The Pragmatic Programmers

Effective Go Recipes

Fast Solutions to Common Tasks

Miki Tebeka

edited by Margaret Eldridge

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Using Composite Keys in Maps

Task

As a hobby, you work on some algorithmic trading code. After a gaming night with your friend Bob, who works in a bank, you decide you need a fast in-memory database for the stock information. You're going to use this database to train stock-trading algorithms that will make you rich!

Stock Trading

This is in no way a recommendation to trade the stock market. I worked for several years in high-frequency trading companies, and the volume of information we had gave us a huge advantage over casual traders.

Automated trading is very risky—the Knight Capital Group lost $440 million in forty-five minutes due to a bug. Do you think you can stand such losses?

Solution

The information for a stock is the opening, low, high, and closing price of the day (known as OHLC).

types/stock_db/stock_db.go

```
// StockInfo is information about stock at a given date.
type StockInfo struct {
    Date   time.Time
    Symbol string
    Open   float64
    High   float64
    Low    float64
    Close  float64
}
```

You want fast access for stock information at a given date, and you decide to use a map and have a struct as a composite key. This is the key:

types/stock_db/stock_db.go

```
type key struct {
    year   int
```

```
    month  time.Month
    day    int
    symbol string
}
```

Then you write an InfoDB struct that will hold the map:

types/stock_db/stock_db.go

```
// InfoDB is in memory stock database.
type InfoDB struct {
    m map[key]StockInfo
}
```

InfoDB provides a Get function that will return stock information for given stock at a given date:

types/stock_db/stock_db.go

```
// Get return information for stock at date. If information is not found the
// second return value will be false.
func (i *InfoDB) Get(symbol string, date time.Time) (StockInfo, bool) {
    k := key{date.Year(), date.Month(), date.Day(), symbol}
    info, ok := i.m[k]
    return info, ok
}
```

Discussion

Most of the time, map keys will be a simple type, but in some cases, you'll need a composite key. A composite key is a key that consists of two or more attributes.

I've seen people who, instead of using composite keys, create a string that represents the composite key. For example, for our database case you might construct a string key with values such as "AAPL:20200302". This approach is error prone; you're inventing a new serialization format and might create the same string for two different keys.

For example, assume that a user is defined by its login name and a user ID. If you create the string "joe42", it might be user joe with user ID 42 or the user joe4 with the ID 2.

We're taking advantage of the fact that a Go struct is comparable if all of its fields are comparable. You can use these structs as map keys without figuring out a bulletproof way of creating strings to represent keys.

Some types, such as slices, are not comparable and can't be used in composite keys. If you need to use a slice as a map key, or a part of a map key, convert

it to a string first. However, since a user might change the original slice, it's better not to use slices as map keys at all.

In general, prefer types that can't be changed (immutable), such as string, int, or rune, as map keys.

You decided not to use time.Time as part of the compound key since comparing with time.Time can be tricky. See Recipe 36, Parsing Time Strings, on page ?, for more information.