

НТТР

Гаратуев Шамиль

План занятия

- HTTP/REST
- net/http client/server
- go-chi/chi router and middlewares
- User sessions
- Graceful shutdown
- Go modules



HTTP :80

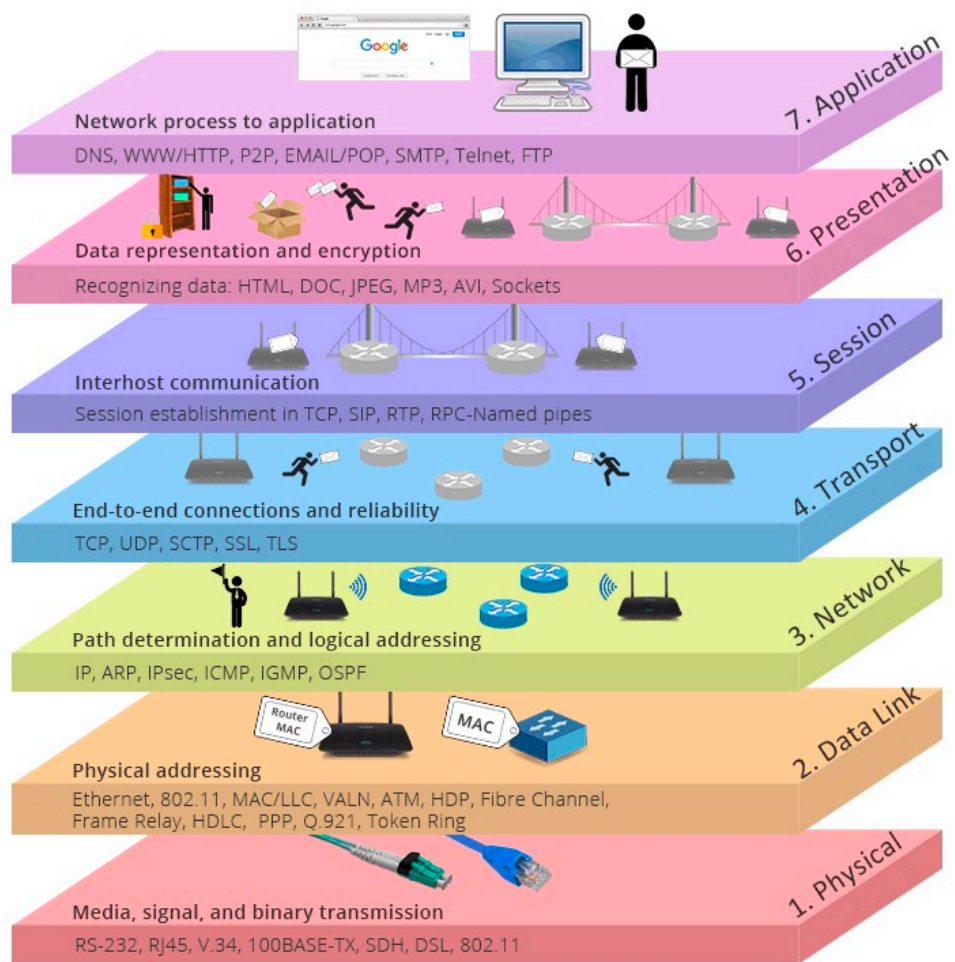
протокол прикладного уровня передачи данных, текстовый.

- Применяется для организации сетевого взаимодействия между приложениями
- Предполагает режим «запрос-ответ»
- Stateless, то есть сервер не хранит контекст с предыдущего запроса



OSI Model

The Open Systems Interconnection model



Структура HTTP-сообщения

Request:

```
POST /api/resource HTTP/1.0
Referer: http://www.example.com/
Content-Type: application/json
Authorization: Basic QWxhZGRpbjpwGVuIHNLc2FtZQ==

{"name": "Mark"}
```

Response:

```
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 1348

{"name": "Mark", "id": 12456}
```

Методы и идемпотентность

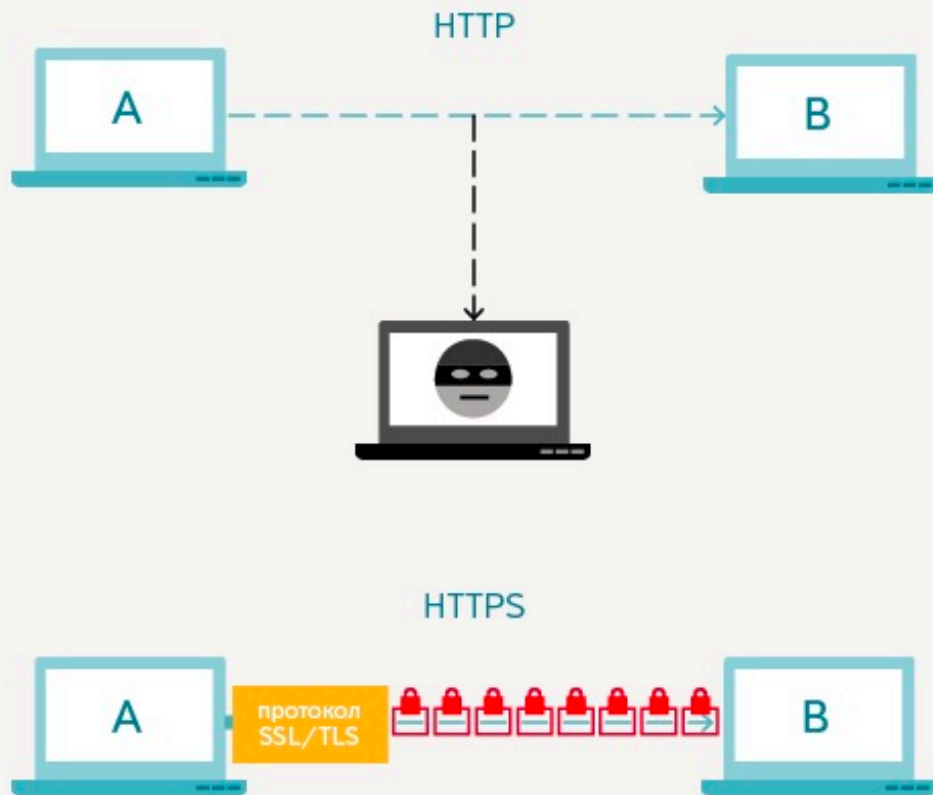
- GET – получить (параметры в path/query)
- POST – создать (параметры в body)
- PUT, PATCH - изменить
- DELETE - удалить
- OPTIONS – проверка допустимых методов
- ...

Response codes

- 1xx – информирование о процессе передачи (101 Switching Protocols)
 - 2xx – успех (200 – OK, 201 – Created, 202 – Accepted)
 - 3xx – перенаправление (301 – Moved Permanently)
 - 4xx – ошибка клиента (400 – Bad request, 401 – Unauthorized, 404 – Not Found)
 - 5xx - ошибка сервера (500 – Internal Server Error)
-
- 402 Payment Required
 - 418 I'm a teapot
 - 451 Unavailable For Legal Reasons

HTTPS :443

обычный HTTP, работающий через
шифрованные транспортные
механизмы [SSL](#) и [TLS](#)



REST *(Representational State Transfer)*

архитектурный стиль взаимодействия
компонентов распределённого приложения в сети

Требования:

1. Модель клиент-сервер
2. Отсутствие состояния
3. Кэширование
4. Единообразие интерфейса
5. Слои
6. Код по требованию (необязательно)

pet Everything about your Pets

POST `/pet/{petId}/uploadImage` uploads an image

POST `/pet` Add a new pet to the store

PUT `/pet` Update an existing pet

GET `/pet/findByStatus` Finds Pets by status

GET `/pet/findByTags` Finds Pets by tags

GET `/pet/{petId}` Find pet by ID

POST `/pet/{petId}` Updates a pet in the store with form data

DELETE `/pet/{petId}` Deletes a pet

store Access to Petstore orders

GET `/store/inventory` Returns pet inventories by status

POST `/store/order` Place an order for a pet

GET `/store/order/{orderId}` Find purchase order by ID

DELETE `/store/order/{orderId}` Delete purchase order by ID

REST и HTTP

Преимущества:

- Надёжность (за счёт отсутствия необходимости сохранять информацию о состоянии клиента);
- Производительность (за счёт использования кэша);
- Масштабируемость;
- Прозрачность системы взаимодействия;
- Простота интерфейсов;
- Портативность компонентов;
- Лёгкость внесения изменений;
- Способность эволюционировать;

example: [swagger petstore](#)

Ограничения HTTP

- взаимодействия синхронного типа «запрос-ответ» <- HTTP/2
- Вопросы производительности (ограничения TCP) <- HTTP/3
- HTTP – текстовый протокол <- HTTP/2

net/http

```
type Request struct {  
    Method  string  
    URL     *net.URL  
    Header  Header // map[string][]string  
    Body    io.ReadCloser  
    ...  
}
```

```
type Response struct {  
    Status      string // "200 OK"  
    StatusCode  int    // 200  
    Header     Header // map[string][]string  
    Body       io.ReadCloser  
    ...  
}
```

net/http Client

```
1  resp, err := http.Get("http://google.com/robots.txt")
2  if err != nil {
3      panic(err)
4  }
5
6  defer resp.Body.Close()
7  data, err := io.ReadAll(resp.Body)
8  if err != nil {
9      panic(err)
10 }
11
12 fmt.Println(string(data))
```

- Мало умеет (GET, HEAD, POST)
- Не передать заголовки
- Нет таймаута (!)

net/http Client

```
1  req, err := http.NewRequest(http.MethodGet, "http://google.com/robots.txt", nil)
2  if err != nil {
3      panic(err)
4  }
5
6  c := http.Client{
7      Timeout: time.Second * 10,
8  }
9
10 resp, err := c.Do(req)
11 if err != nil {
12     panic(err)
13 }
14
15 defer resp.Body.Close()
16 data, err := io.ReadAll(resp.Body)
17 if err != nil {
18     panic(err)
19 }
20
21 fmt.Println(string(data))
```

- Любые методы
- Настраиваемый таймаут

net/http Client

```
1  v := url.Values{}
2  v.Add("id", "1")
3  queryString := v.Encode()
4
5  req, err := http.NewRequest(http.MethodGet, "http://google.com/robots.txt" + "?"
  + queryString, nil)
```

Добавление query параметров

```
1  req, err := http.NewRequest(http.MethodGet, "http://google.com/robots.txt", nil)
2  if err != nil {
3      panic(err)
4  }
5
6  req.Header.Add("User-Agent", "Mozilla/5.0 (X11; Linux i686; rv:2.0.1)
  Gecko/20100101 Firefox/4.0.1")
```

Управление заголовками

debug req/resp

```
1 b, err := httputil.DumpRequestOut(req, true)
2 if err != nil {
3     panic(err)
4 }
5 fmt.Println(string(b))
```

```
POST /robots.txt?id=1 HTTP/1.1
Host: google.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:2.0.1)
Gecko/20100101 Firefox/4.0.1
Content-Length: 18
Accept-Encoding: gzip

Hello and welcome!
```

```
1 b, err = httputil.DumpResponse(resp, true)
2 if err != nil {
3     panic(err)
4 }
5 fmt.Println(string(b))
```

```
HTTP/2.0 200 OK
Accept-Ranges: bytes
Cache-Control: private, max-age=0
Content-Type: text/plain
Cross-Origin-Opener-Policy-Report-Only: same-origin;
report-to="static-on-bigtable"
Cross-Origin-Resource-Policy: cross-origin
...

User-agent: *
Disallow: /search
Allow: /search/about
Allow: /search/static
Allow: /search/howsearchworks
...
```


net/http Server

```
1 func HelloHandler(w http.ResponseWriter, r *http.Request) {  
2     _, _ = w.Write([]byte("hello, World!"))  
3 }
```

```
1 func main() {  
2     http.HandleFunc("/", HelloHandler)  
3     log.Fatal(http.ListenAndServe(":5000", nil))  
4 }
```

```
1 func main() {  
2     s := http.Server{  
3         Addr: ":5000",  
4         Handler: nil,  
5         ReadTimeout: time.Second,  
6     }  
7     http.HandleFunc("/", HelloHandler)  
8     log.Fatal(s.ListenAndServe())  
9 }
```

Parameters handling

```
1 func AdvancedHandler(w http.ResponseWriter, r *http.Request) {
2     fmt.Printf("method: %s\n", r.Method)
3     fmt.Printf("query values: %v\n", r.URL.Query())
4     fmt.Printf("headers: %v\n", r.Header)
5
6     body, err := io.ReadAll(r.Body)
7     if err != nil {
8         w.WriteHeader(http.StatusInternalServerError)
9         return
10    }
11    fmt.Printf("body: %s\n", string(body))
12
13    w.WriteHeader(http.StatusAccepted)
14 }
```

```
1 http.HandleFunc("/", HelloHandler)
2 http.HandleFunc("/advanced", AdvancedHandler)
3 log.Fatal(s.ListenAndServe())
```

Недостатки стандартной библиотеки

- нет возможности указывать в url path parameter
 - нет возможности на уровне роута хэндлить http-методы
 - `/` является роутом по умолчанию – нет 404 статуса
- <- кастомные роутеры

go-chi/chi

```
1 func PostHello(w http.ResponseWriter, r *http.Request) {
2     id := chi.URLParam(r, "id")
3     _, _ = w.Write([]byte("POST hello, " + id))
4 }
5
6 func GetHello(w http.ResponseWriter, r *http.Request) {
7     id := chi.URLParam(r, "id")
8     _, _ = w.Write([]byte("POST hello, " + id))
9 }
10
11 func main() {
12     r := chi.NewRouter()
13     r.Get("/hello/{id}", GetHello)
14     r.Post("/hello/{id}", PostHello)
15
16     log.Fatal(http.ListenAndServe(":5000", r))
17 }
```

middlewares

```
1  r := chi.NewRouter()  
2  r.Use(middleware.RequestID)  
3  r.Use(middleware.Logger)  
4  
5  r.Get("/hello", GetHello)  
6  
7  log.Fatal(http.ListenAndServe(":5000", r))
```

```
2021/10/10 12:21:31 [macbook-s.garatuev/9QyU8dFSm4-000001] "GET  
http://localhost:5000/hello HTTP/1.1" from [::1]:51986 - 200 10B in 17.257µs
```

Аутентификация и авторизация

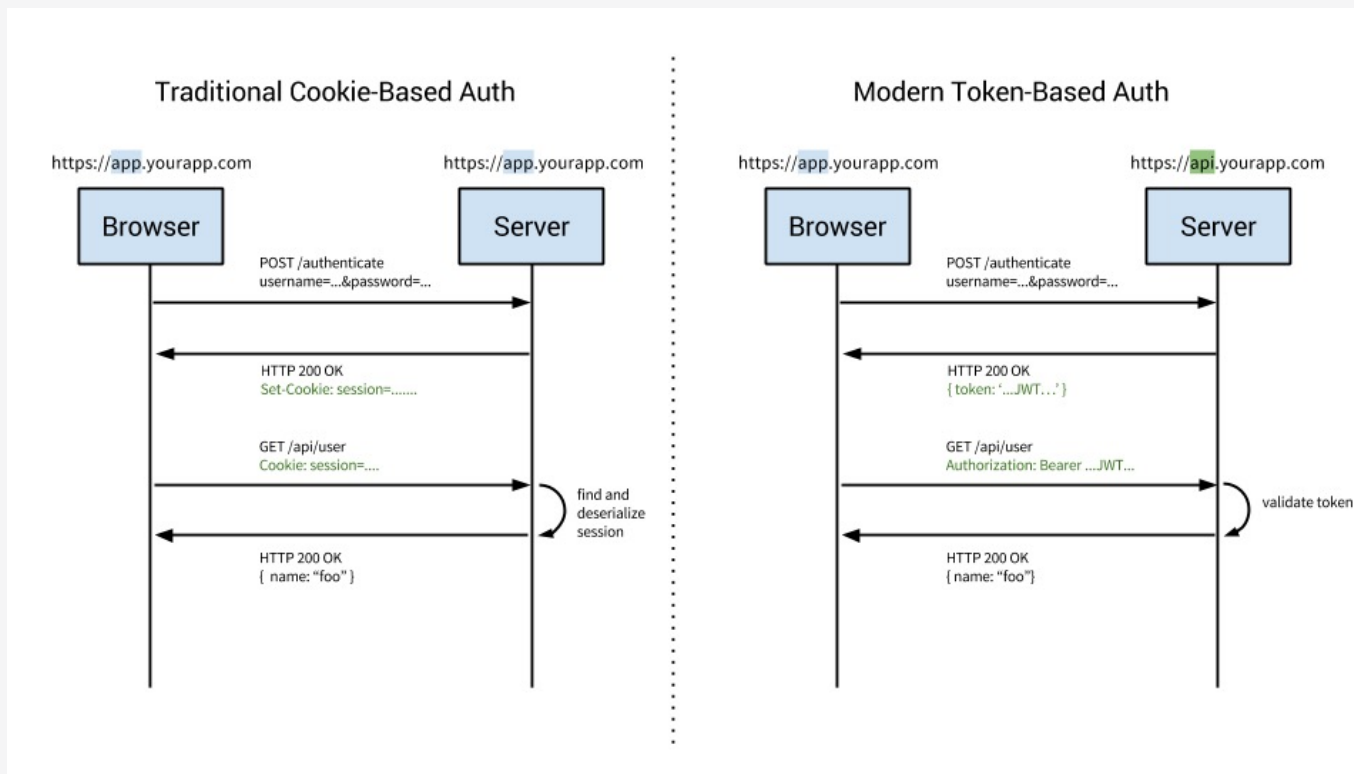
Аутентификация

процедура проверки подлинности, то есть мы идентифицируем пользователя по его логину/ паролю

Авторизация

предоставление права пользователю на получение ресурса или выполнения действия

Аутентификация и авторизация



Graceful shutdown

- Клиент ждет ответа на отправленный запрос, а вместо ответа получает ошибку **connection reset by peer**
- Пайплайн был прерван в середине обработки, **данные были потеряны**
- Приложение **не закрыло коннект** к внешнему ресурсу

Graceful shutdown

```
1 func main() {
2     mainSrv := http.Server{Addr: ":5000"}
3     sigquit := make(chan os.Signal, 1)
4     signal.Ignore(syscall.SIGHUP, syscall.SIGPIPE)
5     signal.Notify(sigquit, syscall.SIGINT, syscall.SIGTERM)
6     stopAppCh := make(chan struct{})
7     go func() {
8         log.Println("Captured signal: ", <-sigquit)
9         log.Println("Gracefully shutting down server...")
10        if err := mainSrv.Shutdown(context.Background()); err != nil {
11            log.Println("Can't shutdown main server: ", err.Error())
12        }
13        stopAppCh <- struct{}{}
14    }()
15    <-stopAppCh
16 }
```

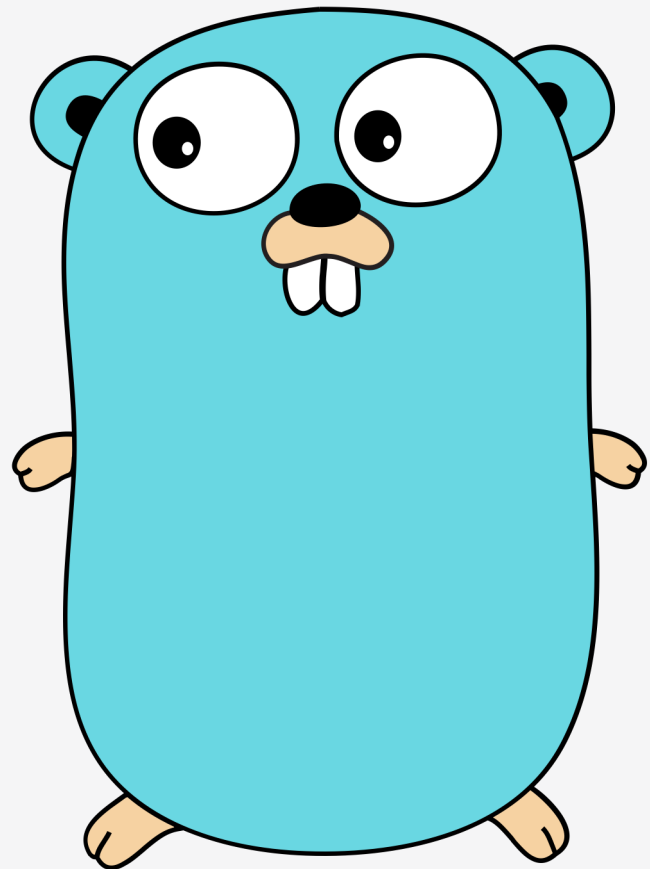
Go modules

1. `go mod init example.com/example`
2. `go get`
3. `go mod tidy`
4. `go mod vendor`
5. ...
6. profit!

golang.org/ref/mod

Итоги

- HTTP/REST
- net/http client/server
- go-chi/chi router and middlewares
- User sessions
- Graceful shutdown
- Go modules



HTTP

Гаратуев Шамиль

s.garatuev@tinkoff.ru

@SquareLemon