Java 的四个基本特性：（1）抽象，抽象是将一类对象的共同特征总结出来构造类的过程，包括数据抽象和行为抽象两部分，抽象只关注对象有哪些属性和行为，不关注行为的具体细节。（2）继承，继承是从已有类得到继承信息创建新类的过程。提供继承信息的类称为父类，得到继承信息的类称为子类。继承让变化中的软件系统有了一定的延续性，同时继承也是封装程序中可变因素的重要手段。（3）封装，通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已经定义的接口。面向对象的本质就是将现实描述为完全自治、封闭的对象。在类中编写的方法就是对实现细节的一种封装，编写一个类就是对数据和数据操作的封装。封装就是隐藏一切可隐藏的东西，只向外界提供最简单的编程接口。（4）多态性是只允许不同子类型的对象对同一消息作出不同的响应。方法重载实现的是编译时的多态性，即前绑定。方法重写实现的是运行时的多态性，即后绑定。方法重写，子类继承父类，并重写父类中已有的或抽象的方法。对象造型是用父类型引用子类型对象，同样的引用调用同样的方法，根据子类对象的不同表现出不同的行为。

面向过程是一种以事件为中心的编程思想，分析出解决问题的步骤，然后用函数把这些步骤实现，并按顺序调用。面向对象是以对象为中心的编程思想。

重载发生在同一个类中，同名的方法如果有不同的参数列表，参数类型，参数个数不同则为重载。重写发生在子类和父类之间，重写要求子类被重写方法与父类被重写方法有相同的返回值类型，比父类被重写方法更好访问，不能比父类被重写方法声明更多的异常。根据不同的子类对象确定调用的哪个方法。

面向对象开发的六个基本原则：（1）单一职责：一个类只做自己该做的事情，高内聚性，在面向对象中，如果只让一个类完成自己该做的事情，而不涉及其他无关的领域就是实现了高内聚性，就是单一职责。（2）开放封闭：软件实体应该对扩展开放，对修改关闭。一个系统中如果没有抽象类或者接口，系统就没有扩展点。将系统中的各种可变因素封装到一个继承结构中。（3）里氏替换：任何时候都可以用子类型替换父类型，子类一定是增加了父类的功能，而不是减少了父类的功能。（4）依赖倒置：面向接口编程，声明方法的参数类型，方法的返回类型，变量的引用类型时，尽可能使用抽象类型，而不是具体类型，因为抽象类型可以被它的任何一个子类型所替代。（5）合成聚合复用：优先使用合成或聚合关系复用代码。（6）接口隔离：一个接口只应该描述一种能力，接口也应该是高度内聚的。（7）迪米特：最少知识原则，一个对象应该对其他对象有尽可能少的了解。

处理构造相同 hash 的字符串进行攻击，当客户端提交一个请求并附带参数的时候，web 应用服务器会把参数转化为一个 HashMap 存储，逻辑结构是 key---value。但是物理存储结构是不同的，key 会被转化为 HashCode，HashCode 又会被转化为数组的下标，0---value。不同的 String 就会产生相同的 HashCode 导致碰撞。之后的物理结构为 0---value1---value2。方法是限制 POST 和 GET 的参数个数，越少越好。或者限制 POST 数据包的大小。

线程安全：定义某个类的行为与其规范一致，不管多个线程是怎样的执行顺序和优先级，

或者是 wait、sleep、join 等控制方式。如果一个类在多线程访问下运转一切正常，并且访问类不需要进行额外的同步处理或者协调，则认为是线程安全的。保证线程安全时可以对变量加 volatile 关键字，对于程序段可以加 synchronized 或者 lock。非线程安全的集合可以在多线程环境下使用，但是不能作为多个线程共享的属性，可以作为某个线程独享的属性。

Java 中内存泄漏：（1）静态集合类，当被定义为静态的时候，由于声明周期和应用程序一样长，可能发生内存泄漏。（2）当增加了监听器但是没有及时释放时，也会产生内存泄漏的问题。（3）物理连接，比如数据库连接和网络连接，除非显式关闭，否则不会被 GC。一般在 try 代码块创建连接，在 finally 中释放连接，避免此类内存泄漏。（4）内部类和外部模块等的引用，内部类的引用一旦没释放，可能会导致后序很多对象没有释放。在调用外部模块时也应该防止内存泄漏。（5）单例模式，单例对象初始化以后会在 JVM 的整个生命周期中都存在，如果其持有一个外部对象的引用，外部对象不能回收，则导致内存泄漏。

volatile 保持可见性的原理是每次访问变量时都会进行一次刷新，因为每次访问都是主内存中最新的版本，volatile 关键字保证内存可见性是其作用之一。当且仅当满足以下条件时才能使用 volatile 变量：（1）对变量的写入操作不依赖变量的当前值，或者能够确保只有单个线程更改变量的值。（2）该变量没有包含在具有其他变量的不变式中。（1）在两个或者更多的线程需要使用的成员变量上使用 volatile，当要访问的变量已经在 synchronized 代码块中，或者为常量时，没必要使用 volatile。（2）volatile 使用屏蔽掉了 JVM 中必要的代码优化，效率上比较低，在必要时再使用。

volatile 不会进行加锁操作，是一种比 synchronized 更轻量级的同步机制，在访问变量时不会加锁，也就不会使得执行线程阻塞。Volatile 变量作用类似于同步变量读写操作。写入变量相当于退出同步代码块，读取变量相当于进入同步代码块。

Sleep 是 Thread 类的方法，wait 是 Object 类的方法。Sleep 方法是线程类的静态方法，调用之后会让当前线程暂停执行指定时间，执行机会让给其他线程，但是对象的锁依然保持，因此休眠时间结束后会自动恢复。线程回到就绪状态。Wait 是 Object 类的方法，调用对象的 wait 方法导致当前线程放弃对象的锁，线程暂停执行，进入对象的等待池。只有调用对象的 notify 或者 notifyAll 方法时才会唤醒等待池中的线程进入等锁池，如果线程重新获得对象的锁就会进入就绪状态。

Synchronized 原始采用的是 CPU 的悲观锁机制，即线程获得的是独占锁。独占锁即其他线程只能依靠阻塞来等待线程释放锁。Lock 用的是乐观锁的方式，每次不加锁而是假设没有冲突而去完成某项操作，如果因为冲突失败就重试，直到成功为止。乐观锁实现的机制就是 CAS 操作。

Synchronized 用在代码块锁的是调用该方法的对象，也可以锁住任何一个对象。Synchronized 用在该方法上锁的是调用该方法的对象。Synchronized 用在代码块可以减小锁的粒度，从而提高并发性能。无论用在代码块上还是方法上，都是获取对象的锁。每一个对

象只有一个锁与之相关联，实现同步需要很大的系统开销作为代价，甚至可能造成死锁。

常见的异常分为 Exception 和 Error。Throwable 是 Java 语言中所有错误和异常的超类，两个子类分别为 Exception 和 Error。（1）Error 为错误，是程序无法处理的，出现这种情况只能交给虚拟机处理，大部分 JVM 会选择终止线程。(2)Exception 是程序可以处理的异常，分为 CheckedException 和 UncheckedException。CheckedException 发生在编译阶段，必须使用 try……catch 或者 throw，否则编译不通过。UncheckedException 发生在运行期，具有不确定性，主要是由于程序的逻辑问题所引起的。

Java 中的（1）BIO，同步并阻塞，服务器实现模式为一个连接一个线程，客户端有连接请求时服务端需要启动一个线程进行处理。BIO 适用于连接数目比较小且固定的架构，对服务器资源要求比较高。（2）NIO，同步非阻塞，服务器实现模式为一个请求一个线程，客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有 I/O 请求时才启动一个线程进行处理。NIO 适用于连接数目多且连接比较短的架构。（3）AIO，异步非阻塞，服务器实现模式为一个有效请求一个线程，客户端的 I/O 请求都是由 OS 先完成了再通知服务器应用去启动线程进行处理。AIO 适用于连接数目多且连接比较长的架构，充分调用 OS 参与并发操作。

设计模式：(1)工厂模式，定义一个用于创建对象的接口，让子类决定实例化哪一个类，工厂模式使一个类的实例化延迟到了子类。（2）单例模式，保证一个类只有一个实例，并提供一个访问它的全局访问点。(3)适配器模式，将一类的接口转换为客户希望的另一个接口，Adapter 模式使得原本由于接口不兼容而不能在一起工作的类可以一起工作。（4）装饰器模式，动态给一个对象增加一些额外的职责，比生成子类更加灵活。（5）代理模式，为其他对象提供一种代理，控制对这个对象的访问。（6）迭代器模式，提供一个人方法顺序访问一个聚合对象的各个元素，不需要暴露该对象的内部表示。

单例模式尽量使用懒加载，双重检测实现线程安全，构造方法为 private，定义静态的 Singleton instance 对象和 getInstance()方法。

匿名内部类是没有访问修饰符的，当所在方法的形参需要被匿名内部类使用，这个形参必须为 final，匿名内部类是没有构造方法的，因为其没有名字。

内存模型就是为了在现代计算机平台中保证程序可以正确的执行，但是不同的平台实现时不同的。编译器中生成的指令顺序可以与源代码中的顺序不同。编译器可能把变量保存在寄存器而不是内存中。处理器可以采用乱序或并行等方式来执行指令。缓存可能会改变将写入变量提交到主内存的顺序。保存在处理器本地缓存中的值，对其他处理器是不可见的。

索引分类：（1）直接创建索引和间接创建索引（2）普通索引和唯一性索引（3）单个索引和复合索引（4）聚簇索引和非聚簇索引。索引失效：如果条件中有 or，即使其中有条件带索引也不会使用。如果列类型是字符串，则一定要在条件中使用引号，否则不会使用索引。对于多列索引，如果不是使用的第一部分，则不会使用索引。

数据库中的范式：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、巴斯-科德范式（BCNF）、第四范式（4NF）、第五范式（5NF）一个数据库如果符合第二范式，则符合第一范式，如果符合第三范式，则符合第二范式。符合第一范式的关系中的每个属性都不可再分。符合第二范式的属性完全依赖于主键[消除部分子函数依赖]。符合第三范式的属性不依赖于其他非主属性[消除传递依赖]。BCNF 是在 1NF 的基础上，任何非主属性不能对主键子集依赖[在 3NF 基础上消除对主码子集的依赖]。4NF 要求把同一表内的多对多关系删除。5NF 从最终结构重新建立原始结构。

数据库中的索引结构，在使用二叉树时由于二叉树的深度过大造成 IO 读写过于频繁，导致效率低下，采用多叉树结构。B 树的各种操作能使 B 树保持较低的高度。B 树又叫平衡多路查找树，一棵 m 阶的 B 树的特性，树中每个节点最多含有 m 个孩子（m>=2），除了根节点和叶子节点外，其他每个节点至少有[ceil(m/2)]个孩子，根节点至少有两个孩子，除非 B 树只包含一个节点，即根节点。所有叶子节点都出现在同一层，叶子节点不包含任何关键字信息，每个非终端节点中包含 n 个关键字信息，且关键字按顺序升序排序。

MyISAM 和 InnoDB 引擎的区别：（1）MyISAM 是非事务安全型的，而 InnoDB 是事务安全型的。（2）MyISAM 锁的粒度是表级，而 InnoDB 锁的粒度支持行级。（3）MyISAM 支持全文类型索引，而 InnoDB 不支持全文索引。（4）MyISAM 相对简单，在效率上优于 InnoDB。（5）MyISAM 表是保存成文件的形式，在跨平台的数据转移中使用 MyISAM 存储会省去不少麻烦。（6）InnoDB 表比 MyISAM 表更安全，可以在保证数据不丢失的情况下，切换非事务表到事务表。应用场景：（1）MyISAM 管理非事务表，提供高速存储和检索以及全文搜索能力。如果需要执行大量的 select 查询，使用 MyISAM 更好。（2）InnoDB 用于事务处理应用程序，具有众多特性，包括 ACID 事务支持。如果应用中需要执行大量的 insert 或 update 操作，应该使用 InnoDB，提高多用户并发操作的性能。

JVM 垃圾处理方法：（1）标记-清除算法，标记阶段先通过根节点，标记所有从根节点开始的对象，未被标记的为垃圾对象。清除阶段清除所有未被标记的对象。（2）复制算法，将原有的内存空间分为两块，每次只使用其中一块，在垃圾回收时，将正在使用的内存中的存活对象复制到未使用的内存块中，然后清除正在使用的内存块中的所有对象。（3）标记-整理算法，标记阶段先通过根节点，标记所有从根节点可达的对象，没有被标记的为垃圾对象，整理阶段将所有的存活对象压缩到内存中的一段，之后清理边界所有的空间。

三种算法比较：（1）效率，复制算法>标记-整理算法>标记-清除算法。（2）内存整齐度，复制算法=标记-整理算法>标记-清除算法。（3）内存利用率，标记-整理算法=标记-清除算法>复制算法。

JVM 垃圾回收：（1）新生代，在方法中去 new 一个对象，方法调用完毕之后，对象就会被回收。（2）老年代，在新生代中经历了 N 次垃圾回收后仍然存活的对象会被放在老年代中，而大的对象直接放在老年代中。当 survivor 空间不够用时，需要依赖老年代进行分配

（3）持久代，即方法区。

GC 用的引用可达性分析算法中，可以被作为 GC Roots 对象的有：（1）Java 虚拟机栈中的对象。（2）方法区中的静态成员（3）方法区中的常量引用对象（4）本地方法区的 JNI 引用对象。

MinGC，新生代中的垃圾收集，采用复制算法。对于较大的对象，在 Minor GC 时可以直接进入老年代。Full GC 是发生在老年代中的垃圾收集动作，采用的是标记-清除或者标记-整理算法。老年代的对象几乎都是在 survivor，Full GC 发生的次数不会有 Minor GC 频繁。Time(Full GC)>Time(Minor GC)。

垃圾收集器：（1）Serial 收集器，是一个单线程的收集器，不是只能使用一个 CPU。在进行垃圾收集时，必须暂停其他所有的工作线程，直到收集结束。新生代采用复制算法，Stop the world。老年代采用标记-整理算法，Stop the world。简单高效，客户端模式下默认的新生代收集器。（2）ParNew 收集器，是 Serial 收集器的多线程版本，新生代采用复制算法，Stop the world。老年代采用标记-整理算法。是服务器模式下的首先新生代收集器，除了 Serial 收集器，只有 ParNew 收集器能与 CMS 收集器配合工作。（3）ParNewScanvenge 收集器，类似 ParNew，更加关注吞吐量，达到一个可控制吞吐量的收集器。停顿时间和吞吐量不可能同时调优，在 GC 时，垃圾回收的工作总量是不变的，如果将停顿时间减少，则频率变高，就会频繁垃圾回收，吞吐量就会减少，性能就会降低。（4）G1 收集器，对垃圾回收进行了划分优先级的操作，保证了效率。最大的优点是结合了空间整合，不会产生大量的碎片，降低了进行 GC 的频率。让使用者明确指定停顿时间。（5）CMS 收集器（Concurrent Mark Sweep，并发标记清除老年代收集器），一种以获得最短回收停顿时间为目标的收集器，适用于互联网或者 B/S 系统的服务器。初始标记（Stop The World）：根可以直接关联到的对象。并发标记（和用户线程一起），主要标记过程，标记全部对象。重新标记（Stop The World），并发标记时用户线程依然执行，在正式清理前再做修正。并发清除（和用户线程一起），基于标记结果直接清理对象。并发收集，低停顿。

JVM 内存划分：（1）程序计数器，线程私有，线程创建时创建，执行本地方法时其值为 undefined。（2）虚拟机栈，线程私有，栈内存为虚拟机执行 Java 方法服务，方法被调用时创建栈帧---局部变量表---局部变量、对象引用。如果线程请求的栈深度超出了虚拟机所允许的深度，则 StackOverFlowError。-Xss 规定了栈的最大空间。虚拟机栈可以动态扩展，如果扩展到无法申请到足够的内存，会 OOM。（3）本地方法栈，线程私有，本地方法栈为虚拟机执行使用到的 native 方法服务。JVM 没有对本地方法栈的使用和数据结构做强制规定。会抛出 StackOverFlowError 和 OutOfMemoryError。（4）Java 堆，被所有线程共享，在 JVM 创建时创建，几乎所有的对象都存储在堆中。是 GC 的主要管理区域。物理上不连续，逻辑上连续，并且可以动态扩展，无法扩展时抛出 OutOfMemoryError。（5）方法区，用于存储已经被虚拟机加载的类信息，常量，静态变量，即时编译器编译后的代码数据，被称为永久区。

（6）运行时常量池，受到方法区的限制，抛出 OutOfMemoryError。

双亲委派：如果一个类加载器收到了类加载的请求，首先不会自己去尝试加载这个类，而是把这个请求委派给父类加载器去完成，每一层的加载器都是如此。因此所有的类加载器请求都会传给顶层的启动类加载器，只有当父加载器反馈自己无法完成该加载请求，子加载器才会尝试自己去加载。

加载器：（1）启动类加载器，用本地代码实现的类装入器，负载将<Java_Runtime_Home>/lib 下面的类库加载到内存中。（2）标准扩展类加载器，负责将<Java_Runtime_Home>/lib/ext 或者由系统变量指定位置中的类库加载到内存中。（3）系统类加载器，负责将系统类路径中指定的类库加载到内存中。（4）线程上下文类加载器。如果加载同一个类，该使用父类的加载器。

Cookie 和 Session 的区别：（1）Session 在服务器端，Cookie 在客户端。（2）Session 的运行依赖 SessionID，SessionID 存储在 Cookie 中，如果浏览器禁用了 Cookie，Session 也会失效，可以在 URL 中传递 SessionID。（2）Session 可以放在文件、数据库、内存中。（3）用户验证一般使用 Session。（4）Session 会在一定时间内保存在服务器上，访问增多比较占用服务器性能，使用 Cookie。（5）单个 Cookie 保存的数据不能超过 4K，最多保存 20 个 Cookie。

IOC 叫控制反转，是 Inversion of Control 的缩写。DI（Dependency Injection）叫依赖注入。控制饭庄是把传统上由程序代码直接操作的对象的调用权给容器，通过容器来实现对象组件的装配和管理。组件之间的依赖关系由容器在运行期决定，由容器动态地将某种依赖关系注入到组件之中。通过反射创建实例，获取需要注入的接口实现类并将其赋值给该接口。

AOP 以一种称为切面的语言构造为基础，用来描述分散在对象、类或方法中的横切关注点。横切关注会影响到整个应用程序的关注功能，跟正常的业务逻辑正交，没有必然的联系。几乎所有的业务逻辑都会涉及到关注功能，事务、日志、安全性等就是横切关注点功能。通过动态代理实现，JDK 的动态代理和 CGLib 的动态代理。

IOC 容器的加载过程：（1）创建 IOC 配置文件的抽象资源（2）创建一个 BeanFactory（3）把读取配置信息的 BeanDefinitionReader 配置给 BeanFactory（4）从定义好的资源位置读取配置信息，具体的解析过程由 XmlBeanDefinitionReader 完成。

JDK 动态代理只能对实现了接口的类生成代理，而不能针对类。CGLib 是针对类实现代理，主要是对指定的类生成一个子类，覆盖其中的方法，由于继承关系，类或方法最好不要声明为 final。JDK 代理不需要依赖第三方库，只需要 JDK 环境就行。CGLib 必须依赖 CGLib 的类库，它需要类来实现任何接口代理的是指定的类生成一个子类，覆盖方法，是一种继承关系。

SpringMVC 工作原理：（1）客户端的所有请求都交给前端控制器 DispatcherServlet 处理，负责调用系统的其它模块，真正处理用户的请求。（2）DispatcherServlet 收到请求后，将根据请求的信息（URL、HTTP 协议方法、请求头、请求参数、Cookie 等）以及 HandlerMapping

的配置找到处理该请求的 Handler（任何一个对象都可以作为请求的 Handler）（3）Spring 会通过 HandlerAdapter 对处理进行封装。（4）HandlerAdapter 是一个适配器，用统一的接口对各种 Handler 中的方法进行调用。（5）Handler 完成对用户请求的处理后，会返回一个 ModelAndView 对象给 DispatcherServlet，包含了数据模型以及相应视图信息。（6）ModelAndView 的视图是逻辑视图，DispatcherServlet 使用 ViewResolver 完成从逻辑视图到真正视图对象的解析工作。（7）当得到真正的视图对象后，DispatcherServlet 会利用视图对象对模型数据进行渲染。（8）客户端得到响应可能是一个 HTML 或者 JSON 或者 XML，或者其他格式信息。

Hibernate 优化策略：（1）制定合理的缓存策略，二级缓存、查询缓存。（2）采用合理的 Session 管理机制。（3）尽量使用延迟加载特性。（4）设定合理的批处理参数。（5）选用 UUID 作为主键生成器。（6）选用基于版本号的乐观锁替代悲观锁。（7）开发过程中，开启 hibernate.show_sql 选项查看生成的 SQL，了解底层的状况，开发完成后关闭选项。（8）合理的索引，恰当的数据分区策略会对持久层的性能产生好的作用。

死锁产生的必要条件：（1）互斥条件，某个资源在一段时间内只能被一个进程占用，不能同时被两个或者两个以上的进程占用。必须在占用该资源的进程主动释放之后，其他进程才能占用。（2）不可抢占条件，进程所获得的资源在未使用完毕之前，资源申请者不能强行的从资源占用者抢夺资源，只能由该资源的占用者自己释放。（3）占用且申请条件，进程至少已经占用一个资源，但又申请新的资源，由于该资源已经被其他线程所占用，所以进程阻塞。但是其在等待新资源时，仍然继续占用已经占用的资源。（4）循环等待条件，存在一个进程等待序列{P1，P2，……Pn}，其中形成了一个进程等待环。

死锁预防：（1）打破互斥条件，允许进程同时访问某些资源。（2）打破不可抢占条件，即允许进程强行从占有者那里剥夺资源。（3）打破占有且申请条件，可是实行资源预先分配策略，进程在运行前一次性的向系统申请所需要的所有资源。如果某个进程所需要的资源得不到满足，则不分配任何资源，进程暂时不运行。只有当系统能够满足当前进程需要的全部资源时，才一次性的将所申请的资源全部分配给该进程。由于运行的进程已经占有了所需要的全部资源，所以不会发生占有且申请，即不会发生死锁。大部分情况下，一个进程在运行前不知道所需要的全部资源，资源利用率降低，并且也降低了进程的并发性。（4）打破循环等待条件，实行资源有序分配策略。所有进程对资源的请求都应该按照资源序号递增的顺序提出。但是限制了进程对资源的请求，同时给系统的进程编号比较困难，增加系统开销。增加了进程对资源的占用时间。

分布式锁是控制分布式系统之间同步访问共享资源的一种方式，如果不同的系统或者同一系统的不同主机之间共享了一个或者一组资源，使用分布式锁来保证一致性。

进程通信：（1）无名管道，用于具有亲缘关系进程间的通信，允许一个进程和另一个与他有共同祖先的进程之间进行通信。（2）有名管道，除了无名管道的功能，还允许无亲缘关

系的进程间通信。通过命令 mkfifo 实现。（3）信号，用于通知接收进程有某种时间发生。（4）消息队列，有足够权限的进程可以向队列中添加消息，有读权限的进程读取队列中的消息。消息队列克服了信号量承载信息少，管道只能承载无格式字节流以及缓冲区大小受限等。（5）共享内存，多个进程可以访问同一块内存空间，是最快的可用的 IPC 形式。是针对其他通信机制运行效率低设置的。可以与其他通信机制结合达到进程间的同步和互斥。（6）内存映射允许任何多个进程间通信，每一个使用该机制的进程通过把一个文件映射到自己的地址空间来实现。（7）信号量，主要作为进程间以及同一进程不同线程之间的同步手段。（8）Socket，可用于不同机器之间的进程通信。

同步和异步关注的是消息通信机制，阻塞和非阻塞关注的是程序在等待调用结果时的状态。阻塞调用，是指调用结果返回之前，当前线程会被挂起。调用线程只有在得到结果之后才返回。非阻塞调用是指不能立即得到结果之前，该调用不会阻塞当前进程。

HTTP 和 HTTPS 的区别：（1）HTTP 是 HTTP 协议运行在 TCP 上，所有传输的内容都是明文，客户端和服务器都无法验证对方的身份。（2）HTTPS 是 HTTP 运行在 SSL/TLS 之上，SSL/TLS 运行在 TCP 之上。所有传输的内容都经过加密，加密采用对称加密，对称加密的秘钥用服务器的证书进行了非对称加密。客户端可以验证服务器端的身份，如果配置了客户端验证，服务器也可以验证客户端的身份。（3）HTTPS 协议需要到 CA 申请证书。（4）HTTP 是超文本传输协议，是明文传输，HTTPS 是具有安全性的 SSL 加密传输协议。（5）HTTP 和 HTTPS 使用不同的连接方法，HTTP 使用 80 端口，HTTPS 使用 443 端口。（6）HTTP 的连接简单，无状态。（7）HTTPS 协议是 SSL+HTTP 协议构建的可进行加密传输、身份验证的网络协议，比 HTTP 更加安全。