给 API 命令增加版本功能

本节核心内容

- 如何给 apiserver 增加版本功能

> 本小节源码下载路径：[demo12 (https://github.com/lexkong/apiserver_demos/tree/mas](https://github.com/lexkong/apiserver_demos/tree/mas)
>
> 可先下载源码到本地，结合源码理解后续内容，边学边练。
>
> 本小节的代码是基于 [demo11 (https://github.com/lexkong/apiserver_demos/tree/mas](https://github.com/lexkong/apiserver_demos/tree/mas) 来开发的。

为什么需要版本

在实际开发中，当开发完一个 apiserver 特性后，会编译 apiserver 二进制文件并发布到生产环境，很多时候为了定位问题和出于安全目的（不能发错版本），我们需要知道当前 apiserver 的版本，以及一些编译时候的信息，如编译时 Go 的版本、Git 目录是否 clean，以及基于哪个 git commmit 来编译的。在一个编译好的可执行程序中，我们通常可以用类似 `./app_name -v` 的方式来获取版本信息。

我们可以将这些信息写在配置文件中，程序运行时从配置文件中取得这些信息进行显示。但是在部署程序时，除了二进制文件还需要额外的配置文件，不是很方便。或者将这些信息写入代码中，这样不需要

额外的配置，但要在每次编译时修改代码文件，也比较麻烦。Go 官方提供了一种更好的方式：通过 `-ldflags -X importpath.name=value`（详见 -ldflags -X importpath.name=value (https://golang.org/cmd/link/)）来给程序自动添加版本信息。

> 在实际开发中，绝大部分都是用 Git 来做源码版本管理的，所以 apiserver 的版本功能也基于 Git。

给 apiserver 添加版本功能

假设我们程序发布的流程是这样：

1. 编码完成，提交测试工程师测试
2. 测试工程师测试代码，提交 bug，更改 bug 并重新测试后验证通过
3. 开发人员把验证通过的代码合并到 master 分支，并打上版本号：`git tag -a v1.0.0`
4. 开发人员将 `v1.0.0` 版本发布到生产环境

最终发布后，我们希望通过 `./apiserver -v` 参数提供如下版本信息：

- 版本号
- git commit
- git tree 在编译时的状态
- 构建时间
- go 版本
- go 编译器
- 运行平台

为了实现这些功能，我们首先要在 main 函数中添加用于接收 -v 参数的入口（详见 [demo12/main.go](https://github.com/lexkong/apiserver_demos/blob/master/

```
package main

import (
    "encoding/json"
    "fmt"
    "os"
    ...
    v "apiserver/pkg/version"
    ...
)

var (
    version = pflag.BoolP("version", "v", false,
"show version info.")
)

func main() {
    pflag.Parse()
    if *version {
        v := v.Get()
        marshalled, err := json.MarshalIndent(&v,
"", "  ")
        if err != nil {
            fmt.Printf("%v\n", err)
            os.Exit(1)
        }

        fmt.Println(string(marshalled))
        return
    }
    ...
}
```

通过 pflag 来解析命令行上传入的 -v 参数。

通过 pkg/version 的 Get() 函数来获取 apiserver 的版本信息。

通过 json.MarshalIndent 来格式化打印版本信息。

pkg/version 的 Get() 函数实现为（详见 [demo12/pkg/version (https://github.com/lexkong/apiserver_demos/tree/master/c](https://github.com/lexkong/apiserver_demos/tree/master/c)

```
func Get() Info {
    return Info{
        GitTag:       gitTag,
        GitCommit:    gitCommit,
        GitTreeState: gitTreeState,
        BuildDate:    buildDate,
        GoVersion:    runtime.Version(),
        Compiler:     runtime.Compiler,
        Platform:     fmt.Sprintf("%s/%s",
runtime.GOOS, runtime.GOARCH),
    }
}
```

其中 gitTag、gitCommit、gitTreeState 等变量的值是通过 -ldflags -X importpath.name=value 在编译时传到程序中的。为此我们需要在编译时传入这些信息，在 Makefile 中添加如下信息（详见 [demo12/Makefile (https://github.com/lexkong/apiserver_demos/blob/master/](https://github.com/lexkong/apiserver_demos/blob/master/)

```
SHELL := /bin/bash
BASEDIR = $(shell pwd)

# build with verison infos
versionDir = "apiserver/pkg/version"
gitTag = $(shell if [ "`git describe --tags --
abbrev=0 2>/dev/null`" != "" ];then git describe
--tags --abbrev=0; else git log --
pretty=format:'%h' -n 1; fi)
buildDate = $(shell TZ=Asia/Shanghai date
+%FT%T%z)
gitCommit = $(shell git log --pretty=format:'%H'
-n 1)
gitTreeState = $(shell if git status|grep -q
'clean';then echo clean; else echo dirty; fi)

ldflags="-w -X ${versionDir}.gitTag=${gitTag} -X
${versionDir}.buildDate=${buildDate} -X
${versionDir}.gitCommit=${gitCommit} -X
${versionDir}.gitTreeState=${gitTreeState}"
```

并在 go build 中添加这些 flag：

```
go build -v -ldflags ${ldflags} .
```

> -w 为去掉调试信息（无法使用 gdb 调试），这样可以使编译后的二进制文件更小。

编译并测试

1. 下载 apiserver_demos 源码包（如前面已经下载过，请忽略

此步骤）

```
$ git clone
https://github.com/lexkong/apiserver_demos
```

2. 将 apiserver_demos/demo12 复制为 $GOPATH/src/apiserver

```
$ cp -a apiserver_demos/demo12/
$GOPATH/src/apiserver
```

3. 在 apiserver 目录下编译源码

```
$ cd $GOPATH/src/apiserver
$ make
```

查看 apiserver 版本

```
$ ./apiserver -v

{
  "gitTag": "7322949",
  "gitCommit":
"732294928b3c4dff5b898fde0bb5313752e1173e",
  "gitTreeState": "dirty",
  "buildDate": "2018-06-05T07:43:26+0800",
  "goVersion": "go1.10.2",
  "compiler": "gc",
  "platform": "linux/amd64"
}
```

可以看到 ./apiserver -v 输出了我们需要的版本信息。

> 在上一小节中我们已经给 `apiserver` 添加过 Makefile 文件。

小结

本小节主要介绍如何用 Makefile 以及 Go 本身所支持的编译特性，实现编译时自动生成版本号的功能。后续小节编译 API 源码均会通过 `make` 来编译。