Antra

(important) Usage of Collections in coding:

List, Set, Map

- minimum ArrayList, LinkedList, HashMap (hashCode equals)

Sorting

a List/Array in a given attribute. Like sorting a list of students by age or name.

```
List<Student> students = new ArrayList<>();
// Add students to the list...

// Sort by age using lambda expression
Collections.sort(students, (s1, s2) -> Integer.compare(s1.getAge(),
s2.getAge()));

// Sort by age in descending order
Collections.sort(students,
Comparator.comparing(Student::getAge).reversed());

// Sort in descending lexicographic order
Collections.sort(strings, (s1, s2) -> s2.compareTo(s1));
```

```
Student[] students = new Student[size];
// Initialize students array...

// Sort by age using lambda expression
Arrays.sort(students, (s1, s2) -> Integer.compare(s1.getAge(),
s2.getAge())); // asc

// Sort by age in descending order using lambda expression
Arrays.sort(students, (s1, s2) -> Integer.compare(s2.getAge(),
s1.getAge()));
```

如果比较年龄和age

#array

```
// Sort by age first, then by name using lambda expression
Arrays.sort(students, Comparator.comparing(Student::getAge)
                                .thenComparing(Student::getName));


方法二:
// Sort by age first, then by name using lambda expression
Arrays.sort(students, (s1, s2) -> {
    // Compare by age
    int ageComparison = Integer.compare(s1.getAge(), s2.getAge());

    // If ages are equal, compare by name
    if (ageComparison == 0) {
        return s1.getName().compareTo(s2.getName());
    } else {
        return ageComparison; // Return the result of age comparison
    }
});
```

#list

```
List<Student> students = new ArrayList<>();
// Add students to the list...

// Sort by age first, then by name using lambda expression
Collections.sort(students, (s1, s2) -> {
    // Compare by age
    int ageComparison = Integer.compare(s1.getAge(), s2.getAge());

    // If ages are equal, compare by name
    if (ageComparison == 0) {
        return s1.getName().compareTo(s2.getName());
    } else {
```

```
            return ageComparison; // Return the result of age comparison
        }
    });
```

(important) Basic java coding

String,

String: In Java, `String` is a built-in class that represents a sequence of characters. Strings in Java are immutable, meaning once a `String` object is created, its value cannot be changed. You can perform various operations on strings such as concatenation, substring, length calculation, etc.

sub-class,

Sub-class: In object-oriented programming, a subclass (or derived class) is a class that inherits properties and behaviors from another class called a superclass (or base class). A subclass can extend the functionality of its superclass by adding new methods or overriding existing ones.

```
class Animal {
    void eat() {
        System.out.println("Animal is eating");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Dog is barking");
    }
}
```

constructor,

Constructor: A constructor in Java is a special type of method that is automatically called when an instance (object) of a class is created. It is used to initialize the object's state. Constructors have the same name as the class and can be overloaded (i.e., a class can have multiple constructors with different parameters).

```
class Person {
    String name;

    // Constructor
    public Person(String n) {
        name = n;
    }
}
```

getter/setter, etc.

Getter/Setter: Getters and setters are methods used to read and modify the values of private fields (variables) in a class, respectively. They are often used to implement encapsulation, which is a fundamental principle of object-oriented programming aimed at hiding the internal state of objects and restricting direct access to it.

```
class Car {
    private String model;

    // Getter
    public String getModel() {
        return model;
    }

    // Setter
    public void setModel(String m) {
        model = m;
    }
}
```

(important) Basic algorithm:

Leetcode easy level

Advanced concepts in java:

Object-Oriented Programming (OOP):

Object-Oriented Programming is a programming paradigm based on the concept of "objects", which can contain data in the form of fields (attributes) and code in the form of procedures (methods). OOP principles include:

- **Encapsulation**: Bundling data and methods that operate on the data into a single unit (class), and restricting access to some of the object's components.
- **Inheritance**: Mechanism by which a class can inherit properties and behavior from another class, facilitating code reuse and establishing hierarchical relationships between classes.
- **Polymorphism**: Ability to present the same interface for different data types, allowing objects to be treated as instances of their parent class or their subclass.

SOLID Principles:

SOLID is an acronym that represents a set of five design principles for writing maintainable and scalable object-oriented software:

- **Single Responsibility Principle (SRP)**: A class should have only one reason to change, meaning it should have only one responsibility.
- **Open/Closed Principle (OCP)**: Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification.
- **Liskov Substitution Principle (LSP)**: Subtypes must be substitutable for their base types without altering the correctness of the program.
- **Interface Segregation Principle (ISP)**: Clients should not be forced to depend on interfaces they do not use.
- **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules; both should depend on abstractions. Abstractions should not depend on details; details should depend on abstractions.

Model-View-Controller (MVC):

MVC is a software architectural pattern commonly used for developing user interfaces. It separates an application into three interconnected components:

- **Model**: Represents the data and business logic of the application.
- **View**: Represents the presentation layer, responsible for displaying the data to the user.
- **Controller**: Acts as an intermediary between the Model and View, handling user input and updating the Model accordingly.

Design Patterns:

Design patterns are reusable solutions to commonly occurring problems in software design. They provide a template for solving particular design problems in a way that is proven to be effective and efficient. Some common design patterns include:

- **Creational Patterns**: Singleton, Factory, Builder, Prototype.
- **Structural Patterns**: Adapter, Decorator, Proxy, Composite.
- **Behavioral Patterns**: Observer, Strategy, Command, Iterator.

Multithreading:

Multithreading allows concurrent execution of multiple threads within a single process. In Java, multithreading is achieved using the `Thread` class or the `Runnable` interface. Java provides built-in support for multithreading through features like synchronization, locks, and concurrent data structures. Multithreading is crucial for improving the performance and responsiveness of Java applications, especially in scenarios where tasks can be executed concurrently.

These advanced concepts form the foundation of robust, scalable, and maintainable Java applications. Understanding and applying these concepts appropriately can significantly enhance the quality and efficiency of software development in Java. If you have any further questions or need clarification on any specific topic, feel free to ask!

Basic network

Hypertext Transfer Protocol (HTTP) and Transmission Control Protocol (TCP)

HTTP (Hypertext Transfer Protocol):

HTTP is an application layer protocol commonly used for transferring hypertext documents on the World Wide Web. It is the foundation of data communication for the World Wide Web, and it defines how messages are formatted and transmitted between web servers and web browsers.

- **Client-Server Model**: HTTP follows a client-server model where a client (typically a web browser) sends requests to a server (typically a web server), and the server responds with the requested resources (such as HTML pages, images, etc.).
- **Stateless Protocol**: HTTP is stateless, meaning each request from a client to a server is independent and does not retain information about previous requests.
- **Request-Response Protocol**: HTTP operates on a request-response model, where clients send HTTP requests to servers and servers respond with HTTP responses, typically containing the requested resources along with status codes and headers.

HTTP/1.0:

HTTP/1.0 was the first version of the Hypertext Transfer Protocol. It was standardized in 1996 by the Internet Engineering Task Force (IETF). Some key features of HTTP/1.0 include:

- **Stateless Protocol**: Each request/response cycle is independent, and the server does not maintain any state between requests.
- **Connection Persistence**: By default, each request/response pair is sent over a separate TCP connection. However, HTTP/1.0 introduced the `Connection: keep-alive` header to allow persistent connections, reducing latency and overhead.
- **Limited Header Fields**: HTTP/1.0 had a limited set of header fields compared to later versions.

HTTP/1.1:

HTTP/1.1, standardized in 1999, is a significant improvement over HTTP/1.0. It introduced several enhancements and optimizations, including:

- **Persistent Connections by Default**: HTTP/1.1 introduced persistent connections by default, eliminating the need for the `Connection: keep-alive` header.
- **Pipelining**: HTTP/1.1 allows multiple requests to be sent over a single TCP connection without waiting for responses, improving performance.
- **Host Header**: HTTP/1.1 introduced the `Host` header, allowing multiple domains to be hosted on the same IP address.

HTTP/2:

HTTP/2 is a major revision of the HTTP protocol, standardized in 2015. It was designed to address the limitations and performance bottlenecks of HTTP/1.x. Some key features of HTTP/2 include:

- **Binary Protocol**: HTTP/2 uses a binary framing layer instead of plain text, reducing overhead and improving efficiency.
- **Multiplexing**: Multiple requests and responses can be sent and received in parallel over a single TCP connection, eliminating the need for pipelining.
- **Header Compression**: HTTP/2 uses HPACK compression to compress header fields, reducing bandwidth usage.
- **Server Push**: Servers can push resources to clients proactively, improving page load times.
- **Stream Prioritization**: HTTP/2 allows prioritization of streams, enabling more efficient resource allocation.

TCP (Transmission Control Protocol):

TCP is a connection-oriented protocol that operates at the transport layer of the OSI model. It provides reliable, ordered, and error-checked delivery of data between applications running on devices connected to a network. TCP ensures that data packets are delivered intact and in order by establishing a connection between the sender and receiver before data transfer.

- **Connection Establishment**: Before data transfer begins, TCP establishes a connection between the client and server using a three-way handshake mechanism (SYN, SYN-ACK, ACK).
- **Reliability**: TCP ensures reliable delivery of data by using acknowledgments, retransmissions, and sequence numbers to detect and recover from packet loss, duplication, and errors.
- **Flow Control**: TCP uses flow control mechanisms to regulate the rate of data transmission between sender and receiver, preventing data overflow and congestion.
- **Connection Termination**: After data transfer is complete, TCP terminates the connection between the client and server using a four-way handshake mechanism (FIN, ACK, FIN, ACK).

Basic DB: SQL, joins, index..

CLUSTERED ： 聚集索引。非聚集索引：NONCLUSTERED。

clustered是物理上实现数据排序，并且同一个表里只能有一个clustered索引，而nonclustered是逻辑上的排序。

微软的SQL Server 支持两种类型的索引:clustered 索引和nonclustered索引。

Clustered索引在数据表中按照物理顺序存储数据。因为在表中只有一个物理顺序，所以在每个表中只能有一个clustered索引。在查找某个范围内的数据时，Clustered索引是一种非常有效的索引，因为这些数据在存储的时候已经按照物理顺序排好序了。

Nonclustered索引不会影响到下面的物理存储，但是它是由数据行指针构成的。如果已经存在一个clustered索引，在nonclustered中的索引指针将包含clustered索引的位置参考。这些索引比数据更紧促，而且对这些索引的扫描速度比对实际的数据表扫描要快得多。
PRIMARY KEY 约束默认为 CLUSTERED；UNIQUE 约束默认为 NONCLUSTERED。

Anything showing on their resumes.

Bytedance

resume

1. why firebase:

    1. document-based, cloud firestore, high scalable, - It has advanced features and tools for web hosting. It can be classed as an efficient database, since it syncs with real-time changes.
2. why nosql:
    1. flexible data model, fast query, horizontally scaling
3. why rxjs:
    1. involves asynchronous operations, observables streamline the administration of asynchronous processes, such as HTTP requests and user input.
4. kalfa
    1. Apache Kafka is a distributed data store optimized for **ingesting and processing streaming data in real-time**.
    2. In scenarios where immediate and reliable data processing is key, **Apache Kafka** shines as a distributed streaming platform that excels in fault tolerance and event-driven architectures.

4.卡夫卡生产者

- 生产者是一个生成并向 Kafka 主题发送消息的应用程序。
- 生产者将数据写入特定主题。
- 它们可以是任何产生数据的应用程序（例如，Web 服务器、传感器、日志）的一部分。

5. Kafka 消费者

- 消费者是从 Kafka 主题读取消息的应用程序。
- 消费者订阅一个或多个主题。
- 它们处理数据并采取适当的行动（例如，发送通知、更新数据库）。
- Kafka 允许多个消费者同时读取同一主题。

6.Kafka分区

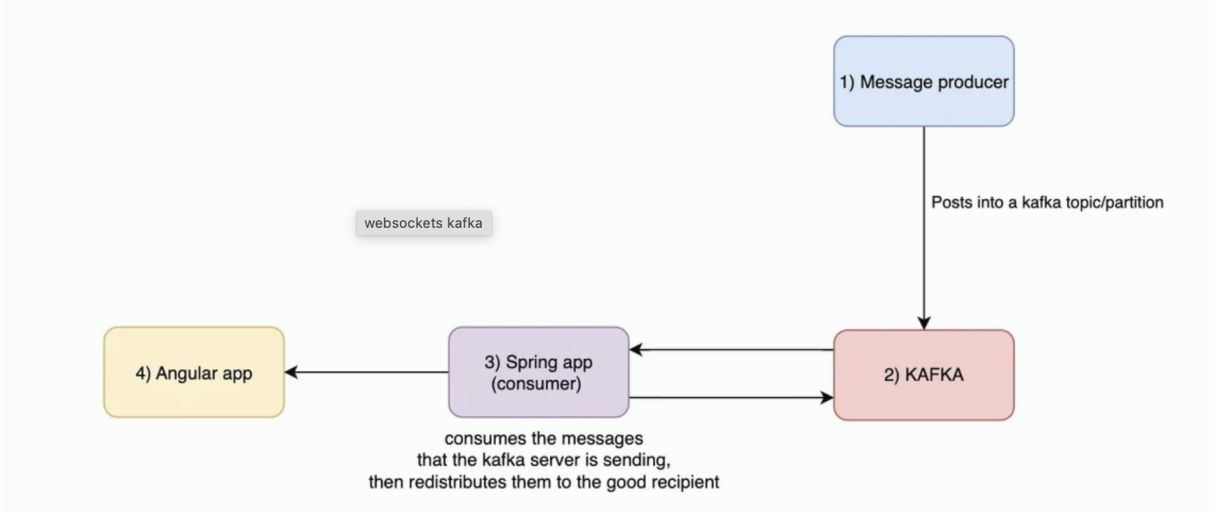
- 分区是主题的逻辑划分。
- 每个分区保存主题数据的一个子集。
- 分区可实现并行性和可扩展性。
- 生产者将消息写入特定分区，消费者从中读取消息。

7.卡夫卡动物园管理员

- **Zookeeper**是 Kafka 使用的分布式协调服务。
- 它管理代理元数据、领导者选举和配置。
- Zookeeper 确保 Kafka 代理能够协同工作。
- 注意：从 Kafka 2.8 开始，Zookeeper 不再是严格要求，因为 Kafka 现在支持自己的元数据管理。

3.

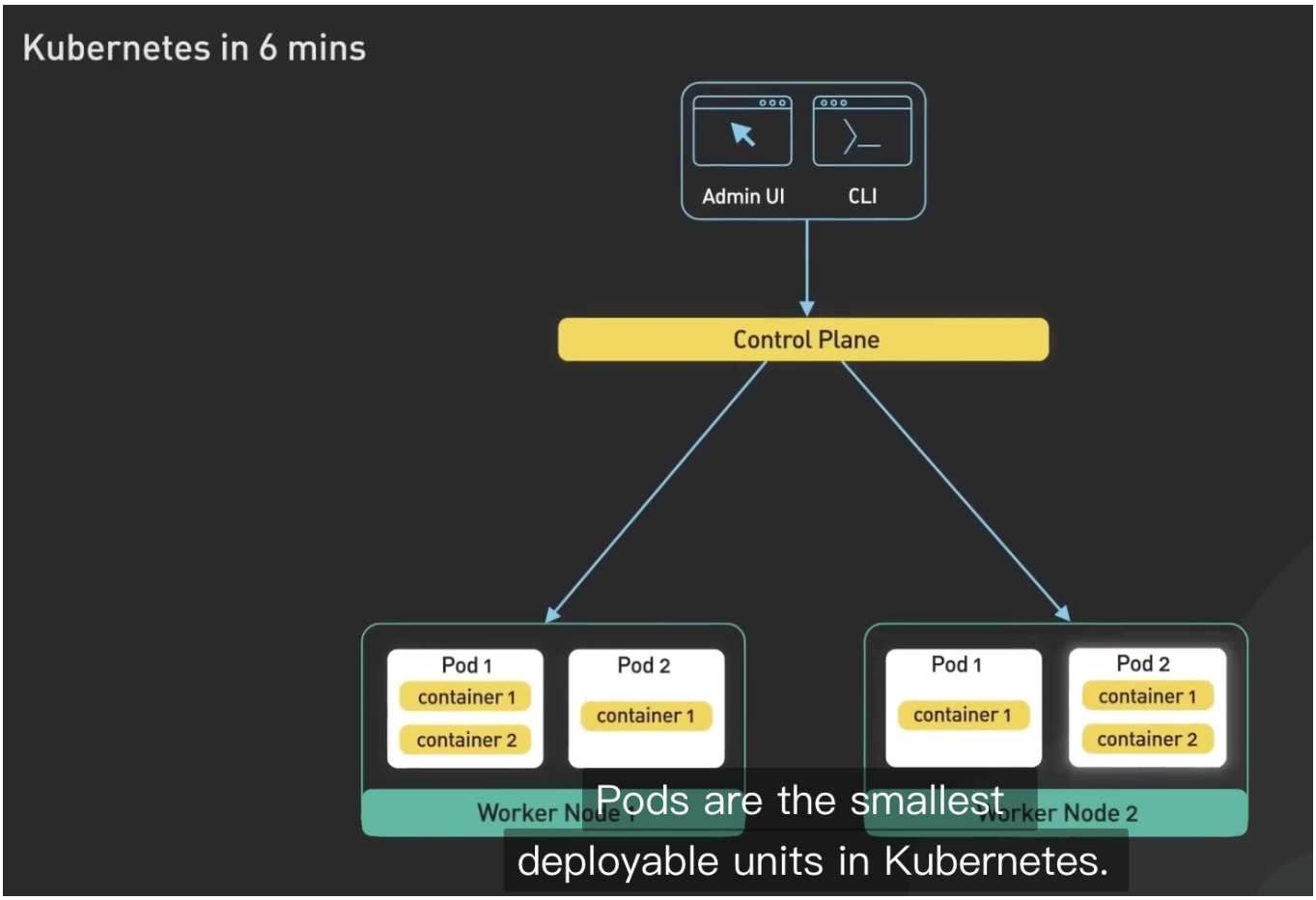
4. https://blog.devgenius.io/creating-a-news-notification-system-with-kafka-and-spring-boot-01b41837e807

5. 

6. ## Server sent event

[](https://github.com/mkapiczy/server-sent-events#server-sent-event

Service is using Server Sent Event for one-way communication with client.



frontend

比如event loop, promise async 判断输出.(网上很多类似题)
React hooks, useEffect, useLayoutEffect什么区别……
如果需要一个customized hook, 去判断user的view port，怎么写

面了promise：
一个function传入一个数组，数组里是已经定义好的receive or reject，return一个数组，数组是receive后返回的值。

HTTP vs HTTP2, angular VS react, 怎么SEO， 什么是CSRF， 给你一堆async promise timeout await， print出来的顺序，怎么migrate project， safe deployment 怎么做， 怎么优化

算法是蠡扣物留，关于int interval overlap的题

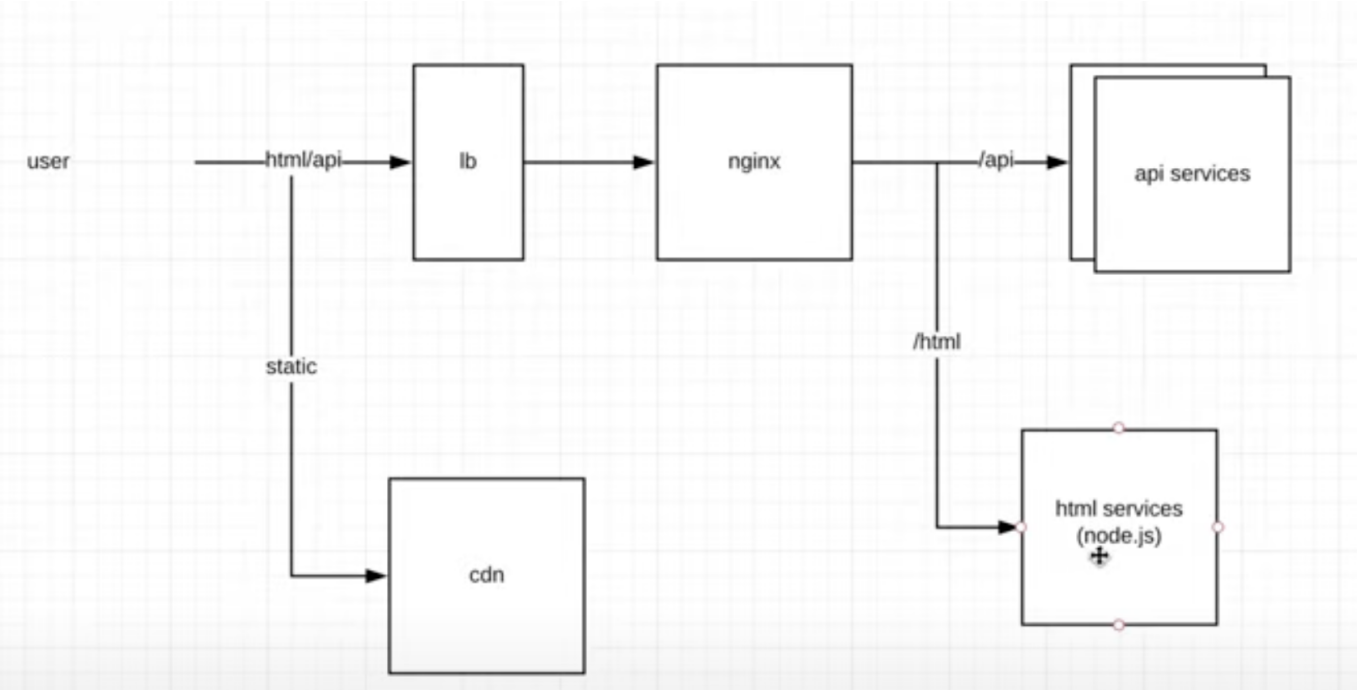
题目是：实现一个tiktok 的投票活动系统，重点在于前端和可复用。不过我从high level讲起，后面大概讲了下API 和数据结构

给了一个数独的网站，参照这个实现一版。

2048游戏

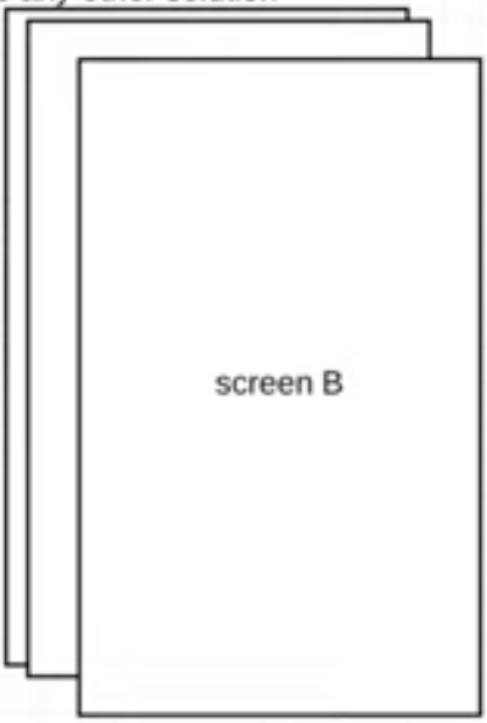
学习前端系统设计的资源不多，Youtube上的JSer和FrontendSystemDesign是很多人都看过的，
https://www.1point3acres.com/bbs/thread-907864-1-1.html

eg instagram



2. instant go back

1) simple improve: cache state / api > still cost time for long pages for DOM
2) Page Stack > beats any other solution



extra lifecycle method
sliding window => 3

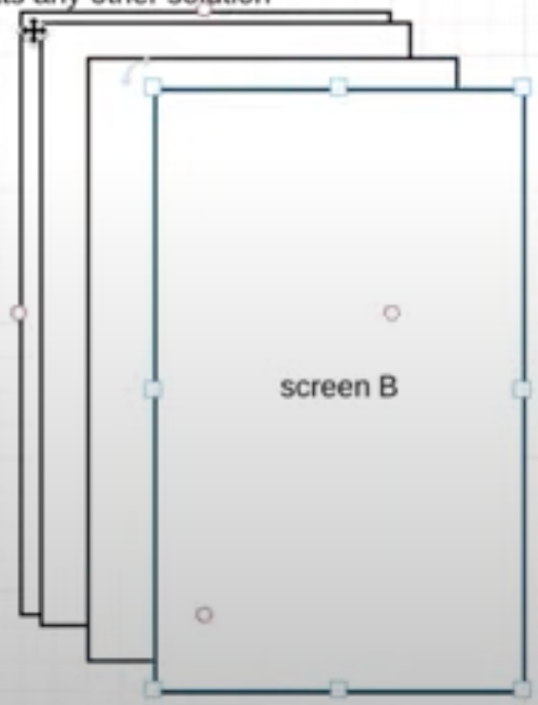
3. images
   1) compress, CDN, srcset, cache
   2) layz-load with dimesntions, avode clikering
   3) skeleton / pregressive

4. timeline
   1) long > infinite scroll or pager , or combine them
   2) update > to avoid reload everthing > server reload button, pull to reload,
      > SSE to notify users about the new updates

1. instant go foward: load the resources fast / show users something
skeleton screen(loading indicator)
   1) seprate component (lazy-load the full version)
   2) same logic with dummy data (data field level lazy-load), => populate the data from list api.

2. instant go back

   1) simple improve: cache state / api > still cost time for long pages for DOM
   2) Page Stack > beats any other solution



extra lifecycle method
sliding window => 3

前端到底用nginx来做啥

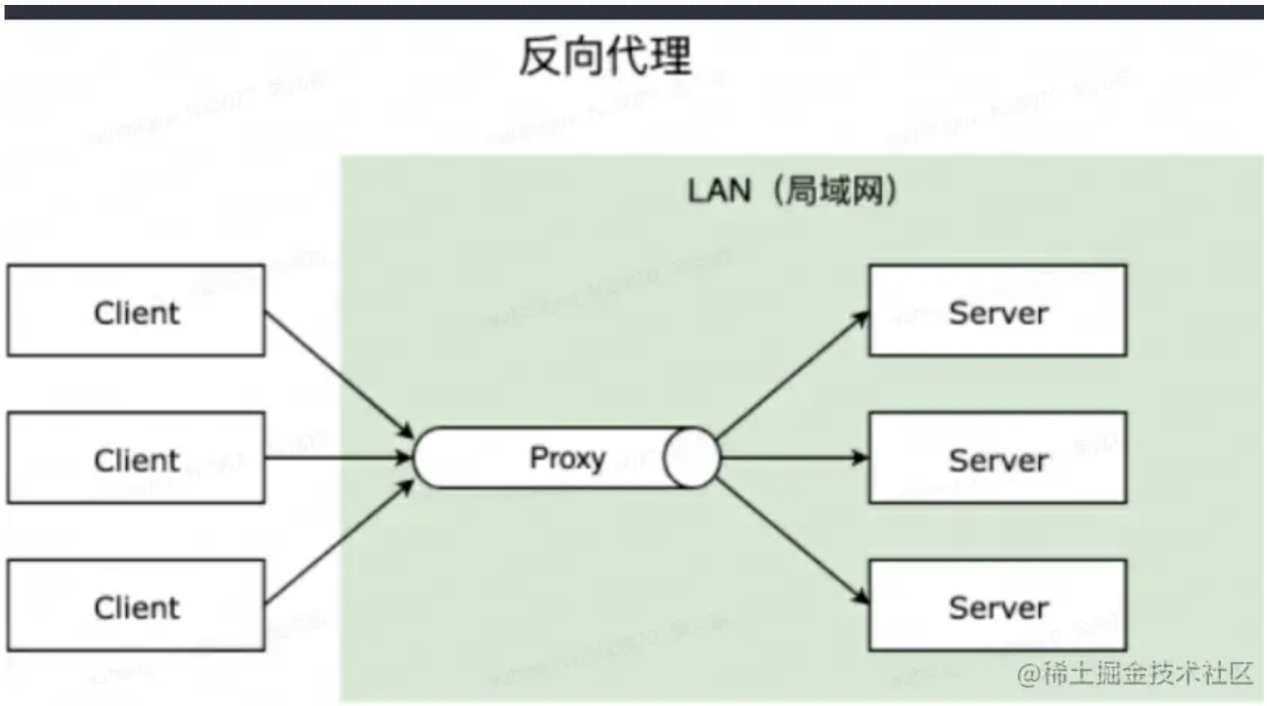
Nginx 是开源、高性能、高可靠的 Web 和反向代理服务器，而且支持热部署，几乎可以做到 7 * 24 小时不间断运行

1. web相关-单页面history配置

2. web相关-跨域问题解决（反向代理或者设置跨域头部）
3. web相关-动静分离 + 缓存
4. 负载均衡
5. 简易的灰度部署
6. 优雅降级-ssr方案的容灾处理

反向代理

反向代理的流程是客户端向代理服务器发送请求，但不需要客户端指定目标服务器，代理服务器根据规则进行客户端的请求转发，获取内容并返回给客户端；反向代理隐藏了真实的服务器，为服务器收发请求，使真实的服务器对客户端不可见。一般在处理跨域请求的时候比较常用。



react相关

function component vs class component

1. **语法**
   - **Functional Component**: 使用函数定义组件,函数返回 JSX 作为组件的输出。
   - **Class-based Component**: 使用 ES6 类定义组件,需要继承 `React.Component` 并实现 `render()` 方法返回 JSX。
2. **状态管理**
   - **Functional Component**: 通常使用 React 的 `useState` hook 来管理组件内部状态。
   - **Class-based Component**: 使用类组件的 `state` 对象来管理组件内部状态。
3. **生命周期钩子**

- **Functional Component**: 使用 React 的生命周期 hook 如 `useEffect`、`useLayoutEffect`、`useMemo`、`useCallback` 等来管理组件的生命周期。
- **Class-based Component**: 使用类组件的生命周期钩子方法,如 `componentDidMount`、`componentDidUpdate`、`componentWillUnmount` 等。

4. 1. **Hook 特性**
   - **Functional Component**: 可以使用 React 提供的各种 Hook 特性,如 `useState`、`useEffect`、`useContext` 等,更加灵活和强大。
   - **Class-based Component**: 无法直接使用 Hook 特性,需要使用其他替代方案,如 HOC 或 Render Props。

八股文

CSS: color 和 background-color的区别：color是text,
box-sizing是什么: 就是包含border margin的width,height
HTML：getElementByID 等等方法；

js相关

`filter()` 用于过滤数组, `reduce()` 用于数组的累计计算

```
const numbers = [1, 2, 3, 4, 5];
const sum = numbers.reduce((acc, cur) => acc + cur, 0);
console.log(sum); // 15


const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers); // [2, 4]
```

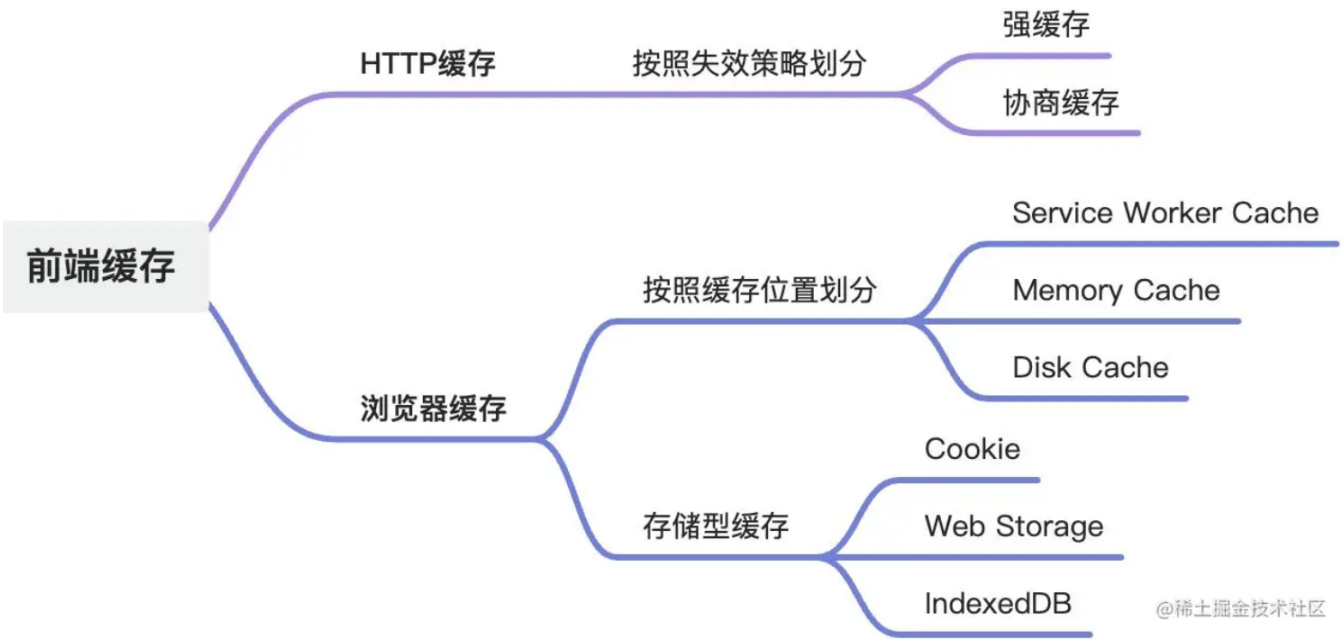
CSS 居中

CSS 居中主要有以下几种实现方式:

- 行内元素/文本居中:使用 `text-align: center;`
- 块级元素居中:
  - 水平居中: `margin: 0 auto;`
  - 垂直居中: 利用 `position: absolute;` 和 `transform: translateY(-50%);`
  - 水平垂直居中: 利用 `position: absolute;` 和 `transform: translate(-50%, -50%);`
- Flex 布局居中:

- 水平居中: `justify-content: center;`
- 垂直居中: `align-items: center;`
- 水平垂直居中: `justify-content: center; align-items: center;`

前端缓存



常用的页面优化实现方案包括:

1. 资源优化:压缩、合并、CDN部署、缓存管理等。
2. 渲染优化:减少DOM操作、合理使用硬件加速、优化关键渲染路径等。
3. 网络优化:HTTP/2、服务端渲染、PWA技术等。
4. 代码优化:Tree Shaking、Code Splitting、懒加载等。
5. 监控优化:定期检查性能指标,持续优化。

对于首页加载优化,可以采取以下方式:

1. 压缩和合并CSS/JS文件,减少HTTP请求数。
2. 使用CDN加速静态资源的加载。
3. 开启Gzip压缩,减小资源体积。
4. 采用懒加载技术,延迟加载非关键资源。
5. 合理利用浏览器缓存,降低重复请求。
6. 优化关键渲染路径,尽早展现页面关键内容。

提高页面渲染速度的常见方式包括:

1. 减少DOM节点数量和DOM深度,优化DOM结构。
2. 合理使用硬件加速,提升GPU渲染能力。
3. 使用骨架屏或shimmer loading效果,改善用户体验。
4. 采用service worker技术实现离线缓存和快速响应。
5. 优化图片格式和尺寸,降低图片资源的加载开销。
6. 使用Web Worker分担主线程的计算任务。

单点登录 与 扫码登录

1. **单点登录的原理**:
   - 用户在第一次登录时,应用程序会颁发一个认证令牌(token)给客户端。
   - 之后客户端在访问其他应用时,会携带这个令牌进行认证。
   - 认证中心负责验证令牌的有效性,并返回登录状态。
2. **扫码登录的原理**:
   - 手机端和网页端通过二维码建立连接。
   - 手机端生成唯一的登录凭证,网页端通过轮询获取该凭证的状态。
   - 当手机端确认登录时,网页端会获取到登录成功的状态,完成登录流程。

web安全

1. **对称加密**:
   - 对称加密是一种加密方式,也称为共享密钥加密。
   - 在对称加密中,发送方和接收方使用相同的密钥进行加密和解密。
   - 这种加密方式相对简单,计算速度快,但需要双方事先协商好密钥,存在密钥管理的问题。
   - 常见的对称加密算法有 AES、DES、Blowfish 等。
2. **XSS (Cross-Site Scripting)**:
   - XSS 是一种Web应用程序安全漏洞,攻击者通过注入恶意的脚本代码到网页中,从而控制用户的浏览器并窃取用户数据。
   - 主要有两种类型:
     - 反射型 XSS: 恶意脚本代码包含在 URL 中,被服务器端返回并执行。
     - 存储型 XSS: 恶意脚本代码被存储在服务器端,每次访问页面时都会执行。
   - 预防 XSS 的关键是对用户输入进行充分的验证和过滤。
3. **CSRF (Cross-Site Request Forgery)**:
   - CSRF 是一种利用受害者已登录的Web应用程序的一种攻击方式。
   - 攻击者构造一个包含恶意请求的URL或表单,诱使用户点击或提交,从而在用户的授权下执行了非法操作。
   - 预防 CSRF 的关键是使用随机令牌(CSRF token)等机制来验证用户的真实意图。

单元测试

Jasmine和Karma是两款广泛应用于前端JavaScript单元测试的工具:

1. **Jasmine**:
   - Jasmine是一个行为驱动开发(BDD)风格的JavaScript测试框架。
   - 它提供了一套易于阅读和编写的语法,用于描述应用程序的行为。
   - Jasmine的主要特点包括:
     - 支持异步测试
     - 提供丰富的断言库
     - 可以编写嵌套的测试套件
     - 支持spies(模拟依赖)和matchers(自定义断言)
   - Jasmine可以独立运行,也可以与其他测试运行器集成使用。
2. **Karma**:
   - Karma是一个JavaScript测试运行器,用于在浏览器环境中执行单元测试。
   - 它的主要功能包括:
     - 管理和运行测试
     - 自动刷新页面并重新执行测试
     - 支持多种浏览器和移动设备
     - 与各种测试框架(如Jasmine、Mocha等)集成
   - Karma的优势在于:
     - 提供灵活的配置,可以针对不同的项目需求进行定制
     - 支持并发测试,加快测试执行速度
     - 提供测试覆盖率报告

通常情况下,Jasmine用于编写测试用例,Karma则负责运行这些测试用例并生成报告。两者结合使用可以构建一个完整的前端单元测试解决方案,帮助开发者编写高质量的JavaScript代码。

design pattern

1. **MVC (Model-View-Controller)**:
   - 模型(Model)负责管理应用程序的数据和逻辑。
   - 视图(View)负责展示数据和处理用户交互。
   - 控制器(Controller)负责协调模型和视图,处理用户输入,更新模型。
   - 控制器直接与视图和模型交互,耦合度较高。
2. **MVP (Model-View-Presenter)**:
   - 与MVC类似,但将控制器拆分为Presenter。
   - 模型(Model)和视图(View)的职责与MVC一致。
   - Presenter负责协调模型和视图,处理用户交互。
   - Presenter与视图和模型的耦合度较低,可测试性更强。
3. **MVVM (Model-View-ViewModel)**:

- 模型(Model)和视图(View)的职责与MVC一致。
- 引入ViewModel作为模型和视图之间的中间层。
- ViewModel负责将模型数据转换为视图所需的格式。
- 视图与模型完全解耦,通过数据绑定实现协作。
- 视图和ViewModel之间的交互更加灵活和可测试。

对比来看:

- MVC中控制器和视图/模型耦合较高,MVVM通过ViewModel实现了视图和模型的完全解耦。
- MVP在MVC基础上引入Presenter,降低了控制器与视图/模型的耦合度。
- MVVM进一步将视图和模型解耦,引入ViewModel作为中间层,实现了更加松耦合的架构。

前端code

Image Carousel, Tab Component, Accordion, Dropdown, Star Rating, Tri-state Checkbox。

async/await就是可以把复杂难懂的异步代码变成类同步语法的语法糖。

promise

```
const onMyBirthday = (isKayoSick) => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (!isKayoSick) {
        resolve(2);
      } else {
        reject(new Error("I am sad"));
      }
    }, 2000);
  });
};
```

async/await

```
const handleGuess = async () => {
  try {
    const result = await enterNumber(); // 代替then方法，我们只需将await放在
promise前，就可以直接获得结果

    alert(\`Dice: ${result.randomNumber}: you got ${result.points}
```

```
points\`);

    const isContinuing = await continueGame();

    if (isContinuing) {
      handleGuess();
    } else {
      alert("Game ends");
    }
  } catch (error) { // catch 方法可以由try, catch函数来替代
    alert(error);
  }
};
```

fetch

```
const fetchCountry = async (alpha3Code) => {
  try {
    const res = await fetch(
      \`https://restcountries.eu/rest/v2/alpha/${alpha3Code}\`
    );
    const data = await res.json();
    return data;
  } catch (error) {
    console.log(error);
  }
};

const fetchCountryAndNeigbors = async () => {
  const china = await fetchCountry("cn");
  const neighbors = await Promise.all(
    china.borders.map((border) => fetchCountry(border))
  );
  console.log(neighbors);
};

fetchCountryAndNeigbors();
```