**Go 的环境变量**

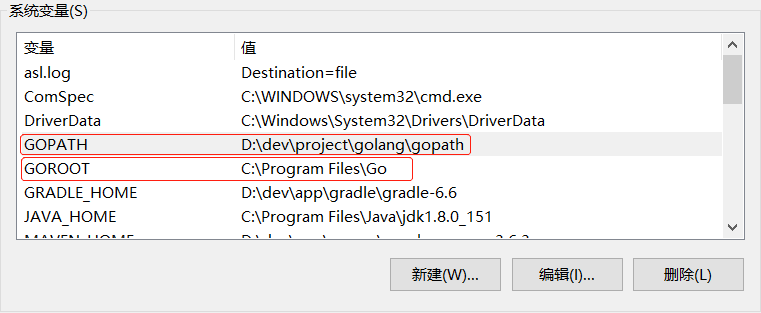
GOROOT：GOROOT就是Go的安装目录，（类似于java的JDK）

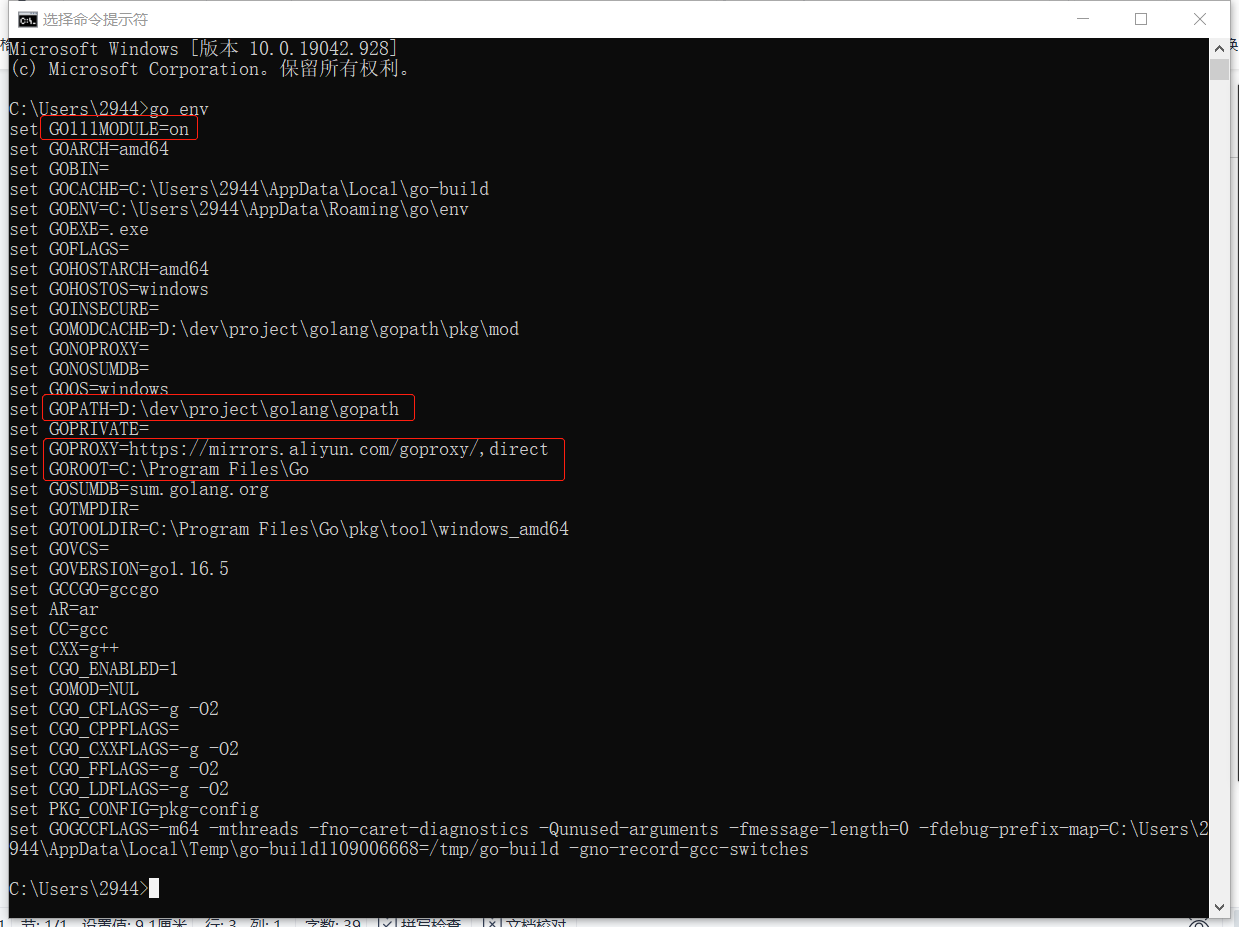
GOPATH：GOPATH是我们的工作空间,保存go项目代码和第三方依赖包

使用GOPATH时，GO会在以下目录中搜索包：

GOROOT/src：该目录保存了Go标准库代码。

GOPATH/src：该目录保存了应用自身的代码和第三方依赖的代码。





**Goland 的环境配置**

* Global GOPATH 选择你在环境变量中配置的GOPATH路径
* Project GOPATH 项目的GOPATH,最好不好设置Global GOPATH,项目将会使用到所用配GOPATH
* Use GOPATH that`s defined in system 将使用系统定义的环境变量，并设置到 Global GOPATH
* Index entire GOPATH: 会将当前项目作为gopath

**Goland 的环境配置与项目的关系**

**1 Goland 没有配置但是项目依赖正常**

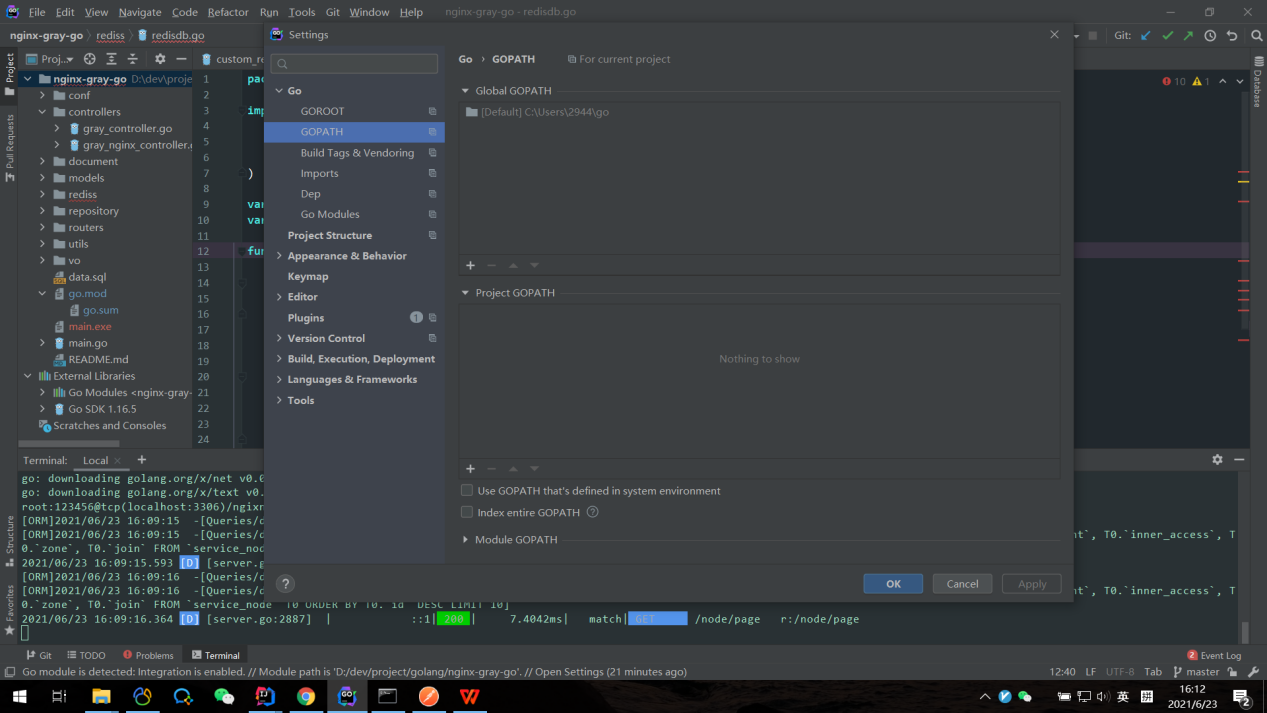
Global GOPATH,

Project GOPATH,

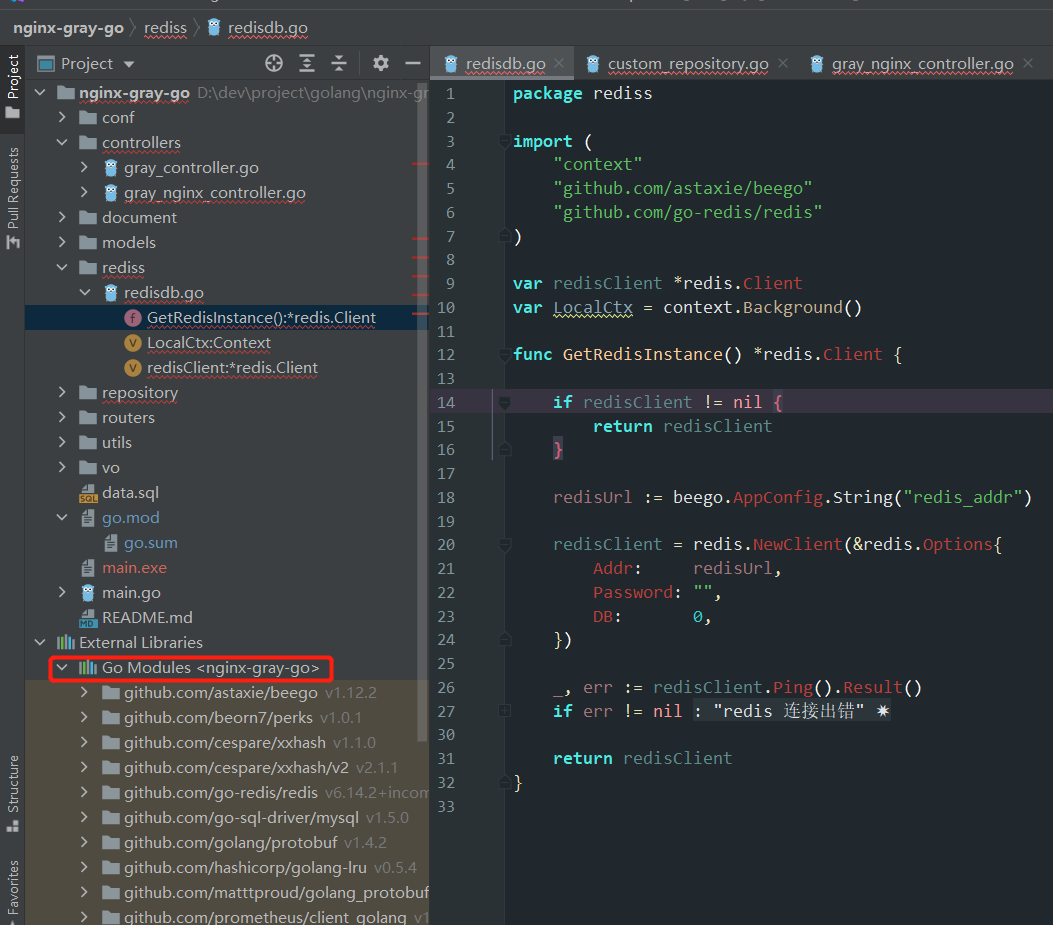
index entire GOPATH,

Use GOPATH that defined in System

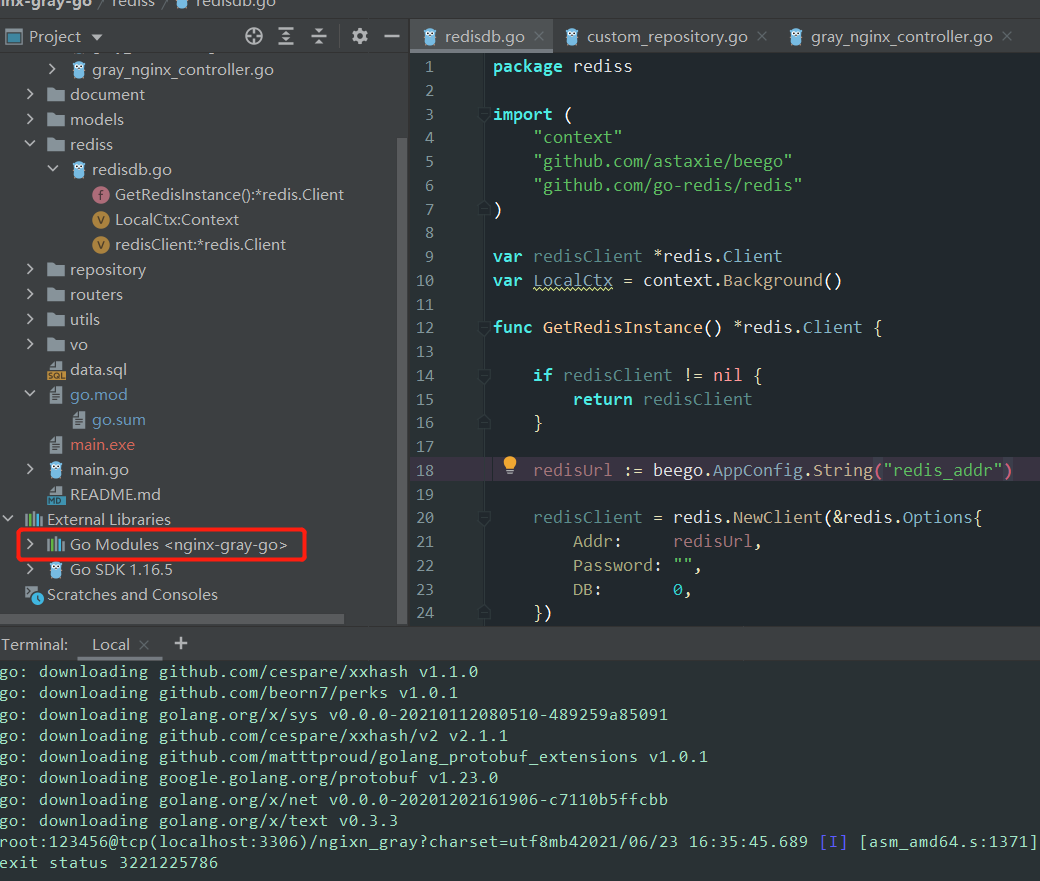
这些都没有配置的情况下，项目的依赖还是能正常下载，GOPATH 的系统设置是 在D盘，项目正常下载好依赖之后，依赖是存在 D盘的 GOPATH 目录下的，说明goland会试别到系统的环境变量 GOPATH



**2 项目是否正常引用依赖与报红色错误没有直接关系**

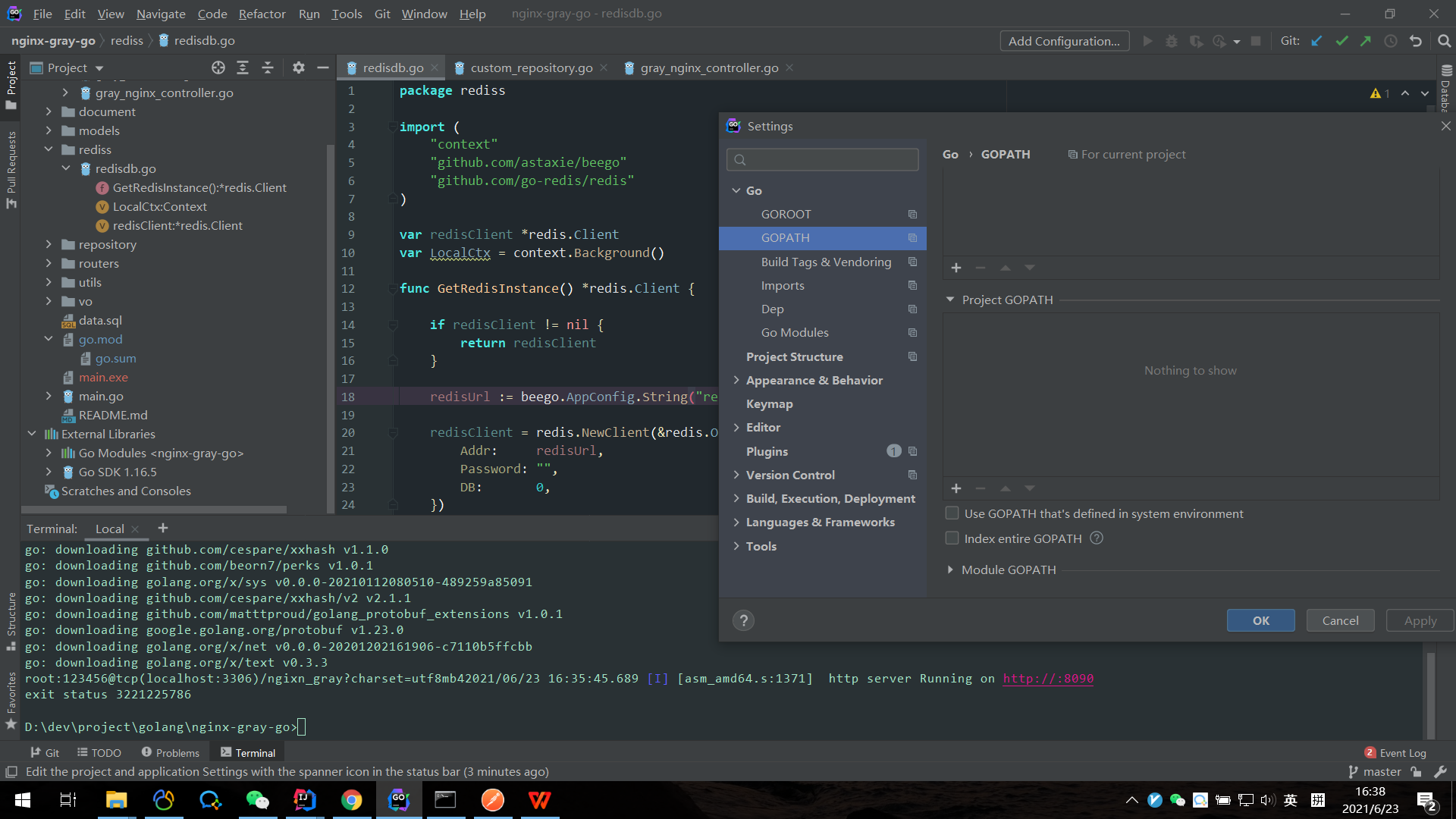


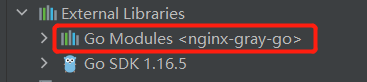
项目虽然有报红色的错误，但是依赖还是引用成功，项目启动正常，可以访问



**3 只要项目的第三方依赖能正确引用，项目代码就不会报红色错误**

项目setting所有 GOPATH 都没有设置，但是依赖引用成功，经过刷新 index, 项目不会报红色错误





只要项目的第三方依赖能正确引用，项目代码就不会报红色错误

项目的 Go Modules 目录是项目必须正常引用依赖的前提条件，这个目录必须真实存在系统的目录中，在goland里面没有设置任何 GOPATH 的情况下，默认是存在系统的环境变量配置的 GOPATH 目录里的，因此 go mod 依赖管理虽然简化了项目依赖，但是go mod 还是需要将依赖包放到 GOPATH 目录的，因此也是需要有 GOPATH 的

**4 不使用 go mod 管理依赖的项目，第三方的包都是需要在goland配置gopath 的**

不使用go mod 依赖的项目， go get 下载下来的依赖都是存放在 项目配置的 GOPATH路径的

