十一、索引添加

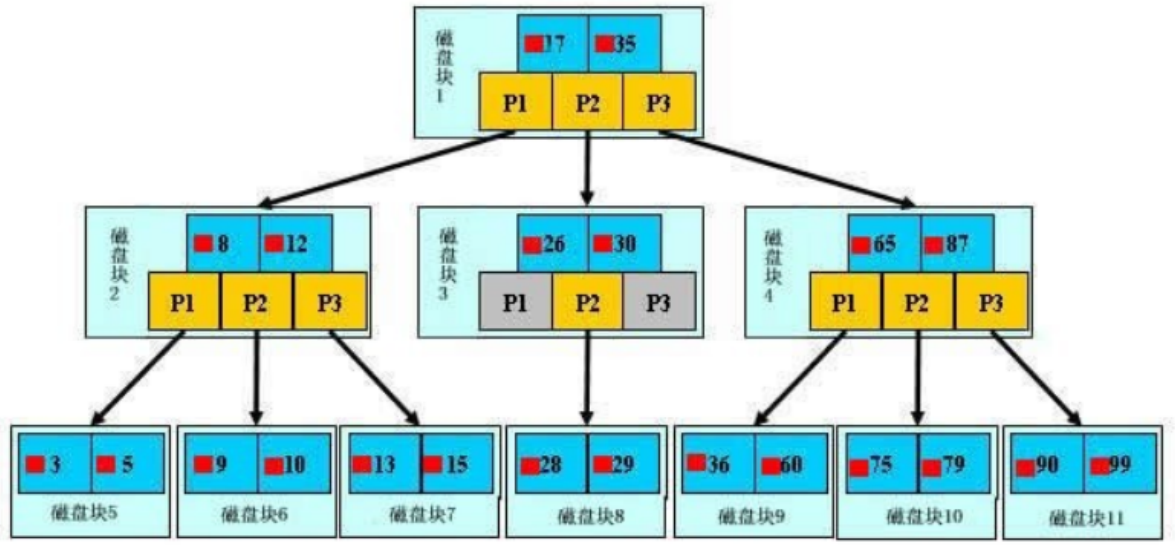
加速查询

原理：

磁盘IO与预读

考虑到磁盘IO是非常高昂的操作，计算机操作系统做了一些优化，**当一次IO时，不光把当前磁盘地址的数据，还把相邻的数据也都读取到内存缓冲区内**，因为局部预读性原理告诉我们，当计算机访问一个地址的数据的时候，与其相邻的数据也会很快被访问到。每一次IO读取的数据我们称之为一页(page)。具体一页有多大数据跟操作系统有关，一般为4k或8k，也就是我们读取一页内的数据时候，实际上才发生了一次IO.

b+树



b+树的查找过程
如图所示，如果要查找数据项29，那么首先会把磁盘块1由磁盘加载到内存，此时发生一次IO，在内存中用二分查找确定29在17和35之间，锁定磁盘块1的P2指针，内存时间因为非常短（相比磁盘的IO）可以忽略不计，通过磁盘块1的P2指针的磁盘地址把磁盘块3由磁盘加载到内存，发生第二次IO，29在26和30之间，锁定磁盘块3的P2指针，通过指针加载磁盘块8到内存，发生第三次IO，同时内存中做二分查找找到29，结束查询，总计三次IO。

b+树性质
1.**索引字段要尽量的小**
2.**索引的最左匹配特性（即从左往右匹配）**：当b+树的数据项是复合的数据结构，比如(name,age,sex)的时候，b+数是按照从左到右的顺序来建立搜索树的，比如当(张三,20,F)这样的数据来检索的时候，b+树会优先比较name来确定下一步的所搜方向，如果name相同再依次比较age和sex，最后得到检索的数据；但当(20,F)这样的没有name的数据来的时候，b+树就不知道下一步该查哪个节点，因为建立搜索树的时候name就是第一个比较因子，必须要先根据name来搜索才能知道下一步去哪里查询。比如当(张三,F)这样的数据来检索时，b+树可以用name来指定搜索方向，但下一个字段age的缺失，所以只能把名字等于张三的数据都找到，然后再匹配性别是F的数据了， 这个是非常重要的性质，即索引的最左匹配特性。

MySQL的索引分类

```
索引分类
1.普通索引index :加速查找
2.唯一索引
    主键索引：primary key ：加速查找+约束（不为空且唯一）
    唯一索引：unique：加速查找+约束 （唯一）
3.联合索引
    -primary key(id,name):联合主键索引
    -unique(id,name):联合唯一索引
    -index(id,name):联合普通索引
4.全文索引fulltext :用于搜索很长一篇文章的时候，效果最好。
5.空间索引spatial :了解就好，几乎不用
```

十二、跨域请求CORS

> 跨域，是指浏览器不能执行其他网站的脚本。它是由浏览器的同源策略造成的，是浏览器对JavaScript实施的安全限制。

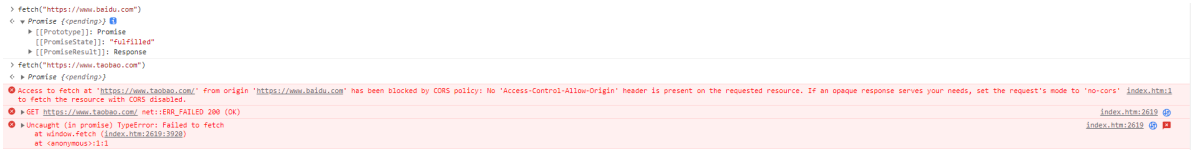

```
 所谓的同源是指域名、协议、端口均为相同。
示例：
       http://www.yiyuanxinghe.com/index.html 调用
http://www.yiyuanxinghe.com/server.jsp 非跨域
       http://www.yiyuanxinghe.com/index.html 调用 http://www.yyxh.com/server.jsp
跨域，主域不同
       http://abc.yiyuanxinghe.com/index.html 调用
http://def.yiyuanxinghe.com/server.jsp 跨域，子域名不同
       http://www.yiyuanxinghe.com:8080/index.html 调用
http://www.yiyuanxinghe.com/server.jsp 跨域，端口不同
       https://www.yiyuanxinghe.com/index.html 调用
http://www.yiyuanxinghe.com/server.jsp 跨域，协议不同
       localhost 调用 127.0.0.1 跨域
```

没有同源策略限制的两大危险场景

一是针对接口的请求，二是针对DOM的查询。

> 没有同源策略限制的接口请求
>
> 有个小小的东西叫Cookie大家应该知道，一般用来处理登录等场景，目的是**让服务端知道谁发出的请求。**如果你请求了接口进行登录，服务端验证通过后会在响应头加入Set-Cookie字段，然后下次再请求的时候，浏览器会自动将Cookie附加在HTTP请求头字段Cookie中，服务器端就能知道

> 这个用户已登录过了。
>
> 知道这个之后，我们来看场景：
> （1）你准备去清空你的购物车，于是打开了淘宝网www.taobao.com，然后登录成功，一看，购物车东西这么少，不行，还得买多点。
> （2）你在看有什么东西买的过程中，你的好基友给你发了一个链接www.1024.com，一脸yin笑地跟你说：“你懂的”，你毫不犹豫打开了。
> （3）你饶有兴趣地浏览www.1024.com，谁知这个网站暗地里做了些不可描述的事情，由于没有同源策略的限制，它向www.taobao.com发起了请求，聪明的你一定想到上面的话“服务器端验证通过后会在响应头加入Set-Cookie字段，然后下次再发请求的时候，浏览器会自动将Cookie附加在HTTP请求的头字段Cookie中”，这样一来，这个不法网站就相当于登录了你的账号，可以为所欲为了！
>
> 原文链接：https://blog.csdn.net/weixin_46178852/article/details/119168365



CORS（Cross-Origin Resource Sharing） 是一种用于允许跨域请求的机制，它允许服务器指定哪些源（域、协议或端口）可以访问其资源。CORS 使用特定的 HTTP 头来实现，主要有以下两个头：

1. `Access-Control-Allow-Origin`：该头用于指定允许访问资源的源。服务器可以设置为允许单个域、多个域或所有域都可以访问资源。
2. `Access-Control-Allow-Methods`：该头用于指定哪些 HTTP 方法（例如 GET、POST、PUT、DELETE）可以用于访问资源。

CORS（Cross-origin resource sharing，跨域源资源共享）是一个 W3C 标准，它是一份浏览器技术的规范，提供了Web服务从不同网域传来沙盒脚本的方法，以避开浏览器的同源策略，这是 JSONP 模式的现代版。

CORS需要浏览器和服务器同时支持。目前，所有浏览器都支持该功能，IE浏览器不能低于IE10。

实现CORS通信的关键是服务器。只要服务器实现了CORS接口，就可以跨源通信。

CORS是一种浏览器安全策略，用于控制一个网页应用是否允许与不同域（或不同协议、不同端口）的服务器进行交互（请求不同服务器中的域名、协议或端口）。当一个网页应用尝试从一个域（例如，`example.com`）请求数据或资源（如图像、API、数据）来自另一个域（例如，`api.example2.com`）时，浏览器会执行CORS策略来决定是否允许这个请求。

GoFrame添加CORS示例

```
import (
    "github.com/gogf/gf/frame/g"
    "github.com/gogf/gf/net/ghttp"
)
```

1. 创建一个中间件函数，该函数用于处理跨域请求。可以根据需要自定义允许的源、HTTP 方法、请求标头等。

```
func CorsMiddleware(r *ghttp.Request) {
    r.Response.CORSDefault()
    r.Middleware.Next()
}
```

在上述示例中，`CORSDefault` 方法会设置一组默认的 CORS 头，允许所有源（"*"）;允许的 HTTP 方法（GET、POST、PUT、DELETE、OPTIONS）;允许的标头等**（默认配置）**。

1. 注册 CORS 中间件函数。在项目初始化中，使用 `ghttp.Middleware` 函数注册上述定义的中间件：

```
func main() {
    s := g.Server()
    // 注册中间件函数
    s.Use(CorsMiddleware)
    // 定义路由规则
    s.BindHandler("/example", YourHandler)
    s.Run()
}
```

通过上述步骤，可以在 GoFrame 项目中配置了 CORS 中间件。这将允许跨域请求访问应用程序，并按默认设置允许来自所有源的请求。

十三、通过鉴权中间件返回执行状态

问题：当前的项目无法返回状态码


```
{
    "code": 0,
    "message": "",
    "data": {
        "message": "用户信息如下: ",
        "userProfileGroup": [
            {
                "userIP": "2020007",
                "userName": "张三",
                "email": "198268743@qq.com",
                "currentNum": 0,
                "historyNum": 0
            },
            {
                "userIP": "2016132",
                "userName": "李四",
                "email": "2698725@qq.com",
                "currentNum": 3,
                "historyNum": 2
            },
            {
                "userIP": "2016132",
                "userName": "李四",
```


中间件有什么功能?

\1. 负载均衡：将请求分配给多个服务器，使得每个服务器都能够充分利用资源，提高系统的并发处理能力和吞吐量。
\2. 缓存：将频繁访问的数据缓存到内存中，减少对数据库的访问，提高系统的响应速度。（**设置缓存就是在中间件中设置的嘛**）
\3. 安全控制：对请求进行安全过滤和访问控制，保护系统的安全性。
\4. 日志记录：记录请求和响应的信息，便于系统的监控和排错。
\5. API 管理：对外提供 API 接口的管理和发布，便于开发者使用和集成。
\6. 服务发现：提供服务发现和注册功能，方便服务的管理和调用。
\7. 数据转换：对请求和响应的数据进行转换和处理，使得系统能够更加灵活和可扩展。

\8. 静态资源处理：对静态资源的访问进行处理和缓存，提高系统的访问速度。
\9. 集成其他服务：与其他服务进行集成和协作，实现系统的功能扩展和整合。

在 GoFrame 框架中，通过鉴权中间件返回状态码通常涉及两个步骤：首先，在鉴权中间件中进行鉴权检查，然后根据鉴权结果设置 HTTP 响应状态码。

1. 创建一个鉴权中间件函数，用于进行鉴权检查。这个函数应该在需要鉴权的路由中使用。例如：

```
func AuthMiddleware(r *ghttp.Request) {
    // 进行鉴权检查
    if !CheckAuth(r) {
        // 如果鉴权失败，设置响应状态码为 403 Forbidden
        r.Response.WriteStatus(http.StatusForbidden)
        return
    }
    r.Middleware.Next()
}
```

在上述示例中，`AuthMiddleware` 函数会检查鉴权条件（在 `CheckAuth` 函数中实现），如果鉴权失败，则设置响应状态码为 403 Forbidden。

1. 注册鉴权中间件函数。在项目初始化中，使用 `ghttp.Middleware` 函数注册上述定义的中间件(得提前声明一下，我要使用鉴权中间件函数了)：

```
func main() {
    s := g.Server()
    // 注册中间件函数
    s.Use(AuthMiddleware)
    // 定义需要鉴权的路由规则
    s.BindHandler("/secure", YourSecureHandler)
    s.Run()
}
```

通过上述步骤，您已经创建了一个简单的鉴权中间件，它在需要鉴权的路由中进行鉴权检查，并根据鉴权结果设置 HTTP 响应状态码。

总结：

1. 创建一个鉴权中间件函数（内部编写需要鉴权的具体逻辑）
2. 在项目初始化中注册鉴权中间件函数
3. 如何调用呢?

十四、r.GetHeader("Authorization")的作用

```
func MyHandler(r *ghttp.Request) {
    // 获取 "Authorization" 头部字段的值
    authorizationHeader := r.GetHeader("Authorization")

    // 在这里可以解析令牌、验证用户身份、授权等操作
    // 例如，您可以检查令牌的有效性并允许或拒绝访问
    if isValidToken(authorizationHeader) {
        // 有效的令牌，执行操作
        r.Response.Write("Authorized")
    } else {
```

```
        // 无效的令牌，拒绝访问
        r.Response.WriteStatus(http.StatusUnauthorized)
    }
}
```

token是啥?

> 1、Token的引入：Token是在客户端频繁向服务端请求数据，服务端频繁的去数据库查询用户名和密码并进行对比，判断用户名和密码正确与否，并作出相应提示，在这样的背景下，Token便应运而生。
>
> 2、Token的定义：Token是**服务端生成的一串字符串**，以作客户端进行请求的一个令牌，当**第一次登录**后，**服务器生成一个Token**并将此Token**返回给客户端**，以后客户端只需带上这个Token前来请求数据即可，无需再次带上用户名和密码。
>
> 3、使用Token的目的：Token的目的是为了减轻服务器的压力，减少频繁的查询数据库，使服务器更加健壮。
> 链接：https://www.jianshu.com/p/24825a2683e6

怎么用?

用session值作为Token

客户端：客户端只需携带用户名和密码登陆即可。

客户端：客户端接收到用户名和密码后并判断，如果正确了就将本地获取sessionID作为Token返回给客户端，客户端以后只需带上请求数据即可。

分析：这种方式使用的好处是方便，不用存储数据，但是缺点就是当session过期后，客户端必须重新登录才能进行访问数据。

该问题的解决方案：将session和Token套用

session、token、cookie都是什么啊?

一、Session、Cookie的作用
Session是**客户端**与**服务器**通讯会话跟踪的一门技术，可以使服务器与客户端保证整个通讯的会话基本信息。

客户端在第一次访问服务器的时候，**服务端**会**响应**一个**sessionId**，并且将它存入到客户端本地的**cookie**中（下次来的时候，我就可以验证了），客户端在之后的访问中会将cookie中的sessionId放入到请求头中去访问服务器，如果服务器通过这个sessionId没有找到对应的数据，那么服务器会创建一个新的sessionId并且响应给客户端。

二、分布式Session存在的问题?
假设第一次访问服务A生成一个sessionId并且存入到cookie中，第二次却访问服务B（可能需要调用到服务器B上的某个模块），客户端会在cookie读取sessionId加入到请求头中，如果在服务器B通过sessionId没有找到对应的数据那么服务器B会创建一个新的sessionId并且将它返回给客户端，这样并不能共享我们的sessionId。**（即通过服务器调用访问了同一个东西，却建立了两个sessionId）**

解决方案：

（1）基于Cookie的Session共享

（2） 基于Redis的Session共享

该方案使用Redis来存储用户的登录状态，**Redis服务器也存在存储数据的淘汰策略**，与Session的过期机制非常类似。目前是互联网公司最为广泛使用的方案之一。

实现原理其实就是把每次用户的请求时候生成的sessionId存储在Redis上，然后在基于Redis的特性设置一个失效时间的机制，这样就能保证客户端在Redis中的session失效时间内，都不需要进行再次登录。

优点：

能适应于负载均衡策略；
服务器重启或宕机不会造成session丢失；
安全性较高；
扩展能力强；
适合集群数量大的使用；

缺点：

对应用有侵入，应加强相关配置；
增加一次网络开销，用户体验降低；
序列化和反序列化消耗CPU性能；

> 再来看上面的Token

session是一个在单个操作人员整个操作过程中，与服务器端保持通信的惟一识别信息。在同一操作人员的多次请求当中，session始终保证是同一个对象，而不是多个对象，因为可以对其加锁。当同一操作人员多个请求进入时，可以通过session限制只能单向通行。
本文正是通过使用session以及在session中加入token，来验证同一个操作人员是否进行了并发重复的请求，在后一个请求到来时，使用session中的token验证请求中的token是否一致，当不一致时，被认为是重复提交，将不准许通过。

Token：为客户端生成Token,客户端以后只需要用Token请求数据

Session：服务端为第一次访问的客户端响应一个sessionId，下次访问就可以把sessionId放到请求头上访问

Cookie：是计算机用户首次访问（登录、注册）某个站点或者特定页面的时候，留存在电脑里的一个文本文件，它用于跟踪记录网站访问者的相关数据信息。（账号密码保存）

- Token 通常存储在客户端，用于实现身份验证和授权。它是一种无状态的机制，服务器不需要保存用户状态。
- Session 是服务器端的会话管理机制，用于维护用户状态和数据。它依赖于 Cookie 来存储会话 ID，但会话数据存储在服务器上。
- Cookie 是客户端存储技术，用于在客户端存储数据，包括会话标识。它通常用于在浏览器和服务器之间传递数据

十五、Json Web Token（JWT）

跨域身份验证解决方法

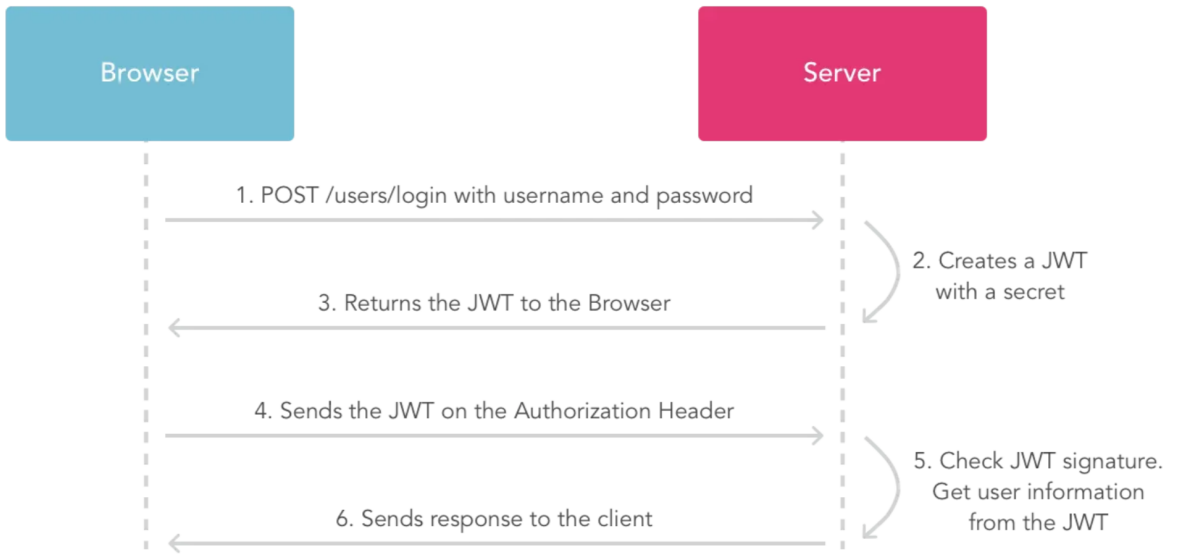
JWT的工作原理

1. 是在服务器身份验证之后，将生成一个JSON对象并将其发送回用户，示例如下：
   {"UserName": "Chongchong","Role": "Admin","Expire": "2018-08-08 20:15:56"}
2. 之后，当用户与服务器通信时，客户在请求中发回JSON对象
3. 为了**防止**用户**篡改**数据，服务器将在**生成对象时添加签名**，并**对发回的数据进行验证**

服务器 ---> JSON对象 ---> 客户端

客户端 ---> JSON对象 ---> 服务器验证签名

> **一个 JWT 实际上就是一个字符串，它由三部分组成：**
>
> **头部(Header)、载荷(Payload)与签名(signature)**



JWT的三个部分如下。JWT头、有效载荷和签名，解析 JWT 令牌是指从一个 JSON Web Token (JWT) 中提取出其中包含的信息和声明。JWT 令牌是一种用于在不同系统之间安全传递信息的标准格式。由三部分组成：头部（Header）、载荷（Payload）、签名（Signature）。

将它们写成一行如下。

使用流程：
用户登录后会将用户信息放到一个加密签名的 token 中，每次请求都把这个串放到 header 或 cookie 内带到服务端，服务端直接将这个 token 解开即可直接获取到用户的信息，无需和用户中心做任何交互请求。

十六、Go语言类型断言

“断言”通常是指类型断言（Type Assertion），允许在运行时检查接口值的实际底层类型（**判断接口类型的底层到底是什么类型**，这里我宁愿把接口类型称为“可变参数”），并将其转换为具体的类型。

在Go语言中类型断言的语法格式如下：

```
value, ok := x.(T)
```

其中，x 表示一个接口的类型，T 表示一个具体的类型（也可为接口类型）。

该断言表达式会**返回 x 的值**（也就是 **value**）和一个布尔值（也就是 ok），可根据该布尔值判断 x 是否为 T 类型：

x肯定是个接口类型

- 如果 T 是具体某个类型，类型断言会检查 x 的动态类型是否等于具体类型 T。如果检查成功，类型断言返回的结果是 x 的动态值，其类型是 T。
- 如果 T 是接口类型，类型断言会检查 x 的动态类型是否满足 T。如果检查成功，x 的动态值不会被提取，返回值是一个类型为 T 的接口值。
- 无论 T 是什么类型，如果 x 是 nil 接口值，类型断言都会失败。

```
package main

import (
    "fmt"
)

func main() {
    var x interface{}
    x = 10
    value, ok := x.(int)
    fmt.Print(value, ",", ok)
}

// 结果
10,true
```

回调函数：

回调函数（Callback Function）是指，在函数执行的过程中，将函数作为参数传递给另一个函数，并在执行过程中被调用的函数。回调函数通常用于处理一些异步或耗时的操作，例如网络请求、文件读写等。回调函数的优点在于避免了函数阻塞，提高了代码的执行效率。

```
// Callback function that should perform the authentication of the user based on
login info.
// Must return user data as user identifier, it will be stored in Claim Array.
Required.
// Check error (e) to determine the appropriate error message.
// 搭了个架子，只要能放进来的(满足参数形式)，就可以放进行
Authenticator func(ctx context.Context) (interface{}, error)
```