JitJSON

By Max Collier

Motivation

Traditional parsing with json.Marshal or json.Unmarshal processes all data immediately, even if it may never be used. Unnecessary parsing leads to wasted CPU cycles on unused data, unnecessary memory allocations, and increased pressure on garbage collection operations. If you intend to parse all your data, jitjson will not provide any benefit.

JITJSON at Starboard

```
type JITJSON struct {
    data   []byte
    keys   []string
    values []interface{}
}
```

Marshal

```go
package main

import (
    "fmt"
        "encoding/json"
)

type Person struct {
        Name    string `json:"name"`
        Age     int    `json:"age"`
        Email   string `json:"email,omitempty"`
}

func main() {
    person := Person{
        Name:    "Alice Johnson",
        Age:     30,
        Email:   "",
    }

    jsonData, err := json.Marshal(person) // []byte, error
    if err != nil {
        panic(err)
    }
    fmt.Printf("%s\n", string(jsonData)) // Output: {"name":"Alice Johnson","age":30}
}
```

Unmarshal

```go
package main

import (
    "fmt"
        "encoding/json"
)

type Person struct {
        Name    string `json:"name"`
        Age     int    `json:"age"`
        Email   string `json:"email,omitempty"`
}

func main() {

    jsonData := []byte(`{
                "age": 32,
        "name": "Augus Fitzgerald",
        "email": "ang.fitz@hotmail.com"
    }`)

    var person Person
    err := json.Unmarshal(jsonData, &person)
    if err != nil {
        panic(err)
    }
    fmt.Printf("%+v\n", person) // Output: {Augus Fitzgerald 32 ang.fitz@hotmail.com }
}
```

Valid JSON

```
// valid json
true
1000.123
"hello W0R1d"
null
[1, 2, 3]
{"hello": "there!"}
[1, "yes", true, null, ["knock knock"]]

// invalid json
My str
{"hello":}
11000.2002.2100
```

json.Marshal

https://pkg.go.dev/encoding/json#Marshaler

```
type Marshaler interface {
        MarshalJSON() ([]byte, error)
}
```

Example

```
package main

import (
    "fmt"
        "encoding/json"
)

type Animal struct {
        Name    string `json:"name"`
        Age     int    `json:"age"`
        Type    string `json:"animalType"`
}

func (p *Animal) MarshalJSON() ([]byte, error) {
        if p.Type == "Dog" {
                return []byte("\"Dog is actually a cat. Meow!\""), nil
        } else {
                return []byte(fmt.Sprintf("\"Woof Woof Woof! said the %s\"", p.Type)), nil
        }
}

// verify interface:
var _ json.Marshaler = (*Animal)(nil)
```

Example

```
var (
    animal1 = &Animal{
        Name: "Roger Rog",
        Age:  13,
        Type: "Dog",
    }
    animal2 = &Animal{
        Name: "Lizzy Lizz",
        Age:  11,
        Type: "Cat",
    }
)

func main() {
    jsonData1, _ := json.Marshal(animal1)
    jsonData2, _ = json.Marshal(animal2)

    fmt.Printf("%s\n", string(jsonData1)) // Output: "Dog is actually a cat. Meow!"
    fmt.Printf("%s\n", string(jsonData2)) // Output: "Woof Woof Woof! said the Cat"
}
```

json.Unmarshal

https://pkg.go.dev/encoding/json#Marshaler

```
type Marshaler interface {
        UnmarshalJSON(data []byte) error
}
```

Example

```
package main

import (
        "encoding/json"
        "fmt"
)

type Animal struct {
        Name string `json:"name"`
        Age  int    `json:"age"`
        Type string `json:"animalType"`
}
```

Example

```
func (a *Animal) UnmarshalJSON(data []byte) error {
        var raw map[string]interface{}
        if err := json.Unmarshal(data, &raw); err != nil {
                return err
        }
        if name, ok := raw["name"].(string); ok {
                a.Name = name
        }
        if age, ok := raw["age"].(float64); ok {
                a.Age = int(age)
        }
        if typ, ok := raw["animalType"].(string); ok {
        select typ {
        case "Dog":
            a.Type = "Cat"
        case "Cat":
            a.Type = "Dog"
                } else {
                        a.Type = "Unsure?"
        }
        }
        return nil
}
```

Example

```
// verify interface
var _ json.Unmarshaler = (*Animal)(nil)

var a1, a2, a3 Animal

func main() {
        jsonDog := `{"name":"Rover","age":5,"animalType":"Dog"}`
        jsonCat := `{"name":"Whiskers","age":3,"animalType":"Cat"}`
        jsonDuck := `{"name":"Donald","age":7,"animalType":"Duck"}`

        _ = json.Unmarshal([]byte(jsonDog), &a1)
        _ = json.Unmarshal([]byte(jsonCat), &a2)
        _ = json.Unmarshal([]byte(jsonDuck), &a3)

        fmt.Printf("%+v\n", a1) // {Name:Rover Age:5 Type:Cat}
        fmt.Printf("%+v\n", a2) // {Name:Whiskers Age:3 Type:Dog}
        fmt.Printf("%+v\n", a3) // {Name:Donald Age:7 Type:Unsure?}
}
```