

COS 316

Precept:

Reflection

Part 2

Embedded Fields

```
type record1 struct {  
    field1 string  
}  
type record2 struct {  
    record1  
    field2 int64  
    field3, field4 float64  
}  
func main() {  
    value := record2{record1{"foo"}, 1, 2, 3}  
    fmt.Println(value.field1)  
    fmt.Println(value.record1.field1)  
}
```

- embedded field since it's declared with type but no name
- "mixin" - record2 gets all the the fields of record1

equivalent to:

```
type record2 struct {  
    field1 string  
    field2 int64  
    field3, field4 float64  
}
```

- Try this example:

<https://play.golang.org/p/gUNLbxvp3Dk>

Tags

```
type record1 struct {  
    field1 string `mytag:"special field"`  
}  
type record2 struct {  
    record1  
    field2    int64    `mytag:"field 2"`  
    field3, field4 float64 `mytag:"field 3"`  
}
```

field declaration may be followed by an optional string literal (**tag**)

Meta information associated with each field

Not much you can do with tags except through using Reflect API

- Try this example:

<https://play.golang.org/p/VjSxP1Cd1tw>

Tags and Reflection

```
type record1 struct {  
    field1 string `mytag:"field 1"`  
}  
type record2 struct {  
    record1  
    field2      int64    `mytag:"field 2"`  
    field3, field4 float64 `mytag:"field 3 & 4"`  
}  
  
func main() {  
    t := reflect.TypeOf(record2{})  
    f1, _ := t.FieldByName("field1")  
    fmt.Println(f1.Tag)  
    f2, _ := t.FieldByName("field2")  
    fmt.Println(f2.Tag)  
    f3, _ := t.FieldByName("field3")  
    fmt.Println(f3.Tag)  
    f4, _ := t.FieldByName("field4")  
    fmt.Println(f4.Tag)  
}
```

returns the struct type
with the given name

returns the field with the
given name

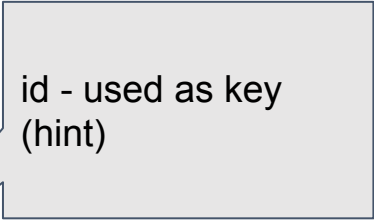
- Try this example:

<https://play.golang.org/p/MBFTjBWTEUT>

returns the tag for a
given field

Tags - Uses

- May want to indicate a field is acting as a primary key



id - used as key
(hint)

```
type movie struct {  
    id int        `dorm:"key"`  
    title string  // title of movie  
    year int      // movie release year  
    genre string  // pipe-separated list of genres  
}
```

- Map between json and struct fields*

```
type Person struct {  
    FirstName string `json:"first_name"`  
    LastName  string `json:"last_name"`  
    MiddleName string `json:"middle_name,omitempty"`  
}  
  
func main {  
    json_string := `  
    {  
        "first_name": "John",  
        "last_name": "Smith"  
    }`  
  
    person := new(Person)  
    json.Unmarshal([]byte(json_string), person)  
    fmt.Println(person)  
  
    new_json, _ := json.Marshal(person)  
    fmt.Printf("%s\n", new_json)  
}
```

- Try this example:

<https://play.golang.org/p/AVb5gQsWUrD>

Recall - reflect.TypeOf, reflect.Kind, reflect.ValueOf

```
package main
```

```
import (  
    "fmt"  
    "reflect")
```

```
type movie struct {  
    id      int    // represents the movie id  
    title   string // title of movie  
    year    int    // movie release year  
    genre   string // pipe-separated list of genres  
}
```

```
func createQuery(q interface{}) {  
    t := reflect.TypeOf(q) // returns type representation  
    k := t.Kind()           // specific kind of type  
    v := reflect.ValueOf(q) // concrete value stored in type  
    fmt.Println("TypeOf  ", t)  
    fmt.Println("Kind    ", k)  
    fmt.Println("ValueOf ", v)  
}
```

```
func main() {  
    joker := movie {movieId: 193612, title:"Joker",  
                    year: 2019, genre: "action|thriller"}  
    createQuery(joker)  
}
```

- <https://play.golang.org/p/eglNsDQs9Hz>

What happens if we pass a reference (instead of a value)

```
package main
```

```
import (  
    "fmt"  
    "reflect")
```

```
type movie struct {  
    id      int    // represents the movie id  
    title   string // title of movie  
    year    int    // movie release year  
    genre   string // pipe-separated list of genres  
}
```

```
func createQuery(q interface{}) {  
    t := reflect.TypeOf(q) // returns type representation  
    k := t.Kind()           // specific kind of type  
    v := reflect.ValueOf(q) // concrete value stored in type  
    fmt.Println("TypeOf ", t)  
    fmt.Println("Kind   ", k)  
    fmt.Println("ValueOf ", v)  
}
```

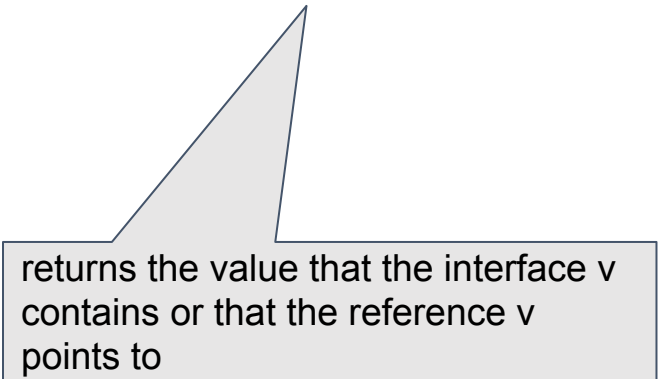
```
func main() {  
    joker := movie {movieId: 193612, title:"Joker",  
                    year: 2019, genre: "action|thriller"}  
    createQuery(&joker)  
}
```

- <https://play.golang.org/p/z16zPNPactX>

Pass
reference of
object

How to get the value at an reference?

- `reflect.Elem(v Value) Value`



returns the value that the interface v contains or that the reference v points to

- Try this example:

<https://play.golang.org/p/hDd05IX1DH2>

```
type Movie struct {
    Id      int
    Title   string
    Year    int
    Genre   string
}

func examine (pointer interface{}) {
    typePointer := reflect.TypeOf(pointer)
    fmt.Print(typePointer)
    fmt.Print(" ")
    fmt.Println(typePointer.Kind())

    addr := reflect.ValueOf(pointer)
    fmt.Println(addr)

    typeDeref := typePointer.Elem()
    fmt.Print(typeDeref)
    fmt.Print(" ")
    fmt.Println(typeDeref.Kind())

    valueDeref := addr.Elem()
    fmt.Println(valueDeref)
}

func main() {
    joker := Movie{Id: 193612, Title: "Joker",
                   Year: 2019, Genre: "action|thriller"}
    examine(&joker)

    i := 316
    examine(&i)
}
```


Final Project!