智能 BI 项目面试题

智能BI项目面试题

后端

你的项目中使用了哪些技术栈？请分别介绍一下 Spring Boot、Redis、RabbitMQ 在项目中的作用。

> 主观回答

项目技术栈：SSM + Spring Boot、Redis、RabbitMQ、MySQL、MyBatis-Plus、Hutool 工具库。

Spring Boot：用于快速构建基础的后端项目，只需要修改配置文件，就能轻松整合 SSM、MySQL、Redis、RabbitMQ 等依赖。

Redis：基于内存的高性能键值对存储，在项目中负责分布式 Session 存储、限流功能的实现。

RabbitMQ：时效性极高的消息队列，在项目中用于将 AIGC 生成图表这一耗时任务进行异步化和解耦。

你是如何使用 AIGC 来生成指定格式的 json 的？请简述整个过程。

> 主观回答

我使用了第三方 AI 助手平台（本质上和 OpenAI GPT 一样的），首先创建了一个智能 BI 助手，并且编写了一段系统预设 prompt，全局指定助手的职责、输入内容和回复格式。

在这个过程中，我不断地更改 prompt 并调试输出效果，最终得到了生成内容比较稳定的 Prompt。可以根据我上传的分析目标和原始数据自动生成 Apache ECharts 可视化库接受的图表配置 JSON。

创建好助手后，我通过 AI 助手平台官方的 SDK 调用该助手完成内容的生成。

值得一提的是，AI 生成内容是一个比较耗时的操作，为了提高用户的体验，我使用消息队列将生成过程从同步改造为异步。

你给 AI 预设的 Prompt 结构是什么样的？如何优化 Prompt?

> 主观回答

示例 Prompt 如下，其中我使用分割符来隔断内容，便于后端程序的处理：

```
你是一个数据分析师和前端开发专家，接下来我会按照以下固定格式给你提供内容：
分析需求：
{数据分析的需求或者目标}
原始数据：
{csv格式的原始数据，用,作为分隔符}
请根据这两部分内容，按照以下指定格式生成内容（此外不要输出任何多余的开头、结尾、注释）
【【【【【
{前端 Echarts V5 的 option 配置对象的json代码，合理地将数据进行可视化，不要生成任何
【【【【【
{明确的数据分析结论、越详细越好，不要生成多余的注释}
```

几个优化 Prompt 的小技巧：

1. 使用系统预设全局改变 AI 助手的行为，并且给 AI 一个角色的定义（比如 “你是一个数据分析师和前端开发专家”）
2. 控制输入格式，便于 AI 精确地理解我们的需求（比如 “接下来我会按照以下固定格式给你提供内容”）
3. 控制输出格式，便于处理 AI 返回的内容（比如 “此外不要输出任何多余的开头、结尾、注释”）
4. 给 AI 指定一组示例问答（few-shot）

你是如何保证用户上传的数据文件的安全性的?

> 主观回答

只要涉及到用户自主上传文件的操作，就一定要进行校验！

我在项目中主要采用以下几种方式校验文件：

1. 文件类型验证：通过检查文件的扩展名或 MIME 类型实现，比如后缀名必须为 xlsx。
2. 文件大小限制：从 MultipartFile 对象中获取文件大小信息，限制不能超过 1 - 10 MB。

此外，还有一些更严格的校验方式，比如：

1. 文件格式校验：防止用户改文件后缀名上传非法格式的文件，可以通过匹配特定格式文件的开头或结尾来校验内容
2. 文件内容安全：防止用户上传一些敏感违规内容，可以使用腾讯云数据万象等第三方内容安全服务自动检测

什么是限流？有哪些常见的限流算法？

> 背诵类题目，也可以有主观回答

限流是一种用于控制请求或事件流量的策略，防止系统在短时间内接收过多的请求或事件，从而导致系统过载或崩溃，从而确保系统的稳定性和可用性。

在本项目中，由于 AIGC 是一个消耗资源和成本的重操作，所以我使用 Redisson 提供的 RateLimiter 实现了对单用户使用 AI 生成图表功能的限流，从而保护系统。

我了解的限流算法：

1. 固定窗口计数限流算法：最简单的限流算法之一，它将请求或事件的到达速率限制在固定的窗口内。例如，每秒最多允许处理 10 个请求。这个算法的问题在于它无法平滑处理请求，因为在窗口边界可能会出现瞬间的高负载。
2. 滑动窗口计数限流算法：这种算法改进了固定窗口算法，使用滑动窗口来平滑处理请求。窗口内的请求计数按照时间的流逝而衰减，从而减少了窗口边界的尖峰负载。
3. 令牌桶算法：令牌桶算法使用令牌桶来控制请求速率。令牌以固定的速率被添加到令牌桶中，每个请求需要从令牌桶中获取一个令牌才能被处理。如果令牌桶中没有足够的令牌，请求将被延迟或拒绝。这种算法平滑控制了请求速率，并且可以处理突发请求。
4. 漏桶算法：漏桶算法以恒定的速率漏水，当请求到达时，会尝试向漏桶中添加请求，如果漏桶已满，则请求被拒绝。漏桶算法可以平滑请求，但不能处理突发请求。

我在项目中使用的 Redisson 的 RateLimiter 底层是基于令牌桶算法实现的。

> 建议阅读这篇文章：https://juejin.cn/post/6967742960540581918 <https://juejin.cn/post/6967742960540581918>

什么是 Redisson?

> 背诵类题目

参考官方文档：https://github.com/redisson/redisson <https://github.com/redisson/redisson>

Redisson 是一个开源的 Java Redis 客户端，也是一个分布式数据网格，提供了基于 Redis 的各种分布式数据结构和便捷的操作接口，比如分布式集合、分布式锁、分布式队列等，让开发者能够更轻松地使用 Redis 进行数据存储和中间件处理。

在本项目中，我使用 Redisson 自带的 RateLimiter 实现了针对单个用户生成图表操作的限流。

什么是 Redisson 的 RateLimiter？它在项目中具体如何实现分布式限流？你的限流策略是什么？

主观回答

RateLimiter 是 Redisson 基于 Redis 实现的分布式限流组件，底层使用令牌桶算法，能够控制某个操作或服务在一定时间内的请求频率，保护系统不被过多的请求压垮。

在项目中，考虑到 AI 生成图表是一个耗时且耗费资源的操作，我决定给 AI 生成图表接口增加限流，具体的策略是：单个用户每秒内最多执行 2 次生成图表操作。

具体的实现方式如下：

1）集中管理限流器：创建一个 RedisLimiterManager 类，集中管理整个项目中所有的限流器，并提供创建限流器的接口。

2）创建限流器：通过向 redissonClient 传入指定的 key 来创建限流器，每个用户对应的 key 不同，对应的限流器也不同，从而实现不同用户的独立限流

3）设置限流规则：通过 rateLimiter 的 trySetRate 方法制定限流规则，每秒内最多获取 2 个令牌

4）请求令牌：当用户要执行操作时，会执行对应 rateLimiter 的 tryAcquire 方法尝试获取令牌，如果能够获取到，可以执行后续操作，否则抛出 TOO_MANY_REQUEST 异常。

什么是分库分表？为什么你选择对每份原始数据进行分表存储？有什么优缺点？

前半句背诵类题目，后半句主观回答

分库分表是一种数据库架构设计方法，通过将单一的数据库划分为多个子数据库（分库）和多个表（分表），将数据进行分散存储，从而提高查询性能、扩展性和负载均衡，解决大规模数据存储和查询的性能问题。

项目中，我对每份用户上传的原始表格数据进行分表存储，不仅能够防止单表数据量过大对整个 Chart 表的查询性能造成影响；还能够支持用户更灵活地管理自己上传的数据，比如根据某个字段对表格内的数据进行过滤查询等，通过分表也能大幅提高每份数据的查询检索性能。

分库分表的缺点是会增加项目和架构的复杂性，如果业务涉及关联查询、跨分表查询，查询逻辑会变得更加复杂。所以还是要根据自己的实际业务场景和数据特点来决定是否选用分库分表。

什么是同步和异步？什么是阻塞和非阻塞？

> 背诵类题目

同步和异步：关注消息的通知机制。

同步是指完成一个任务并得到返回结果后，才能进行下一个任务；异步是指下一个任务不需要等待上一个任务的完成和返回结果，等任务完成后通过回调机制通知调用者。

用打电话来举例：

- 同步：当你拨打电话并等待对方回应时，你必须一直等待对方说完并回答你的问题，然后才能继续进行下一步操作，比如提出下一个问题。这就是同步操作，你的行动是与对方的回应直接相关的，一次只能处理一个问题。
- 异步：你可以拨打电话并留下问题，然后立即挂断电话。此时，你可以继续进行其他活动，而不必等待对方的回应。对方在有回应时会发短信通知你，等你有空了再查看短信。这样，你的行动与对方的回应是分离的，你可以并行处理多个问题。

阻塞和非阻塞：关注线程等待消息通知时的状态。

阻塞是指当前执行的任务结果返回之前，该线程会一直等待返回结果，不能执行其他任务；非阻塞是指当前任务执行的结果返回之前，线程不用等待返回结果，可以执行其他任务。

用打电话来举例：

- 阻塞：当你拨打电话向对方求助问题时，无论对方是立刻回复你、还是稍后给你回电，你在这期间都要一直等待，不能做其他的事情。
- 非阻塞：当你拨打电话向对方求助问题后，你不需要一直等着别人的回复，可以去做其他的事情，定期来查看手机上有无回复即可。

建议阅读这篇文章：https://blog.csdn.net/qq_41687938/article/details/119606435 <https://blog.csdn.net/qq_41687938/article/details/119606435>

什么是线程池？使用它有哪些好处？

> 背诵类题目

线程池是一种用于管理和复用线程的机制，它允许在应用程序中创建一组可用线程，并在需要执行任务时将任务分配给这些线程。

线程池的主要目的是优化线程的创建、销毁和管理，以提高多线程应用程序的性能和效率。

可以把线程池理解为公司（或者管理机制），线程理解为员工，任务理解为公司的工作

具体的好处如下：

1. 降低线程创建和销毁的开销：线程的创建和销毁是开销较大的操作，线程池可以重复利用线程，减少这些开销。（理解为公司不会每个任务都去招新人、用完再开除）
2. 控制并发度：通过配置线程池的大小，可以限制并发执行的任务数量，防止系统过载。
3. 统一管理和监控：线程池提供了统一的管理和监控接口，方便对线程的状态和执行情况进行监控和调整。
4. 避免资源竞争：在多线程环境中执行任务，可能会导致资源竞争和线程冲突，线程池可以帮助避免这些问题。

线程池有哪些核心参数？为什么在本项目中选择 IO 密集型线程池？

前半句背诵类题目，后半句主观回答

项目中，我使用 ThreadPoolExecutor 实现灵活的自定义线程池，并通过 ArrayBlockingQueue 存放任务。针对每类不同的业务，我分别定义不同的线程池，让它们互不影响。

示例代码：

```
1   ExecutorService executorService = new ThreadPoolExecutor(40, 1000, 10000, Time
```

其中，自定义线程池的核心参数如下：

1. 核心线程数（corePoolSize）：线程池中一直保持活动的线程数。可以使用corePoolSize方法来设置。一般情况下，可以根据系统的资源情况和任务的特性来设置合适的值。
2. 最大线程数（maximumPoolSize）：线程池中允许存在的最大线程数。可以使用maximumPoolSize方法来设置。如果所有线程都处于活动状态，而此时又有新的任务提交，线程池会创建新的线程，直到达到最大线程数。

3．空闲线程存活时间（keepAliveTime）：当线程池中的线程数量超过核心线程数时，如果这些线程在一定时间内没有执行任务，则这些线程会被销毁。可以使用keepAliveTime和TimeUnit方法来设置。

4．阻塞队列（workQueue）：用于存放等待执行的任务的阻塞队列。可以根据任务的特性选择不同类型的队列，如LinkedBlockingQueue、ArrayBlockingQueue等。默认情况下，使用无界阻塞队列，即LinkedBlockingQueue，但也可以根据需要设置有界队列。

5．线程工厂（threadFactory）：用于创建线程的工厂。可以通过实现ThreadFactory接口自定义线程的创建逻辑。

6．拒绝策略（rejectedExecutionHandler）：当线程池无法接受新的任务时，会根据设置的拒绝策略进行处理。常见的拒绝策略有 AbortPolicy、DiscardPolicy、DiscardOldestPolicy 和 CallerRunsPolicy。

我是根据任务的类型以及消耗资源的情况来调整线程池的参数。比如本项目中，需要线程池执行的任务为 “AI 生成内容”，该任务需要等待第三方 AI 接口的返回，需要消耗较长的时间，在这期间不需要高强度的 CPU 计算。所以我选择 IO 密集型线程池，将核心线程数设置得更大，比如 CPU 核心数的 2 - 4 倍，能够提高系统的并发性能。

分布式消息队列有哪些应用场景?

> 背诵类题目，也可以有主观回答

分布式消息队列的作用是将消息从一个发送者传递给一个或多个接收者，从而实现解耦、异步、可靠的系统通信。

常见的应用场景有：

1）应用解耦：解耦系统内的不同模块，使它们能够独立开发、部署和扩展。一个模块可以将消息发送到队列，而其他模块则可以异步地接收并处理这些消息，而不需要直接调用模块的 API。

2）异步处理：可以将耗时的任务作为消息放入消息队列中，然后由单个或多个工作线程来处理，从而提高系统性能和响应速度。

3）流量削峰：当系统面临突发的高峰流量时，分布式消息队列可以通过请求排队的方式实现流量的平滑处理，防止系统过载。

4）日志和监控：分布式消息队列可以高效地收集和传输日志数据、监控数据和指标数据，这是 Kafka 的一个典型应用场景。

5）跨语言和跨平台通信：分布式消息队列通常提供多语言和多平台的客户端，因此可以用于不同编程语言和操作系统之间的通信，本质上也是应用解耦。

6）分布式事务：有些分布式消息队列支持分布式事务，可以用于确保消息的可靠传递和处理，同时保持一致性。

在本项目中，使用 RabbitMQ 消息队列来对耗时的 AI 生成任务进行异步化处理，同时可以解耦创建任务和执行任务模块，在系统负载增大时，可以开启多个任务执行服务，用轮训的方式消费消息、并发处理任务。

你在项目中为什么使用分布式消息队列来存储任务消息?

> 背诵类题目

这题和上一题有一些类似，更侧重于考察分布式消息队列的优点。

相比于通过本地内存开启队列来存储任务消息，使用分布式消息队列有如下好处：

1. 分布式存储：消息存储在分布式的消息队列中而不是本地，有利于分布式系统的扩展。
2. 提高系统可靠性：分布式消息队列通常会保证消息的可靠传递，确保消息不会丢失，未即时处理的任务可以在消费者准备好时再次处理。
3. 高可扩展性：使用分布式消息队列，可以轻松地添加多个任务消费者，提高系统的并发处理能力。
4. 任务重试：分布式消息队列通常支持消息的重试机制。如果某个任务由于某种原因未能成功处理，消息队列可以重新将其推送给消费者，直到成功为止。

在本项目中，使用了 RabbitMQ 的消息确认机制，只有接受消息并处理成功的情况下，才会确认消息；否则拒绝消息并通过死信队列进行降级处理（比如修改任务状态为失败）。通过这种机制，可以确保每一个任务都被系统处理，不会出现丢失。

为什么使用 RabbitMQ 这个消息队列？它相比于其他的消息队列有哪些优点和缺点?

> 背诵类题目，可以加主观回答

在选用 RabbitMQ 消息队列前，我做过充分的技术选型和调研。发现 RabbitMQ 不仅简单易用（通过阅读官方文档就能上手开发），而且其时效性极低（延迟最低，微秒级）。此外，RabbitMQ 支持消息确认机制、延迟队列、死信队列等特性，能够满足业务对于消息可靠性的需求，这是我选择它的原因。

具体的优点和缺点请见消息队列技术对比表格：

技术名称	吞吐量	时效性	可用性	可靠性	优势	应用场景
activemq	万级	高	高	高	简单易学	中小型企业、项目
rabbitmq	万级	**极高（微秒）**	高	高	生态好（基本什么语言都支持）、时效性高、易学	适合绝大多数分布式的应用，这也是先学他的原因
kafka	**十万级**	高（毫秒以内）	**极高**	**极高**	吞吐量大、可靠性、可用性，强大的数据流处理能力	适用于 **大规模**处理数据的场景，比如构建日志收集系统、实时数据流传输、事件流收集传输
rocketmq	十万级	高（ms）	**极高**	**极高**	吞吐量大、可靠性、可用性，可扩展性	适用于 **金融** 、电商等对可靠性要求较高的场景，适合 **大规模** 的消息处理。
pulsar	十万级	高（ms）	**极高**	**极高**	可靠性、可用性很高，基于发布订阅模型，新兴（技术架构先进）	适合大规模、高并发的分布式系统（云原生）。适合实时分析、事件流处理、IoT 数据处理等。

💡 多说几句：

绝大多数的面试官其实心里都清楚，你为啥做项目的时候用某个技术，肯定要么就是看了网上的教程、要么就是你只学过这个技术。虽说如此，如果你直白地回答：“因为我学的就是RabbitMQ！” 那就寄了。

这其实是一个开放性问题，面试官主要想考察的是你的知识广度、以及技术选型的能力，因为像我们在实际开发中，一定都是基于业务需要才使用某个技术，而不能太死板地为了用技术而用技术、学过什么就只用什么。

应该怎么回答呢？我的建议是：

1）先说一下自己了解过的所有的主流消息队列，比如 Kafka、RocketMQ 等

2）分别说明每个消息队列的核心优缺点以及主流应用场景，比如 Kafka 吞吐量极高，适用于大规模处理数据、构建日志收集系统等

3）结合实际应用场景，说明自己的选择。RabbitMQ 的优势是易学易用、并且有着极高的时效性，我们的 OJ / 智能 BI 项目系统用户量并不多，使用它完全能够满足需求。而且它的生态好，支持多语言的客户端、支持分布式，也有利于我后续的学习以及项目扩展。

然后面试官可能就会根据你的回答，继续问你一些消息队列的问题啦，做好准备即可。

RabbitMQ 有哪几种交换机？你在项目中选择了哪个交换机？

> 前半句背诵类题目，后半句主观回答

RabbitMQ 主要有 4 种交换机：

1．Direct Exchange（直连交换机）：根据消息的路由键（Routing Key）将消息发送到与之完全匹配的队列上。通常用于点对点通信。
2．Fanout Exchange（广播交换机）：将收到的消息发送到所有与之绑定的队列上，忽略路由键。通常用于广播消息给多个消费者。
3．Topic Exchange（主题交换机）：根据消息的路由键和绑定队列的通配符模式来进行匹配，将消息发送到符合匹配规则的队列上。通常用于支持灵活的消息路由、相对复杂的业务场景。
4．Headers Exchange（头交换机）：根据消息的头部信息来进行匹配，将消息发送到符合匹配规则的队列上。这种方式用的不多，适用于基于消息头部信息进行路由、相对复杂的业务场景。

我在项目中使用了 Direct 类型的交换机，因为项目只有 1 种生产者、1 组消费者、1 种消息类型，选用 Direct 交换机的点对点通信已经能够满足需求，且便于理解。

你在 Spring Boot 项目中是如何使用 RabbitMQ 的？请简述消息发送和处理流程。

> 主观回答

1）引入 spring-boot-starter-amqp 依赖，并且在 application.yml 中编写 rabbitmq 连接配置

2）编写初始化队列类，创建 Direct 交换机、队列，以及交换机和队列的绑定

3）创建生产者类：通过 Spring Boot 整合的 RabbitTemplate 类来操作 MQ，比如调用其 convertAndSend 方法，向指定的交换机发送指定路由键的消息，比如发送 AI 生成任务的 id。

4）创建消费者类：通过 @RabbitListener 注解标识处理消息的方法，并在方法内编写对应的业务逻辑（比如获取 AI 生成任务的 id、消息确认等）

这个项目的性能是否有遇到瓶颈？如何优化项目？

1）由于 OpenAI 的 API QPS 有限，项目实际运行时出现了调用超限的报错，所以我们需要通过限流、或者增加负载均衡（轮询调用多个 OpenAI 账号）的方式来保证系统的稳定性。

2）由于 OpenAI 单次能处理的数据量有限，所以我们需要通过数据格式压缩，来让系统同时处理更多行列的数据。如果发现压缩效果差，可以优化算法；如果发现压缩效率低，可以用文件分片 + 并发压缩。

3）由于 AI 响应时间较长，用户要等待很久，我们需要将同步等待优化为异步执行，能立刻给用户提交成功的提示，从而在提高用户体验的同时，支持更多用户使用系统

4）随着用户图表数的增多，可能前端查询速度会越来越慢，这时就需要通过缓存来优化查询性能。

当然，上面几点都是围绕性能优化，其实项目的优化远不止性能，还有可用性、稳定性、可观测性等等，之前都分享过，可以阅读《如何增加项目亮点？》这篇文章：https://www.code-nav.cn/course/1789178931875651586/section/1789181560999616514?type=<https://www.code-nav.cn/course/1789178931875651586/section/1789181560999616514?type=>

举个例子，比如成本的优化。随着用户图表数据量的增大，可能后端的存储成本越来越高，这时就需要通过数据压缩、转储冷备、数据清理等机制来降低成本。

前端

> 由于技术栈相同，和用户中心、API 开放平台项目前端面试题几乎一致

请介绍你在项目中使用的 React 框架的优势和适用场景?

> 背诵类题目

React 是主流的前端开发框架。

优势：

1. 组件化开发： React 的核心概念是组件化开发，将UI拆分为独立的可重用组件，使代码更易于理解、维护和测试。
2. 虚拟 DOM： React引入了虚拟DOM的概念，通过在内存中维护虚拟DOM树，最小化了实际DOM操作，提高了性能和响应速度。
3. 单向数据流： React采用了单向数据流模型，使数据流动可控，易于跟踪和调试。
4. 生态系统：React拥有丰富的生态系统，包括大量的第三方库和组件，可以加速开发并提供各种解决方案。
5. 跨平台支持： React可以用于构建Web应用、移动应用（React Native）、桌面应用（Electron）等，使得开发团队可以在不同平台上共享代码和技能。
6. 社区支持： React拥有庞大的开发者社区和活跃的维护团队，可以获得大量的教程、文档和支持。

简单来说，就是用的人多、性能很好、生态成熟。

适用场景：

1．单页面应用（SPA）： React非常适合构建单页面应用，其中所有交互都在同一个页面中完成，无需每次加载新页面。
2．大规模应用： React 的性能和可维护性使其成为开发大规模应用的理想选择，特别是需要多人协作的项目。
3．跨平台开发： 如果需要同时支持Web、iOS和Android，可以使用React和React Native来共享代码和技能。
4．服务端渲染（SSR）： React可以用于服务器端渲染，提高应用的性能和搜索引擎优化（SEO）。

简单来说，适合开发企业级、高性能应用。

项目中为什么选择了 Ant Design Pro 脚手架？可以谈谈你对 Ant Design Pro 的使用体会和优缺点。

> 主观题目

为了提高开发效率。

Ant Design Pro 是一个开箱即用的企业级前端应用开发脚手架，基于 React 和 Ant Design 构建，能够通过命令行选项的方式，快速创建一个包含示例页面、全局通用布局、权限管理、路由管理、国际化、前端工程化的默认项目。我只需要在此基础上编写和业务相关的页面即可，大幅节省时间。

Ant Design Pro 虽然功能强大且方便，但是对新手不够友好，一方面是项目功能太多，理解项目中的代码文件、系统阅读官方文档都要花费不少时间；另一方面是框架封装的代码较多，如果现有的功能不满足要求，自己定制开发新能力的成本较大。而且 Ant Design Pro 框架更新迭代太快，阅读文档和安装依赖时一定要选择指定的版本，否则可能就会出现依赖冲突、代码不兼容的情况。

你是如何根据业务定制前端项目模板的？这个模板具体有哪些功能？

> 主观回答

1）项目瘦身：首先我使用 Ant Design Pro 提供的脚手架创建了一个基础的中后台项目，默认拥有了项目全局布局、前端工程化校验能力。然后我根据自己的实际需要，移除了 mock、国际化、测试相关的代码，精简了项目。

2）全局异常处理：我修改了 app.tsx 入口文件的 request 对象，自定义了全局请求响应拦截器，能够自动根据后端业务返回的错误码进行相应的处理。比如后端返回 code = 40100 未登录时，

全局异常处理逻辑会自动给前端用户弹出 Message 提示、并且重定向到登录页。

3）通用业务功能：我开发了用户登录注册页，并且将用户登录态托管到 Ant Design Pro 提供的 initialState 全局状态中进行管理，通过修改 access.ts 和路由配置文件实现了自动化权限管理。

4）请求代码生成：我在 config 配置文件中添加了根据后端 Swagger 接口文档自动生成前端请求后端代码的配置，只需要执行一行 OpenAPI 插件命令，就能自动生成代码。

当我需要开发一个新项目时，只需要 git clone（或复制）项目模板，全局替换页面的标题和 Logo 等基础信息，就可以根据需求开发页面了，大幅提高了开发效率。

你是如何保证项目编码规范的？用到了哪些技术？每个技术分别有什么作用？

> 主观回答

我通过一系列的工具和技术保证项目编码规范，包括 TypeScript、ESLint、Prettier 和 Husky。

各技术的作用如下：

- TypeScript：支持静态类型检查功能，用于增强代码的类型安全性，减少类型相关的错误。
- ESLint：代码检查工具，用于检测 JavaScript 的代码质量、风格等问题，强制开发者遵守编码规范。
- Prettier：代码格式化工具，能够根据配置自动格式化代码，确保代码的可读性和一致性。
- Husky：用于管理 Git 钩子的工具，在 Git 提交操作前自动运行代码检查脚本，确保提交的代码质量和规范。

如果是团队协作开发的项目，还可以使用代码审查机制，让其他人帮忙审查代码，减少潜在的 Bug 和编码不规范问题。

什么是 Husky？它有什么作用？

> 背诵类题目，也可以加主观回答

Husky 是一个用于管理 Git 钩子的工具。Git 钩子是一些在特定 Git 操作（如提交、推送、合并等）发生时自动触发的脚本或命令。Husky 的主要作用是帮助开发团队在代码提交前执行预定义的自定义脚本，以确保代码的一致性、质量和安全性。

Husky 通常与其他工具（如 lint-staged + ESLint + Prettier、commitlint 等）一起使用。开发者需要在项目中配置钩子脚本，然后在执行 Git 操作时，这些脚本将自动运行，执行代码检查和验证流程。

举个例子，如果项目使用 ESLint 进行代码风格检查，可以通过 Husky 在提交代码前运行 ESLint 检查，如果代码不符合规范，将阻止提交操作。这有助于维护代码库的一致性，确保团队成员都遵循代码规范。

什么是 Umi OpenAPI 插件？它有什么作用？你是如何使用它来生成代码的，请介绍整个流程。

> 主观回答

Umi OpenAPI 插件是一个用于 UmiJS（一个基于 React 的前端开发框架）的插件，它的主要作用是根据 OpenAPI 规范生成前端代码，包括 API 请求函数和数据模型定义。

使用这个插件，可以大大简化前端和后端 API 集成的过程，提高开发效率。

项目中，我使用的 Ant Design Pro 框架默认整合了 Umi OpenAPI 插件，无需手动引入依赖。

只需要在 config 配置文件中指定后端 Swagger 接口文档的地址、以及生成的规则（比如使用 request 库），示例代码如下：

```
openAPI: [
  {
    requestLibPath: "import { request } from '@umijs/max'",
    // 指定接口文档地址
    schemaPath: "http://localhost:8080/api/v2/api-docs",
    projectName: 'yubi',
    mock: false,
  }
],
```

然后执行 package.json 中定义的 openapi 命令，即可自动生成请求代码。

你在项目中使用了 ECharts 可视化库来实现接口调用的分析图表，请解释一下 ECharts 的主要功能？为什么说它的兼容性较好？

> 主观回答

ECharts 是一个开源的数据可视化库，开发者只需要传入图表配置和数据，就能得到精美的交互式数据可视化图表。

选用 ECharts 主要是因为它的生态广、兼容性强、图表丰富，并且官网还提供了 Playground 页面来调试图表的样式。

在项目中，我使用 ECharts 展示 AIGC 生成的图表。

ECharts 的兼容性好主要体现在以下几个方面：

1. 版本更新稳定：2020 年 12 月 ECharts 发布了 v5 版本，且至今 v5 仍是主流版本号。
2. 跨平台：ECharts 可以在各种主流 Web浏览器中运行，包括 Chrome、Firefox、Safari、Edge 等。
3. 响应式设计：ECharts 的图表可以适应不同屏幕尺寸和设备类型，从桌面到移动端都能够正常显示。
4. 开源社区支持：ECharts 拥有庞大的开源社区支持，持续维护和更新，大多数的 Bug 都能得到解决和修复。

你是如何使用 ECharts 接受后端的动态 JSON 自动渲染可视化图表的？请简述整个流程。

> 主观回答

首先我通过阅读 ECharts 官方文档和 Playground 调试明确了后端要返回的 JSON 数据格式，包括图表名称、横纵坐标、数据内容等。

在接收到后端返回的 JSON 后，我会先进行返回值校验，然后使用 JSON.parse 将 JSON 字符串转化为 JS 对象，最后将该 JS 对象作为 options 参数传递给 ECharts 组件，组件会自动渲染可视化图表。

请解释什么是 React 组件的生命周期，以及生命周期函数的执行顺序是怎样的?

> 背诵类题目

React 组件的生命周期指的是组件在不同阶段（或生命周期阶段）内执行的一系列特定函数。这些生命周期函数允许你在组件的不同生命周期阶段执行特定的操作，例如初始化组件、处理数据、更新UI等。在 React 16.3 版本之后，生命周期函数分为三类：

1）挂载阶段（Mounting）：这是组件被创建并插入到DOM中的阶段。在挂载阶段，有三个生命周期函数：

- `constructor()`：构造函数，用于初始化组件的状态和绑定事件处理程序。在组件生命周期中只会被调用一次。
- `render()`：渲染函数，用于生成虚拟DOM。它在组件挂载之前和每次更新时都会被调用。
- `componentDidMount()`：组件挂载后立即调用。通常用于执行DOM操作、网络请求、订阅事件等初始化操作。

2）更新阶段（Updating）：这是组件重新渲染并更新到DOM中的阶段。在更新阶段，有两个生命周期函数：

- `shouldComponentUpdate(nextProps, nextState)`：用于决定组件是否应该更新。默认情况下，它返回 `true`，但你可以根据新的 `props` 和 `state` 与当前的 `props` 和 `state` 进行比较，来决定是否需要更新组件。
- `componentDidUpdate(prevProps, prevState)`：组件更新后调用，通常用于执行与DOM相关的操作、网络请求、订阅事件等。

3）卸载阶段（Unmounting）：这是组件从DOM中移除的阶段。在卸载阶段，只有一个生命周期函数：

- `componentWillUnmount()`：组件即将被卸载时调用。通常用于清理工作，如取消订阅、清除定时器等。

此外，React 16.3 版本后引入了一些新的生命周期函数，例如 `getDerivedStateFromProps` 和 `getSnapshotBeforeUpdate`，用于更好地管理组件的状态和副作用。

生命周期函数的执行顺序如下：

1）挂载阶段：

- `constructor()`
- `render()`
- `componentDidMount()`

2）更新阶段：

- `shouldComponentUpdate(nextProps, nextState)`
- `render()`
- `componentDidUpdate(prevProps, prevState)`

3）卸载阶段：

- `componentWillUnmount()`

需要注意的是，React 17 及以后的版本中，部分生命周期函数被标记为不推荐使用，并有可能在未来的版本中被移除。因此，在新项目中建议使用更现代的生命周期方法，如 `componentDidCatch`、`getDerivedStateFromProps` 等，以更好地处理组件的状态和副作用。

在 React 中，什么是 Virtual DOM？它的作用是什么？与真实 DOM 相比，Virtual DOM 有什么优势？

> 背诵类题目

Virtual DOM（虚拟 DOM）是React中的一个关键概念，它是一个轻量级的内存中表示真实 DOM 树的数据结构，其作用是优化页面渲染性能和提高开发效率。

Virtual DOM 的作用和优势如下：

1）性能优化：

- 快速更新： 当组件的状态发生变化时，React首先创建一个虚拟DOM树，然后将新旧虚拟DOM树进行比较（称为协调或调和），找出需要更新的部分，并将这些更新一次性批量操作到真实DOM中，而不是每次都直接操作真实DOM。这样可以减少真实DOM操作的次数，提高性能。
- 批处理更新： React会将多个状态更新合并为一个更新，以减少重排（reflow）和重绘（repaint）的开销，从而提高渲染效率。

2）跨平台支持：Virtual DOM的设计使得React不仅可以用于Web开发，还可以用于构建移动应用（React Native）和桌面应用（Electron），因为虚拟DOM树与平台无关。

3）提高开发效率：使用虚拟DOM可以使开发人员更容易管理和维护UI组件，因为它将UI的状态和视图分离，并提供了声明式的UI编程模型，使得UI的开发更直观和可预测。

4）减少直接DOM操作：直接操作真实DOM通常会导致性能问题，而Virtual DOM可以帮助开发人员避免直接DOM操作，从而提高代码的可维护性。

5）跨浏览器兼容性：虚拟DOM使得React可以处理不同浏览器之间的差异，开发人员无需担心不同浏览器的兼容性问题。

在前端开发中，如何处理跨域请求？请描述常见的跨域解决方案以及它们的优缺点。

> 背诵类题目，也可以包含主观回答

可以使用 JSONP 或者配置代理服务器来实现。比如在 Ant Design Pro 框架中，提供了一个 proxy 配置，可以直接开启对后端服务器的代理，从而解决跨域。

JSONP 的优点是简单易用，缺点是仅支持 GET 请求，且容易受到跨站脚本攻击。真实业务场景中，更推荐在服务端解决跨域，前端本地开发时通过代理绕过跨域即可。

请解释一下 React Hooks，并举例说明如何使用 useState 和 useEffect 这两个常用的 Hooks。

12650字

背诵类题目

React Hooks 是 React 16.8 版本引入的一项功能，它们允许函数组件（无状态组件）使用状态管理和其他 React 特性，以前这些功能只能在类组件中使用。React Hooks 的目标是使组件的状态逻辑更容易复用、理解和测试。两个最常用的 React Hooks 是 `useState` 和 `useEffect`。

1) `useState` Hook:

`useState` Hook 用于在函数组件中添加状态管理。它返回一个状态变量和一个更新该变量的函数，可以用来在函数组件中管理和更新状态。

示例:

```
import React, { useState } from 'react';

function Counter() {
  // 使用useState来声明一个状态变量count，初始值为0
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>

      <button onClick={() => setCount(count + 1)}>Increment</button>

      <button onClick={() => setCount(count - 1)}>Decrement</button>

    </div>

  );
}
```

在上面的示例中，`useState` Hook 用于创建一个名为 `count` 的状态变量和一个名为 `setCount` 的函数。通过 `setCount` 函数，我们可以更新 `count` 的值。每次点击按钮时，`count` 的值会更新，并重新渲染组件。

2) `useEffect` Hook:

`useEffect` Hook 用于在函数组件中执行副作用操作，例如数据获取、订阅事件、手动管理 DOM 等。它接受两个参数，第一个参数是一个函数，用于执行副作用操作，第二个参数是一个数组，用于指定依赖项。

示例:

```jsx
import React, { useState, useEffect } from 'react';

function DataFetching() {
  const [data, setData] = useState([]);

  useEffect(() => {
    // 在组件挂载后，使用fetch API获取数据
    fetch('https://jsonplaceholder.typicode.com/posts')
      .then((response) => response.json())
      .then((data) => setData(data))
      .catch((error) => console.error('Error:', error));
  }, []); // 空数组表示只在组件挂载后执行一次

  return (
    <div>
      <h2>Posts</h2>

      <ul>
        {data.map((post) => (
          <li key={post.id}>{post.title}</li>

        ))}
      </ul>

    </div>

  );
}
```

在上面的示例中，`useEffect` Hook 用于在组件挂载后执行数据获取操作。我们指定了一个空数组作为第二个参数，这表示副作用操作只在组件挂载后执行一次。如果指定了依赖项数组，`useEffect` 将在依赖项发生变化时重新执行。

在前端开发中，如何优化网页的加载性能和渲染性能？请提供一些常见的优化策略和技巧？

> 背诵类题目，但尽量多包含主观回答

优化网页的加载性能和渲染性能是前端开发中的关键任务之一，它可以提高用户体验并降低网站的跳出率。以下是一些常见的优化策略和技巧：

1）优化图片：

- 使用适当的图片格式：选择合适的图片格式（如WebP、JPEG、PNG）以减小文件大小。
- 压缩图片：使用压缩工具或在线服务来减小图片文件的大小。

- 使用响应式图片：为不同屏幕大小提供不同尺寸的图片，以避免加载过大的图片。

2）使用CDN（内容分发网络）：

- 将静态资源（如CSS、JavaScript、图片等）托管到CDN上，加速资源的加载。
- CDN可以将内容缓存到全球各地的服务器上，减少距离造成的延迟。

3）懒加载和预加载：

- 使用懒加载技术延迟加载图片和其他资源，只有当用户滚动到可见区域时才加载。
- 使用 `<link rel="preload">` 标签预加载关键资源，以提前下载可能需要的资源。

4）压缩和合并文件：

- 压缩JavaScript和CSS文件，减小文件大小。
- 合并多个CSS或JavaScript文件为一个，减少HTTP请求次数。

Ant Design Pro 使用 Webpack 打包工具，会自动对代码文件进行打包合并。

5）使用浏览器缓存：

- 使用 HTTP缓存头（如 `Cache-Control` 和 `ETag` ）来允许浏览器缓存资源。
- 使用 Service Worker 来创建离线应用程序，使用户在离线状态下也能访问网站。

6）减少重排和重绘：

- 避免频繁修改DOM，可以使用文档片段（DocumentFragment）进行批量操作。
- 使用CSS动画而不是JavaScript来实现动画效果，以减少重排和重绘。

7）代码分割（Code Splitting）：

- 将应用程序拆分为多个模块或块，按需加载模块，减小初始加载时间。
- 使用工具如 Webpack 的 `import()` 语法来实现代码分割。

Ant Design Pro 支持在项目的全局配置文件中配置按需加载。

8）使用异步加载：

- 将非关键的JavaScript代码标记为异步加载，以防止它们阻塞页面加载。

- 使用 `<script async>` 标签来异步加载脚本。

9）前端框架和库的优化：使用轻量级的前端框架或库，以减少不必要的代码。

Ant Design Pro 其实是一个比较重的开发框架，更适合中后台管理系统。

10）监控和性能分析：

- 使用工具如 Google PageSpeed Insights、Lighthouse 等来检查和分析性能问题。
- 使用性能监控工具来实时追踪网站的性能指标，以便及时优化。

注意，性能优化是一个持续的过程，需要不断地监测、测试和调整，以确保网站保持高性能。

通用

什么是 AIGC?

> 背诵类题目

AIGC（Artificial Intelligence Generated Content）生成式人工智能，是一种新型的技术，是指利用人工智能自动生成内容，比如文章、音频、视频等，大幅提高内容生产的效率。

ChatGPT 就是 AIGC 的一种应用，我也已经将其运用到自己的生活和工作中，比如用 ChatGPT 帮我解决 Bug、回答一些问题。

请介绍一下本项目的完整业务流程?

区别于传统的 BI 可视化系统，在本项目中，用户只需要导入 Excel 原始数据集、并输入自己的分析诉求，就能利用 AI 自动生成可视化图表和分析结论，实现数据分析的降本增效。

此外，用户还可以在 “我的图表” 页面内查看已有的图表，集中管理。

在开发过程中，你遇到过比较复杂的技术问题或挑战吗？如果有，请谈谈你是如何解决这些问题的?

可以从以上任意一道主观的面试题出发去讲，比如你在使用 AI 生成文章时，发现这个操作需要消耗很长的时间，用户一直在前端等待体验不是很好，所以使用消息队列将生成操作异步化。