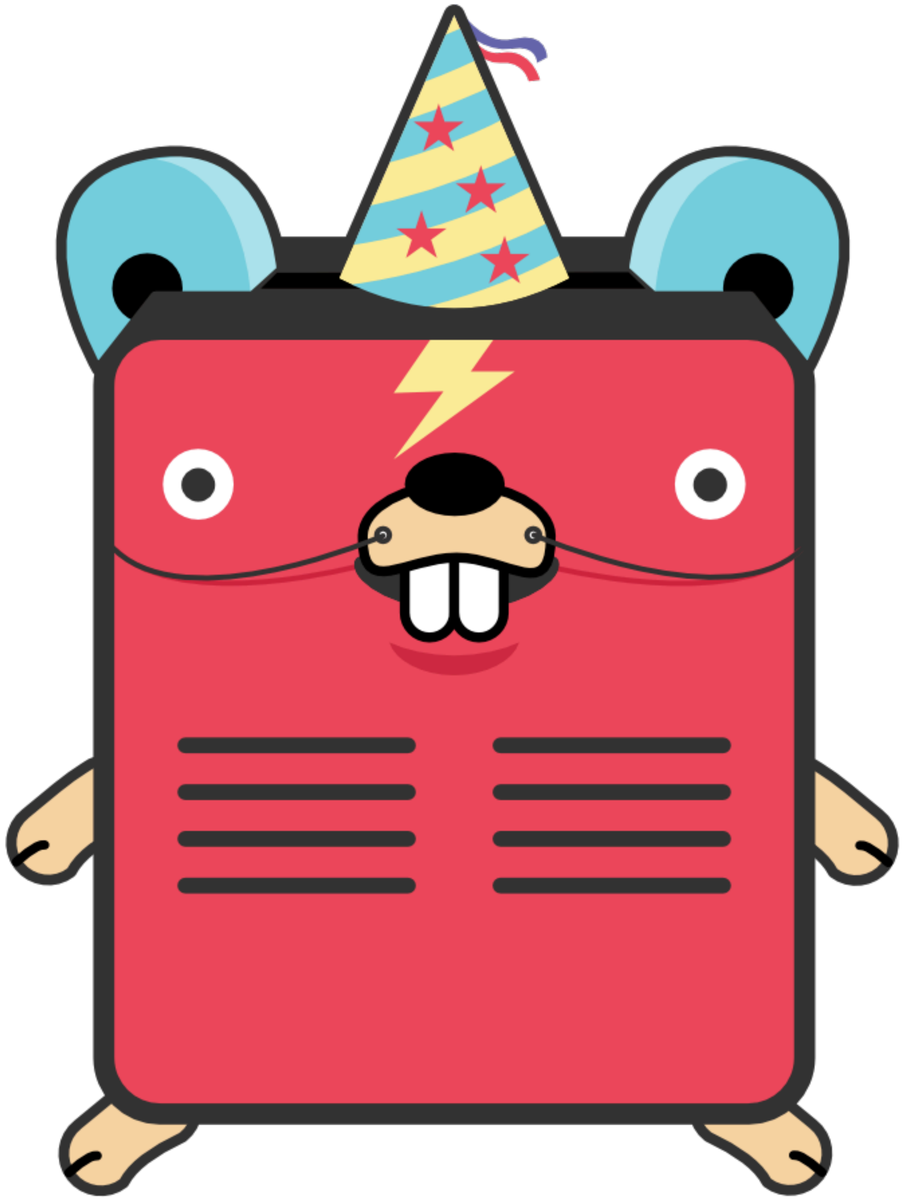
Using Go to Guide API Design Decisions

dotGo 2016 lightning talk



This is Tiny P.,

our little resolver,

pretending to be a Gopher.

2 years ago we started to redesign our HTTP API.

We decided to develop official clients in several languages.

The Go client was leading the development.

My approach was to develop a feature in Go, and then in other languages.

Writing Go is certainly a different way of coding.

Some design decisions we made in the past also made the API harder to use with certain kinds of languages.

Let me show a few examples.

Consuming our API with Go allowed us to understand flaws in our response formats and design.

```json
{
  "name": "www",
  "type": "A",
  "content": "127.0.0.1",
  "regions": "global"
}

{
  "name": "www",
  "type": "A",
  "content": "127.0.0.1",
  "regions": ["TKO", "AMS"]
}
```

```go
// ZoneRecord represents a DNS record in DNSimple.
type ZoneRecord struct {
  ID          int      `json:"id,omitempty"`
  Type        string   `json:"type,omitempty"`
  Name        string   `json:"name",omitempty`
  Content     string   `json:"content,omitempty"`
  // ...
  Regions     []string `json:"regions,omitempty"`
}
```

```json
{
  "name": "www",
  "type": "A",
  "content": "127.0.0.1",
  "regions": ["global"]
}

{
  "name": "www",
  "type": "A",
  "content": "127.0.0.1",
  "regions": ["TKO", "AMS"]
}
```

```go
// ZoneRecord represents a DNS record in DNSimple.
type ZoneRecord struct {
  ID          int      `json:"id,omitempty"`
  Type        string   `json:"type,omitempty"`
  Name        string   `json:"name",omitempty`
  Content     string   `json:"content,omitempty"`
  // ...
  Regions     []string `json:"regions,omitempty"`
}
```

```go
type Domain struct {
    ID           int    `json:"id,omitempty"`
    Name         string `json:"name,omitempty"`
    // ...
}

// Both accountIdentifier and domainIdentifier
// may be either an int ID or string Identifier.
func (s *DomainsService) GetDomain(accountIdentifier, domainIdentifier string)
(*DomainResponse, error) {
    // ...
}
```

```go
type Domain struct {
    ID           int    `json:"id,omitempty"`
    Name         string `json:"name,omitempty"`
    // ...
}

// Both accountIdentifier and domainIdentifier
// may be either an int ID or string Identifier.
func (s *DomainsService) GetDomain(accountIdentifier, domainIdentifier string)
(*DomainResponse, error) {
    // ...
}
```

```go
// ListDomains lists the domains for an account.
func (s *DomainsService) ListDomains(ID string, o *DomainListOptions) (*DomainsResponse, error) {
    // ...
}

// DomainListOptions specifies the optional parameters you can provide
// to customize the DomainsService.ListDomains method.
type DomainListOptions struct {
    NameLike string `url:"name_like,omitempty"`
    RegistrantID int `url:"registrant_id,omitempty"`

    ListOptions
}

// ListOptions contains the common options you can pass to a List method
// in order to control parameters such as paginations and page number.
type ListOptions struct {
    Page int `url:"page,omitempty"`
    PerPage int `url:"per_page,omitempty"`
    Sort string `url:"sort,omitempty"`
}
```

The simplicity of Go,
along with some of its constraints,
are features that forced us to rethink
the way we designed our API.

We used our Go client as the leading implementation.

```go
// Get the list of domains in Go
domainsResponse, err := client.Domains.ListDomains(accountID, nil)
```

```ruby
# Get the list of domains in Ruby
response = client.domains.list_domains(account_id)
```

```elixir
# Get the list of domains in Elixir
{:ok, response} = Dnsimple.Domains.list_domains(%Dnsimple.Client{}, account_id)
```

```node
// Get the list of domains in Node.js
client.domains.listDomains(accountId).then(function(response) { });
```

```java
// Get the list of domains in Java
ListDomainsResponse response = client.domains.listDomains(accountId);
```

```go
// Get the list of domains in Go
domainsResponse, err := client.Domains.ListDomains(accountID, nil)
```

```ruby
# Get the list of domains in Ruby
response = client.domains.list_domains(account_id)
```

```elixir
# Get the list of domains in Elixir
{:ok, response} = Dnsimple.Domains.list_domains(%Dnsimple.Client{}, account_id)
```

```node
// Get the list of domains in Node.js
client.domains.listDomains(accountId).then(function(response) { });
```

```java
// Get the list of domains in Java
ListDomainsResponse response = client.domains.listDomains(accountId);
```

Thanks!

Simone Carletti

@weppos

https://dnsimple.com/dotgo