# cmd\prometheus\main.go

E:\workspace\go\prometheus\prometheus-fork\cmd\prometheus\main.go

webHandler := web.New(&cfg.web)

# web\web.go

E:\workspace\yh\OpenBridge-passos-proxy\open-falcon\src\github.com\prometheus\prometheus\web\web.go

## Web.go.New

// New initializes a new web Handler.  
**func** New(o \*Options) \*Handler {  
 router := route.New()  
 cwd, err := os.Getwd()  
  
 **if** err != nil {  
 cwd = "<error retrieving current working directory>"  
 }  
  
 h := &Handler{  
 router: router,  
 listenErrCh: make(**chan** error),  
 quitCh: make(**chan struct**{}),  
 reloadCh: make(**chan chan** error),  
 options: o,  
 versionInfo: o.Version,  
 birth: time.Now(),  
 cwd: cwd,  
 flagsMap: o.Flags,  
  
 context: o.Context,  
 targetManager: o.TargetManager,  
 ruleManager: o.RuleManager,  
 queryEngine: o.QueryEngine,  
 storage: o.Storage,  
 notifier: o.Notifier,  
  
 apiV1: api\_v1.NewAPI(o.QueryEngine, o.Storage, o.TargetManager, o.Notifier),  
 now: model.Now,  
 }  
  
 **if** o.RoutePrefix != "/" {  
 // If the prefix is missing for the root path, prepend it.  
 router.Get("/", **func**(w http.ResponseWriter, r \*http.Request) {  
 http.Redirect(w, r, o.RoutePrefix, http.*StatusFound*)  
 })  
 router = router.WithPrefix(o.RoutePrefix)  
 }  
  
 instrh := prometheus.InstrumentHandler  
 instrf := prometheus.InstrumentHandlerFunc  
  
 router.Get("/", **func**(w http.ResponseWriter, r \*http.Request) {  
 router.Redirect(w, r, path.Join(o.ExternalURL.Path, "/graph"), http.*StatusFound*)  
 })  
  
 router.Get("/alerts", instrf("alerts", h.alerts))  
 router.Get("/graph", instrf("graph", h.graph))  
 router.Get("/status", instrf("status", h.status))  
 router.Get("/flags", instrf("flags", h.flags))  
 router.Get("/config", instrf("config", h.config))  
 router.Get("/rules", instrf("rules", h.rules))  
 router.Get("/targets", instrf("targets", h.targets))  
 router.Get("/version", instrf("version", h.version))  
  
 router.Get("/heap", instrf("heap", dumpHeap))  
  
 router.Get(o.MetricsPath, prometheus.Handler().ServeHTTP)  
  
 router.Get("/federate", instrh("federate", httputil.CompressionHandler{  
 Handler: http.HandlerFunc(h.federation),  
 }))  
  
 h.apiV1.Register(router.WithPrefix("/api/v1"))  
  
 router.Get("/consoles/\*filepath", instrf("consoles", h.consoles))  
  
 router.Get("/static/\*filepath", instrf("static", serveStaticAsset))  
  
 **if** o.UserAssetsPath != "" {  
 router.Get("/user/\*filepath", instrf("user", route.FileServe(o.UserAssetsPath)))  
 }  
  
 **if** o.EnableQuit {  
 router.Post("/-/quit", h.quit)  
 }  
  
 router.Post("/-/reload", h.reload)  
 router.Get("/-/reload", **func**(w http.ResponseWriter, r \*http.Request) {  
 w.WriteHeader(http.*StatusMethodNotAllowed*)  
 fmt.Fprintf(w, "This endpoint requires a POST request.\n")  
 })  
  
 router.Get("/debug/\*subpath", http.DefaultServeMux.ServeHTTP)  
 router.Post("/debug/\*subpath", http.DefaultServeMux.ServeHTTP)  
  
 **return** h  
}

### 调用

#### api\_v1.NewAPI

apiV1: api\_v1.NewAPI(o.QueryEngine, o.Storage, o.TargetManager, o.Notifier),

#### apiV1.Register

//添加前缀/api/v1

h.apiV1.Register(router.WithPrefix("/api/v1"))

# web\api\v1\api.go

E:\workspace\go\prometheus\prometheus-fork\web\api\v1\api.go

## api.go.NewAPI

// NewAPI returns an initialized API type.  
**func** NewAPI(qe \*promql.Engine, st local.Storage, tr targetRetriever, ar alertmanagerRetriever) \*API {  
 **return** &API{  
 QueryEngine: qe,  
 Storage: st,  
 targetRetriever: tr,  
 alertmanagerRetriever: ar,  
 now: model.Now,  
 }  
}

## API.Register

// Register the API's endpoints in the given router.  
**func** (api \*API) Register(r \*route.Router) {  
 instr := **func**(name string, f apiFunc) http.HandlerFunc {  
 hf := http.HandlerFunc(**func**(w http.ResponseWriter, r \*http.Request) {  
 setCORS(w)  
 **if** data, err := f(r); err != nil {  
 respondError(w, err, data)  
 } **else if** data != nil {  
 respond(w, data)  
 } **else** {  
 w.WriteHeader(http.*StatusNoContent*)  
 }  
 })  
 **return** prometheus.InstrumentHandler(name, httputil.CompressionHandler{  
 Handler: hf,  
 })  
 }  
  
 r.Options("/\*path", instr("options", api.options))  
  
 r.Get("/query", instr("query", api.query))  
 r.Get("/query\_range", instr("query\_range", api.queryRange))  
  
 r.Get("/label/:name/values", instr("label\_values", api.labelValues))  
  
 r.Get("/series", instr("series", api.series))  
 r.Del("/series", instr("drop\_series", api.dropSeries))  
  
 r.Get("/targets", instr("targets", api.targets))  
 r.Get("/alertmanagers", instr("alertmanagers", api.alertmanagers))  
}