奇安信 （9/10）

1、为什么选择测试开发岗，为什么?

> - **更早更频繁的去思考价值问题**，包括艰难的人生： 就Bug在整个开发活动中**造成的损失来看，在分析、设计处是最小的**，而在测试时是最大的，依赖于具体使用的开发工程模型。每个环节引入的Bug都会在下一环节被放大，导致修正成本迅速扩大，到了测试这里再发现和修正Bug，返工成本相当高，拦截和修正的责任重大。
> - **新鲜感**： 新功能的探索和发现，是我个人一直爱接新功能胜过做回归的主要原因。**新工具新技术的发现和学习是个有趣的过程。测试其实是个目的驱动的事情**，基于这一点，没人会要求你从头造轮子，能拿来用的现成都得学会捡，不然什么都要从main写起，黄花菜都凉了。囤新奇工具、学新鲜技术，都是有趣的事情。
> - **成就感**： **在参与设计讨论时指出可能存在的设计缺陷，在功能开发之前提供建议避免功能误读和错误风险评估**，一个描述清晰、根源挖掘准确充分的defect送到dev处被干净利落地斩草除根，当support team来征询产品功能的相关问题时，当用户来寻求解决方案时，是不是都有一种叫成就感的东东在心里撒了欢地奔走呢。

2、给了一个表，有学生和班级字段，写出统计每个班级的总人数并按人数的倒叙输出的SQL语句？

3、了解测试嘛，说说测试的流程，什么是性能测试，怎么测试一个网站的稳定性？

测试的流程：

- 测试**需求分析**阶段：阅读需求，理解需求，主要就是对业务的学习，**分析需求点。**
- **测试计划**阶段：主要任务是编写测试计划，参考软件需求规格说明书、项目总体计划，内容包括测试范围(来自需求文档)、**进度的安排，人力物力的分配，整体测试策略的制定**，和风险的评估与规避措施有一个制定，一般有测试负责人编写，当然我们也会参与相关的评审工作。
- **测试设计**阶段：主要任务是**编写测试用例**，会参考需求文档(原型图)、概要设计、详细设计等文档，有不明确的也会及时和开发、产品经理沟通。用例编写完成后会进行评审。
- **测试执行**阶段：首先**搭建测试环境**，执行预测(冒烟)，以判定当前版本可测与否，如果预测通过，正式进入系统测试，遇到问题提交Bug到缺陷管理平台，并对bug进行跟踪，直到被测软件达到测试需求要求，没有重大bug，测试结束。
- **测试评估**阶段：出测试报告，对整个测试的过程和版本质量做一个详细的评估。确认是否可以上线。

性能测试是通过自动化的测试工具**模拟多种正常、峰值以及异常负载条件**来对系统的各项性能指标进行测试。

> 1 稳定性测试就测试系统的长期稳定运行能力。在系统运行过程中，**对系统施压，观察系统的各种性能指标，以及服务器的指标。**
>
> 2 测试场景：模拟平常的压力，模拟实际中日常的用户数进行操作。数据库要存有一定的数据。
>
> 3 稳定性测试是概率性的测试，就是说即使稳定性测试通过，也不能保证系统实际运行的时候不出问题。所以要尽可能的提高测试的可靠性。可以通过多次测试，延长测试时间，增大测试压力来提高测试的可靠性。
>
> 4 稳定性测试的测试时间和压力存在一定的关系。在测试时间不能保证的情况下，可以通过增强压力在一定程度上来挽救。
>
> 观察系统的各种监控指标曲线，预测系统的发展状况。响应时间是否有增长，可用内存是否在减少，CPU利用率是否在上升等等都可以说明系统是否存在问题 。

4、了解网络安全嘛？

网络安全是指**网络系统的硬件、软件及其系统中的数据受到保护**，不因偶然的或者恶意的原因而遭受到破坏、更改、泄露，系统连续可靠正常地运行，**网络**服务不中断。

保护网络安全：

- 制定网络安全的管理措施。
- 使用防火墙
- 建立可靠的识别和鉴别机制。
- 加密技术：对称、非对称
- 认证技术：数字签名、数字证书
- 安全套接字协议SSL

5、子网掩码的最大长度？

30。首先要理解掩码的作用，子网掩码**将某个IP地址划分成网络地址和主机地址**两部分，**在一个网段中，有2个地址是被固定占用的，一个是网段地址，一个是网段内广播地址**，其他是主机可用的地址，至少一个，不然就没有意义了。也就是说，被掩码所分的网段至少要包含3个地址。掩码越长，可用的地址越少。

6、重载和重写：

- 重写指的是**在Java的子类与父类中有两个名称、参数列表都相同的方法的情况。**由于他们具有相同的方法签名，所以**子类中的新方法将覆盖父类中原有的方法。**
- 重载，就是**方法有同样的名称，但是参数列表不相同的情形**，这样的同名不同参数的方法之间，互相称之为重载方法。 应该注意的是，返回值不同，其它都相同不算是重载。

区别：位置不一样 ，一个是说父子类，一个是当前类中；方法参数变化，一个不变化，一个是参数个数可以不一样。

> 1、重载是一个**编译期**概念、重写是一个**运行期间**概念。
>
> 2、重载遵循所谓"编译期绑定"，即在编译时根据参数变量的类型判断应该调用哪个方法。
>
> 3、重写遵循所谓"运行期绑定"，即在运行的时候，根据引用变量所指向的实际对象的类型来调用方法。
>
> 4、因为在编译期已经确定调用哪个方法，所以重载并不是多态。而重写是多态。重载只是一种语言特性，是一种语法规则，与多态无关，与面向对象也无关。（注：严格来说，**重载是编译时多态**，即静态多态。但是，Java中提到的多态，在不特别说明的情况下都指动态多态）

7、Java中接口与抽象类区别?

- 抽象类是用来**捕捉子类的通用特性**的。它不能被实例化，只能被用作子类的超类。抽象类是被用来创建继承层级里子类的模板。
- **接口是抽象方法的集合。**如果一个类实现了某个接口，那么它就继承了这个接口的抽象方法。

区别：

> 从语法层面来说：
>
> - 抽象类可以提供成员方法的实现细节，而接口中只能存在抽象方法
> - 抽象类中成员变量可以是多种类型，接口中成员变量必须用public，static，final修饰
> - 一个类只能继承一个抽象类，但可以实现多个接口
> - 抽象类中允许含有静态代码块和静态方法，接口不能
>
> 从设计层面而言：

> - 抽象类是对整一个类的属性，行为等方面进行抽象，而接口则是对行为抽象。就好比飞机和鸟，抽象类抽象出的是飞行物类。而接口则是抽闲出飞行方法。
> - **抽象类是一个模板式**的设计，当在开发过程中出现需求更改的情况，只需要更改抽象类而不需要更改它的子类。**接口是一种辐射性**设计，当接口的内容发生改变时，需要同时对实现它的子类进行相应的修改。
> - 抽象类可以类比为模板，而接口可以类比为协议。

8、多线程sleep和wait区别?

- **wait() 是 Object 的方法，而 sleep() 是 Thread 的静态方法**；
- **sleep()方法导致了程序暂停执行指定的时间，让出 cpu 给其他线程**，但是他的监控状态依然 保持者，当指定的时间到了又会自动恢复运行状态；
- **wait() 会释放锁，sleep() 不会**；
- 当调用 wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用 notify() 方法后本线程才进入对象锁定池准备获取对象锁进入运行状态。

9、线程同步问题如何解决?

为了解决并发编程中存在的**原子性、可见性和有序性**问题，提供了一系列和并发处理相关的关键字，比如`synchronized`、`volatile`、`final`、`concurren包`等。

synchronized：

- 被`synchronized`修饰的代码块及方法，**在同一时间，只能被单个线程访问。**
- `synchronized`，是Java中用于**解决并发情况下数据同步访问**的一个很重要的关键字。当我们想要**保证一个共享资源在同一时间只会被一个线程访问到时，我们可以在代码中使用`synchronized`关键字对类或者对象加锁。**

> synchronized与原子性：原子性是指**一个操作是不可中断的，要全部执行完成，要不就都不执行。** 线程是CPU调度的基本单位。CPU有时间片的概念，会根据不同的调度算法进行线程调度。当一个线程获得时间片之后开始执行，在时间片耗尽之后，就会失去CPU使用权。所以在多线程场景下，由于时间片在线程间轮换，就会发生原子性问题。
>
> 被`synchronized`修饰的代码在同一时间只能被一个线程访问，在锁未释放之前，无法被其他线程访问到 。
>
> synchronized与可见性：可见性是指**当多个线程访问同一个变量时，一个线程修改了这个变量的值，其他线程能够立即看得到修改的值。**Java内存模型规定了所有的变量都存储在主内存中，每条线程还有自己的工作内存，线程的工作内存中保存了该线程中是用到的变量的主内存副本拷贝，线程对变量的所有操作都必须在工作内存中进行，而不能直接读写主内存。不同的线程之间也无法直接访问对方工作内存中的变量，线程间变量的传递均需要自己的工作内存和主存之间进行数据同步进行。所以，就可能出现线程1改了某个变量的值，但是线程2不可见的情况。
>
> 被`synchronized`修饰的代码，在开始执行时会加锁，执行完成后会进行解锁。而为了保证可见性，有一条规则是这样的：**对一个变量解锁之前，必须先把此变量同步回主存中。**这样解锁后，后续线程就可以访问到被修改后的值。
>
> synchronized与有序性：有序性即**程序执行的顺序按照代码的先后顺序执行。** 除了引入了时间片以外，由于处理器优化和指令重排等，CPU还可能对输入代码进行乱序执行，比如load->add->save 有可能被优化成load->save->add 。这就是可能存在有序性问题。
>
> `synchronized`是无法禁止指令重排和处理器优化的。也就是说，`synchronized`无法避免上述提到的问题。Java程序中天然的有序性可以总结为一句话：如果在本线程内观察，所有操作都是天然有序的。如果在一个线程中观察另一个线程，所有操作都是无序的。 由于`synchronized`修饰的代码，同一时间只能被同一线程访问。那么也就是单线程执行的。所以，可以保证其有序性。

volatile：`volatile`通常被比喻成"轻量级的`synchronized`"，也是Java并发编程中比较重要的一个关键字。和`synchronized`不同，`volatile`是一个变量修饰符，**只能用来修饰变量。无法修饰方法及代码块等。** `volatile`的用法比较简单，只需要**在声明一个可能被多线程同时访问的变量时，使用`volatile`修饰**就可以了。

> volatile与可见性：当多个线程访问同一个变量时，一个线程修改了这个变量的值，其他线程能够立即看得到修改的值。
>
> Java中的`volatile`关键字提供了一个功能，那就是**被其修饰的变量在被修改后可以立即同步到主内存，被其修饰的变量在每次是用之前都从主内存刷新。**因此，可以使用`volatile`来保证多线程操作时变量的可见性。
>
> volatile与有序性：有序性即程序执行的顺序按照代码的先后顺序执行。
>
> 而`volatile`除了可以保证数据的可见性之外，还有一个强大的功能，那就是他可以**禁止指令重排优化**等。通过禁止指令重排优化，就可以保证代码程序会严格按照代码的先后顺序执行。 volatile是通过**内存屏障**来来禁止指令重排的。
>
> **内存屏障（Memory Barrier）**是一类同步屏障指令，是CPU或编译器在对内存随机访问的操作中的一个同步点，使得此点之前的所有读写操作都执行后才可以开始执行此点之后的操作。
>
> volatile与原子性：原子性是指一个操作是不可中断的，要全部执行完成，要不就都不执行。
>
> **volatile是不能保证原子性的** 。

比较：

- synchronized是一种锁机制，存在阻塞问题和性能问题，而volatile并不是锁，所以不存在阻塞和性能问题。
- volatile借助了内存屏障来帮助其解决可见性和有序性问题，而内存屏障的使用还为其带来了一个禁止指令重排的附件功能，所以在有些场景中是可以避免发生指令重排的问题的。

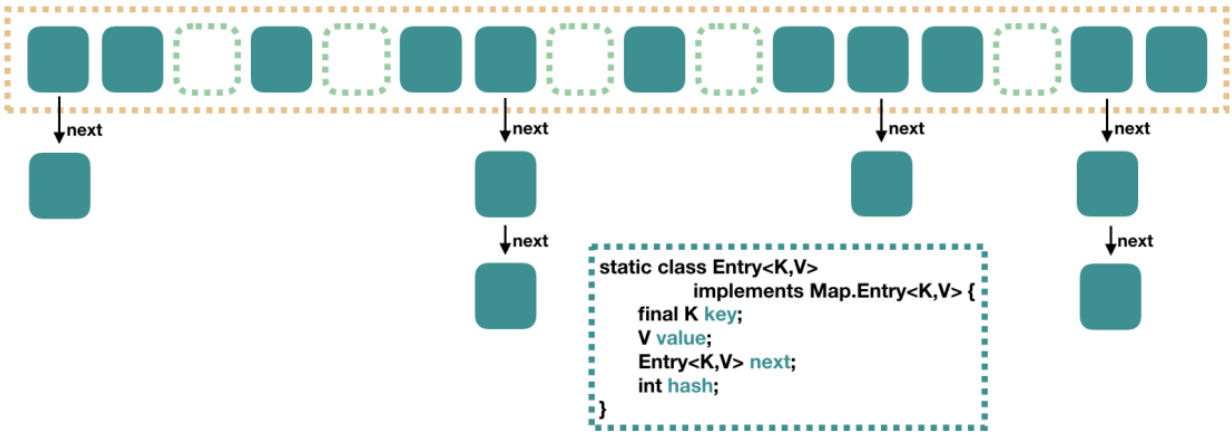
10、Hashmap、HashTable、ConCurrentHashMap区别?

HashMap：**允许键、值为null；基于数组+链表实现；非线程安全。**在HashMap**进行扩容重哈希时导致Entry链形成环。一旦Entry链中有环，会导致在同一个桶中进行插入、查询、删除等操作时陷入死循环。**

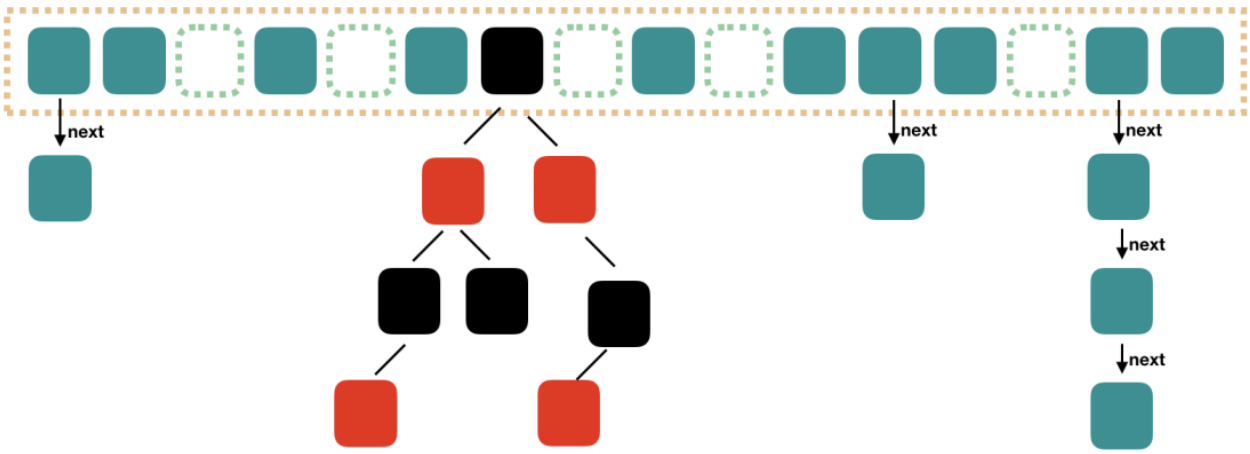
如何实现线程安全：使用 java.util.Collections.synchronizedMap() 方法包装 HashMap object，得到线程安全的Map，并在此Map上进行操作。

- 内部存储结构：数组+链表+红黑树（JDK8）；
- 默认容量16，默认装载因子0.75；
- key和value对数据类型的要求都是泛型；
- key可以为null，放在table[0]中；
- hashcode：计算键的hashcode作为存储键信息的数组下标用于查找键对象的存储位置。equals：HashMap使用equals()判断当前的键是否与表中存在的键相同。

Java7 HashMap 结构



Java8 HashMap 结构



> - put：将指定的键、值对添加到map里，根据key值计算出hashcode，根据hashcode定位出所在桶。如果桶是一个链表则需要遍历判断里面的hashcode、key值是否和传入key相等，如果相等则进行覆盖，否则插入（头插法）新的Entry。如果桶是空的，说明当前位置没有数据存入，新增一个Entry对象写入当前位置。
> - get：根据指定的key值返回对应的value，根据key计算出hashcode，定位到桶的下标，如果桶是一个链表，依次遍历冲突链表，通过key.equals（k）方法来判断是否是要找的那个entry。如果桶不是链表，根据key是否相等返回值。

注：当hash冲突严重时，**在桶上形成的链表会变的越来越长，这样在查询时的效率就会越来越低。JDK1.8的优化：数组+链表/红黑树。判断当前链表的大小是否大于预设的阈值，大于时就要转换为红黑树。**

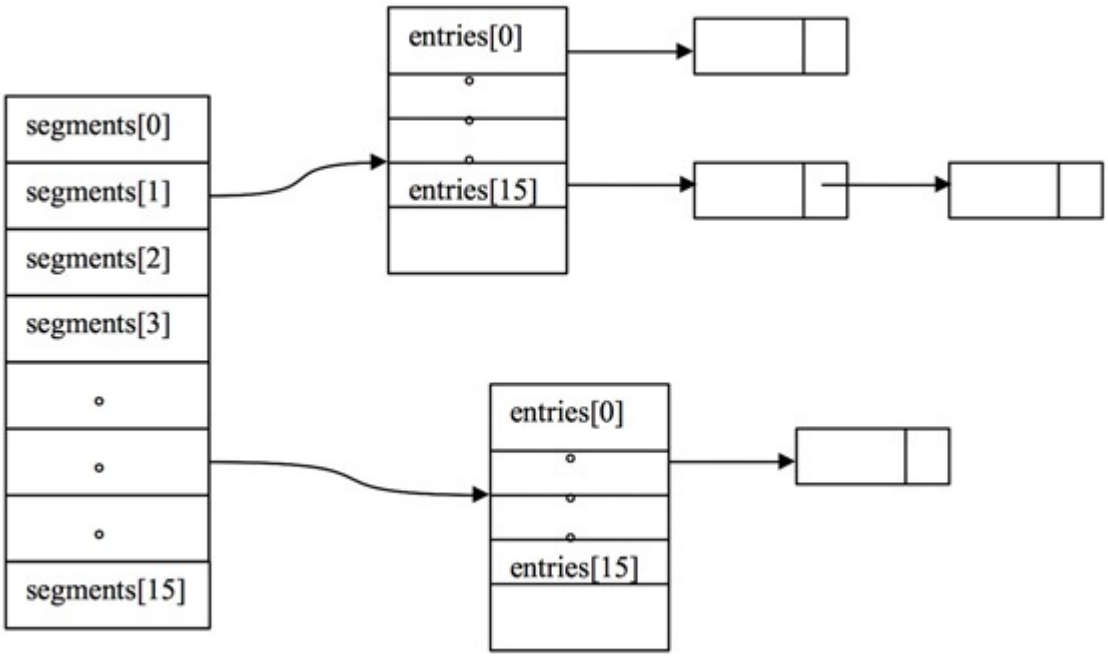
HashTable：**HashTable是线程安全；键、值均不可为null**，其余与HashMap大致相同。

HashTable容器使用Synchronized来保证线程安全，但在线程竞争激烈的情况下，HashTable的效率非常低下。因为当一个线程访问HashTable的同步方法时，其他线程访问HashTable的同步方法时，**可能会进入阻塞或轮询状态。**

ConCurrentHashMap：**不允许键、值为null；使用锁分段技术**（容器有多把锁，每一把锁用于锁容器其中一部分数据）。由segment数组结构和HashEntry数组结构组成。**segment是一种可重入锁Reentantlock，在concurrentHashMap里扮演锁的角色。HashEntry则用于存储键值对数据。**

为了能通过按位与的哈希算法来定位 segments 数组的索引，必须保证 segments 数组的长度是 2 的 N 次方 。

一个ConcurrentHashMap里包含一个**segment数组，segment的结构和HashMap类似，是一种数组和链表结构**，一个segment里包含一个HashEntry数组，每个HashEntry是一个链表结构的元素。



> 定位segment：
>
> concurrentHashMap使用分段锁segment来保护不同段的数据，那么**在插入和获取元素的时候，必须先通过哈希算法定位到segment。**
>
> get：
>
> - 先经过一次再哈希，然后使用这个哈希值通过哈希运算定位到segment，再通过哈希算法定位到元素。
> - get操作的高效之处在于**整个get过程不需要加锁，除非读到的值是空的才会加锁重读。**它的get方法里**将要使用的共享变量都定义成Volatile**，如用于统计当前segment大小的count字段和用于存储值的HashEntry的value。**定义成Volatile的变量，能够在线程之间保持可见性，能够被多线程同时读，并且保证不会读到过期的值。**但是只能被单线程写（有一种情况可以被多线程写，就是写入的值不依赖于原值），之所以不会读到过期的值，是根据**java内存模型的happen-before原则，对volatile字段的写入操作先于读操作**，即使两个线程同时修改和获取volatile变量，get操作也能拿到最新的值。（将key通过Hash之后定位到具体的segment，再通过一次Hash定位到具体的元素上）。
>
> put：
>
> 需要对共享变量进行写入操作，所以**为了线程安全，在操作共享变量时必须得加锁。put方法首先定位到segment，然后再segment里进行插入操作。**
>
> - 判断是否需要对segment里的HashEntry数组进行扩容；
> - 定位添加元素的位置，然后放在HashEntry数组里。
>
> segment的扩容判断比HashMap更恰当：**HashMap是在插入元素后判断元素是否已经到达容量的，如果到达了就进行扩容，但是很有可能扩容之后就没有新元素插入，这时HashMap就进行了一次无效的扩容。**ConcurrentHashMap**扩容的时候首先会创建一个两倍于原容量的数组，然后将原数组里的元素进行再hash后插入到新的数组里，不会对整个容器进行扩容，而只对某个segment进行扩容。**

> put流程：将当前segment中的table通过key的hashcode定位到HashEntry，遍历该HashEntry，如果不为空则判断传入的key和当前遍历的key是否相等，相等则覆盖旧的value。为空则需新建一个HashEntry并加入到segment中，同时会先判断是否需要扩容，最后解除当前segment锁。
>
> 查询遍历链表效率太低，**JDK1.8抛弃了原有的segment分段锁，采用了CAS+Synchronized保证并发安全性。**新版的JDK中对Synchronized优化是很到位的。

总结：读操作（几乎）不需要加锁，而**在写操作时通过锁分段技术只对所操作的段加锁而不影响客户端对其它段的访问。**ConcurrentHashMap本质上是一个segment数组，而一个segment实例又包含若干个桶，每个桶中都包含一条由若干个HashEntry对象链接起来的链表，ConcurrentHashMap的高效并发机制是通过以下三方面来保证的；

- **通过锁分段技术保证并发环境下的写操作；**
- **通过 HashEntry的不变性、Volatile变量的内存可见性和加锁重读机制保证高效、安全的读操作；**
- **通过不加锁和加锁两种方案控制跨段操作的的安全性。**

11、防火墙：通过有机**结合各类用于安全管理与筛选的软件和硬件设备，帮助计算机网络于其内、外网之间构建一道相对隔绝的保护屏障，以保护用户资料与信息安全性的一种技术。**

防火墙技术的功能主要在于**及时发现并处理计算机网络运行时可能存在的安全风险、数据传输等问题**，其中处理措施包括隔离与保护，同时可对计算机网络安全当中的各项操作实施记录与检测，以确保计算机网络运行的安全性，保障用户资料与信息的完整性，为用户提供更好、更安全的计算机网络使用体验。

12、判断两台机器的可达性：Java 类库中 java.net.InetAddress 类来实现 。枚举本地的每个网络地址，建立本地 Socket，在某个端口上尝试连接远程地址，如果可以连接上，则说明该本地地址可达远程网络。

13、说一说**简单用户界面登录过程**都需要做哪些分析?

14、junit：开源的Java单元测试框架、设计思想是通过从零开始来应用设计模式，然后一个接一个，直至获得最终合适的系统架构。

- Template Method（模板方法）：需要复用的是算法的结构，也就是步骤，而步骤的实现可以在子类中完成。

  在Junit中的应用：@Before、@Test、@After
- Command（命令）模式：将一个请求封装成一个对象，从而使你可用不同的请求对客户进行参数化，可以为一个操作生成一个对象并给出它的一个execute（执行）方法。Command模式**使新的TestCase很容易的加入**，无需改变已有的类，只需继承TestCase类即可，这样方便了测试人员。
- Composite（组合）：把多个测试用例进行组合成为一个符合的测试用例，当作一个请求发送给Junit。
- Dapter（适配器）：将一个类的接口转换成客户希望的另外一个接口，使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。在Junit中的应用：testXXX（）。通过**runTest方法将我们自己编写的testXXX方法进行了适配**，使得junit可以执行我们自己编写的Test Case。

计网五层协议

1、应用层：通过应用进程间的交互完成特定网络应用.应用层定义的是**应用进程间通信和交互的规则。**

- 常用协议：HTTP、SMTP、FTP、ping、telnet、DNS、DHCP等

HTTP协议（超文本传输协议）

主要特点：

- **支持客户/服务器模式**
- **简单快速**：客户向服务器请求服务时，**只需传送请求方法和路径**；请求方法常用GET、HEAD、POST等，每种方法规定了客户与服务器联系的不同类型；HTTP协议简单，服务器程序规模小，通信速度较快
- **灵活**：HTTP**允许传输任意类型的数据对象**；正在传输的数据类型由Content-Type加以标记
- **无连接**：无连接是指**每次连接只处理一个请求**；服务器处理完客户请求，并收到客户应答后，即断开连接，节省传输时间
- **无状态**：**无状态是指协议对于事务处理没有记忆能力；应答较快，但传输数据量较大**

HTTP URL:定位网络资源

- http://host[:port][abs_path]

HTTP请求

- 三部分组成：请求行、消息报头、请求正文
- 格式：Method Request-URI HTTP-Version CRLF
- Method:请求方法，GET、POST等
- Request-URI:请求的HTTP协议版本
- CRLF:回车换行

HTTP响应

- 由三部分组成：状态行、消息报头、响应正文
- 状态行格式：HTTP-Version Status-Code Reason-Phrase CRLF
- HTTP-Version:服务器HTTP协议版本
- Status-Code:服务器返回的响应状态码

HTTP状态码

- 由三位数字组成，首数字定义响应类别

- 1xx:指示信息，表示请求已接收，继续处理；
- 2xx:成功
- 3xx:重定向，要完成请求必须进行更进一步的操作；
- 4xx:客户端错误，请求有语法错误或请求无法实现
- 5xx：服务器端错误：服务器未能实现合法的请求

常见状态代码

- 200：OK,请求成功；
- 302：跳转，重定向
- 400：Bad Request,客户端有语法错误
- 401：Unauthorized,请求未经授权；
- 403：Forbidden,服务器收到请求，但是拒绝提供服务；
- 404：Not Found,请求资源不存在；
- 500：Internet Server Error,服务器内部错误
- 503：Server Unavailable,服务器不能处理客户请求

2、传输层：负责向两个主机中进程之间的通信提供通用数据服务（**为两台主机的应用程序提供端到端通信**）

主要使用以下两种协议： **传输控制协议TCP** ：提供面向连接的 、可靠的、基于流的数据传输服务，数据传输的单位是**报文段。**使用超时重发、数据确认等方式确保数据被正确发送至目的地 **用户数据报协议UDP**：提供无连接的、不可靠的、基于数据报的数据传输服务；数据传输的单位是**用户数据报**

3、网络层：负责**为分组交换网上的不同主机提供通信服务。**在发送数据时，网络层把运输层产生的**报文段**和**用户数据报**封装成*分组*（**IP数据报**）或*包*进行传送。

- 实现**数据包的路由和转发**
- IP协议:逐跳发送模式；根据数据包的目的地IP地址决定数据如何发送；如果数据包不能直接发送至目的地，IP协议负责寻找一个合适的下一跳路由器，并将数据包交付给该路由器转发
- ICMP协议：因特网控制报文协议，用于检测网络连接

4、数据链路层：**负责建立和管理节点间的链路。**该层的主要功能是：**通过各种控制协议，将有差错的物理信道变为无差错的、能可靠传输数据帧的数据链路。**

- 网卡接口的网络驱动程序，处理数据在物理媒介上的传输；不同的物理网络具有电气特性，网络驱动程序隐藏实现细节，为上层协议提供一致接口
- 常用协议：地址解析协议（ARP）和反地址解析协议（RARP）,实现IP地址与机器物理地址（MAC地址）之间的转换

5、物理层：设备之间比特流的传输。**利用传输介质为数据链路层提供物理连接，实现比特流的透明传输。**实现相邻计算机节点之间比特流的透明传送，尽可能屏蔽掉具体传输介质和物理设备的差异。

TCP和UDP

区别：

> - **TCP提供面向对象的连接，通信前要建立三次握手机制的连接，UDP提供无连接的传输，传输前不用建立连接**
> - **TCP提供可靠的，有序的，不丢失的传输，UDP提供不可靠的传输**
> - **TCP提供面向字节流的传输，它能将信息分割成组，并在接收端将其充足，UDP提供面向数据报的传输，没有分组开销**
> - **TCP提供拥塞控制，流量控制机制，UDP没有**
> - **每一条TCP连接只能是点到点的；UDP支持一对一，一对多，多对一和多对多的交互通信**

TCP协议作为一个可靠的面向流的传输协议，其可靠性和流量控制由滑动窗口协议保证，而拥塞控制则由控制窗口结合一系列的控制算法实现。

TCP拥塞控制-慢启动、拥塞避免、快重传、快启动：

如果网络出现拥塞，分组将会丢失，此时发送方会继续重传，从而导致网络拥塞程度更高。因此当出现拥塞时，应当控制发送方的速率。这一点和流量控制很像，但是出发点不同。流量控制是为了让接收方能来得及接收，而**拥塞控制是为了降低整个网络的拥塞程度。**

> 慢开始与拥塞避免：
>
> 发送的最初执行慢开始，令 cwnd = 1，发送方只能发送 1 个报文段；当收到确认后，将 cwnd 加倍，因此之后发送方能够发送的报文段数量为：2、4、8 ...
>
> 注意到慢开始每个轮次都将 cwnd 加倍，这样会让 cwnd 增长速度非常快，从而使得发送方发送的速度增长速度过快，网络拥塞的可能性也就更高。设置一个慢开始门限 ssthresh，当 cwnd >= ssthresh 时，进入拥塞避免，每个轮次只将 cwnd 加 1。
>
> 如果出现了超时，则令 ssthresh = cwnd / 2，然后重新执行慢开始。
>
> 快重传与快恢复：
>
> 在接收方，要求每次接收到报文段都应该对最后一个已收到的有序报文段进行确认。例如已经接收到 M1 和 M2，此时收到 M4，应当发送对 M2 的确认。
>
> 在发送方，如果收到三个重复确认，那么可以知道下一个报文段丢失，此时执行快重传，立即重传下一个报文段。例如收到三个 M2，则 M3 丢失，立即重传 M3。
>
> 在这种情况下，只是丢失个别报文段，而不是网络拥塞。因此执行快恢复，令 ssthresh = cwnd / 2 ，cwnd = ssthresh，注意到此时直接进入拥塞避免。
>
> 慢开始和快恢复的快慢指的是 cwnd 的设定值，而不是 cwnd 的增长速率。慢开始 cwnd 设定为 1，而快恢复 cwnd 设定为 ssthresh。

http和tcp区别

TPC/IP协议是传输层协议，主要解决数据如何在网络中传输，

HTTP是应用层协议，主要解决如何包装数据。

关于TCP/IP和HTTP协议的关系，网络有一段比较容易理解的介绍：“我们在传输数据时，可以只使用（传输层）TCP/IP协议，但是那样的话，如果没有应用层，便无法识别数据内容，如果想要使传输的数据有意义，则必须使用到应用层协议，应用层协议有很多，比如HTTP、FTP、TELNET等，也可以自己定义应用层协议。WEB使用HTTP协议作应用层协议，以封装HTTP 文本信息，然后使用TCP/IP做传输层协议将它发到网络上。”

术语TCP/IP代表传输控制协议/网际协议，指的是一系列协议。“IP”代表网际协议，TCP和UDP使用该协议从一个网络传送数据包到另一个网络。把**IP想像成一种高速公路**，它允许其它协议在上面行驶并找到到其它电脑的出口。**TCP和UDP是高速公路上的“卡车”，它们携带的货物就是像HTTP**，文件传输 协议FTP这样的协议等。

你应该能理解，TCP和UDP是FTP，HTTP和SMTP之类使用的传输层协议。虽然TCP和UDP都是用来传输其他协议的，它们却有一个显著的不同：TCP提供有保证的数据传输，而UDP不提供。这意味着TCP有一个特殊的机制来确保数据安全的不出错的从一个端点传到另一个端点，而UDP不提供任何这样的保证。

HTTP(超文本传输协议)是利用TCP在两台电脑(通常是Web服务器和客户端)之间传输信息的协议。客户端使用Web浏览器发起HTTP请求给Web服务器，Web服务器发送被请求的信息给客户端。

1、HTTP协议的几个重要概念

1.连接(Connection)：一个传输层的实际环流，它是建立在两个相互通讯的应用程序之间。

2.消息(Message)：HTTP通讯的基本单位，包括一个结构化的八元组序列并通过连接传输。

3.请求(Request)：一个从客户端到服务器的请求信息包括应用于资源的方法、资源的标识符和协议的版本号

4.响应(Response)：一个从服务器返回的信息包括HTTP协议的版本号、请求的状态(例如"成功"或"没找到")和文档的MIME类型。

5.资源(Resource)：由URI标识的网络数据对象或服务。

6.实体(Entity)：数据资源或来自服务资源的回映的一种特殊表示方法，它可能被包围在一个请求或响应信息中。一个实体包括实体头信息和实体的本身内容。

7.客户机(Client)：一个为发送请求目的而建立连接的应用程序。

8.用户代理(Useragent)：初始化一个请求的客户机。它们是浏览器、编辑器或其它用户工具。

9.服务器(Server)：一个接受连接并对请求返回信息的应用程序。

10.源服务器(Originserver)：是一个给定资源可以在其上驻留或被创建的服务器。

11.代理(Proxy)：**一个中间程序，它可以充当一个服务器，也可以充当一个客户机，为其它客户机建立请求。**请求是通过可能的翻译在内部或经过传递到其它的服务器中。一个代理在发送请求信息之前，必须解释并且如果可能重写它。 代理经常作为通过防火墙的客户机端的门户，代理还可以作为一个帮助应用来通过协议处理没有被用户代理完成的请求。

12.网关(Gateway)：**一个作为其它服务器中间媒介的服务器。**与代理不同的是，网关接受请求就好象对被请求的资源来说它就是源服务器；发出请求的客户机并没有意识到它在同网关打交道。 网关经常作为通过防火墙的服务器端的门户，网关还可以作为一个协议翻译器以便存取那些存储在非HTTP系统中的资源。

13.通道(Tunnel)：**是作为两个连接中继的中介程序。**一旦激活，通道便被认为不属于HTTP通讯，尽管通道可能是被一个HTTP请求初始化的。当被中继的连接两端关闭时，通道便消失。当一个门户(Portal)必须存在或中介(Intermediary)不能解释中继的通讯时通道被经常使用。

14.缓存(Cache)：反应信息的局域存储。

交换机、路由器、网桥、网卡、集线器的区别

网卡(NIC)：负责接收网络上传过来的数据包，解包后将数据通过主板上的总线传输给本地计算机；另一方面它将本地计算机上的数据经过打包后送入网络。

集线器(HUB)：**是一个多端口的信号放大设备。**能够改变网络传输信号、扩展网络规模、构建网络、连接PC、服务器和外设。它不需任何软件支持，工作在局域网(LAN)环境，像网卡一样，应用于OSI参考模型第一层，因此又被称为物理层设备。

交换机：也称为交换式集线器，是简化（典型）的网桥，一般用于**互连相同类型的LAN**（例如：以太网—以太网的互连）。交换机和网桥的不同在于：交换机端口数较多；交换机的数据传输效率较高。

网桥：用于连接两个具有相同或相似体系结构的局域网，它是**数据链路层**的连接设备。

路由器：主要作用就是进行网络路由选择。常用于相同类型的局域网互连，划分子网段等。如：当一个网络中的主机要给另一个网络中的主机发送分组信息时，它首先把分组送给同一网络中用于网间连接的路由。

网关（Gateway）：是连接两个具有**不同体系结构**的计算机网络的设备。在OSI / RM中，网关属于最高层—**应用层的设备。**（注：现在的新型路由器也兼有网关的功能） **网关与网桥区别：** （1）网关是用来实现不同局域网(LAN)的连接，而网桥用于同类LAN的连接；　　（2）网关建在应用层，网桥建在数据链路层；　　（3）网关比起网桥有一个主要的优势，它可以将具有不相容的地址格式的网络相连起来。 **中继器：**（RP）是最简单的网络互联设备，主要完成物理层的功能，负责在两个节点的物理层上按二进制位传递信息，对衰减的信号进行放大，保持与原数据相同，以此来延长网络的长度。

HTTP协议

特点：

1、简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。

2、灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。

3.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

4.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。 5、支持B/S及C/S模式。

HTTP之URL

HTTP使用统一资源标识符（Uniform Resource Identifiers, URI）来传输数据和建立连接。URL是一种特殊类型的URI，包含了用于查找某个资源的足够的信息

URL,全称是UniformResourceLocator, 中文叫统一资源定位符,是互联网上用来标识某一处资源的地址。以下面这个URL为例，介绍下普通URL的各部分组成：

http://www.aspxfans.com:8080/news/index.asp?boardID=5&ID=24618&page=1#name

从上面的URL可以看出，一个完整的URL包括以下几部分： **1.协议部分**：该URL的协议部分为“http：”，这代表网页使用的是HTTP协议。在Internet中可以使用多种协议，如HTTP，FTP等等本例中使用的是HTTP协议。
在"HTTP"后面的“//”为分隔符

2.域名部分：该URL的域名部分为“www.aspxfans.com”。一个URL中，也可以使用IP地址作为域名使用

3.端口部分：跟在域名后面的是端口，域名和端口之间使用“:”作为分隔符。端口不是一个URL必须的部分，如果省略端口部分，将采用默认端口

4.虚拟目录部分：从域名后的第一个“/”开始到最后一个“/”为止，是虚拟目录部分。虚拟目录也不是一个URL必须的部分。本例中的虚拟目录是“/news/”

5.文件名部分：从域名后的最后一个“/”开始到“？”为止，是文件名部分，如果没有“?”,则是从域名后的最后一个“/”开始到“#”为止，是文件部分，如果没有“？”和“#”，那么从域名后的最后一个“/”开始到结束，都是文件名部分。本例中的文件名是“index.asp”。文件名部分也不是一个URL必须的部分，如果省略该部分，则使用默认的文件名

6.锚部分：从“#”开始到最后，都是锚部分。本例中的锚部分是“name”。锚部分也不是一个URL必须的部分

7.参数部分：从“？”开始到“#”为止之间的部分为参数部分，又称搜索部分、查询部分。本例中的参数部分为“boardID=5&ID=24618&page=1”。参数可以允许有多个参数，参数与参数之间用“&”作为分隔符。

HTTP之请求消息Request

客户端发送一个HTTP请求到服务器的请求消息包括以下格式：

请求行（request line）、请求头部（header）、空行和请求数据四个部分组成。

Get请求例子，使用Charles抓取的request：

```
GET /562f25980001b1b106000338.jpg HTTP/1.1
Host    img.mukewang.com
User-Agent    Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/51.0.2704.106 Safari/537.36
Accept    image/webp,image/*,*/*;q=0.8
Referer    http://www.imooc.com/
Accept-Encoding    gzip, deflate, sdch
Accept-Language    zh-CN,zh;q=0.8
```

第一部分：请求行，用来说明请求类型,要访问的资源以及所使用的HTTP版本.

GET说明请求类型为GET,[/562f25980001b1b106000338.jpg]为要访问的资源，该行的最后一部分说明使用的是HTTP1.1版本。

第二部分：请求头部，紧接着请求行（即第一行）之后的部分，用来说明服务器要使用的附加信息

从第二行起为请求头部，HOST将指出请求的目的地.User-Agent,服务器端和客户端脚本都能访问它,它是浏览器类型检测逻辑的重要基础.该信息由你的浏览器来定义,并且在每个请求中自动发送等等

第三部分：空行，请求头部后面的空行是必须的

即使第四部分的请求数据为空，也必须有空行。

第四部分：请求数据也叫主体，可以添加任意的其他数据。

这个例子的请求数据为空。

POST请求例子，使用Charles抓取的request：

```
POST / HTTP1.1
Host:www.wrox.com
User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR
3.0.04506.648; .NET CLR 3.5.21022)
Content-Type:application/x-www-form-urlencoded
Content-Length:40
Connection: Keep-Alive

name=Professional%20Ajax&publisher=Wiley
```

第一部分：请求行，第一行明了是post请求，以及http1.1版本。 第二部分：请求头部，第二行至第六行。 第三部分：空行，第七行的空行。 第四部分：请求数据，第八行。

HTTP请求方法：

根据HTTP标准，HTTP请求可以使用多种请求方法。 HTTP1.0定义了三种请求方法： GET, POST 和 HEAD方法。 HTTP1.1新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

```
GET       请求指定的页面信息，并返回实体主体。
HEAD      类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头
POST      向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。
PUT       从客户端向服务器传送的数据取代指定的文档的内容。
DELETE        请求服务器删除指定的页面。
CONNECT       HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。
OPTIONS       允许客户端查看服务器的性能。
TRACE       回显服务器收到的请求，主要用于测试或诊断。
```

HTTP工作原理：

HTTP协议定义Web客户端如何从Web服务器请求Web页面，以及服务器如何把Web页面传送给客户端。HTTP协议采用了请求/响应模型。客户端向服务器发送一个请求报文，请求报文包含请求的方法、URL、协议版本、请求头部和请求数据。服务器以一个状态行作为响应，响应的内容包括协议的版本、成功或者错误代码、服务器信息、响应头部和响应数据。

以下是 HTTP 请求/响应的步骤：

> **1、客户端连接到Web服务器**
>
> 一个HTTP客户端，通常是浏览器，与Web服务器的HTTP端口（默认为80）建立一个TCP套接字连接。例如，http://www.oakcms.cn。
>
> **2、发送HTTP请求**
>
> 通过TCP套接字，客户端向Web服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据4部分组成。
>
> **3、服务器接受请求并返回HTTP响应**
>
> Web服务器解析请求，定位请求资源。服务器将资源复本写到TCP套接字，由客户端读取。一个响应由状态行、响应头部、空行和响应数据4部分组成。
>
> **4、释放连接TCP连接**
>
> 若connection 模式为close，则服务器主动关闭TCP连接，客户端被动关闭连接，释放TCP连接;若connection 模式为keepalive，则该连接会保持一段时间，在该时间内可以继续接收请求;
>
> **5、客户端浏览器解析HTML内容**
>
> 客户端浏览器首先解析状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的HTML文档和文档的字符集。客户端浏览器读取响应数据HTML，根据HTML的语法对其进行格式化，并在浏览器窗口中显示。
>
> 例如：在浏览器地址栏键入URL，按下回车之后会经历以下流程：
>
> 1、浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址;
>
> 2、解析出 IP 地址后，根据该 IP 地址和默认端口 80，和服务器建立TCP连接;
>
> 3、浏览器发出读取文件(URL 中域名后面部分对应的文件)的HTTP 请求，该请求报文作为 TCP 三次握手的第三个报文的数据发送给服务器;
>
> 4、服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器;

> 5、释放 TCP连接;
>
> 6、浏览器将该 html 文本并显示内容;

单例模式

单例模式(Singleton Pattern)：单例模式**确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例**，这个类称为单例类，它**提供全局访问的方法。**

三要素：

- 一个静态类变量；
- 一个私有构造方法；
- 一个全局静态的类方法。

优点：**节约资源，节省时间。**

- 由于频繁使用的对象，可以省略创建对象所花费的时间，这对于那些重量级的对象而言，是很重要的。
- 因为不需要频繁创建对象，我们的GC压力也减轻了。

缺点：简单的单例模式设计开发都比较简单，但是**复杂的单例模式需要考虑线程安全等并发问题**，引入了部分复杂度。**职责过重**（既充当工厂角色，提供工厂方法，同时又充当了产品角色。包含一些业务方法）。

类型：懒汉式（线程不安全）、饿汉式（天生线程安全）。

饿汉式：初始化类时，直接就创建唯一实例；

（1）

```
public class Singleton {
    private static Singleton instance = new Singleton();////私有静态变量
    private Singleton() {}//私有构造函数
    public static Singleton getInstance() {//公有静态函数
        return instance;
    }
}
```

（2）枚举方式：（线程安全，**可防止反射构建**），在枚举类对象被反序列化的时候，保证反序列的返回结果是同一对象。

```
public enum Singleton {
    INSTANCE;
}
```

懒汉式：

（1）双重校验锁：

```
//第一次判断 instance是否为空是为了确保返回的实例不为空
//第二次判断 instance是否为空是为了防止创建多余的实例
public class Singleton {
    private volatile static Singleton instance = null;
    private Singleton() {}
```

```
    public static Singleton getInstance() {
        if(instance == null) {
            synchronized(Singleton.class){
                if(instance == null) {//线程B
                    instance = new Singleton();//并非是一个原子操作     线程A
                    /**
                    * 会被编译器编译成如下JVM指令：
                    * memory = allocate(); - 1、分配对象的内存空间
                    * ctorInstance(memory); - 2、初始化对象
                    * instance = memory; - 3、设置instance指向刚分配的内存地址
                    */
                }
            }
        }
        return instance;
    }
}
```

synchronized同步块里面能够保证只创建一个对象。但是**通过在synchronized的外面增加一层判断，就可以在对象一经创建以后，不再进入synchronized同步块。这种方案不仅减小了锁的粒度，保证了线程安全，性能方面也得到了大幅提升。** 进入Synchronized 临界区以后，还要再做一次判空。因为当两个线程同时访问的时候，线程A构建完对象，线程B也已经通过了最初的判空验证，不做第二次判空的话，线程B还是会再次构建instance对象。

Volatile：

- 可见性；
- 防止指令重排序：**防止new Singleton时指令重排序导致其他线程获取到未初始化完的对象。**指令顺序并非一成不变，有可能会经过JVM和CPU的优化，指令重排成下面的顺序：1，3，2。在3执行完毕，2未执行之前，被线程2抢占了，这时instance已经是非null了（但却没有初始化），所以线程2会直接返回instance，然后使用，报错。

（2）静态内部类：线程安全，懒加载

```
//INSTANCE对象初始化的时机并不是在单例类Singleton被加载的时候，而是在调用getInstance方法，使得静态内部类SingletonHolder被加载的时候。因此这种实现方式是利用classloader的加载机制来实现懒加载，并保证构建单例的线程安全。
public class Singleton {
    private static class SingletonHolder {
        private static final Singleton INSTANCE = new Singleton();
    }
    private Singleton() {}
    public static Singleton getInstance() {
        return SingletonHolder.INSTANCE;
    }
}
```

数据库的安全性

- 系统运行安全，系统运行安全通常受到的威胁如下，一些网络不法分子**通过网络，局域网等途径通过入侵电脑使系统无法正常启动，或超负荷让机子运行大量算法，并关闭cpu风扇，使cpu过热烧坏等破坏性活动**；

- 系统信息安全，系统安全通常受到的威胁如下，**黑客对数据库入侵，并盗取想要的资料**数据库系统的安全特性主要是针对数据而言的，包括数据独立性、数据安全性、数据完整性、并发控制、故障恢复等几个方面。

数据库安全的防护技术有：数据库加密（核心数据存储加密）、数据库防火墙（防漏洞、防攻击）、数据脱敏（敏感数据匿名化）等。

SQL注入

SQL注入，就是**通过把SQL命令插入到Web表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。**具体来说，它是利用现有应用程序，将（恶意的）SQL命令注入到后台数据库引擎执行的能力，它可以通过在Web表单中输入（恶意）SQL语句得到一个存在安全漏洞的网站上的数据库，而不是按照设计者意图去执行SQL语句。 比如先前的很多影视网站泄露VIP会员密码大多就是通过WEB表单递交查询字符暴出的，这类表单特别容易受到SQL注入式攻击.

防护：

- 永远不要信任用户的输入。对用户的输入进行校验，可以通过正则表达式，或限制长度；对单引号和双"-"进行转换等。
- 永远不要使用动态拼装sql，可以使用参数化的sql或者直接使用存储过程进行数据查询存取。
- 永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。
- 不要把机密信息直接存放，加密或者hash掉密码和敏感的信息。
- 应用的异常信息应该给出尽可能少的提示，最好使用自定义的错误信息对原始错误信息进行包装 。
- sql注入的检测方法一般采取辅助软件或网站平台来检测，软件一般采用sql注入检测工具jsky，网站平台就有亿思网站安全平台检测工具。MDCSOFT SCAN等。采用MDCSOFT-IPS可以有效的防御SQL注入，XSS攻击等。

路由选择协议

1、AS(Autonomous System)自治系统 自治系统是在单一技术管理体系下的多个路由器的集合，在自治系统内部使用内部网关协议（IGP）和通用参数来决定如何路由数据包，在自治系统间则使用AS间路由协议来路由数据包。1）AS内部使用IGP(Interior Gateway Protocol)协议进行自我相互之间更新路由表信息。 2）不同AS之间使用EGP(External Gateway Protocol)协议进行AS之间的路由信息更新。

2、常用路由选择协议 IGP：RIP和OSPF ***1）RIP协议***

IGP协议的一种，只和相邻路由器，在固定时间间隔中，交换各自整个路由表。基于UDP协议进行数据报发送。**RIP协议的距离**：实际就是值经过多少个路由器可到达网络，即跳数。一条路径最大只能是15，16表示不可达。 ***优点：***收敛速度快，刚刚开始路由器只知道于自己直连的路由信息，但是通过定时与相邻路由器交换路由表，可以快速在AS中达到收敛。 ***缺点：***好消息传播快，坏消息传播慢。因为坏消息有可能需要经过多次更新后直到距离达到16才会被AS中其它路由都发现。 ***RIP1和RIP2区别：*** RIP1不发送子网掩码，RIP2发送掩码，支持VLSN（可变长子网掩码） RIP1不支持认证，RIP2支持认证。

RIP协议更新路由表算法： 路由条目格式：<net-id,distance,next-hop,...> 令当前路由器路由表为CurT *a. 收到相邻路由器R的路由表T信息后，将下一跳全部改为R，每项距离都加1。 b. for each x in T loop if CurT don't contain x.net-id then CurT.addItem(x); else y = CurT.getItemByNetID(x.net-id) if x.next-hop == y.next-hop then update y.distance = x.distance;//更新路由项距离 else y = x.distance < y.distance ? x : y;//选择距离小的对进行更新 endif endif endloop c. 如果在规定的时间中没有收到相邻路由器R的路由信息，则标记该路由器不可达即设置距离为16。*

2）OSPF(Open Shortest Path First)协议 使用洪泛法（即向所有其它路由器）发送更新信息，基于IP数据报进行信息发送。

3）BGP协议

是AS系统之间进行路由信息交换的协议，BGP协议的一种。 原理：每个AS系统指派一个或者多个系统内的代表作为该AS的发言人，不同AS之间通过发言人之间进行信息交流，然后在由各自的更新算法，根据收到的其它AS的路由信息，来更新自身的路由信息。 BGP协议是唯一个使用TCP协议进行路由更新的协议。

3、 总结 RIP是应用层IGP路由选择协议，使用UDP进行路由信息更新。 OSPF是网络层IGP路由选择协议，直接使用IP协议发送更新数据。 BGP是应用层EGP路由选择协议，使用TCP协议更新路由信息。