**一、String,StringBuffer, StringBuilder 的区别是什么？String为什么是不可变的？**

答：   1、String是字符串常量，StringBuffer和StringBuilder都是字符串变量。后两者的字符内容可变，而前者创建后内容不可变。

2、String不可变是因为在JDK中String类被声明为一个final类。

3、StringBuffer是线程安全的，而StringBuilder是非线程安全的。

ps：线程安全会带来额外的系统开销，所以StringBuilder的效率比StringBuffer高。如果对系统中的线程是否安全很掌握，可用StringBuffer，在线程不安全处加上关键字Synchronize。

**二、Vector,ArrayList, LinkedList的区别是什么？**

答：   1、Vector、ArrayList都是以类似数组的形式存储在内存中，LinkedList则以链表的形式进行存储。

2、List中的元素有序、允许有重复的元素，Set中的元素无序、不允许有重复元素。

3、Vector线程同步，ArrayList、LinkedList线程不同步。

4、LinkedList适合指定位置插入、删除操作，不适合查找；ArrayList、Vector适合查找，不适合指定位置的插入、删除操作。

5、ArrayList在元素填满容器时会自动扩充容器大小的50%，而Vector则是100%，因此ArrayList更节省空间。

          详见：<http://www.cnblogs.com/mgod/archive/2007/08/05/844011.html>

**三、HashTable, HashMap，TreeMap区别？**

答：   1、HashTable线程同步，HashMap非线程同步。

2、HashTable不允许<键,值>有空值，HashMap允许<键,值>有空值。

3、HashTable使用Enumeration，HashMap使用Iterator。

4、HashTable中hash数组的默认大小是11，增加方式的old\*2+1，HashMap中hash数组的默认大小是16，增长方式一定是2的指数倍。

5、TreeMap能够把它保存的记录根据键排序，默认是按升序排序。

详见：<http://www.zlmind.com/?p=679>

**五、Tomcat，Apache，JBoss的区别？**

答：   1、Apache是Http服务器，Tomcat是web服务器，JBoss是应用服务器。

2、Apache解析静态的html文件；Tomcat可解析jsp动态页面、也可充当servlet容器。

详见：<http://foohsinglong.iteye.com/blog/1195780>

**六、GET，POST区别？**

答：   基础知识：Http的请求格式如下。

<request line>           主要包含三个信息：1、请求的类型（GET或POST），2、要访问的资源（如\res\img\a.jif），3、Http版本（http/1.1）

<header>                 用来说明服务器要使用的附加信息

<blank line>              这是Http的规定，必须空一行

[<request-body>]      请求的内容数据

区别：

1、Get是从服务器端获取数据，Post则是向服务器端发送数据。

2、在客户端，Get方式通过URL提交数据，在URL地址栏可以看到请求消息，该消息被编码过；Post数据则是放在Html header内提交。

3、对于Get方式，服务器端用Request.QueryString获取变量的值；对用Post方式，服务器端用Request.Form获取提交的数据值。

4、Get方式提交的数据最多1024字节，而Post则没有限制。

5、Get方式提交的参数及参数值会在地址栏显示，不安全，而Post不会，比较安全。

详见：<http://www.cnblogs.com/wxf0701/archive/2008/08/17/1269798.html>

**七、Session, Cookie区别**

答：   1、Session由应用服务器维护的一个服务器端的存储空间；Cookie是客户端的存储空间，由浏览器维护。

2、用户可以通过浏览器设置决定是否保存Cookie，而不能决定是否保存Session，因为Session是由服务器端维护的。

3、Session中保存的是对象，Cookie中保存的是字符串。

4、Session和Cookie不能跨窗口使用，每打开一个浏览器系统会赋予一个SessionID，此时的SessionID不同，若要完成跨浏览器访问数据，可以使用       Application。

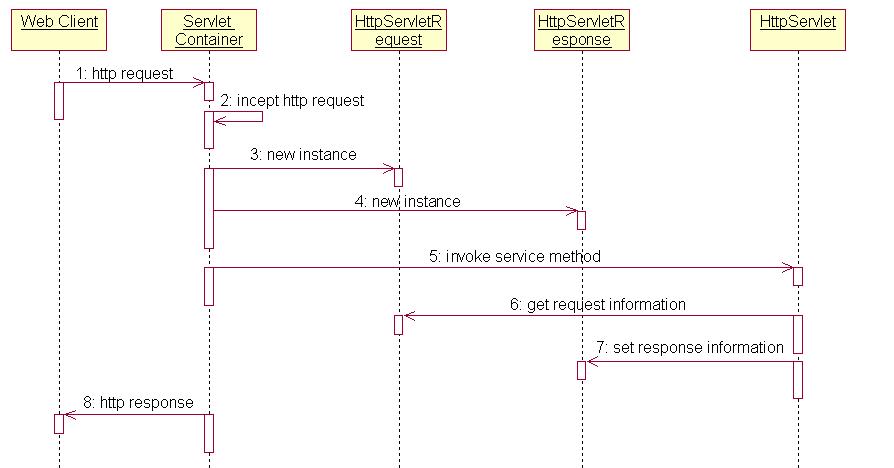
5、Session、Cookie都有失效时间，过期后会自动删除，减少系统开销。

详见：<http://www.chinahtml.com/1007/128010707619425.html>

**八、Servlet的生命周期**

答：   大致分为4部：Servlet类加载-->实例化-->服务-->销毁

        下图是Tomcat中Servlet时序图。



1、Web Client向Servlet容器(Tomcat)发出Http请求。

2、Servlet容器接收Client端的请求。

3、Servlet容器创建一个HttpRequest对象，将Client的请求信息封装到这个对象中。

4、Servlet创建一个HttpResponse对象。

5、Servlet调用HttpServlet对象的service方法，把HttpRequest对象和HttpResponse对象作为参数传递给HttpServlet对象中。

6、HttpServlet调用HttpRequest对象的方法，获取Http请求，并进行相应处理。

7、处理完成HttpServlet调用HttpResponse对象的方法，返回响应数据。

8、Servlet容器把HttpServlet的响应结果传回客户端。

        其中的3个方法说明了Servlet的生命周期：

1、init()：负责初始化Servlet对象。

2、service()：负责响应客户端请求。

3、destroy()：当Servlet对象推出时，负责释放占用资源。

详见：<http://sunnylocus.iteye.com/blog/342996>

**九、HTTP 报文包含内容**

答：   主要包含四部分：

1、request line

2、header line

3、blank line

4、request body

详见：<http://blog.csdn.net/heruiup/article/details/6072961>

**十、Statement与PreparedStatement的区别,什么是SQL注入，如何防止SQL注入**

答：   1、PreparedStatement支持动态设置参数，Statement不支持。

2、PreparedStatement可避免如类似 单引号 的编码麻烦，Statement不可以。

3、PreparedStatement支持预编译，Statement不支持。

4、在sql语句出错时PreparedStatement不易检查，而Statement则更便于查错。

5、PreparedStatement可防止Sql助于，更加安全，而Statement不行。

 详见：<http://blog.163.com/xiaokangzhijia@126/blog/static/1659548562010927222912/>

 什么是SQL注入：

 通过sql语句的拼接达到无参数查询数据库数据目的的方法。

 如将要执行的sql语句为 select \* from table where name = "+appName+"，利用appName参数值的输入，来生成恶意的sql语句，如将['or'1'='1']  传入可在数据库中执行。

 因此可以采用PrepareStatement来避免Sql注入，在服务器端接收参数数据后，进行验证，此时PrepareStatement会自动检测，而Statement不  行，需要手工检测。

**十一、sendRedirect, foward区别**

答：   1、foward是服务器端控制页面转向，在客户端的浏览器地址中不会显示转向后的地址；sendRedirect则是完全的跳转，浏览器中会显示跳转的地址并重             新发送请求链接。

原理：forward是服务器请求资源，服务器直接访问目标地址的URL，把那个URL的响应内容读取过来，然后再将这些内容返回给浏览器，浏览器根本不知         道服务器发送的这些内容是从哪来的，所以地址栏还是原来的地址。

        redirect是服务器端根据逻辑，发送一个状态码，告诉浏览器重新去请求的那个地址，浏览器会用刚才的所有参数重新发送新的请求。

详见：<http://dubaopeng.iteye.com/blog/1056564>

**十二、关于JAVA内存模型，一个对象（两个属性，四个方法）实例化100次，现在内存中的存储状态，几个对象，几个属性，几个方法。**

答：   由于JAVA中new出来的对象都是放在堆中，所以如果要实例化100次，将在堆中产生100个对象，一般对象与其中的属性、方法都属于一个整体，但如果         属性和方法是静态的，就是用static关键字声明的，那么属于类的属性和方法永远只在内存中存在一份。

详见：<http://wenwen.soso.com/z/q252323203.htm?sp=4000>

**十三、谈谈Hibernate的理解，一级和二级缓存的作用，在项目中Hibernate都是怎么使用缓存的。**

答：   Hibernate是一个开发的对象关系映射框架（ORM）。它对JDBC进行了非常对象封装，Hibernate允许程序员采用面向对象的方式来操作关系数据库。

Hibernate的优点：

1、程序更加面向对象

2、提高了生产率

3、方便移植

4、无入侵性。

缺点：

1、效率比JDBC略差

2、不适合批量操作

3、只能配置一种关联关系

Hibernate有四种查询方式：

1、get、load方法，根据id号查询对象。

2、Hibernate query language

3、标准查询语言

4、通过sql查询

Hibernage工作原理：

1、配置hibernate对象关系映射文件、启动服务器

2、服务器通过实例化Configuration对象，读取hibernate.cfg.xml文件的配置内容，并根据相关的需求建好表以及表之间的映射关系。

3、通过实例化的Configuration对象建立SeesionFactory实例，通过SessionFactory实例创建Session对象。

4、通过Seesion对象完成数据库的增删改查操作。

Hibernate中的状态转移

临时状态（transient）

1、不处于session缓存中

2、数据库中没有对象记录

java是如何进入临时状态的：1、通过new语句创建一个对象时。2、刚调用session的delete方法时，从seesion缓存中删除一个对象时。

持久化状态(persisted)

1、处于session缓存中

2、持久化对象数据库中没有对象记录

3、seesion在特定的时刻会保存两者同步

java如何进入持久化状态：1、seesion的save()方法。2、seesion的load().get()方法返回的对象。3、seesion的find()方法返回的list集合中存放的对象。4、Session的update().save()方法。

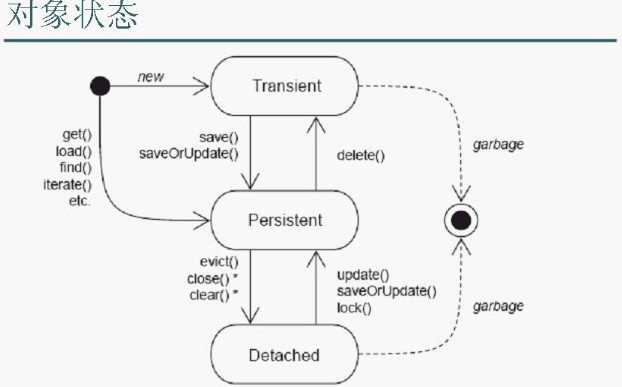
流离状态（detached）

1、不再位于session缓存中

2、游离对象由持久化状态转变而来，数据库中还没有相应记录。

java如何进入流离状态：1、Session的close()。Session的evict()方法，从缓存中删除一个对象。

具体如下图所示：



Hibernate中的缓存主要有Session缓存（一级缓存）和SessionFactory缓存（二级缓存，一般由第三方提供）。

详见：<http://wxg6203.iteye.com/blog/766947>     <http://jyjiao2007.iteye.com/blog/760554>

**十四、反射讲一讲，主要是概念,都在哪需要反射机制，反射的性能，如何优化**

答：   反射机制的定义：

是在运行状态中，对于任意的一个类，都能够知道这个类的所有属性和方法，对任意一个对象都能够通过反射机制调用一个类的任意方法，这种动态获取类信息及动态调用类对象方法的功能称为java的反射机制。

反射的作用：

1、动态地创建类的实例，将类绑定到现有的对象中，或从现有的对象中获取类型。

2、应用程序需要在运行时从某个特定的程序集中载入一个特定的类

详见：<http://blog.csdn.net/p785810989/article/details/6495956>

**十五、谈谈Hibernate与Ibatis的区别，哪个性能会更高一些**

答：   1、Hibernate偏向于对象的操作达到数据库相关操作的目的；而ibatis更偏向于sql语句的优化。

2、Hibernate的使用的查询语句是自己的hql，而ibatis则是标准的sql语句。

3、Hibernate相对复杂，不易学习；ibatis类似sql语句，简单易学。

性能方面：

1、如果系统数据处理量巨大，性能要求极为苛刻时，往往需要人工编写高性能的sql语句或存错过程，此时ibatis具有更好的可控性，因此性能优于Hibernate。

2、同样的需求下，由于hibernate可以自动生成hql语句，而ibatis需要手动写sql语句，此时采用Hibernate的效率高于ibatis。  
详见：<http://blog.csdn.net/cdh1213/article/details/5967405>     <http://wenku.baidu.com/view/06c4486aaf1ffc4ffe47acfb.html>

**十六、对Spring的理解，项目中都用什么？怎么用的？对IOC、和AOP的理解及实现原理**

答：   Spring是一个开源框架，处于MVC模式中的控制层，它能应对需求快速的变化，其主要原因它有一种面向切面编程（AOP）的优势，其次它提升了系统性能，因为通过          依赖倒置机制（IOC），系统中用到的对象不是在系统加载时就全部实例化，而是在调用到这个类时才会实例化该类的对象，从而提升了系统性能。这两个优秀的性能            使得Spring受到许多J2EE公司的青睐，如阿里里中使用最多的也是Spring相关技术。

Spring的优点：

1、降低了组件之间的耦合性，实现了软件各层之间的解耦。

2、可以使用容易提供的众多服务，如事务管理，消息服务，日志记录等。

3、容器提供了AOP技术，利用它很容易实现如权限拦截、运行期监控等功能。

Spring中AOP技术是设计模式中的动态代理模式。只需实现jdk提供的动态代理接口InvocationHandler，所有被代理对象的方法都由InvocationHandler接管实际的处理任务。面向切面编程中还要理解切入点、切面、通知、织入等概念。

Spring中IOC则利用了Java强大的反射机制来实现。所谓依赖注入即组件之间的依赖关系由容器在运行期决定。其中依赖注入的方法有两种，通过构造函数注入，通过set方法进行注入。

详见：<http://www.360doc.com/content/08/0113/20/7635_970484.shtml>    <http://www.zlmind.com/?p=694>

**十七、线程同步，并发操作怎么控制**

答：   Java中可在方法名前加关键字syschronized来处理当有多个线程同时访问共享资源时候的问题。syschronized相当于一把锁，当有申请者申请该

资源时，如果该资源没有被占用，那么将资源交付给这个申请者使用，在此期间，其他申请者只能申请而不能使用该资源，当该资源被使用完成后将释放该资源上的锁，其他申请者可申请使用。

并发控制主要是为了多线程操作时带来的资源读写问题。如果不加以空间可能会出现死锁，读脏数据、不可重复读、丢失更新等异常。

并发操作可以通过加锁的方式进行控制，锁又可分为乐观锁和悲观锁。

悲观锁：

悲观锁并发模式假定系统中存在足够多的数据修改操作，以致于任何确定的读操作都可能会受到由个别的用户所制造的数据修改的影响。也就是说悲观锁假定冲突总会发生，通过独占正在被读取的数据来避免冲突。但是独占数据会导致其他进程无法修改该数据，进而产生阻塞，读数据和写数据会相互阻塞。

乐观锁：

乐观锁假定系统的数据修改只会产生非常少的冲突，也就是说任何进程都不大可能修改别的进程正在访问的数据。乐观并发模式下，读数据和写数据之间不会发生冲突，只有写数据与写数据之间会发生冲突。即读数据不会产生阻塞，只有写数据才会产生阻塞。

详见：<http://www.cnblogs.com/ego/articles/1514350.html>      <http://blog.csdn.net/niu870781892/article/details/6140755>

**十八、描述struts的工作流程**

答：   1、在web应用启动时，加载并初始化ActionServlet，ActionServlet从struts-config.xml文件中读取配置信息，将它们存放到各个配置对象中。

2、当ActionServlet接收到一个客户请求时，首先检索和用户请求相匹配的ActionMapping实例，如果不存在，就返回用户请求路径无效信息。

3、如果ActionForm实例不存在，就创建一个ActionForm对象，把客户提交的表单数据保存到ActionForm对象中。

4、根据配置信息决定是否需要验证表单，如果需要，就调用ActionForm的validate()方法，如果ActionForm的validate()方法返回null或返回一个不包含ActionMessage的ActionErrors对象，就表示表单验证成功。

5、ActionServlet根据ActionMapping实例包含的映射信息决定请求转发给哪个Action，如果相应的Action实例不存在，就先创建一个实例，然后调用Action的execute()方法。

6、Action的execute()方法返回一个ActionForward对象，ActionServlet再把客户请求转发给ActionForward对象指向的JSP组件。

7、ActionForward对象指向的JSP组件生成动态网页，返回给客户。

**十九、Tomcat的session处理，如果让你实现一个tomcatserver，如何实现session机制**

答：   没有找到合适的答案。

**二十、关于Cache(Ehcache,Memcached)**

答：   详见：<http://xuezhongfeicn.blog.163.com/blog/static/2246014120106144143737/>

**二一、sql的优化相关问题**

答：   这篇文章写的真心不错，值得仔细拜读，所以将其转载过来了。

近期因工作需要，希望比较全面的总结下SQL SERVER数据库性能优化相关的注意事项，在网上搜索了一下,发现很多文章,有的都列出了上百条,但是仔细看发现，有很多似是而非或者过时(可能对SQL SERVER6.5以前的版本或者ORACLE是适用的)的信息，只好自己根据以前的经验和测试结果进行总结了。

我始终认为，一个系统的性能的提高，不单单是试运行或者维护阶段的性能调优的任务，也不单单是开发阶段的事情，而是在整个软件生命周期都需要注意，进行有效工作才能达到的。所以我希望按照软件生命周期的不同阶段来总结数据库性能优化相关的注意事项。

一、             分析阶段

一 般来说，在系统分析阶段往往有太多需要关注的地方，系统各种功能性、可用性、可靠性、安全性需求往往吸引了我们大部分的注意力，但是，我们必须注意，性能 是很重要的非功能性需求，必须根据系统的特点确定其实时性需求、响应时间的需求、硬件的配置等。最好能有各种需求的量化的指标。

另一方面，在分析阶段应该根据各种需求区分出系统的类型，大的方面，区分是OLTP（联机事务处理系统）和OLAP（联机分析处理系统）。

二、             设计阶段

设计阶段可以说是以后系统性能的关键阶段，在这个阶段，有一个关系到以后几乎所有性能调优的过程—数据库设计。

在数据库设计完成后，可以进行初步的索引设计，好的索引设计可以指导编码阶段写出高效率的代码，为整个系统的性能打下良好的基础。

以下是性能要求设计阶段需要注意的：

1、 数据库逻辑设计的规范化

数据库逻辑设计的规范化就是我们一般所说的范式，我们可以这样来简单理解范式：

第1规范：没有重复的组或多值的列，这是数据库设计的最低要求。

第2规范: 每个非关键字段必须依赖于主关键字，不能依赖于一个组合式主关键字的某些组成部分。消除部分依赖，大部分情况下，数据库设计都应该达到第二范式。

第3规范: 一个非关键字段不能依赖于另一个非关键字段。消除传递依赖，达到第三范式应该是系统中大部分表的要求，除非一些特殊作用的表。

更高的范式要求这里就不再作介绍了，个人认为，如果全部达到第二范式，大部分达到第三范式，系统会产生较少的列和较多的表，因而减少了数据冗余，也利于性能的提高。

2、 合理的冗余

完全按照规范化设计的系统几乎是不可能的，除非系统特别的小，在规范化设计后，有计划地加入冗余是必要的。

冗余可以是冗余数据库、冗余表或者冗余字段，不同粒度的冗余可以起到不同的作用。

冗余可以是为了编程方便而增加，也可以是为了性能的提高而增加。从性能角度来说，冗余数据库可以分散数据库压力，冗余表可以分散数据量大的表的并发压力，也可以加快特殊查询的速度，冗余字段可以有效减少数据库表的连接，提高效率。

3、 主键的设计

主键是必要的，SQL SERVER的主键同时是一个唯一索引，而且在实际应用中，我们往往选择最小的键组合作为主键，所以主键往往适合作为表的聚集索引。聚集索引对查询的影响是比较大的，这个在下面索引的叙述。

在有多个键的表，主键的选择也比较重要，一般选择总的长度小的键，小的键的比较速度快，同时小的键可以使主键的B树结构的层次更少。

主键的选择还要注意组合主键的字段次序，对于组合主键来说，不同的字段次序的主键的性能差别可能会很大，一般应该选择重复率低、单独或者组合查询可能性大的字段放在前面。

4、 外键的设计

外键作为数据库对象，很多人认为麻烦而不用，实际上，外键在大部分情况下是很有用的，理由是：

外键是最高效的一致性维护方法，数据库的一致性要求，依次可以用外键、CHECK约束、规则约束、触发器、客户端程序，一般认为，离数据越近的方法效率越高。

谨慎使用级联删除和级联更新，级联删除和级联更新作为SQL SERVER 2000当年的新功能，在2005作 了保留，应该有其可用之处。我这里说的谨慎，是因为级联删除和级联更新有些突破了传统的关于外键的定义，功能有点太过强大，使用前必须确定自己已经把握好 其功能范围，否则，级联删除和级联更新可能让你的数据莫名其妙的被修改或者丢失。从性能看级联删除和级联更新是比其他方法更高效的方法。

5、 字段的设计

字段是数据库最基本的单位，其设计对性能的影响是很大的。需要注意如下：

A、数据类型尽量用数字型，数字型的比较比字符型的快很多。

B、 数据类型尽量小，这里的尽量小是指在满足可以预见的未来需求的前提下的。

C、 尽量不要允许NULL，除非必要，可以用NOT NULL+DEFAULT代替。

D、少用TEXT和IMAGE，二进制字段的读写是比较慢的，而且，读取的方法也不多，大部分情况下最好不用。

E、 自增字段要慎用，不利于数据迁移。

6、 数据库物理存储和环境的设计

在设计阶段，可以对数据库的物理存储、操作系统环境、网络环境进行必要的设计，使得我们的系统在将来能适应比较多的用户并发和比较大的数据量。

这里需要注意文件组的作用，适用文件组可以有效把I/O操作分散到不同的物理硬盘，提高并发能力。

7、 系统设计

整个系统的设计特别是系统结构设计对性能是有很大影响的，对于一般的OLTP系统，可以选择C/S结构、三层的C/S结构等，不同的系统结构其性能的关键也有所不同。

系统设计阶段应该归纳一些业务逻辑放在数据库编程实现，数据库编程包括数据库存储过程、触发器和函数。用数据库编程实现业务逻辑的好处是减少网络流量并可更充分利用数据库的预编译和缓存功能。

8、 索引的设计

在设计阶段，可以根据功能和性能的需求进行初步的索引设计，这里需要根据预计的数据量和查询来设计索引，可能与将来实际使用的时候会有所区别。

关于索引的选择，应改主意：

A、              根据数据量决定哪些表需要增加索引，数据量小的可以只有主键。

B、              根据使用频率决定哪些字段需要建立索引，选择经常作为连接条件、筛选条件、聚合查询、排序的字段作为索引的候选字段。

C、              把经常一起出现的字段组合在一起，组成组合索引，组合索引的字段顺序与主键一样，也需要把最常用的字段放在前面，把重复率低的字段放在前面。

D、              一个表不要加太多索引，因为索引影响插入和更新的速度。

三、             编码阶段

编码阶段是本文的重点，因为在设计确定的情况下，编码的质量几乎决定了整个系统的质量。

编码阶段首先是需要所有程序员有性能意识，也就是在实现功能同时有考虑性能的思想，数据库是能进行集合运算的工具，我们应该尽量的利用这个工具，所谓集合运算实际是批量运算，就是尽量减少在客户端进行大数据量的循环操作，而用SQL语句或者存储过程代替。关于思想和意识，很难说得很清楚，需要在编程过程中来体会。

下面罗列一些编程阶段需要注意的事项：

1、 只返回需要的数据

返回数据到客户端至少需要数据库提取数据、网络传输数据、客户端接收数据以及客户端处理数据等环节，如果返回不需要的数据，就会增加服务器、网络和客户端的无效劳动，其害处是显而易见的，避免这类事件需要注意：

A、横向来看，不要写SELECT \*的语句，而是选择你需要的字段。

B、 纵向来看，合理写WHERE子句，不要写没有WHERE的SQL语句。

C、 注意SELECT INTO后的WHERE子句，因为SELECT INTO把数据插入到临时表，这个过程会锁定一些系统表，如果这个WHERE子句返回的数据过多或者速度太慢，会造成系统表长期锁定，诸塞其他进程。

D、对于聚合查询，可以用HAVING子句进一步限定返回的行。

2、 尽量少做重复的工作

这一点和上一点的目的是一样的，就是尽量减少无效工作，但是这一点的侧重点在客户端程序，需要注意的如下：

A、              控制同一语句的多次执行，特别是一些基础数据的多次执行是很多程序员很少注意的。

B、              减少多次的数据转换，也许需要数据转换是设计的问题，但是减少次数是程序员可以做到的。

C、              杜绝不必要的子查询和连接表，子查询在执行计划一般解释成外连接，多余的连接表带来额外的开销。

D、              合并对同一表同一条件的多次UPDATE，比如

1. UPDATE EMPLOYEE SET FNAME=’HAIWER’ WHERE EMP\_ID=’ VPA30890F’
2. UPDATE EMPLOYEE SET LNAME=’YANG’ WHERE EMP\_ID=’ VPA30890F’

这两个语句应该合并成以下一个语句

1. UPDATE EMPLOYEE SET FNAME=’HAIWER’,LNAME=’YANG’
2. WHERE EMP\_ID=’ VPA30890F’

E、               UPDATE操作不要拆成DELETE操作+INSERT操作的形式，虽然功能相同，但是性能差别是很大的。

F、               不要写一些没有意义的查询，比如

SELECT \* FROM EMPLOYEE WHERE 1=2

3、 注意事务和锁

事务是数据库应用中和重要的工具，它有原子性、一致性、隔离性、持久性这四个属性，很多操作我们都需要利用事务来保证数据的正确性。在使用事务中我们需要做到尽量避免死锁、尽量减少阻塞。具体以下方面需要特别注意：

A、事务操作过程要尽量小，能拆分的事务要拆分开来。

B、 事务操作过程不应该有交互，因为交互等待的时候，事务并未结束，可能锁定了很多资源。

C、 事务操作过程要按同一顺序访问对象。

D、提高事务中每个语句的效率，利用索引和其他方法提高每个语句的效率可以有效地减少整个事务的执行时间。

E、 尽量不要指定锁类型和索引，SQL SERVER允许我们自己指定语句使用的锁类型和索引，但是一般情况下，SQL SERVER优化器选择的锁类型和索引是在当前数据量和查询条件下是最优的，我们指定的可能只是在目前情况下更有，但是数据量和数据分布在将来是会变化的。

F、 查询时可以用较低的隔离级别，特别是报表查询的时候，可以选择最低的隔离级别（未提交读）。

4、 注意临时表和表变量的用法

在复杂系统中，临时表和表变量很难避免，关于临时表和表变量的用法，需要注意：

A、如果语句很复杂，连接太多，可以考虑用临时表和表变量分步完成。

B、 如果需要多次用到一个大表的同一部分数据，考虑用临时表和表变量暂存这部分数据。

C、 如果需要综合多个表的数据，形成一个结果，可以考虑用临时表和表变量分步汇总这多个表的数据。

D、其他情况下，应该控制临时表和表变量的使用。

E、 关于临时表和表变量的选择，很多说法是表变量在内存，速度快，应该首选表变量，但是在实际使用中发现，这个选择主要考虑需要放在临时表的数据量，在数据量较多的情况下，临时表的速度反而更快。

F、 关于临时表产生使用SELECT INTO和CREATE TABLE + INSERT INTO的选择，我们做过测试，一般情况下，SELECT INTO会比CREATE TABLE + INSERT INTO的方法快很多，但是SELECT INTO会锁定TEMPDB的系统表SYSOBJECTS、SYSINDEXES、SYSCOLUMNS，在多用户并发环境下，容易阻塞其他进程，所以我的建议是，在并发系统中，尽量使用CREATE TABLE + INSERT INTO，而大数据量的单个语句使用中，使用SELECT INTO。

G、  注意排序规则，用CREATE TABLE建立的临时表，如果不指定字段的排序规则，会选择TEMPDB的默认排序规则，而不是当前数据库的排序规则。如果当前数据库的排序规则和TEMPDB的排序规则不同，连接的时候就会出现排序规则的冲突错误。一般可以在CREATE TABLE建立临时表时指定字段的排序规则为DATABASE\_DEFAULT来避免上述问题。

5、 子查询的用法

子查询是一个 SELECT 查询，它嵌套在 SELECT、INSERT、UPDATE、DELETE 语句或其它子查询中。任何允许使用表达式的地方都可以使用子查询。

子查询可以使我们的编程灵活多样，可以用来实现一些特殊的功能。但是在性能上，往往一个不合适的子查询用法会形成一个性能瓶颈。

如果子查询的条件中使用了其外层的表的字段，这种子查询就叫作相关子查询。相关子查询可以用IN、NOT IN、EXISTS、NOT EXISTS引入。

关于相关子查询，应该注意：

A、NOT IN、NOT EXISTS的相关子查询可以改用LEFT JOIN代替写法。比如：

1. SELECT PUB\_NAME
2. FROM PUBLISHERS
3. WHERE PUB\_ID NOT IN
4. (SELECT PUB\_ID
5. FROM TITLES
6. WHERE TYPE = ’BUSINESS’)

可以改写成：

1. SELECT A.PUB\_NAME
2. FROM PUBLISHERS A LEFT JOIN TITLES B
3. ON        B.TYPE = ’BUSINESS’ AND
4. A.PUB\_ID=B. PUB\_ID
5. WHERE B.PUB\_ID IS NULL
6. SELECT TITLE
7. FROM TITLES
8. WHERE NOT EXISTS
9. (SELECT TITLE\_ID
10. FROM SALES
11. WHERE TITLE\_ID = TITLES.TITLE\_ID)

可以改写成：

1. SELECT TITLE
2. FROM TITLES LEFT JOIN SALES
3. ON SALES.TITLE\_ID = TITLES.TITLE\_ID
4. WHERE SALES.TITLE\_ID IS NULL

B、 如果保证子查询没有重复 ，IN、EXISTS的相关子查询可以用INNER JOIN 代替。比如：

1. SELECT PUB\_NAME
2. FROM PUBLISHERS
3. WHERE PUB\_ID IN
4. (SELECT PUB\_ID
5. FROM TITLES
6. WHERE TYPE = ’BUSINESS’)

可以改写成：

1. SELECT DISTINCT A.PUB\_NAME
2. FROM PUBLISHERS A INNER JOIN TITLES B
3. ON        B.TYPE = ’BUSINESS’ AND
4. A.PUB\_ID=B. PUB\_ID

C、 IN的相关子查询用EXISTS代替，比如

1. SELECT PUB\_NAME
2. FROM PUBLISHERS
3. WHERE PUB\_ID IN
4. (SELECT PUB\_ID
5. FROM TITLES
6. WHERE TYPE = ’BUSINESS’)

可以用下面语句代替：

1. SELECT PUB\_NAME
2. FROM PUBLISHERS
3. WHERE EXISTS
4. (SELECT 1
5. FROM TITLES
6. WHERE TYPE = ’BUSINESS’ AND
7. PUB\_ID= PUBLISHERS.PUB\_ID)

D、不要用COUNT(\*)的子查询判断是否存在记录，最好用LEFT JOIN或者EXISTS，比如有人写这样的语句：

1. SELECT JOB\_DESC FROM JOBS
2. WHERE (SELECT COUNT(\*) FROM EMPLOYEE WHERE JOB\_ID=JOBS.JOB\_ID)=0

应该改成：

1. SELECT JOBS.JOB\_DESC FROM JOBS LEFT JOIN EMPLOYEE
2. ON EMPLOYEE.JOB\_ID=JOBS.JOB\_ID
3. WHERE EMPLOYEE.EMP\_ID IS NULL
4. SELECT JOB\_DESC FROM JOBS
5. WHERE (SELECT COUNT(\*) FROM EMPLOYEE WHERE JOB\_ID=JOBS.JOB\_ID)<>0

应该改成：

1. SELECT JOB\_DESC FROM JOBS
2. WHERE EXISTS (SELECT 1 FROM EMPLOYEE WHERE JOB\_ID=JOBS.JOB\_ID)

6、 慎用游标

数据库一般的操作是集合操作，也就是对由WHERE子句和选择列确定的结果集作集合操作，游标是提供的一个非集合操作的途径。一般情况下，游标实现的功能往往相当于客户端的一个循环实现的功能，所以，大部分情况下，我们把游标功能搬到客户端。

游标是把结果集放在服务器内存，并通过循环一条一条处理记录，对数据库资源（特别是内存和锁资源）的消耗是非常大的，所以，我们应该只有在没有其他方法的情况下才使用游标。

另外，我们可以用SQL SERVER的一些特性来代替游标，达到提高速度的目的。

A、字符串连接的例子

这是论坛经常有的例子，就是把一个表符合条件的记录的某个字符串字段连接成一个变量。比如需要把JOB\_ID=10的EMPLOYEE的FNAME连接在一起，用逗号连接，可能最容易想到的是用游标：

1. DECLARE @NAME VARCHAR(20)
2. DECLARE @NAME VARCHAR(1000)
3. DECLARE NAME\_CURSOR CURSOR FOR
4. SELECT FNAME FROM EMPLOYEE WHERE JOB\_ID=10 ORDER BY EMP\_ID
5. OPEN NAME\_CURSOR
6. FETCH NEXT FROM RNAME\_CURSOR INTO @NAME
7. WHILE @@FETCH\_STATUS = 0
8. BEGIN
9. SET @NAMES = ISNULL(@NAMES+’,’,’’)+@NAME
10. FETCH NEXT FROM NAME\_CURSOR  INTO @NAME
11. END
12. CLOSE NAME\_CURSOR
13. DEALLOCATE NAME\_CURSOR

可以如下修改，功能相同：

1. DECLARE @NAME VARCHAR(1000)
2. SELECT @NAMES = ISNULL(@NAMES+’,’,’’)+FNAME
3. FROM EMPLOYEE WHERE JOB\_ID=10 ORDER BY EMP\_ID

B、 用CASE WHEN 实现转换的例子

很多使用游标的原因是因为有些处理需要根据记录的各种情况需要作不同的处理，实际上这种情况，我们可以用CASE WHEN语句进行必要的判断处理，而且CASE WHEN是可以嵌套的。比如:

表结构:

1. CREATE TABLE 料件表(
2. 料号           VARCHAR(30),
3. 名称           VARCHAR(100),
4. 主单位         VARCHAR(20),
5. 单位1         VARCHAR(20),
6. 单位1参数      NUMERIC(18,4),
7. 单位2         VARCHAR(20),
8. 单位2参数      NUMERIC(18,4)
9. )
10. GO
11. CREATE TABLE 入库表(
12. 时间               DATETIME,
13. 料号               VARCHAR(30),
14. 单位               INT,
15. 入库数量           NUMERIC(18,4),
16. 损坏数量           NUMERIC(18,4)
17. )
18. GO

其中，单位字段可以是0，1，2，分别代表主单位、单位1、单位2，很多计算需要统一单位，统一单位可以用游标实现：

1. DECLARE @料号     VARCHAR(30),
2. @单位   INT,
3. @参数      NUMERIC(18,4),
4. DECLARE CUR CURSOR FOR
5. SELECT 料号,单位 FROM 入库表 WHERE 单位 <>0
6. OPEN CUR
7. FETCH NEXT FROM CUR INTO @料号,@单位
8. WHILE @@FETCH\_STATUS<>-1
9. BEGIN
10. IF @单位=1
11. BEGIN
12. SET @参数=(SELECT 单位1参数 FROM 料件表 WHERE 料号 =@料号)
13. UPDATE 入库表 SET 数量=数量\*@参数,损坏数量=损坏数量\*@参数,单位=1 WHERE CURRENT OF CUR
14. END
15. IF @单位=2
16. BEGIN
17. SET @参数=(SELECT 单位1参数 FROM 料件表 WHERE 料号 =@料号)
18. UPDATE 入库表 SET 数量=数量\*@参数,损坏数量=损坏数量\*@参数,单位=1 WHERE CURRENT OF CUR
19. END
20. FETCH NEXT FROM CUR INTO @料号,@单位
21. END
22. CLOSE CUR
23. DEALLOCATE CUR

可以改写成：

1. UPDATE A SET
2. 数量=CASE A.单位 WHEN 1 THEN      A.数量\*B. 单位1参数
3. WHEN 2 THEN         A.数量\*B. 单位2参数
4. ELSE A.数量
5. END,
6. 损坏数量= CASE A.单位 WHEN 1 THEN    A. 损坏数量\*B. 单位1参数
7. WHEN 2 THEN         A. 损坏数量\*B. 单位2参数
8. ELSE A. 损坏数量
9. END,
10. 单位=1
11. FROM入库表 A, 料件表 B
12. WHERE    A.单位<>1      AND
13. A.料号=B.料号

C、 变量参与的UPDATE语句的例子

SQL ERVER的语句比较灵活，变量参与的UPDATE语句可以实现一些游标一样的功能，比如：

在

1. SELECT A,B,C,CAST(NULL AS INT) AS 序号
2. INTO #T
3. FROM 表
4. ORDER BY A ,NEWID()

产生临时表后，已经按照A字段排序，但是在A相同的情况下是乱序的，这时如果需要更改序号字段为按照A字段分组的记录序号，就只有游标和变量参与的UPDATE语句可以实现了，这个变量参与的UPDATE语句如下：

1. DECLARE @A INT
2. DECLARE @序号 INT
3. UPDATE #T SET
4. @序号=CASE WHEN A=@A THEN @序号+1 ELSE 1 END,
5. @A=A,
6. 序号=@序号

D、如果必须使用游标，注意选择游标的类型，如果只是循环取数据，那就应该用只进游标（选项FAST\_FORWARD），一般只需要静态游标（选项STATIC）。

E、 注意动态游标的不确定性，动态游标查询的记录集数据如果被修改，会自动刷新游标，这样使得动态游标有了不确定性，因为在多用户环境下，如果其他进程或者本身更改了纪录，就可能刷新游标的记录集。

7、 尽量使用索引

建立索引后，并不是每个查询都会使用索引，在使用索引的情况下，索引的使用效率也会有很大的差别。只要我们在查询语句中没有强制指定索引，索引的选择和使用方法是SQLSERVER的优化器自动作的选择，而它选择的根据是查询语句的条件以及相关表的统计信息，这就要求我们在写SQL语句的时候尽量使得优化器可以使用索引。

为了使得优化器能高效使用索引，写语句的时候应该注意：

A、不要对索引字段进行运算，而要想办法做变换，比如

SELECT ID FROM T WHERE NUM/2=100

应改为:

SELECT ID FROM T WHERE NUM=100\*2

SELECT ID FROM T WHERE NUM/2=NUM1

如果NUM有索引应改为:

SELECT ID FROM T WHERE NUM=NUM1\*2

如果NUM1有索引则不应该改。

发现过这样的语句：

1. SELECT 年,月,金额 FROM 结余表
2. WHERE 100\*年+月=2007\*100+10

应该改为：

1. SELECT 年,月,金额 FROM 结余表
2. WHERE 年=2007 AND
3. 月=10

B、 不要对索引字段进行格式转换

日期字段的例子：

WHERE CONVERT(VARCHAR(10), 日期字段,120)=’2008-08-15’

应该改为

WHERE日期字段〉=’2008-08-15’         AND   日期字段<’2008-08-16’

ISNULL转换的例子：

WHERE ISNULL(字段,’’)<>’’应改为:WHERE字段<>’’

WHERE ISNULL(字段,’’)=’’不应修改

WHERE ISNULL(字段,’F’) =’T’应改为: WHERE字段=’T’

WHERE ISNULL(字段,’F’)<>’T’不应修改

C、 不要对索引字段使用函数

WHERE LEFT(NAME, 3)=’ABC’ 或者WHERE SUBSTRING(NAME,1, 3)=’ABC’

应改为:

WHERE NAME LIKE ‘ABC%’

日期查询的例子：

WHERE DATEDIFF(DAY, 日期,’2005-11-30′)=0应改为:WHERE 日期 >=’2005-11-30′ AND 日期 <’2005-12-1‘

WHERE DATEDIFF(DAY, 日期,’2005-11-30′)>0应改为:WHERE 日期 <’2005-11-30‘

WHERE DATEDIFF(DAY, 日期,’2005-11-30′)>=0应改为:WHERE 日期 <’2005-12-01‘

WHERE DATEDIFF(DAY, 日期,’2005-11-30′)<0应改为:WHERE 日期>=’2005-12-01‘

WHERE DATEDIFF(DAY, 日期,’2005-11-30′)<=0应改为:WHERE 日期>=’2005-11-30‘

D、不要对索引字段进行多字段连接

比如：

WHERE FAME+ ’.’+LNAME=‘HAIWEI.YANG’

应改为:

WHERE FNAME=‘HAIWEI’ AND LNAME=‘YANG’

8、 注意连接条件的写法

多表连接的连接条件对索引的选择有着重要的意义，所以我们在写连接条件条件的时候需要特别的注意。

A、多表连接的时候，连接条件必须写全，宁可重复，不要缺漏。

B、 连接条件尽量使用聚集索引

C、 注意ON部分条件和WHERE部分条件的区别

9、 其他需要注意的地方

经验表明，问题发现的越早解决的成本越低，很多性能问题可以在编码阶段就发现，为了提早发现性能问题，需要注意：

A、程序员注意、关心各表的数据量。

B、 编码过程和单元测试过程尽量用数据量较大的数据库测试，最好能用实际数据测试。

C、 每个SQL语句尽量简单

D、不要频繁更新有触发器的表的数据

E、 注意数据库函数的限制以及其性能

10、              学会分辩SQL语句的优劣

自己分辨SQL语句的优劣非常重要，只有自己能分辨优劣才能写出高效的语句。

A、              查看SQL语句的执行计划，可以在查询分析其使用CTRL+L图形化的显示执行计划，一般应该注意百分比最大的几个图形的属性，把鼠标移动到其上面会显示这个图形的属性，需要注意预计成本的数据，也要注意其标题，一般都是CLUSTERED INDEX SEEK 、INDEX SEEK 、CLUSTERED INDEX SCAN 、INDEX SCAN 、TABLE SCAN等，其中出现SCAN说明语句有优化的余地。也可以用语句

SET SHOWPLAN\_ALL ON

要执行的语句

SET SHOWPLAN\_ALL OFF

查看执行计划的文本详细信息。

B、               用事件探查器跟踪系统的运行，可疑跟踪到执行的语句，以及所用的时间，CPU用量以及I/O数据，从而分析语句的效率。

C、               可以用WINDOWS的系统性能检测器，关注CPU、I/O参数

四、             测试、试运行、维护阶段

测试的主要任务是发现并修改系统的问题，其中性能问题也是一个重要的方面。重点应该放在发现有性能问题的地方，并进行必要的优化。主要进行语句优化、索引优化等。

试运行和维护阶段是在实际的环境下运行系统，发现的问题范围更广，可能涉及操作系统、网络以及多用户并发环境出现的问题，其优化也扩展到操作系统、网络以及数据库物理存储的优化。

这个阶段的优花方法在这里不再展开，只说明下索引维护的方法：

A、              可以用DBCC DBREINDEX语句或者SQL SERVER维护计划设定定时进行索引重建，索引重建的目的是提高索引的效能。

B、               可以用语句UPDATE STATISTICS或者SQL SERVER维护计划设定定时进行索引统计信息的更新，其目的是使得统计信息更能反映实际情况，从而使得优化器选择更合适的索引。

C、               可以用DBCC CHECKDB或者DBCC CHECKTABLE语句检查数据库表和索引是否有问题，这两个语句也能修复一般的问题。

D、

五、             网上资料中一些说法的个人不同意见

1、 “应尽量避免在 WHERE 子句中对字段进行 NULL 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

SELECT ID FROM T WHERE NUM IS NULL

可以在NUM上设置默认值0，确保表中NUM列没有NULL值，然后这样查询：

SELECT ID FROM T WHERE NUM=0”

个人意见：经过测试，IS NULL也是可以用INDEX SEEK查找的，0和NULL是不同概念的，以上说法的两个查询的意义和记录数是不同的。

2、 “应尽量避免在 WHERE 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。”

个人意见：经过测试，<>也是可以用INDEX SEEK查找的。

3、 “应尽量避免在 WHERE 子句中使用 OR 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：

SELECT ID FROM T WHERE NUM=10 OR NUM=20

可以这样查询：

SELECT ID FROM T WHERE NUM=10

UNION ALL

SELECT ID FROM T WHERE NUM=20”

个人意见：主要对全表扫描的说法不赞同。

4、 “IN 和 NOT IN 也要慎用，否则会导致全表扫描，如：

SELECT ID FROM T WHERE NUM IN(1,2,3)

对于连续的数值，能用 BETWEEN 就不要用 IN 了：

SELECT ID FROM T WHERE NUM BETWEEN 1 AND 3”

个人意见：主要对全表扫描的说法不赞同。

5、 “如果在 WHERE 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：

SELECT ID FROM T WHERE NUM=@NUM

可以改为强制查询使用索引：

SELECT ID FROM T WITH(INDEX(索引名)) WHERE NUM=@NUM”

个人意见：关于局部变量的解释比较奇怪，使用参数如果会影响性能，那存储过程就该校除了，我坚持我上面对于强制索引的看法。

6、 “尽可能的使用 VARCHAR/NVARCHAR 代替 CHAR/NCHAR ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。”

个人意见：“在一个相对较小的字段内搜索效率显然要高些”显然是对的，但是字段的长短似乎不是由变不变长决定，而是业务本身决定。在SQLSERVER6.5或者之前版本，不定长字符串字段的比较速度比定长的字符串字段的比较速度慢很多，所以对于那些版本，我们都是推荐使用定长字段存储一些关键字段。而在2000版本，修改了不定长字符串字段的比较方法，与定长字段的比较速度差别不大了，这样为了方便，我们大量使用不定长字段。

7、 关于连接表的顺序或者条件的顺序的说法，经过测试，在SQL SERVER，这些顺序都是不影响性能的，这些说法可能是对ORACLE有效。

**二二、oracle中 rownum与rowid的理解，一千条记录我查200到300的记录怎么查？**

答：   详见：<http://www.sunxin.org/article/242.html>

如何查200到300行的记录，可以通过top关键字辅助：select top 100 \* from table where id is not in (select top 200 id from table);

查询n到m行记录的通用公式：select top m \* from table where id is not in (select top n \* from table)

**二三、如何分析ORACLE的执行计划？**

答：   详见：<http://blog.chinaunix.net/space.php?uid=22539324&do=blog&id=2644068>

**二四、 DB中索引原理，种类，使用索引的好处和问题是什么？**

答：    经常需要修改的表不易对改变建立索引，因为数据库对索引进行维护需要额外的开销。对经常需要查询的大数据表需要建立索引，这样会增加

查询的效率。

索引的原理：没有找到合适的答案。

索引的种类：B\*Tree、反向索引、降序索引、位图索引、函数索引。详见：<http://wenku.baidu.com/view/d4d6ee1ba76e58fafab00336.html>

**二五、JVM垃圾回收实现原理。垃圾回收的线程优先级。**

答：   详见：<http://chenchendefeng.iteye.com/blog/455883>

**二六、jvm 最大内存设置。设置的原理。结合垃圾回收讲讲。**

答：   详见：<http://www.open-open.com/lib/view/open1324736648468.html>

| **车次** | **出发站 到达站** | | **出发时间 到达时间** | **历时** | **商务座** | **特等座** | **一等座** | **二等座** | **高级 软卧** | **软卧** | **硬卧** | **软座** | **硬座** | **无座** | **其他** | **备注** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**G101**](javascript:)  **北京南徐州东**  **07:0009:57**  **02:57**当日到达 | | [**G143**](javascript:)  **北京南徐州东**  **14:4117:31**  **02:50**当日到达 | | | | [**G135**](javascript:)  **北京南徐州东**  **13:0015:51**  **02:51**当日到达 | [**G35**](javascript:)  **北京南徐州东**  **15:0518:09**  **03:04**当日到达 |  |  | -- | -- | -- | -- | -- | -- | 14点起售 |
| [**G57**](javascript:)  **北京南徐州东**  **07:2510:23**  **02:58**当日到达 | | [**G117**](javascript:)  **北京南徐州东**  **09:4412:57**  **03:13**当日到达 | | | | [**G137**](javascript:)  **北京南徐州东**  **13:0615:57**  **02:51**当日到达 | [**G145**](javascript:)  **北京南徐州东**  **15:2918:21**  **02:52**当日到达 |  |  | -- | -- | -- | -- | -- | -- | 14点起售 |
| [**G263**](javascript:)  **北京南徐州东**  **07:4810:48**  **03:00**当日到达 | | [**G119**](javascript:)  **北京南徐州东**  **10:0513:03**  **02:58**当日到达 | | | | [**G139**](javascript:)  **北京南徐州东**  **13:4016:41**  **03:01**当日到达 |  |  |  | -- | -- | -- | -- | -- | -- | 14点起售 |
| [**G107**](javascript:)  **北京南徐州东**  **08:0811:01**  **02:53**当日到达 | | [**G265**](javascript:)  **北京南徐州东**  **10:1013:08**  **02:58**当日到达 | | | | [**G59**](javascript:)  **北京南徐州东**  **13:4516:49**  **03:04**当日到达 | \* |  |  | -- | -- | -- | -- | -- | -- | 14点起售 |
| [**G55**](javascript:)  **北京南徐州东**  **08:1311:19**  **03:06**当日到达 | | [**G121**](javascript:)  **北京南徐州东**  **10:4513:41**  **02:56**当日到达 | | | | [**G43**](javascript:)  **北京南徐州东**  **14:0516:54**  **02:49**当日到达 | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
| [**G113**](javascript:)  **北京南徐州东**  **09:0511:49**  **02:44**当日到达 | | [**G33**](javascript:)  **北京南徐州东**  **11:1914:24**  **03:05**当日到达 | | | | [**G141**](javascript:)  **北京南徐州东**  **14:1616:59**  **02:43**当日到达 | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
| [**G41**](javascript:)  **北京南徐州东**  **09:3312:31**  **02:58**当日到达 | | [**G127**](javascript:)  **北京南徐州东**  **11:3514:36**  **03:01**当日到达 | | | | [**G267**](javascript:)  **北京南徐州东**  **14:4617:49**  **03:03**当日到达 | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | |  | | | | \* | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | |  | | | | \* | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | -- | | | | \* | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | | \* | | | | -- | \* | \* | -- | -- | -- | -- | -- | -- | -- | 14点起售 |
|  | |  | | | |  |  |  |  |  |  |  |  |  |  |  |