# api\api.go

E:\workspace\go\prometheus\alertmanager\api\api.go

## API.Register

// Register registers the API handlers under their correct routes  
// in the given router.  
**func** (api \*API) Register(r \*route.Router) {  
 ihf := **func**(name string, f http.HandlerFunc) http.HandlerFunc {  
 **return** prometheus.InstrumentHandlerFunc(name, **func**(w http.ResponseWriter, r \*http.Request) {  
 setCORS(w)  
 f(w, r)  
 })  
 }  
  
 r.Options("/\*path", ihf("options", **func**(w http.ResponseWriter, r \*http.Request) {}))  
  
 // Register legacy forwarder for alert pushing.  
 r.Post("/alerts", ihf("legacy\_add\_alerts", api.legacyAddAlerts))  
  
 // Register actual API.  
 r = r.WithPrefix("/v1")  
  
 r.Get("/status", ihf("status", api.status))  
 r.Get("/alerts/groups", ihf("alert\_groups", api.alertGroups))  
  
 r.Get("/alerts", ihf("list\_alerts", api.listAlerts))  
 r.Post("/alerts", ihf("add\_alerts", api.addAlerts))  
  
 r.Get("/silences", ihf("list\_silences", api.listSilences))  
 r.Post("/silences", ihf("add\_silence", api.addSilence))  
 r.Get("/silence/:sid", ihf("get\_silence", api.getSilence))  
 r.Del("/silence/:sid", ihf("del\_silence", api.delSilence))  
}

r.Post("/alerts", ihf("add\_alerts", api.addAlerts))

## API.addAlets

**func** (api \*API) addAlerts(w http.ResponseWriter, r \*http.Request) {  
 **var** alerts []\*types.Alert  
 **if** err := receive(r, &alerts); err != nil {  
 respondError(w, apiError{  
 typ: *errorBadData*,  
 err: err,  
 }, nil)  
 **return** }  
  
 api.insertAlerts(w, r, alerts...)  
}

## API.insertAlerts

**func** (api \*API) insertAlerts(w http.ResponseWriter, r \*http.Request, alerts ...\*types.Alert) {  
 now := time.Now()  
  
 **for** \_, alert := **range** alerts {  
 alert.UpdatedAt = now  
  
 // Ensure StartsAt is set.  
 **if** alert.StartsAt.IsZero() {  
 alert.StartsAt = now  
 }  
 // If no end time is defined, set a timeout after which an alert  
 // is marked resolved if it is not updated.  
 **if** alert.EndsAt.IsZero() {  
 alert.Timeout = true  
 alert.EndsAt = now.Add(api.resolveTimeout)  
  
 numReceivedAlerts.WithLabelValues("firing").Inc()  
 } **else** {  
 numReceivedAlerts.WithLabelValues("resolved").Inc()  
 }  
 }  
  
 // Make a best effort to insert all alerts that are valid.  
 **var** (  
 validAlerts = make([]\*types.Alert, 0, len(alerts))  
 validationErrs = &types.MultiError{}  
 )  
 **for** \_, a := **range** alerts {  
 **if** err := a.Validate(); err != nil {  
 validationErrs.Add(err)  
 numInvalidAlerts.Inc()  
 **continue** }  
 validAlerts = append(validAlerts, a)  
 }  
 **if** err := api.alerts.Put(validAlerts...); err != nil {  
 respondError(w, apiError{  
 typ: *errorInternal*,  
 err: err,  
 }, nil)  
 **return** }  
  
 **if** validationErrs.Len() > 0 {  
 respondError(w, apiError{  
 typ: *errorBadData*,  
 err: validationErrs,  
 }, nil)  
 **return** }  
  
 respond(w, nil)  
}

### 调用

#### 根据EndsAt时间判断Firing和resolved

// If no end time is defined, set a timeout after which an alert  
// is marked resolved if it is not updated.  
**if** alert.EndsAt.IsZero() {  
 alert.Timeout = true  
 alert.EndsAt = now.Add(api.resolveTimeout)  
  
 numReceivedAlerts.WithLabelValues("firing").Inc()  
} **else** {  
 numReceivedAlerts.WithLabelValues("resolved").Inc()  
}