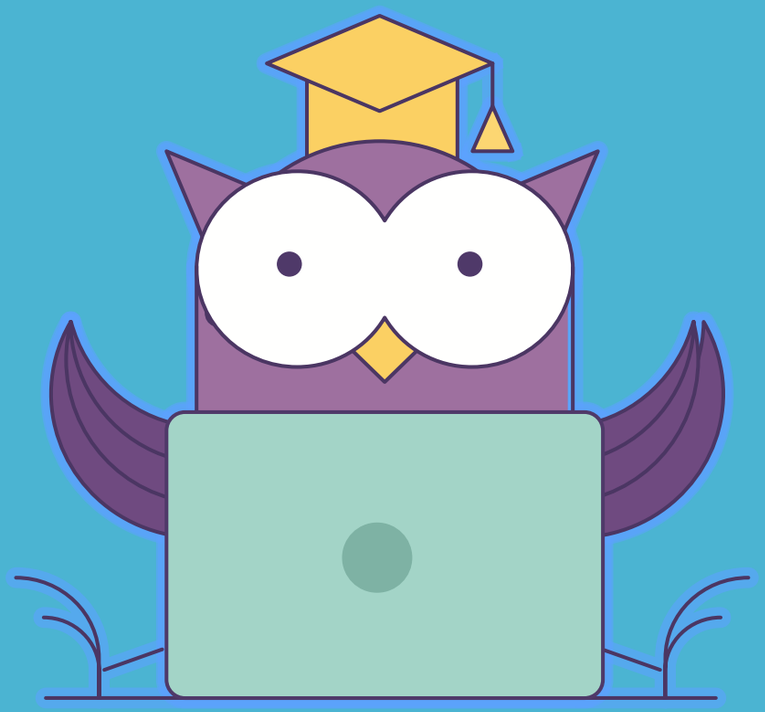
ОНЛАЙН-ОБРАЗОВАНИЕ

Форматирование данных

Желтак Артем

Как меня слышно и видно?



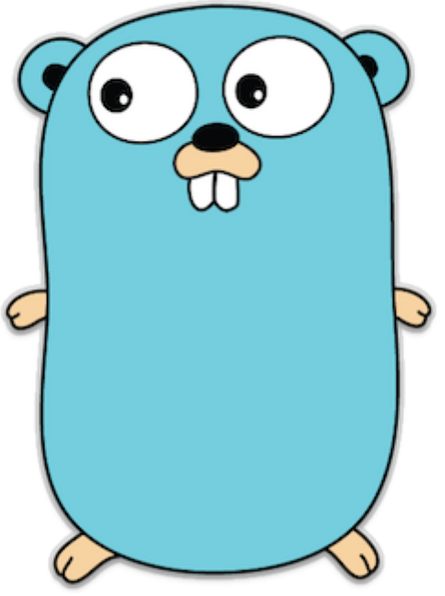
> Напишите в чат

+ если все хорошо

– если есть проблемы со звуком или с видео

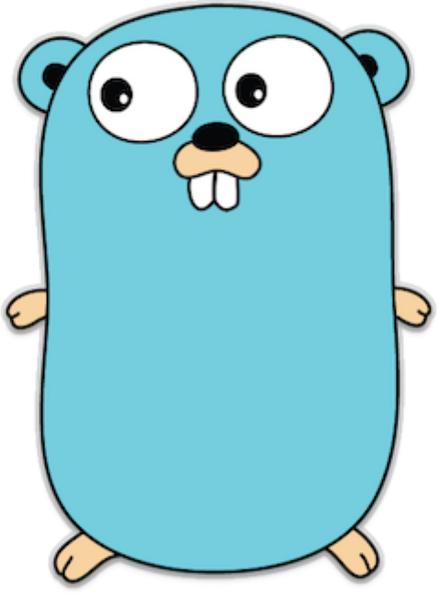
Цель занятия

- Изучить возможности кодирования бинарных данных в текстовой форме
- Научиться использовать стандартную библиотеку для кодирования в формате base64
- Изучить форматы JSON, XML, YAML.
- Изучить подходы к парсингу XML.
- Научиться парсить JSON через стандартную библиотеку
- Изучить библиотеку easyjson
- Изучить библиотеки для работы с MsgPack и Protobuf



А зачем?

Зачем кодировать бинарные данные в текстовый вид?



Кодировка quoted-printable

```
Content-Disposition: inline
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain; charset="UTF-8"

=D0=91=D0=BE=D0=BB=D1=8C=D1=88=D0=B8=D0=BD=D1=81=D1=82=D0=B2=D0=BE =D0=BA=
=D0=BE=D0=BD=D1=84=D0=B5=D1=80=D0=B5=D0=BD=D1=86=D0=B8=D0=B9 =D1=81=D1=82=
=D0=B0=D1=80=D1=82=D1=83=D0=B5=D1=82 =D1=80=D0=B0=D0=BD=D0=BE =D1=83=D1=82=
=D1=80=D0=BE=D0=BC, =D0=BA=D0=BE=D0=B3=D0=B4=D0=B0 =D1=83 =C2=AB=D1=81=D0=
=BE=D0=B2=C2=BB =0D=0A=D0=B5=D1=89=D1=91 =D1=81=D0=BB=D0=B8=D0=BF=D0=B0=D1=
=8E=D1=82=D1=81=D1=8F =D0=B3=D0=BB=D0=B0=D0=B7=D0=B0=0D=0A=0D=0A=D0=9C=D1=
=8B =D1=83=D1=81=D1=82=D0=B0=D0=BB=D0=B8 =D1=82=D0=B5=D1=80=D0=BF=D0=B5=D1=
=82=D1=8C =D1=8D=D1=82=D1=83 =D0=BD=D0=B5=D1=81=D0=BF=D1=80=D0=B0=D0=B2=D0=
=B5=D0=B4=D0=BB=D0=B8=D0=B2=D0=BE=D1=81=D1=82=D1=8C =D0=B8 =D1=83=D1=81=D1=
```

Какие минусы?

Кодировка quoted-printable

```
Content-Disposition: inline
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain; charset="UTF-8"

=D0=91=D0=BE=D0=BB=D1=8C=D1=88=D0=B8=D0=BD=D1=81=D1=82=D0=B2=D0=BE =D0=BA=
=D0=BE=D0=BD=D1=84=D0=B5=D1=80=D0=B5=D0=BD=D1=86=D0=B8=D0=B9 =D1=81=D1=82=
=D0=B0=D1=80=D1=82=D1=83=D0=B5=D1=82 =D1=80=D0=B0=D0=BD=D0=BE =D1=83=D1=82=
=D1=80=D0=BE=D0=BC, =D0=BA=D0=BE=D0=B3=D0=B4=D0=B0 =D1=83 =C2=AB=D1=81=D0=
=BE=D0=B2=C2=BB =0D=0A=D0=B5=D1=89=D1=91 =D1=81=D0=BB=D0=B8=D0=BF=D0=B0=D1=
=8E=D1=82=D1=81=D1=8F =D0=B3=D0=BB=D0=B0=D0=B7=D0=B0=0D=0A=0D=0A=D0=9C=D1=
=8B =D1=83=D1=81=D1=82=D0=B0=D0=BB=D0=B8 =D1=82=D0=B5=D1=80=D0=BF=D0=B5=D1=
=82=D1=8C =D1=8D=D1=82=D1=83 =D0=BD=D0=B5=D1=81=D0=BF=D1=80=D0=B0=D0=B2=D0=
=B5=D0=B4=D0=BB=D0=B8=D0=B2=D0=BE=D1=81=D1=82=D1=8C =D0=B8 =D1=83=D1=81=D1=
```

Избыточность = 300%

Кодировка base64

```
TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0
aGlzIHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBhIGx1
c3Qgb2YgdGhlIG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0
aGUgY29udGludWVkIGFuZCBpbmRlZmF0aWdhYmxlIGdlbmVyYXRpb24gb2Yga25vd2xlZGdl
LCBleGNlZWRzIHRoZSBzaG9ydCB2ZWhlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbGVhc3VyZS4=
```

Избыточность = 1/3

Работа с base64

```
package main

import b64 "encoding/base64"
import "fmt"

func main() {

    data := "Hello world"

    sEnc := b64.StdEncoding.EncodeToString([]byte(data))
    fmt.Println(sEnc)

    sDec, _ := b64.StdEncoding.DecodeString(sEnc)
    fmt.Println(string(sDec))
    fmt.Println()

    uEnc := b64.URLEncoding.EncodeToString([]byte(data))
    fmt.Println(uEnc)
    uDec, _ := b64.URLEncoding.DecodeString(uEnc)
    fmt.Println(string(uDec))
}
```

https://play.golang.org/p/4oFM2M2Sirq

А какие недостатки?

Поточная работа с base64 (кодирование)

```go
package main

import (
    "encoding/base64"
    "os"
)

func main() {
    input := []byte("foo\x00bar")
    encoder := base64.NewEncoder(base64.StdEncoding, os.Stdout)
    encoder.Write(input)
    // Must close the encoder when finished to flush any partial blocks.
    // If you comment out the following line, the last partial block "r"
    // won't be encoded.
    encoder.Close()
}
```

https://play.golang.org/p/GwrvXsSzeN7

Поточная работа с base64 (декодирование)

```
package main

import (
    "encoding/base64"
    "os"
    "io"
    "strings"
)

func main() {
    input := "Zm9vAGJhcg=="
    r := base64.NewDecoder(base64.StdEncoding, strings.NewReader(input))
        io.Copy(os.Stdout, r)
}
```

https://play.golang.org/p/uxmmi_OX42i

JSON

XML

YAML

Какие цели у сериализации?

```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Konfabulator Widget",
        "name": "main_window",
        "width": 500,
        "height": 500
    },
    "image": {
        "src": "Images/Sun.png",
        "name": "sun1",
        "hOffset": 250,
        "vOffset": 250,
        "alignment": "center"
    },
    "text": {
        "data": "Click Here",
        "size": 36,
        "style": "bold",
        "name": "text1",
        "hOffset": 250,
        "vOffset": 100,
        "alignment": "center",
        "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"
    }
}}
```

```
---
widget:
  debug: 'on'
  window:
    title: Sample Konfabulator Widget
    name: main_window
    width: 500
    height: 500
  image:
    src: Images/Sun.png
    name: sun1
    hOffset: 250
    vOffset: 250
    alignment: center
  text:
    data: Click Here
    size: 36
    style: bold
    name: text1
    hOffset: 250
    vOffset: 100
    alignment: center
    onMouseUp: sun1.opacity = (sun1.opacity / 100) * 90;
```

```
<widget>
    <debug>on</debug>
    <window title="Sample Konfabulator Widget">
        <name>main_window</name>
        <width>500</width>
        <height>500</height>
    </window>
    <image src="Images/Sun.png" name="sun1">
        <hOffset>250</hOffset>
        <vOffset>250</vOffset>
        <alignment>center</alignment>
    </image>
    <text data="Click Here" size="36" style="bold">
        <name>text1</name>
        <hOffset>250</hOffset>
        <vOffset>100</vOffset>
        <alignment>center</alignment>
        <onMouseUp>
            sun1.opacity = (sun1.opacity / 100) * 90;
        </onMouseUp>
    </text>
</widget>
```

```
func main() {
    p1 := &Person{
        Name: "Vasya",
        age: 36,
        Job: struct {
            Department string
            Title      string
        }{Department: "Operations", Title: "Boss"},
    }

    j, err := json.Marshal(p1)
    if err != nil {
        fmt.Printf("%v\n", err)
        return
    }
    fmt.Printf("p1 json %s\n", j)

    var p2 Person
    json.Unmarshal(j, &p2)
    fmt.Printf("p2: %v\n", p2)

}
```

https://play.golang.org/p/p9uRcgPUX8B (полная версия)

Работа с JSON через interface{}

```
package main

import (
    "encoding/json"
    "fmt"
)

func main() {
    j := []byte(`{"Name":"Vasya",
        "Job":{"Department":"Operations","Title":"Boss"}}`)

    var p2 interface{}
    json.Unmarshal(j, &p2)
    fmt.Printf("p2: %v\n", p2)

    person := p2.(map[string]interface{})
    fmt.Printf("name=%s\n", person["Name"])
}
```

https://play.golang.org/p/mGVtP-hSQjq

Работа с тегами пакета json

```
type Person struct {
    Name    string `json:"fullname,omitempty"`
    Surname string `json:"familyname,omitempty"`
    Age     int    `json:"-"`
    Job     struct {
        Department string
        Title      string
    }
}
```

https://play.golang.org/p/RxcV-MjmgAm (полная версия)

Кодирование в xml

```
type Address struct {
    City, State string
}
type Person struct {
    XMLName   xml.Name `xml:"person"`
    Id        int      `xml:"id,attr"`
    FirstName string   `xml:"name>first"`
    LastName  string   `xml:"name>last"`
    Age       int      `xml:"age"`
    Height    float32  `xml:"height,omitempty"`
    Married   bool
    Address
    Comment string `xml:",comment"`
}
```

https://play.golang.org/p/QbfwL44vjJU (полная версия)

Декодирование из xml

```
type Address struct {
    City, State string
}
type Person struct {
    XMLName   xml.Name `xml:"person"`
    Id        int      `xml:"id,attr"`
    FirstName string   `xml:"name>first"`
    LastName  string   `xml:"name>last"`
    Age       int      `xml:"age"`
    Height    float32  `xml:"height,omitempty"`
    Married   bool
    Address
    Comment string `xml:",comment"`
}
```

https://play.golang.org/p/FekJkpuj9KT (полная версия)

А что если данные не помещаются в оперативную память?

```
for {
        token, _ := decoder.Token()

        switch se := token.(type) {
        case xml.StartElement:
            fmt.Printf("Start element: %v Attr %s\n",
                       se.Name.Local, se.Attr)
            inFullName = se.Name.Local == "FullName"
        case xml.EndElement:
            fmt.Printf("End element: %v\n", se.Name.Local)
        case xml.CharData:
            fmt.Printf("Data element: %v\n", string(se))
            if inFullName {
                names = append(names, string(se))
            }
        default:
            fmt.Printf("Unhandled element: %T", se)
        }

    }
```

https://play.golang.org/p/cuSIsVyZpD-

Что это такое?

Какие плюсы?

А какие минусы?

Бинарные сериализаторы

- gob
- msgpack
- protobuf

```
type Person struct {
    Name        string
    Surname     string
    Age         uint8
    ChildrenAge map[string]uint8
}
func main() {
    p := Person{Name:  "Ivan",
        Surname: "Remen", Age: 27,
    }
    p.ChildrenAge = make(map[string]uint8)
    p.ChildrenAge["Alex"] = 5
    p.ChildrenAge["Maria"] = 2

    marshaled, _ := msgpack.Marshal(&p)

    fmt.Printf("Length of marshaled: %v
       IMPL: %v\n", len(marshaled), string(marshaled))

    p2 := &Person{}
    msgpack.Unmarshal(marshaled, p2)
    fmt.Printf("Unmarshled: %v\n", p2)
}
```

https://play.golang.org/p/4pYvh-Qa_wg

protobuf (proto-файл)

```
syntax = "proto3";

package main;

message Person {
    string name = 1;
    string surname = 2;
    uint32 age = 3;

    map<string, uint32> children_age = 4;
}
```

Сборка: protoc --go_out=. *.proto

```go
package main

import (
    "fmt"
    "github.com/golang/protobuf/proto"
)

func main() {
    p := &Person{
        Age:          27,
        Name:         "Ivan",
        Surname:      "Remen",
        ChildrenAge: make(map[string]uint32),
    }
    p.ChildrenAge["Maria"] = 2
    p.ChildrenAge["Alex"] = 5

    marshaled, _ := proto.Marshal(p)

    fmt.Printf("marshaled len %d message = %s\n", len(marshaled), string(marshaled))

    p2 := &Person{}
    proto.Unmarshal(marshaled, p2)

    fmt.Printf("Unmarshaled %v", p2)

}
```

Изучаем easyjson

https://forms.gle/SiDmYTPUU5La3rA88

- Изучили возможности кодирования бинарных данных в текстовой форме
- Научились использовать стандартную библиотеку для кодирования в формате base64
- Изучили форматы JSON, XML, YAML.
- Изучили подходы к парсингу XML.
- Научились использовать стандартную библиотеку для кодирования в формате base64
- Научиться парсить JSON через стандартную библиотеку
- Изучили библиотеку easyjson
- Изучили библиотеки для работы с MsgPack и Protobuf

Вопросы?

Опрос

Не забудьте заполнить опрос. https://otus.ru/polls/4901/

Спасибо за внимание!