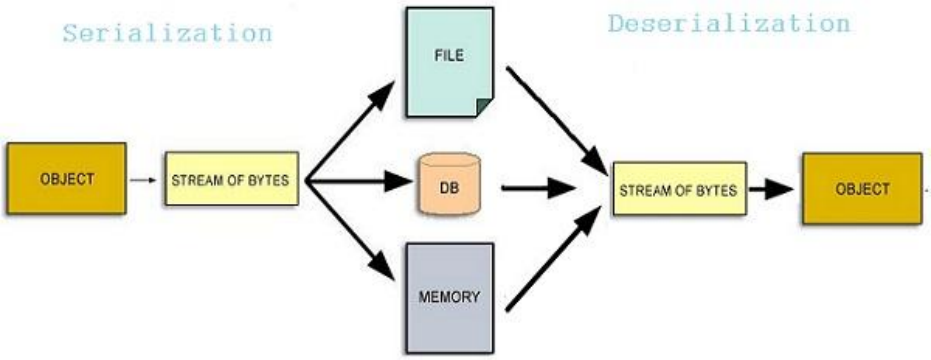
Serijalizacija

Serijalizacija predstavlja postupak prevođenja stanja objekta u niz bajtova sa ciljem njihovog trajnog skladištenja ili prenosa preko mreže, tako da se to stanje kasnije može rekonstruisati procesom deserijalizacije, često i u drugačijem okruženju.

Formati za serijalizaciju su brojni i mogu se podeliti na tekstualne i binarne. Neki od često korišćenih tekstualnih formata su JSON, XML, CSV, YAML, dok su neki od binarnih formata BSON, MessagePack, protobuf.



Podrška za serijalizaciju u Go-u nalazi se u okviru paketa encoding i njegovih potpaketa kao što su json, xml, csv, binary, gob itd. Sem ponuđenih načina za serijalizaciju, moguće je formirati i sopstvene formate, sve dok oni zadovoljavaju sve pred njih postavljene zahteve.

- **encoding/binary paket**

Paket encoding/binary implementira jednostavnu translaciju između brojeva i sekvenci bajtova. Translacija se vrši čitanjem i pisanjem vrednosti fiksne dužine. Vrednost fiksne dužine predstavlja ili aritmetički tip fiksne dužine (bool, int8, int16, uint8, float32, complex64 …) ili niz ili strukturu koja sadrži samo vrednosti fiksne dužine.

Važan deo paketa predstavlja interfejs ByteOrder koji specificira kako sekvence bajtova konvertovati u 16, 32, ili 64-bitne unsigned integer-e, dok su njegove implementacije BigEndian i LittleEndian. Treba voditi računa da se prilikom serijalizacije i deserijalizacije koristi ista implementacija kako bi se očuvalo značenje podatka.

```
b := []byte{1, 2, 3, 4}
val := binary.LittleEndian.Uint32(b)
fmt.Printf("Little endian: %b\n", val)
//output: Little endian: 00000100000000110000001000000001
val2 := binary.BigEndian.Uint32(b)
fmt.Printf("Big endian: %b\n", val2)
//output: Big endian: 00000001000000100000001100000100
binary.LittleEndian.PutUint32(b, val2)
fmt.Println(b)
//output: [4 3 2 1]
```

Strukture je moguće ručno serijalizovati polje po polje. Dat je primer rada sa strukturom Vertex koja ima polja X i Y tipa int32.

```
file, err := os.OpenFile("folder/somefile", os.O_RDWR, 0777)
if err != nil {
   panic(err.Error())
}
defer file.Close()

vertex := Vertex{1, 2}
bytes := make([]byte, 8)
binary.LittleEndian.PutUint32(bytes[:], uint32(vertex.X))
binary.LittleEndian.PutUint32(bytes[4:], uint32(vertex.Y))
fmt.Printf("%v\n", bytes)
//output: [1 0 0 0 2 0 0 0]
file.Write(bytes)
file.Sync()

result := make([]byte, 4)
file.Seek(4, 0)
_, err = file.Read(result)
if err != nil {
   panic(err.Error())
}
fmt.Println(result)
//output: [2 0 0 0]
```

Kako bi proces bio jednostavniji, binary paket nudi funkcije Read i Write kojima se iz io.Reader-a čita ili u io.Writer piše određeni podatak, uz zadavanje odgovarajućeg ByteOrder-a. Kako fajl implementira i io.Reader i io.Writer interfejse, na ovaj način je moguće direktno čitanje ili pisanje određene strukture u fajl. U daljem tekstu nalazi se nastavak primera sa početka strane.

```
vertex.X = 3
err = binary.Write(file, binary.LittleEndian, vertex)
if err != nil {
   panic(err.Error())
}
file.Seek(8, 0)
newv := &Vertex{}
err = binary.Read(file, binary.LittleEndian, newv)
if err != nil {
   panic(err.Error())
}
fmt.Println(*newv)
//output: {3 2}
file.Seek(0, 0)
result, err = ioutil.ReadAll(file)
if err != nil {
   panic(err.Error())
}
fmt.Println(result)
//output: [1 0 0 0 2 0 0 0 3 0 0 0 2 0 0 0]
```

Sem pojedinačnih vrednosti (u ovom slučaju strukture Vertex), moguće je direktno upisivati ili čitati i nizove/slice-ove struktura.

Paket predstavlja pogodan način za encoding struktura podataka sa numeričkim vrednostima u njihovu binarnu reprezentaciju. Može predstavljati i osnovu za kreiranje sopstvenih binarnih protokola.

- ## encoding/gob paket

Paket gob deo je standardne Go-ove biblioteke i služi za enkodiranje i dekodiranje kompleksnih struktura podataka. Trenutno je ekskluzivan za Go i ne poseduje implementaciju u nekom drugom programskom jeziku.

Transformacija podataka obavlja se pomoću struktura Encoder i Decoder koje se kreiraju tako što se metodi NewEncoder prosledi io.Writer implementacija, odnosno tako što se metodi NewDecoder prosledi neka vrednost čiji tip implementira io.Reader. Encoder poseduje metodu Encode() kojoj se kao argument prosleđuje bilo šta (što nije funkcija ili channel) i ta vrednost biva enkodirana. Prikazan je primer upisivanja rezultata enkodiranja u fajl.

```
student := Student{"Ketan Parmar",35}
file, err := os.OpenFile("./student.gob", os.O_RDWR, 0777)

encoder := gob.NewEncoder(file)
err = encoder.Encode(student)
if err != nil{
   fmt.Println(err)
}
student.Age = 12
err = encoder.Encode(student)
if err != nil{
   fmt.Println(err)
}
```

Decoder vrši dekodiranje pomoću metode Decode() kojoj se prosleđuje promenljiva u koju će rezultat biti upisan (mora biti prosleđen pokazivač na željeni tip), a sama metoda vraća grešku do koje je potencijalno došlo. Primer prikazuje rezultat dekodiranja sadržaja fajla.

```
decoder := gob.NewDecoder(file)
var studentRead = new(Student)
file.Seek(0, 0)
for  {
   err = decoder.Decode(studentRead)
   if err != nil {
      break
   }
   fmt.Println(*studentRead)
}
```

Prikazani kod je nastavak primera sa početka strane. Nakon instanciranja dekodera za fajl, kreira se pokazivač na studenta u koji će biti upisivane vrednosti. Nakon toga se vrši pozicioniranje na početak fajla kako bi kompletan sadržaj bio iščitan. For petlja zadužena je da prođe kroz sve vrednosti studenta i ispiše ih.

Kada se vrednost određenog tipa enkodira, nije je neophodno dekodirati u vrednost potpuno istog tipa. Ukoliko se određeno polje nalazi u polaznoj strukturi, ali ne u odredišnoj i obrnuto, to ne predstavlja problem. Neophodno je ispoštovati uslov da dve strukture imaju barem neko zajedničko polje, a to znači polje istog naziva i istog tipa (računa se i pokazivač na tip, gob će odraditi potrebnu indirekciju i dereferenciranje pokazivača). Problem nastaje u situacijama kada u preseku dve strukture ne postoji nijedno polje istog naziva ili ukoliko su tipovi takvog polja nekompatibilni.

Ukoliko je polazišna struktura sledećeg oblika:

```
struct { A, B int }
```

sledeće odredišne strukture su validne:

```
struct { A, B int }         // the same
*struct { A, B int }        // extra indirection of the struct
struct { *A, *B int }       // extra indirection of the fields
struct { A, B int64 }       // different concrete value type
struct { B, A int }         // ordering doesn't matter; matching is by name
struct { A, B, C int }      // extra field (C) ignored
struct { B int }            // missing field (A) ignored; data will be dropped
struct { B, C int }         // missing field (A) ignored; extra field (C) ignored.
```

dok navedene nisu:

```
struct { A int; B uint }            // change of signedness for B
struct { A int; B float }           // change of type for B
struct { }                          // no field names in common
struct { C, D int }                 // no field names in common
```

Sve signed celobrojne vrednosti mogu se predstaviti bilo kojim intX tipom sve dok je tu vrednost moguće uskladištiti, u suprotnom dolazi do greške. Isto važi i za uint i float vrednosti.

Kada se vrši enkoding strukture, enkodiraju se samo ona polja koja su exported, dok se unexported polja zanemaruju. Ukoliko se to pak želi promeniti moguće je na nivou tipa implementirati interfejse GobEncoder (sa metodom GobEncode) i GobDecoder (sa metodom GobDecode) čime se može postići enkodiranje/dekodiranje i unexported polja, kao i funkcija i kanala, što inače nije podržano. Kroz ove metode pruža se sopstvena implementacija reprezentacije vrednosti nekog tipa, što daje veću kontrolu nad procesom serijalizacije. Dokumentacija za GobEncoder i GodDecoder interfejse je sledeća:

```
type GobEncoder interface {
        // GobEncode returns a byte slice representing the encoding of the
        // receiver for transmission to a GobDecoder, usually of the same
        // concrete type.
        GobEncode() ([]byte, error)
}

type GobDecoder interface {
        // GobDecode overwrites the receiver, which must be a pointer,
        // with the value represented by the byte slice, which was written
        // by GobEncode, usually for the same concrete type.
        GobDecode([]byte) error
}
```