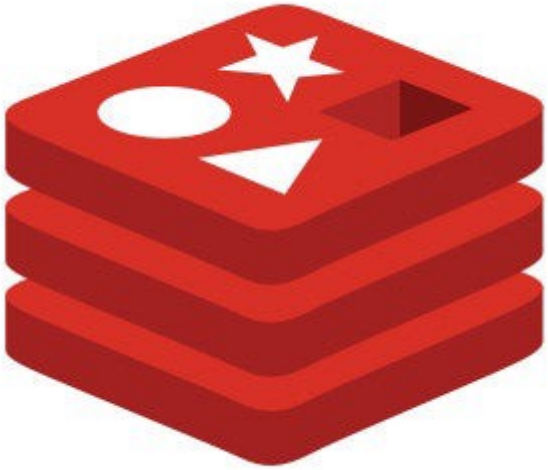
Zayn Korai Follow
Jan 15 · 3 min read · Listen

Golang with Redis 101



Redis is an in-memory data store that is commonly used for caching, real-time data processing, and message queues. It supports a wide range of data structures such as strings, hashes, lists, and sets, making it a versatile and powerful tool for a variety of use cases.

Golang, also known as Go, is a modern programming language that is known for its simplicity, performance, and scalability. It is often used for building high-performance and concurrent systems, making it a great choice for working with Redis.

To interact with Redis from Golang, you can use a popular Redis client library called go-redis. This library provides a convenient and powerful API for interacting with

Redis, including the abil[illegible] as well as perform more complex operations suc[illegible]

Here is an example of h[illegible]ey-value pair in Redis:

```
package main

import (
    "fmt"
    "github.com/go-redis/redis/v8"
)

func main() {
    // Connect to Redis
    client := redis.NewClient(&redis.Options{
        Addr: "localhost:6379",
    })

    // Set a key-value pair
    err := client.Set(context.Background(), "mykey", "myvalue", 0).Err()
    if err != nil {
        panic(err)
    }

    fmt.Println("Key-value pair set!")
}
```

This example shows how to use the go-redis library to create a new Redis client, set a key-value pair, and then retrieve the value. You can also use this library to perform other operations such as getting, deleting, and expiring keys, as well as more advanced operations like transactions and Lua scripting.

This example shows how to use the go-redis library to create a new Redis client, create a key-value pair, read the value of the key, update the value of the key, and delete the key. It also illustrates how to handle errors that can occur during these operations.

```
package main

import (
    "context"
    "fmt"
```

```
    "github.com/go-redis/redis/v8"
)

func main() {
    // Connect to R
    client := redis.NewClient(&redis.Options{
        Addr: "localhost:6379",
    })

    // Create a key-value pair
    err := client.Set(context.Background(), "mykey", "myvalue", 0).Err()
    if err != nil {
        panic(err)
    }
    fmt.Println("Key-value pair created!")

    // Read the value of the key
    val, err := client.Get(context.Background(), "mykey").Result()
    if err != nil {
        panic(err)
    }
    fmt.Println("Value of the key:", val)

    // Update the value of the key
    err = client.Set(context.Background(), "mykey", "newvalue", 0).Err()
    if err != nil {
        panic(err)
    }
    fmt.Println("Value of the key updated!")

    // Delete the key
    err = client.Del(context.Background(), "mykey").Err()
    if err != nil {
        panic(err)
    }
    fmt.Println("Key deleted!")
}
```


In addition, Redis also supports Publish/Subscribe pattern, which is a pattern where senders (in redis called publishers) sends messages without the need of a receiver and receivers (in redis called subscribers) receives messages without the need of a sender. This can be useful in cases where multiple systems need to communicate with each other in real-time.

In conclusion, Redis and Golang are a great combination for building high-performance and scalable systems. The go-redis library provides a convenient and powerful API for interacting with Redis, making it easy to work with this powerful

data store from Golang. With the addition of Pub/Sub pattern, it becomes a powerful tool for real-time data p

Golang Redis Cache Crud