HTTP/2的特性：

- 压缩HTTP头
- 实现服务器推送
- 在单个连接上多路复用请求。

HTTP/2 服务器

HTTP/2强制使用TLS。为了实现这一点，我们首先需要一个私钥和一个证书。在Linux上，下面的命令执行这个任务。

openssl req -newkey rsa:2048 -nodes -keyout server.key -x509 -days 365 -out server.crt

该命令将生成两个文件：server.key 以及 server.crt

申请证书的时候如果是本地测试，需指定服务名称为localhost，不然就会有上述错误

现在，对于服务器代码，以最简单的形式，我们将使用Go的标准库HTTP服务器，并启用TLS与生成的SSL文件。

```
package main

import (
```

```
    "log"
    "net/http"
)

func main() {
    // 在 8000 端口启动服务器
    // 确切地说，如何运行HTTP/1.1服务器。

    srv := &http.Server{Addr: ":8000", Handler: http.HandlerFunc(handle)}
    // 用TLS启动服务器，因为我们运行的是http/2，它必须是与TLS一起运行。
    // 确切地说，如何使用TLS连接运行HTTP/1.1服务器。
    log.Printf("Serving on https://0.0.0.0:8000")
    //私钥和证书在linux中可以通过下面命令生成
    //openssl req -newkey rsa:2048 -nodes -keyout server.key -x509 -days 365 -out
server.crt
    log.Fatal(srv.ListenAndServeTLS("server.crt", "server.key"))
}

func handle(w http.ResponseWriter, r *http.Request) {
    // 记录请求协议
    log.Printf("Got connection: %s", r.Proto)
    // 向客户发送一条消息
    w.Write([]byte("Hello world"))
}
```

HTTP/2 客户端

我们生成的服务器证书是“自签名”的，这意味着它不是由一个已知的证书颁发机构（CA）签署的。这将导致我们的客户端不相信它：

```
package main

import (
    "fmt"
    "net/http"
)
```

```
const url = "https://localhost:8000"

func main() {
    _, err := http.Get(url)
    fmt.Println(err)
}
```

上面代码时无法连接到服务端的，客户端会出现下面的错误：

为了解决这个问题，我们可以用定制的TLS配置去配置我们的客户端。我们将把服务器证书文件添加到客户端“证书池”中，因为我们信任它，即使它不是由已知CA签名的。

```
package main

import (
    "crypto/tls"
    "crypto/x509"
    "flag"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "golang.org/x/net/http2"
)

const url = "https://localhost:8000"
//通过命令行接受协议版本号
```

```
var httpVersion = flag.Int("version", 2, "HTTP version")

func main() {
    flag.Parse()
    client := &http.Client{}
//    server.crt为服务端证书
    caCert, err := ioutil.ReadFile("server.crt")
        if err != nil {
         log.Fatalf("读取服务端证书失败: %s", err)
    }
    //创建一个证书池
    caCertPool := x509.NewCertPool()
    caCertPool.AppendCertsFromPEM(caCert)
    // 创建tls配置项
    tlsConfig := &tls.Config{
        RootCAs: caCertPool,
    }

    switch *httpVersion {
     case 1:
    //Transport可用于缓冲连接，以便充分使用
    //Transport并发安全的
        client.Transport = &http.Transport{
            TLSClientConfig: tlsConfig,
        }
    case 2:
        client.Transport = &http2.Transport{
            TLSClientConfig: tlsConfig,
        }
    }

    resp, err := client.Get(url)

    if err != nil {
        log.Fatalf("%s", err)
```

```
    }
    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        log.Fatalf("读取返回信息失败%s", err)
    }
    fmt.Printf(
        "获取返回信息 %d: %s %s\n",
        resp.StatusCode, resp.Proto, string(body)
        )
}
```