Alibaba Go Compile Instrumentation Internals

Yi Yang

2025.2.10

Goals

- No code modification should be required from users
- Use same injection mechanism for both standard libraries and third-party libraries
- Instrumentation should be version-aware, we can inject different hook codes for different versions of third-party libraries
- The modified or generated code should be inspectable and debuggable
- Instrumentation point should be configurable by user at compile time

Project Structure

```
$ tree –d .
├──── docs
├──── example
├──── pkg                          # sdk
│     ├──── api
│     ├──── inst-api               # encapsulation of instrumentation
│     ├──── inst-api-semconv       #  encapsulation of OpenTelemetry semconv
│     └──── rules                  #  hook code
├──── test
└──── tool                         # instrumentation tool
```

We put both hook code and instrumentation tool into **one repository**

otel-java-instrumentation puts them into **one repository and splits into different modules**

Orchestration puts them into **different repositories**

This is a point of divergence that needs to be discussed and resolved.

Workflow

Prefixing otel with original go build command

otel go build –o cmd/app

Copy hook code into project and configure them properly

Preprocess

Intercept go build with –toolexec. Find interested compile command and inject our code

go build –toolexec –o cmd/app

We start by adding **otel** prefix to original user build command

Something like **sudo** command

It's easy to use and easy to change their build pipeline

Workflow

Prefixing otel with original go build command

otel go build –o cmd/app

Copy hook code into project and configure them properly

Preprocess

Run dry build(go build –a –x –n) to find all dependencies

Match dependencies with user defined rules.json

If matched, copy hook codes to project and generate necessary code

Intercept go build with –toolexec. Find interested compile command and inject our code

go build –toolexec –o cmd/app

```
{
"ImportPath": "net/http",
"Function": "RoundTrip",
"ReceiverType": "*Transport",
"OnEnter": "clientOnEnter",
"OnExit": "clientOnExit",
"Path": ”/path/to/code"
}
```

(or “StructType” “StructName” fields for struct manipulation)

Workflow

Prefixing otel with original go build command

otel go build –o cmd/app

Copy hook code into project and configure them properly

Preprocess

Intercept go build with –toolexec. Find interested compile command and inject our code

go build –toolexec –o cmd/app

Invoke go build with –toolexec, it intercepts all build commands

Find our interested build commands

Code injection

Invoke compilation command with injected code

```
compile –p slices a.go b.go
link –p slices a.o b.o
compile –p net/http a.go b.go
link –p net/http a.o b.o
compile –p runtime a.go b.go
link –p runtime a.o b.o
```

we only interested in net/http compilation command

```
compile –p slices a.go b.go
link –p slices a.o b.o
compile –p net/http a’.go b’.go generated.go
link –p a.o b.o
compile –p runtime a.go b.go
link –p runtime a.o b.o
```

Code Injection

compile –p net/http a.go b.go → compile –p net/http a'.go b'.go generated.go

inject **complex-if** statement at func entry, jump to trampoline func

```go
func (t *Transport) RoundTrip(req *Request) (retVal0 *Response, retVal1 error) {
    if callContext91363, _ := OtelOnEnterTrampoline_RoundTrip91363(&t, &req); false {
    } else {
        defer OtelOnExitTrampoline_RoundTrip91363(callContext91363, &retVal0, &retVal1)
        if callContext21479, _ := OtelOnEnterTrampoline_RoundTrip21479(&t, &req); false {
        } else {
            defer OtelOnExitTrampoline_RoundTrip21479(callContext21479, &retVal0, &retVal1)
        }
    }
    return t.roundTrip(req)
}
```

```go
func OtelOnEnterTrampoline_RoundTrip91363(t **Transport, req **Request) (CallContext, bool) {
    defer func() {
        if err := recover(); err != nil {
            println("failed to exec onEnter hook", "httpClientEnterHook")
            if e, ok := err.(error); ok {
                println(e.Error())
            }
            fetchStack, printStack := OtelGetStackImpl, OtelPrintStackImpl
            if fetchStack != nil && printStack != nil {
                printStack(fetchStack())
            }
        }
    }()
    callContext := &CallContextImpl91363{}
    callContext.Params = []interface{}{t, req}
    if HttpClientEnterHookImpl != nil {
        HttpClientEnterHookImpl(callContext, *t, *req)
    }
    return callContext, callContext.SkipCall
}

var HttpClientEnterHookImpl func(callContext CallContext, t *Transport, req *Request)
var HttpClientExitHookImpl func(callContext CallContext, retVal0 *Response, retVal1 error)
```

```go
func httpClientEnterHook(call api.CallContext, t *http.Transport, req *http.Request) {
    header, _ := json.Marshal(req.Header)
    fmt.Println("request header is ", string(header))
}
```

Within trampoline func, we catch all panic to ensure that, even in the worst case, our hook code doesn't crash user application. Then we set up the context and jump to hook func

```
func init() {
 HttpClientEnterHookImpl = pkga.httpClientEnterHook
 HttpClientExitHookImpl = pkgb.httpClientExitHook
}
```

Code Injection

Target Function:

```
func (t *Transport) RoundTrip(req *Request) (*Response, error) {
        return t.roundTrip(req)
}
```

Hook Function:

```
func httpClientEnterHook(call api.CallContext, t *http.Transport, req *http.Request) {
  call.SetKeyData("hello", "world")
}
func httpClientExitHook(call api.CallContext, res *http.Response, err error) {
  ...
  api.SetReturnVal(1, ...)
  api.SetParam(1, ...)
  api.GetKeyData("hello")
}
```

```
type CallContext interface {
        SetSkipCall(bool)
        IsSkipCall() bool
        SetData(interface{})
        GetData() interface{}
        GetKeyData(key string) interface{}
        SetKeyData(key string, val interface{})
        HasKeyData(key string) bool
        GetParam(idx int) interface{}
        SetParam(idx int, val interface{})
        GetReturnVal(idx int) interface{}
        SetReturnVal(idx int, val interface{})
}
```

We can change target function's parameter value or return value by using CallContext APIs. Furthermore, we can carry data between enter and exit hook.

Live Demo