinate of the specified resource collection, using some server-side algorithm.
- PUT = Create if you are sending the full content of the specified resource (URI).
- PUT = Update if you are updating the full content of the specified resource.
- DELETE = Delete if you are requesting the server to delete the resource
- PATCH = Update partial content of a resource
- OPTIONS = Get information about the communication options for the request URI

Example

```
ws := new(restful.WebService)
ws.
    Path("/users").
    Consumes(restful.MIME_XML, restful.MIME_JSON).
    Produces(restful.MIME_JSON, restful.MIME_XML)

ws.Route(ws.GET("/{user-id}").To(u.findUser).
    Doc("get a user").
    Param(ws.PathParameter("user-id", "identifier of the user").DataType("string")).
    Writes(User{}))
...

func (u UserResource) findUser(request *restful.Request, response *restful.Response)
{
    id := request.PathParameter("user-id")
    ...
}
```

Full API of a UserResource

Features

- Routes for request → function mapping with path parameter (e.g. {id}) support
- Configurable router:
  - (default) Fast routing algorithm that allows static elements, regular expressions and dynamic parameters in the URL path (e.g. /meetings/{id} or /static/{subpath:*}
  - Routing algorithm after JSR311 that is implemented using (but does **not** accept) regular expressions

- Request API for reading structs from JSON/XML and accesing parameters (path,query,header)
- Response API for writing structs to JSON/XML and setting headers
- Customizable encoding using EntityReaderWriter registration
- Filters for intercepting the request → response flow on Service or Route level
- Request-scoped variables using attributes
- Containers for WebServices on different HTTP endpoints
- Content encoding (gzip,deflate) of request and response payloads
- Automatic responses on OPTIONS (using a filter)
- Automatic CORS request handling (using a filter)
- API declaration for Swagger UI (go-restful-openapi, see go-restful-swagger12)
- Panic recovery to produce HTTP 500, customizable using RecoverHandler(...)
- Route errors produce HTTP 404/405/406/415 errors, customizable using ServiceErrorHandler(...)
- Configurable (trace) logging
- Customizable gzip/deflate readers and writers using CompressorProvider registration

How to customize

There are several hooks to customize the behavior of the go-restful package.

- Router algorithm
- Panic recovery
- JSON decoder
- Trace logging
- Compression
- Encoders for other serializers
- Use jsoniter by build this package using a tag, e.g. `go build -tags=jsoniter .`

TODO: write examples of these.

Resources

- Example posted on blog
- Design explained on blog
- sourcegraph
- showcase: Zazkia - tcp proxy for testing resiliency
- showcase: Mora - MongoDB REST Api server

Type `git shortlog -s` for a full list of contributors.

© 2012 - 2018, http://ernestmicklei.com. MIT License. Contributions are welcome.