1. **一个URL包括哪些**

协议（Protocol）：URL以协议开头，指定了访问资源所使用的协议，常见的协议有HTTP、HTTPS、FTP等。

域名（Domain Name）：域名是URL中的主机名部分，用于标识资源所在的服务器。例如，www.example.com是一个域名。

端口号（Port Number）：端口号可选，用于标识服务器上的具体服务。如果未指定端口号，默认使用协议的默认端口号（如HTTP默认端口号为80）。

路径（Path）：路径指定了服务器上资源的具体位置或路径。例如，/path/to/resource是一个路径。

查询参数（Query Parameters）：查询参数是可选的，用于传递额外的参数给服务器。查询参数以问号（?）开头，多个参数之间使用&符号分隔。例如，?key1=value1&key2=value2。

锚点（Anchor）：锚点也是可选的，用于在页面内跳转到指定位置。锚点以井号（#）开头，后面跟着锚点标识符。例如，#section1。

http://www.example.com:8080/path/to/resource?key1=value1&key2=value2#section1

在这个示例URL中，协议是HTTP，域名是www.example.com，端口号是8080，路径是/path/to/resource，查询参数是key1=value1和key2=value2，锚点是section1。

需要注意的是，URL中的各个部分之间使用特定的符号进行分隔，同时每个部分可能有一些规范和约束。对于一些特殊字符，还需要进行URL编码以确保其正确传输和解析。

1. **URL的相关测试点**

协议（Protocol）：检查URL是否以正确的协议开头，如http://、https://等。

域名（Domain Name）：验证URL的域名部分是否符合规范，包括域名格式、顶级域名等。

路径（Path）：检查URL的路径部分是否正确，包括路径格式、斜杠使用等。

查询参数（Query Parameters）：验证URL中的查询参数是否正确，包括参数名称、值的编码等。

锚点（Anchor）：检查URL中的锚点部分是否正确，包括锚点标识符的格式和位置。

用户认证（User Authentication）：如果URL包含用户认证信息，可以检查认证信息是否正确。

端口号（Port Number）：验证URL中的端口号是否正确，特别是非标准端口号的使用。

URL长度：测试URL的长度限制，确保URL不超过服务器或浏览器的限制。

URL编码：测试URL中的特殊字符是否正确编码，以避免引起解析错误或安全问题。

重定向（Redirect）：如果URL会进行重定向，验证重定向是否按预期进行，并检查重定向后的URL是否正确。

边界条件：测试URL的边界条件，如空URL、只包含协议的URL、只包含域名的URL等。

兼容性：测试URL在不同浏览器和操作系统上的兼容性，确保在各种环境下都能正常解析和访问。

这些测试点可以帮助确保URL的正确性和可用性。根据具体的需求和应用场景，可以选择适当的测试点进行测试，并使用相应的工具和技术来执行测试。

**3、不可迭代对象**

整数、浮点数、布尔值、None

字符串的每个字符本身并不是可迭代对象

**4、可迭代对象和迭代器**

在Python中，可迭代对象（Iterable）是指可以通过迭代器（Iterator）进行遍历的对象。可迭代对象包括但不限于以下几种：

序列类型：如字符串（str）、列表（list）、元组（tuple）、字节串（bytes）等。这些对象可以按照索引的方式访问元素，并且支持切片操作。

集合类型：如集合（set）和冻结集合（frozenset）。集合是无序的，不包含重复元素的可迭代对象。

字典类型：字典（dict）是由键值对组成的可迭代对象，可以通过键访问对应的值。

文件对象：打开的文件对象也是可迭代对象，可以按行遍历文件内容。

生成器（Generator）：生成器是一种特殊的可迭代对象，它的元素是通过函数的yield语句产生的。

迭代器对象

1、通过自定义类来实现可迭代性，只需要在类中定义一个\_\_iter\_\_()方法并返回一个迭代器对象即可。迭代器对象（Iterator）是实现了\_\_iter\_\_()和\_\_next\_\_()方法的对象，用于实现迭代过程。

2、可以使用iter()函数将可迭代对象转换为迭代器对象，然后使用next()函数来获取迭代器的下一个元素。

3、每次调用iter()函数都会返回一个新的迭代器对象

可迭代对象是具有\_\_iter\_\_()方法或\_\_getitem\_\_()方法的对象，它们可以通过iter()函数获取一个迭代器。迭代器是具有\_\_iter\_\_()和\_\_next\_\_()方法的对象，用于遍历可迭代对象的元素。可迭代对象可以被多次迭代，而迭代器只能被迭代一次

**5、如何判断一个对象是可迭代对象**

可以使用iter()函数来判断一个对象是否是可迭代对象

自定义类可以通过实现\_\_iter\_\_()方法来使其变成可迭代对象

isinstance()函数检查对象obj是否是Iterable类的实例

Iterable是Python标准库collections.abc模块中的一个抽象基类，用于表示可迭代对象

**6、序列化和反序列化**

pickle模块：pickle模块是Python标准库中用于序列化和反序列化的模块。它可以将对象转换为字节流（二进制数据），也可以从字节流中还原对象。pickle模块支持几乎所有的Python对象

json模块：json模块是Python标准库中用于处理JSON格式数据的模块。它可以将对象转换为JSON字符串，也可以将JSON字符串转换为对象。json模块只能序列化一些基本类型（如字典、列表、字符串等），不支持自定义类的序列化

**7、全局变量和局部变量的区别**

作用域：全局变量的作用域是整个程序，可以在任何地方访问。而局部变量的作用域只限于定义它们的函数内部。

生命周期：全局变量的生命周期从程序开始到结束，而局部变量的生命周期只在定义它们的函数执行期间。

访问范围：全局变量可以在程序的任何地方被访问，包括函数内部。但是，在函数内部如果有与全局变量同名的局部变量，局部变量会覆盖全局变量。

内存消耗：全局变量会一直占用内存，直到程序结束。而局部变量在函数执行完毕后会被销毁，释放内存空间。

变量名冲突：如果在函数内部使用了与全局变量同名的局部变量，那么在函数内部访问该变量时，将会使用局部变量而不是全局变量。

**8、python的锁和死锁**

可以使用acquire()方法获取锁，release()方法释放锁。

threading.Lock：简单互斥锁一次只允许一个线程访问被保护的代码块

threading.RLock：可重入锁允许同一个线程多次获取锁而不会造成死锁

死锁:当线程持有一个锁并试图获取另一个锁时，而另一个线程持有了需要的锁并试图获取第一个线程持有的锁，彼此等待对方释放资源，导致所有线程无法继续执行的状态

**9、mysql什么情况下会出现慢查询？如何解决？**

缺乏合适的索引：如果查询中涉及到的列没有合适的索引，MySQL会进行全表扫描，导致查询变慢。

复杂的连接查询：当查询中包含多个表的连接操作（如JOIN操作）时，如果没有合适的索引或者连接条件不恰当，查询性能会受到影响。

数据库统计信息不准确：MySQL使用统计信息来优化查询计划，如果统计信息不准确或者过时，可能会导致MySQL选择不合适的查询计划，从而导致慢查询。

大数据量查询：当查询涉及到的数据量较大时，无论是否有索引，查询的执行时间都可能变长。

锁竞争：如果查询需要获取锁并且存在锁竞争的情况，可能会导致其他查询被阻塞，从而导致慢查询。

不合理的查询设计：如果查询的条件不合理、查询语句写法不规范或者查询语句存在错误，都可能导致慢查询

如何解决？

优化查询语句，确保查询条件合理、索引使用正确，并尽量避免复杂的连接查询。

确保数据库统计信息准确，定期更新统计信息或者使用自动化工具进行统计信息收集。

优化数据库配置参数，如调整缓冲区大小、并发连接数等。

使用性能监控工具，及时发现慢查询，并对慢查询进行分析和优化。

避免长事务和锁竞争，合理设计事务边界和锁粒度。

**10、mysql约束**

主键约束：主键约束用于唯一标识表中的每一行数据。它要求被约束的列具有唯一性和非空性，一个表只能有一个主键。主键可用于快速查找和关联其他表。

唯一约束：唯一约束要求被约束的列具有唯一性，但允许为空值。一个表可以有多个唯一约束，用于确保数据的唯一性。

外键约束：外键约束用于建立表之间的关联关系。它指定了一个表中的列与另一个表中的主键或唯一键之间的关系。外键约束确保了数据的引用完整性，防止无效的关联操作。

非空约束：非空约束要求被约束的列不允许为空值。它保证了必需的数据完整性，防止插入或更新操作中的空值错误。

默认约束：默认约束为列指定了一个默认值，当插入新行时如果没有提供该列的值，则会自动使用默认值。它确保了数据的一致性，避免了空值或缺失值的情况。

检查约束：检查约束用于定义一个表中的列必须满足的条件。它可以是一个表达式、逻辑条件或函数。检查约束用于限制列值的范围或特定格式。

**11、mysql事务的特性和作用**

**特性:**

原子性（Atomicity）：事务中的操作要么全部成功执行，要么全部失败回滚。如果其中一个操作失败，则整个事务会被回滚到初始状态，不会对数据库产生影响。

一致性（Consistency）：事务执行前后，数据库的状态必须保持一致。在事务开始和结束时，数据库必须满足所有定义的完整性约束。

隔离性（Isolation）：并发执行的多个事务之间应该互相隔离，每个事务都应该感觉不到其他事务的存在。隔离性确保了事务在并发环境下的正确性。

持久性（Durability）：一旦事务提交，其所做的修改将永久保存在数据库中，并且对于系统故障或崩溃也是安全的。

**作用:**

数据库一致性：通过将相关的数据库操作组合为一个事务，可以确保数据的一致性。事务的原子性和一致性特性能够保证数据库在任何情况下都保持有效的状态。

并发控制：事务的隔离性特性确保了并发执行的多个事务之间相互隔离，避免了数据的不一致和冲突。通过锁定机制和其他并发控制技术，可以防止并发事务对同一数据进行冲突操作。

数据完整性：事务的一致性特性能够保证在事务开始和结束时，数据库满足定义的完整性约束。这样可以避免数据的不完整或错误状态。

错误回滚：如果一个事务中的某个操作失败，可以通过回滚操作将数据库恢复到事务开始前的状态。这样可以确保数据库的完整性，并且避免了因为部分操作失败而导致的数据错误。

**12、mysql视图**

虚拟的表，它基于一个或多个表的查询结果而创建。视图并不实际存储数据，而是根据定义的查询逻辑在查询时动态生成结果

作用:通过对数据的封装来简化查询的复杂度，从而更安全的访问一些不敏感的数据来保证每次查询的结果都是最新的一致结果

可以简化复杂查询、提供更安全的数据访问、实现数据逻辑抽象和保证数据一致性

数据安全性：通过使用视图，可以隐藏底层表的一些敏感数据，只向用户暴露需要访问的数据。视图可以对表进行过滤、限制和重组，以提供更安全的数据访问机制。

简化复杂查询：当某个查询操作需要多个表的联合查询、聚合操作或者其他复杂的逻辑时，可以将这些操作封装到一个视图中。这样，在实际查询时只需要引用视图，而不需要编写复杂的查询语句。

数据逻辑抽象：视图可以将多个底层表的数据组合为一个逻辑上的数据集合。通过视图，可以将底层数据的逻辑结构进行抽象和封装，简化了应用程序对数据的访问和操作。

数据一致性：当某个视图依赖于多个表，并且这些表的数据可能经常变化时，通过查询视图可以保证查询结果的一致性。即使底层表的数据发生了变化，通过查询视图仍然可以获得最新的一致结果。

简化权限管理：通过使用视图，可以对用户分配更细粒度的数据访问权限。管理员可以控制用户只能查询特定视图，而无需直接访问底层表，从而提高了数据安全性和权限管理的灵活性。

**13、mysql索引类型**

B-Tree索引:

B-Tree索引适用于查找某个范围内的记录，例如根据名字或者日期范围来查询记录。B-Tree索引可以优化等值查询和范围查询，并且支持多列索引。常见的包括:PRIMARY KEY（主键）、 UNIQUE（唯一）、INDEX（普通）等。

Hash索引:

这种索引适用于等值查询，例如根据ID来查询某个记录。Hash索引将索引列的值作为输入，通过Hash算法计算出一个指针，指向对应的记录。Hash索引无法优化范围查询，也无法支持排序，因此Hash索引在实际应用中使用较少;

Full-text索引:

这种索引适用于全文搜索，例如在文章中查找包含某个关键词的记录。Full-text索引可以提高全文搜索的效率，并且支持模糊匹配和相关度排序。

R-Tree索引:

这种索引适用于地理信息系统和空间数据存储。R-Tree索引可以高效地查询某个范围内的记录，并且支持空间数据的相关操作，例如查询某个点是否在某个多边形内;

**14、http错误码相关**

1 表示消息

2 表示成功

3 表示重定向

4 表示请求错误

5 表示服务器错误

1xx 代表请求已被接受，需要继续处理，这类响应是临时响应，只包含状态行和某些可选的响应信息，并一空行结束 常见的有：

100 （客户继续发送请求，这是临时响应） 这个临时响应是用来通知客户端它的部分请求已经被服务器接收，且仍未被拒绝。客户端印当据需发送请求的剩余部分，或者如果请求已经完成，忽略这个响应，服务器必须在请求完成后向客户端发送一个最终响应

101 服务器根据客户端的请求切换协议，主要用于 websocket 或 HTTP2 升级

2xx 代表请求已成功被服务器接收，处理，并接受

200 （成功） 请求已成功，请求所希望的响应头或数据体将随此响应返回 201 （已创建）请求成功并且服务器创建了新的资源 202 （已创建）服务器已经接受请求，但尚未处理 203 （非授权信息）服务器已成功处理请求，但返回的信息可能来自另一来源 204 （无内容）服务器成功处理请求，但没有返回任何内容 205 （重置内容）服务器成功处理请求，但没有返回任何内容 206 （部分内容）服务器成功处理了部分请求

3xx 表示要完成请求，需要进一步操作，通常这些状态代码用来重定向

300 （多种选择）针对请求，服务器可执行多种操作。 301 （永久移动）请求的网页已永久移动到新位置。 302 （临时移动）服务器目前从不同位置的网页响应请求，但请求者应该继续使用原有位置来进行以后的请求 303 （查看其它位置）请求者应当对不同位置使用单独的 GET 请求来检索响应时，服务器返回此代码 305 （使用代理）请求者只能使用代理访问请求的网页。 307 （临时重定向）服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来进行以后的请求

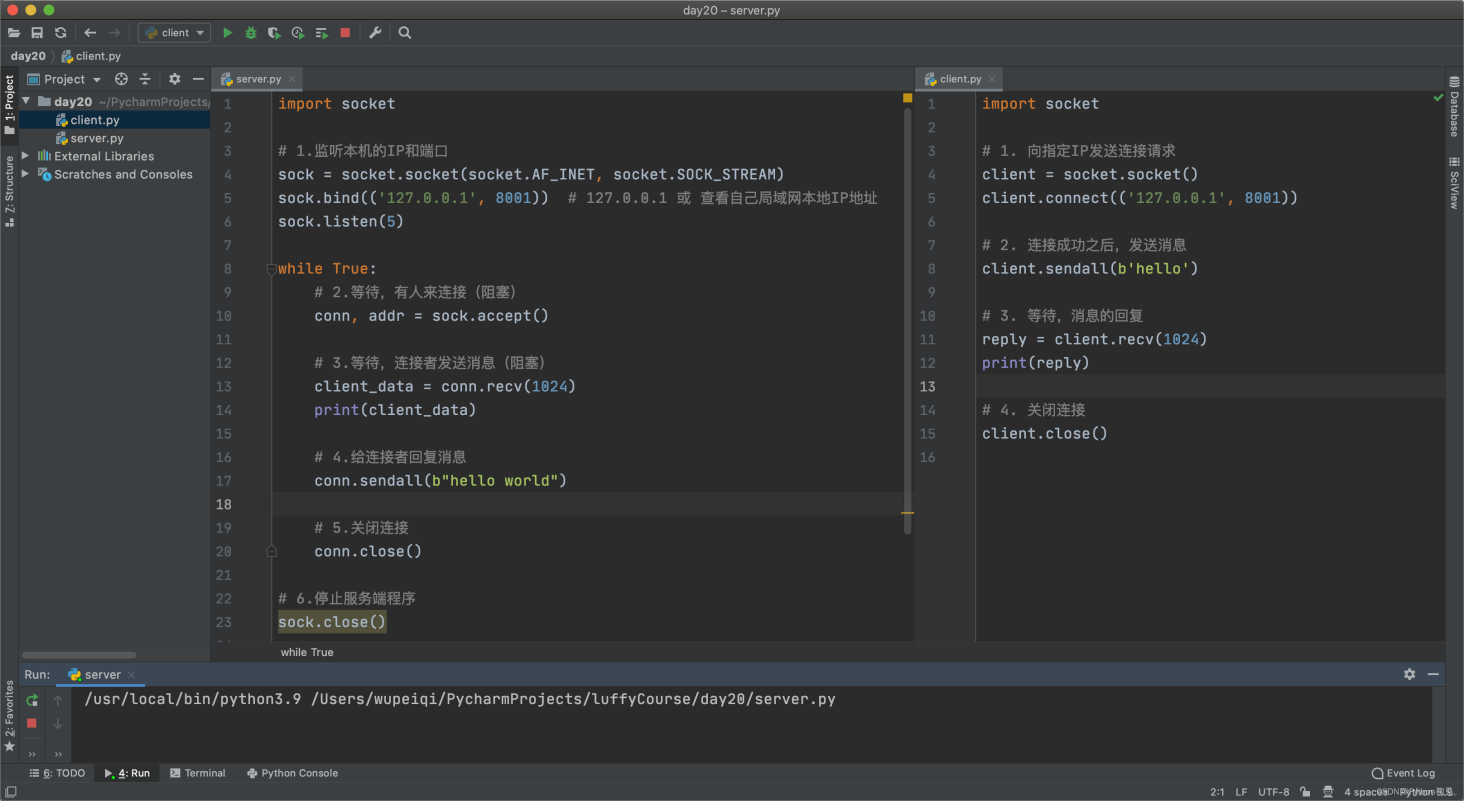
4xx 代表了客户端看起来可能发生了错误，妨碍了服务器的处理

400 （错误请求）服务器不理解请求的语法 401 （未授权）请求要求身份验证。 403 （禁止）服务器拒绝请求 404 （未找到）服务器找不到请求的网页 405 （方法禁用）禁用请求中指定的方法 406 （不接受）无法使用请求的内容特性响应请求的网页 407 （需要代理授权）此状态代码与 401（未授权）类似，但指定请求者应当授权使用代理 408 （请求超时）服务器等候请求时发生超时

5xx 表示服务器无法完成明显有效的请求。这类状态代码代表了服务器在处理请求的过程中有错误或异常状态发生

500 （服务器内部错误）服务器遇到错误，无法完成请求 501 （尚未实施）服务器服务器不具备完成请求的功能 502 （错误网关）服务器作为网关或代理，从上游服务器收到无效响应 503 （服务不可用）服务器目前无法使用，（由于超载或停机维护） 504 （网关超时）服务器作为网关或代理，但是没有及时从上游服务器收到请求 505 （ HTTP 版本不受支持）服务器不支持请求中所用的 HTTP 协议版本

**15、网络间数据传输**



**16、OSI七层模型**

报文

应用层：根据协议进行请求和收到响应从而规定数据的格式。

表示层：对应用层数据的编码、压缩（解压缩）、分块、加密（解密）等任务。

会话层：负责与目标建立、中断连接。

传输层：建立端口到端口的通信，其实就确定双方的端口信息。 段 服务进程

网络层：标记目标IP信息（IP协议层） 包 端

数据链路层：对数据进行分组并设置源和目标mac地址 帧 跳

物理层：将二进制数据在物理媒体上传输。比特

**17、TCP和UDP（传输层协议）**

在TCP建立连接后，使用UDP来传送数据单元

连接性：

TCP：TCP是一种面向连接的协议。在进行通信之前，需要先建立TCP连接，通过三次握手来确保双方的可靠连接。在数据传输完成后，通过四次挥手关闭连接。

UDP：UDP是一种无连接的协议，发送端直接将数据报发送给接收端，不需要事先建立连接。因此，UDP的传输效率较高，但不保证可靠性。

可靠性：

TCP：TCP提供可靠的数据传输，通过使用序列号、确认应答、重传机制和流量控制等方式，确保数据按照正确的顺序到达目的地，且不丢失和重复。

UDP：UDP不提供数据的可靠传输，数据报可能会在传输过程中丢失、重复或乱序。它适用于对实时性要求较高，但对数据准确性要求相对较低的应用。

通信效率：

TCP：由于TCP的可靠性机制和拥塞控制算法，会引入一定的开销，导致传输效率较低。但TCP能够适应网络状况的变化，保证网络的稳定性和公平性。

UDP：UDP没有复杂的机制，传输效率较高，适用于对实时性要求较高的应用，如视频流、音频流等。

**18、TCP三次握手和四次挥手**

防止已失效的连接请求再次传送到服务器，以及避免资源浪费

**客户端 服务端**

1. **SYN-SENT --> <seq=100><CTL=SYN> --> SYN-RECEIVED**

**客户端发送一个SYN（同步序列号编号）的报文，同时选择一个初始序列号**

1. **ESTABLISHED <-- <seq=300><ack=101><CTL=SYN,ACK> <-- SYN-RECEIVED**

**收到报文后，服务端发送SYN/ACK报文作为响应并确认客户端的一个ack同时自己也选择一个初始序列号**

1. **ESTABLISHED --> <seq=101><ack=301><CTL=ACK> --> ESTABLISHED**

**收到报文后，发送一个确认报文，然后确认服务器ack**

**TCP A（客户端） TCP B（服务器）**

**1. FIN-WAIT-1 --> <seq=100><ack=300><CTL=FIN,ACK> --> CLOSE-WAIT**

**2. FIN-WAIT-2 <-- <seq=300><ack=101><CTL=ACK> <-- CLOSE-WAIT**

**3. TIME-WAIT <-- <seq=300><ack=101><CTL=FIN,ACK> <-- LAST-ACK**

**4. TIME-WAIT --> <seq=101><ack=301><CTL=ACK> --> CLOSED**

**主动关闭方发送FIN标志，被动关闭方回确认ACK，被动关闭方发送FIN标志，主动关闭方回复确认ACK**

**19、粘包，如何解决？**

对于发送者，执行 sendall/send 发送消息时，是将数据先发送至自己网卡的 写缓冲区 ，再由缓冲区将数据发送给到对方网卡的读缓冲区。

对于接受者，执行 recv 接收消息时，是从自己网卡的读缓冲区获取数据。

如果发送者连续快速的发送了2条信息，接收者在读取时会认为这是1条信息，即：2个数据包粘在了一起

将消息划分为 头部（固定字节长度） 和 数据 两部分

发送数据，先发送数据的长度，再发送数据（或拼接起来再发送）。

接收数据，先读4个字节就可以知道自己这个数据包中的数据长度，再根据长度读取到数据。

header1 = conn.recv(4)

data\_length1 = struct.unpack('i', header1)[0] *# 数据字节长度 21*

**20、非阻塞**

**Sock.setblocking（False） #加上就变成非阻塞了**

程序运行时一旦遇到 accept、recv、connect 就会抛出 BlockingIOError 的异常。

这不是代码编写的有错误，而是原来的IO阻塞变为非阻塞之后，由于没有接收到相关的IO请求抛出的固定错误

**21、IO多路复用**

I/O多路复用指：通过一种机制，可以****监视多个描述符****，一旦某个描述符就绪（一般是读就绪或者写就绪），能够通知程序进行相应的读写操作。

Select + IO多路复用 + 非阻塞，可以实现让TCP的客户端同时发送多个请求

**22、RPC协议用于远程通信**

远程调用：RPC协议允许客户端应用程序发起远程调用请求，将参数传递给服务器端，并接收服务器端返回的结果。客户端和服务器端可以位于不同的物理机器或网络节点上。

透明性：RPC协议提供了透明性，使得远程调用对于客户端来说就像调用本地函数一样，隐藏了底层的网络通信细节

序列化和反序列化：在RPC协议中，参数和返回值需要在客户端和服务器端之间进行序列化和反序列化操作，以便在网络中传输。

通信协议：RPC协议可以基于不同的通信协议实现，如TCP/IP、HTTP等。常见的RPC框架如gRPC、Apache Thrift、Dubbo等都提供了不同的通信协议支持。

接口定义：RPC协议通常使用接口定义语言（IDL）来描述远程服务的接口，以便客户端和服务器端之间进行通信和协调。

远程异常处理：RPC协议需要支持远程异常处理机制，以便在远程调用过程中能够处理和传递异常信息。

并发和线程安全性：RPC协议需要考虑并发访问和线程安全性，确保在多个并发请求下能够正确处理和保护共享资源。

1. **TCP发送速率和窗口的关系**

TCP协议中的拥塞控制机制会根据窗口大小来调整发送速度。

TCP窗口是一个动态的概念，用于控制发送方可以连续发送多少个数据包而不需要等待确认。窗口大小由接收方通知发送方，发送方根据窗口大小来决定发送的数据量。

当窗口大小较小时，发送方每次只能发送少量的数据包，然后等待接收方的确认，再继续发送下一批数据包。这会导致发送速度相对较慢。

当窗口大小较大时，发送方可以连续发送更多的数据包，而无需等待接收方的确认。这会使得发送速度更快。

**24、TCP拥塞控制**

慢启动：在连接刚建立时，发送方会以较低的速率发送数据，并逐渐增加发送速率，以便快速找到网络的可用带宽。

拥塞避免：一旦发送方确定网络的可用带宽，它将以较慢的速率逐渐增加发送速率，以避免过多的数据包进入网络，从而导致拥塞。

快速重传和快速恢复：当发送方收到重复确认时，它会立即重传丢失的数据包，并根据接收方的确认信息调整窗口大小，以快速恢复数据传输。

超时重传：如果发送方在一定时间内未收到确认，它将假设数据包丢失，并重新发送这些数据包，以确保数据的可靠传输。