
Reflections



The `reflections` library provides high-level abstractions on top of the go language standard `reflect` library.

In practice, the `reflect` library's API proves somewhat low-level and un-intuitive. Using it can turn out pretty complex, daunting, and scary, especially when doing simple things like accessing a structure field value, a field tag, etc.

The `reflections` package aims to make developers' life easier when it comes to introspect struct values at runtime. Its API takes inspiration in the python language's `getattr`, `setattr`, and `hasattr` set of methods and provides simplified access to structure fields and tags.

- Reflections
 - Documentation
 - Usage
 - * `GetField`
 - * `GetFieldKind`
 - * `GetFieldType`
 - * `GetFieldTag`
 - * `HasField`
 - * `Fields`
 - * `Items`
 - * `Tags`
 - * `GetFieldNameByTagValue`
 - Important notes
 - Contribute

Documentation

Head to the documentation to get more details on the library's API.

Usage

GetField

`GetField` returns the content of a structure field. For example, it proves beneficial when you want to iterate over struct-specific field values. You can provide `GetField` a structure or a pointer to a

struct as the first argument.

```
1 s := MyStruct {
2     FirstField: "first value",
3     SecondField: 2,
4     ThirdField: "third value",
5 }
6
7 fieldsToExtract := []string{"FirstField", "ThirdField"}
8
9 for _, fieldName := range fieldsToExtract {
10     value, err := reflections.GetField(s, fieldName)
11     DoWhateverWithThatValue(value)
12 }
```

GetFieldKind

`GetFieldKind` returns the `reflect.Kind` of a structure field. You can use it to operate type assertion over a structure field at runtime. You can provide `GetFieldKind` a structure or a pointer to structure as the first argument.

```
1 s := MyStruct{
2     FirstField: "first value",
3     SecondField: 2,
4     ThirdField: "third value",
5 }
6
7 var firstFieldKind reflect.String
8 var secondFieldKind reflect.Int
9 var err error
10
11 firstFieldKind, err = GetFieldKind(s, "FirstField")
12 if err != nil {
13     log.Fatal(err)
14 }
15
16 secondFieldKind, err = GetFieldKind(s, "SecondField")
17 if err != nil {
18     log.Fatal(err)
19 }
```

GetFieldType

`GetFieldType` returns the string literal of a structure field type. You can use it to operate type assertion over a structure field at runtime. You can provide `GetFieldType` a structure or a pointer to structure as the first argument.

```
1 s := MyStruct{
2     FirstField: "first value",
3     SecondField: 2,
4     ThirdField: "third value",
5 }
6
7 var firstFieldKind string
8 var secondFieldKind string
9 var err error
10
11 firstFieldKind, err = GetFieldType(s, "FirstField")
12 if err != nil {
13     log.Fatal(err)
14 }
15
16 secondFieldKind, err = GetFieldType(s, "SecondField")
17 if err != nil {
18     log.Fatal(err)
19 }
```

GetFieldTag

`GetFieldTag` extracts a specific structure field tag. You can provide `GetFieldTag` a structure or a pointer to structure as the first argument.

```
1 s := MyStruct{}
2
3 tag, err := reflections.GetFieldTag(s, "FirstField", "matched")
4 if err != nil {
5     log.Fatal(err)
6 }
7 fmt.Println(tag)
8
9 tag, err = reflections.GetFieldTag(s, "ThirdField", "unmatched")
10 if err != nil {
11     log.Fatal(err)
12 }
13 fmt.Println(tag)
```

HasField

`HasField` asserts a field exists through the structure. You can provide `HasField` a struct or a pointer to a struct as the first argument.

```
1 s := MyStruct {
2     FirstField: "first value",
```

```
3     SecondField: 2,  
4     ThirdField: "third value",  
5 }  
6  
7 // has == true  
8 has, _ := reflections.HasField(s, "FirstField")  
9  
10 // has == false  
11 has, _ := reflections.HasField(s, "FourthField")
```

Fields

`Fields` returns the list of structure field names so that you can access or update them later. You can provide `Fields` with a struct or a pointer to a struct as the first argument.

```
1 s := MyStruct {  
2     FirstField: "first value",  
3     SecondField: 2,  
4     ThirdField: "third value",  
5 }  
6  
7 var fields []string  
8  
9 // Fields will list every structure exportable fields.  
10 // Here, it's content would be equal to:  
11 // []string{"FirstField", "SecondField", "ThirdField"}  
12 fields, _ = reflections.Fields(s)
```

Items

`Items` returns the structure's field name to the values map. You can provide `Items` with a struct or a pointer to structure as the first argument.

```
1 s := MyStruct {  
2     FirstField: "first value",  
3     SecondField: 2,  
4     ThirdField: "third value",  
5 }  
6  
7 var structItems map[string]interface{}  
8  
9 // Items will return a field name to  
10 // field value map  
11 structItems, _ = reflections.Items(s)
```

Tags

`Tags` returns the structure's fields tag with the provided key. You can provide `Tags` with a struct or a pointer to a struct as the first argument.

```
1 s := MyStruct {
2     FirstField: "first value",    `matched:"first tag"`
3     SecondField: 2,              `matched:"second tag"`
4     ThirdField: "third value",    `unmatched:"third tag"`
5 }
6
7 var structTags map[string]string
8
9 // Tags will return a field name to tag content
10 // map. N.B that only field with the tag name
11 // you've provided will be matched.
12 // Here structTags will contain:
13 // {
14 //   "FirstField": "first tag",
15 //   "SecondField": "second tag",
16 // }
17 structTags, _ = reflections.Tags(s, "matched")
```

SetField

`SetField` updates a structure's field value with the one provided. Note that you can't set un-exported fields and that the field and value types must match.

```
1 s := MyStruct {
2     FirstField: "first value",
3     SecondField: 2,
4     ThirdField: "third value",
5 }
6
7 //To be able to set the structure's values,
8 // it must be passed as a pointer.
9 _ := reflections.SetField(&s, "FirstField", "new value")
10
11 // If you try to set a field's value using the wrong type,
12 // an error will be returned
13 err := reflection.SetField(&s, "FirstField", 123) // err != nil
```

GetFieldNameByTagValue

`GetFieldNameByTagValue` looks up a field with a matching `{tagKey}:{tagValue}` tag in the provided `obj` item. If `obj` is not a `struct`, nor a `pointer`, or it does not have a field tagged

with the `tagKey`, and the matching `tagValue`, this function returns an error.

```
1 s := MyStruct {
2     FirstField: "first value",    `matched:"first tag"`
3     SecondField: 2,              `matched:"second tag"`
4     ThirdField: "third value",    `unmatched:"third tag"`
5 }
6
7 // Getting field name from external source as json would be a headache
8 // to convert it manually,
9 // so we get it directly from struct tag
10 // returns fieldName = "FirstField"
11 fieldName, _ = reflections.GetFieldNameByTagValue(s, "matched", "first
12 tag");
13
14 // later we can do GetField(s, fieldName)
```

Important notes

- **Un-exported fields** can't be accessed nor set using the `reflections` library. The Go lang standard `reflect` library intentionally prohibits un-exported fields values access or modifications.

Contribute

- Check for open issues or open a new issue to start a discussion around a feature idea or a bug.
- Fork the repository on GitHub to start making your changes to the **master** branch, or branch off of it.
- Write tests showing that the bug was fixed or the feature works as expected.
- Send a pull request and bug the maintainer until it gets merged and published. :) Make sure to add yourself to [AUTHORS](#).