Concurrency Patterns in Go

Introduction to Concurrency in Go

Go provides rich support for concurrency using goroutines and channels. Concurrency is not parallelism, but it enables better utilization of resources.

Goroutines

Goroutines are lightweight threads managed by the Go runtime.

Usage: go func() { ... }()

They allow asynchronous execution of functions.

Worker Pool Pattern

A worker pool is a collection of goroutines that process tasks from a shared channel.

Example:

```
jobs := make(chan int, 100)
results := make(chan int, 100)

for w := 1; w <= 3; w++ {
  go worker(w, jobs, results)
}
for j := 1; j <= 5; j++ {
  jobs <- j
}
close(jobs)
```

Fan-Out, Fan-In Pattern

Fan-Out: Multiple functions read from the same input channel and perform parallel computation.

Fan-In: Multiple channels are multiplexed into a single channel.

Example:

```
out1 := square(in)
out2 := square(in)
for n := range merge(out1, out2) {
  fmt.Println(n)
}
```

Context Package - Introduction

The context package in Go is used to carry deadlines, cancellation signals, and other request-scoped values across API boundaries.

Context - WithCancel

```
ctx, cancel := context.WithCancel(context.Background())
go func(ctx context.Context) {
  <-ctx.Done()
  fmt.Println("Goroutine exited")
}(ctx)
cancel()
```

Context - WithTimeout & WithDeadline

```
ctx, cancel := context.WithTimeout(context.Background(), 2*time.Second)
defer cancel()
select {
case <-time.After(3 * time.Second):
  fmt.Println("Done")
case <-ctx.Done():
  fmt.Println("Timeout exceeded")
}
```

Context - WithValue

```
ctx := context.WithValue(context.Background(), "userID", 12345)
val := ctx.Value("userID")
fmt.Println("User ID:", val)
```

Best Practices

- Always cancel context to release resources
- Avoid using context.Value for passing optional parameters
- Use context for controlling lifecycles of goroutines

Conclusion

Concurrency in Go is powerful when used correctly. Using patterns like worker pool, fan-out/fan-in, and proper context usage can help write robust and efficient programs.