====2020/08/21====

一、事务隔离性问题：

1. 脏读：一个事务读取到了另一个未提交的数据；

2. 不可重复读：一个事务内，多次读取同一个数据，在这个过程中，另一个事务对数据进行了修改，第一个事务前后读到的数据不一样；

3. 幻读：事务不是独立执行的，一个事务对表中的数据进行修改，涉及到表中的全部数据行，事务开启，这时，第二个事务向表中插入了一行新数据。那么，第一个事务提交后，用户会发现表中还有没有修改的数据。

二、事务隔离级别：

1. 未提交读：允许脏读，一个事务可以读到另一个事务未提交的数据；

2. 提交读：避免脏读，一个事务等待另一个事务提交后才能读取数据，oralce默认；

3. 可重复读：避免不可重复读，开始读数据时，事务开启时，不再允许修改操作，innodb默认；

4. 序列化：事务串行执行，可避免脏读不可重复读幻读。

三、csrf：

1. cross-site request forgery，跨站点请求伪造；

2. 网站A，恶意网站B，用户C：

a. 用户C打开浏览器访问网站A，验证信息返回cookie，用户A登录成功；

b. 同时，用户A再同一浏览器访问恶意网站B，网站B返回一些攻击性代码，发送请求访问网站A；

c. 浏览器收到攻击性代码，在用户不知情的情况下携带Cookie向网站A发送请求，网站A会根据cookie信息，以用户的权限处理该请求，导致网站B的恶意代码被执行。

3. 防御：验证HTTP Referer字段，在请求地址中添加token，在HTTP头中自定义属性并验证。

四、cookie、session、token：

1. cookie：是一个数据，由服务端生成，发送给浏览器，在浏览器存储，以kv的形式存储，下次访问同一网站时，把cookie发送给服务器。

2. session：服务端保存用户信息，保存在内存中，当访问数量多时，会占用服务器性能，用户离开网站的时候，session会被销毁；当用户第一次通过浏览器使用用户名和密码访问服务器时，服务器会验证用户数据，验证成功后在服务器端写入session数据，向客户端浏览器返回sessionid，浏览器将sessionid保存在cookie

中，当用户再次访问服务器时，会携带sessionid，服务器会拿着sessionid从数据库获取session数据，然后进行用户信息查询，查询到，就会将查询到的用户信息返回，从而实现状态保持。缺点：服务器压力大，CSRF跨站点请求伪造，扩展性差。

3. token：

a. 服务器认证成功后，会对用户信息进行加密，生成加密字符串token，返回给客户端；浏览器会将接收到的token存储在Local Storage中；再次访问服务端，服务端对浏览器的token进行解密，对解密后的数据进行验证。

b. 是服务端生成的字符串，作为身份认证的令牌，发送给客户端，客户端使用token使用数据，不需要再输入账号密码验证数据库，减轻服务器压力，减少数据库频繁查询。

4. cookie和token的区别：

a. cookie：用户登录成功后，会在服务端生成session，返回cookie携带sessionId，客户端和服务端同时保存，用户再次操作时，需要带上cookie，在服务端进行验证，cookie有状态。

b. token：只保存在客户端，服务器收到数据后，进行解密验证，token时无状态的，适合跨平台。

====2020/08/21====

一、数据库的三范式

1. 第一范式，字段不可分，原子性，数据库表的每一列都是不可分割的原子数据，而不能是集合，数组等非原子数据项；

2. 第二范式，非主键字段完全依赖于主键，通过主键能确定所有其他字段，部分依赖，由部分主键确定，多对多；

3. 第三范式，非主键字段不能相互依赖，不能传递依赖，a推出b，b推出c，一对多，分2张表。

4. 反范式，减少冗余会产生笛卡尔积，所以用空间换时间。

二、sql查询语句：select id from student order by score desc group by class limit 5;

====2020/08/22====

一、事务隔离级别：

1. 隔离级别：

a. 未提交读：允许脏读，可能读到其他会话中未提交事务修改的数据；

b. 提交读：只能读到已经提交的数据，Oracle；

c. 可重复度：在同一事务内的查询都是事务开始时刻一致的，Innodb。消除了不可重复度，存在幻读；

d. 序列化：串行化读，每次读都需要获得表级共享锁，读写相互都会阻塞。

====2020/08/21====

一、进程和线程：

1. 区别：

a. 进程是程序的**一次执行**，是**资源分配**的基本单位；线程是**CPU调度**的基本单位，是进程中的**一条执行流程**，线程间**共享地址空间和文件资源**；

b. 一个进程可以包含**多个线程**；

c. 线程的**切换、创建、上下文切换**的开销小；

d. 当进程只有**一个**线程时，可以认为进程就**等于**线程；有**多个**线程时，线程之间**共享虚拟内存和文件资源。**

2. 上下文切换：

a. 进程的**上下文切换**：进程间共享CPU资源，CPU从一个进程切换到另一个进程；

b. 两个线程属于同一进程：因为虚拟内存是共享的，所以资源不变，只需要切换线程的**寄存器和私有数据。**

3. 进程是程序的一次执行，是资源分配的基本单元；线程是进程当中的一条执行流程，线程之间可以并发运行且共享相同的**地址空间和文件资源**，每个线程都有独立的寄存器和栈，确保线程的控制流是相对独立的，线程是cpu调度和分派的基本单元，线程缺点是一个**线程挂掉**，进程中的其他线程也会挂掉。

4. 进程的数据结构是，PCB进程控制块，是进程的唯一标识，包含进程描述信息、进程控制信息、资源分配信息、CUP状态信息（断点处继续执行）。PCB如何组织，每个PCB通过链表的方式，把相同状态的进程链接在一起，组成队列，就绪队列、阻塞队列；另一种方式是索引表。

5. 进程的创建：**分配进程标识符，申请PCB，分配资源，加入就绪队列**

a. 分配一个唯一的**进程标识符**，并申请一个空白的PCB**进程控制块**，PCB是有限的，申请失败则创建失败；

b. 为进程**分配资源**，如果资源不足，则会进入等待状态；

c. 初始化PCB；

d. 如果进程的**就绪队列**能够接纳新进程，就插入到就绪队列，等待被调度。

6. 进程终止：**回收资源、撤销PCB，如果有子进程，会终止所有子进程**

a. 包括**正常结束、异常结束、外界干预（kill信号）**

b. 查找终止进程的**PCB**进程控制块；

c. 如果处于执行状态，则**立即结束执行**，将**CPU资源**分配给其他进程；

d. 如果有**子进程**，则终止所有子进程；

e. 将进程**资源**归还给父进程或操作系统；

f. 将PCB从队列中删除。

7. 阻塞，当进程需要等待某一时间完成时，可以调用阻塞语句把自己阻塞等待，必须由其他进程唤醒。

8. 唤醒：从阻塞队列，插入到就绪队列。

9. 阻塞和挂起：

a. 相同点：都会释放CPU；

b. 阻塞的进程在内存中，进程等待资源时发生，

c. 挂起的进程在外存中；

10. 孤儿进程是指，父进程**退出**，子进程仍在运行，子进程将将为孤儿进程，孤儿进程被init进程收养，由init进程对他们完成回收工作；僵尸进程是指，子进程退出，父进程没有调用wait或者waitpid去获取子进程的状态信息，子进程的进程描述符仍然保存在系统中，他就是一个僵尸进程。

11. 调度算法分类：（**时钟中断**）

a. 非抢占式调度算法，一个进程运行，直到**阻塞或退出**，才会调用另一个进程；

b. 抢占式调度算法，时间片机制，在时间间隔的末端发生时钟中断，把CPU控制返回给调度程序，一个进程只允许运行某段时间，如果时间片结束，仍在运行，则**挂起**，调度程序从就绪队列中挑选另一个进程。

12. 调度原则：

a. CPU利用率；

b. 系统吞吐量，单位时间CPU完成进程的数量；

c. 周转时间，进程运行时间+等待时间；

d. 等待时间，进程处于就绪队列的等待时间；

e. 响应时间，交互式强的应用（鼠标键盘），响应时间要短。

13. 调度算法：

a. 先来先服务，非抢占式，先进入就绪队列先运行；

b. 最短作业优先调度算法，优先选择运行时间最短的进程；

c. 高响应比优先调度算法，响应比=（等待时间+执行时间）/执行时间；

d. 时间片轮转调度算法，进程分配一个时间片，一个时间片内没有运行完的进程回到就绪队列尾部，20~50ms；

e. 最高优先级调度算法，静态优先级，动态优先级是随着时间推移，增加等待进程的优先级；

f. 多级反馈队列调度算法：

i. 设置多个队列，优先级从高到低，优先级高时间片短；

ii. 新进程放在第一级队列末尾，先来先服务原则排队等待调度，如果第一级队列的进程在规定时间片没有运行完，会转到第二级队列末尾；

iii. 当较高优先级的队列为空，才调度较低优先级的队列；

iv. 进程运行时，有新进程进入较高优先级队列，则停止执行当前进程，并移入到原队列末尾，让较高优先级进程运行。

v. 兼顾长短作业，且有较好的响应时间。

二、参考资料：

1. https://mp.weixin.qq.com/s/52Ud2ebDbsoLztVjrd-QNA

====2020/08/19====

一、工厂设计模式：

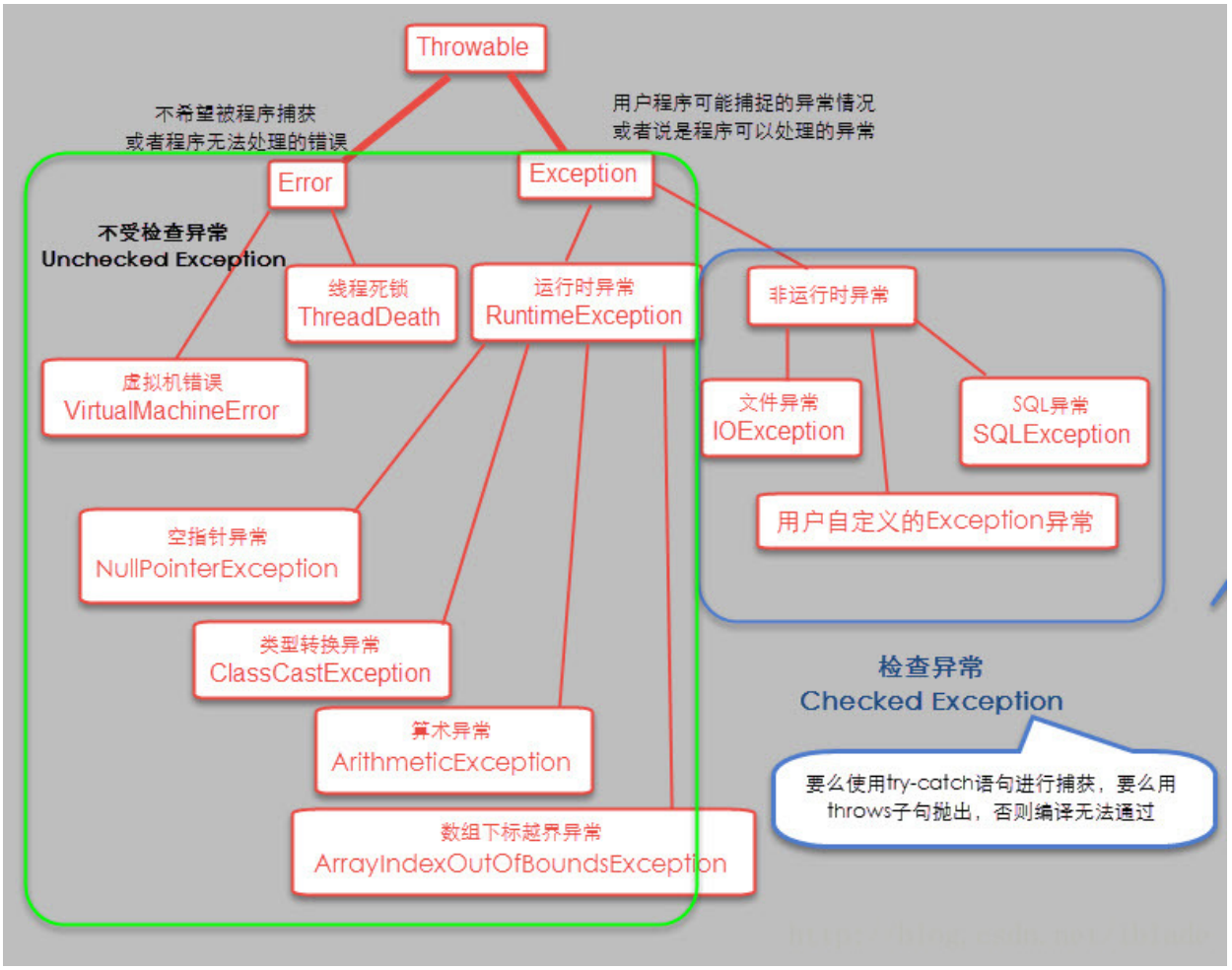
1. 构造一个对象，不必关心构造对象的细节和过程，把细节和过程交给工厂类；

2. 简单工厂顾明思议，实现比较简单，只需要传入一个特定参数即可，不用知道具体的工厂实现类名，缺点就是违背了开闭原则。

3. 工厂方法和抽象工厂，遵守开闭原则，解耦了类关系。但是，扩展比较复杂，需要增加一系列的类。

4. 工厂方法和抽象工厂的区别就在于，抽象工厂关注于某一个产品族，当产品对象之间是有关联关系的一个产品族时用这种方式，而工厂方法没有产品族的概念。

二、Throwable，Error，Exception



====2020/08/18====

一、 索引的创建删除，基于mysql（https://www.jb51.net/article/73372.htm）

1. 索引作用：提高查询效率，适用于数据量大、查询涉及多个表。利用索引加速了where子句满足条件行的搜索，在多表连接查询时，在执行连接时加快了与其他表中的行匹配的速度。

2. 唯一索引，保证没有重复的值；主键索引，是一个唯一索引，一个表只有一个主键索引。

====2020/08/17====

1. 权限管理：

a. 多级树形结构；

b. 过程：

i. 设计数据库的表结构，id，name，parent_id（根节点为0），parent_code（所有上级节点-隔开）；

ii. 首先查询数据库，把所有信息放到List中，遍历list找到根节点（parent_id=0），从根节点出发，查找子节点。

iii. 执行selectCilrd()方法，传入根节点，返回根节点，在中间对根节点操作。拿到根节点的id，通过模糊查询parent_code找到所有子节点放到List1中，再通过一个方法，找到根节点直属的下一级子节点List2，将List2中的子节点加到根节点的children上，再对list2中的每个子节点selectChild()查找其子节点，递归调用，能将孙子节点加到子节点的children上，遍历完所有子节点，return回来拿到根节点。

2. 聚簇索引和非聚簇索引：

a. 聚簇索引的索引和数据保存在一个文件，索引顺序和数据物理存放数据一致，一般使用主键作为索引，主键自增使索引结构紧凑；

====2020/08/14====

1. 垃圾回收方式：

a. 标记清除：两次扫描，第一次进行标记，第二次进行回收，适用于垃圾少；

b. 复制收集：一次扫描，准备新空间，存活复制到新空间，适用于垃圾比例大。有局部性优点，在复制的过程中，会按照**对象被引用**

的顺序将对象复制到新空间，于是，关系较近的对象被放在距离近的内存空间可能性高，内存缓存有更高的效率；

c. 引用计数：每个对象有一个引用计数，对象的引用增加，计数增加；引用计数为0，则回收对象；缺点：循环引用，不适合并行；

2. char和varchar：

a. 定长和变长；char是**固定长度**，插入长度小于定长时，会用**空格补齐**；varchar时按**实际长度存储**；因此char的速度快，**空间换时间**；

b. 存储容量；char最多存**255个字符**，varchar在**5.0**以前是255字符，在5.0以后是**65532字节**；

c. varchar2支持**null**；

d. 对于进程修改长度的数据，varchar会产生**行迁移**；一行数据存在一个block中，修改后block空间不足以存储，产生行迁移，数据库会将整行数据迁移到新的block中；行链接，初始插入一行数据，大小大于block，会链接多个block开存储这一行。

3. 工厂设计模式：

a. 建造一个工厂来创建对象；

b. 构造对象的实例，而不用关系构造对象实例的细节和过程。

4. hashMap、hasgTable、concurrentHashHap：

a. hashMap：数组+链表；

b. hashTable：通过synchronized来保证线程安全；

c. concurrentHashMap：线程安全，通过**锁分段技术**提高并发访问率，hashTable在并发环境下效率低，是因为所有的hashTable线程都必须竞争同一把锁；concurrentHashMap使用锁分段技术，将数据分成小段的存储，给每一段数据配一把锁，一个线程占用锁访问一个数据，其他分段的数据还能够被访问。

5. spring：

a. IOC（控制反转，Inversion ofController）：把对象的控制权交给容器，通过容器来实现对象的装配和管理；

b. AOP（面向切面编程，Aspect-Oriented Programing）：把通用的功能提取出来，织入到应用程序中，比如事务、权限、日志；

6. 孤儿进程和僵尸进程：

a. 孤儿进程是指，父进程退出，子进程仍在运行，子进程将将为孤儿进程，孤儿进程被init进程收养，由init进程对他们完成回收工作；僵尸进程是指，子进程退出，父进程没有调用wait或者waitpid去获取子进程的状态信息，子进程的进程描述符仍然保存在系统中，他就是一个僵尸进程。

b. 孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被init进程所收养，并由init进程对它们完成状态收集工作。

c. 僵尸进程：一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵尸进程。

7. 聚簇索引和非聚簇索引

a. 聚集索引包含索引和数据，索引的叶子节点就是对应的数据。索引顺序和表中记录的物理顺序是一致的。

b. 非聚集索引将索引和数据分开，索引的叶子节点指向数据的对应行，等于做了一个映射。索引顺序和物理顺序不一致。

c. 一个表只能又一个聚集索引，通常默认是主键。

d. 聚簇索引的顺序就是数据的物理存储顺序，非聚簇索引的索引顺序和物理顺序无关，因此只能由一个聚簇索引；在B+树中，聚簇索引的叶子节点就是数据节点，非聚簇索引的叶子节点仍然是索引节点，只不过由一个指针指向对应的数据块。

e. MyISAM索引文件和数据文件是分离的，索引文件仅保存数据记录的地址。而在InnoDB中，表数据文件本身就是按B+树组织的一个索引结构，这棵树的叶子节点的data域保存了完整的数据记录。

f. Innodb的二级索引和主键索引有很大不同，Innodb的二级索引的叶子节点包含主键值，而不是行指针（row pointers），减小移动数据维护二级索引的开销，不需要更新索引的行指针。（列值=索引值=主键值）

g. Innodb的二级索引叶子节点存放key字段+主键值，myisam存放的是列值与行号的组合。

h. 主索引是系统创建，二级索引是我们自己创建；主索引是表的主键；

====2020/08/13====

1. 进程和线程：
    a. 进程是程序的一次执行，是资源分配的基本单元；
    b. 线程是cpu调度和分派的基本单元。
2. 状态：
    a. 线程：创建，就绪，运行，阻塞，死亡；New，Runnable，Running，Blocked，Dead；
    b. 进程：创建，就绪，运行，阻塞，结束；
3. blocked，waiting，waiting-timed：
    a. 阻塞状态等待事件发生，转换为就绪状态；
    b. blocked：等待monitor lock；
    c. waiting：等待notify；
    d. waiting-timed：等待notify或者定时器数到0；
4. 文件描述符：fd（file descriptor）；linux中每打开一个文件，都有一个小的整数与之对应，就是文件描述符，0标准输入，1标准输出，2标准保存输出，<输入重定向符，>输出重定向符；
5. IO多路复用：通过一种机制，监视多个文件描述符，一旦某个文件描述符就绪，就能通知程序进行相应的IO操作。数据从内核到用户空间。
6. IO多路复用模型：
    a. 多个io复用一个线程；
    b. select：轮询；
    c. poll：最大连接数无上限；
    d. epoll：事件通知。
7. select：
    a. select函数，返回就绪的文件描述符fd的个数，找到就绪的文件描述符，保存在fdset中。每次调用select，需要轮询一遍fd，查看

就绪状态；且支持的最大fd数量有限，32位系统默认是1024；需要把fdset拷贝从用户态拷贝到内核态，开销大；

b. poll：不存在最大文件描述符限制；

c. epoll：事件通知的方式。包含epoll_create，epoll_ctl，epoll-wait三个方法；

i. epoll_create：创建一个句柄（类似于指针），占用一个fd；

ii. epoll_ctl：注册监听事件，把所有fd拷贝进内核一次，并为每一个fd指定一个回调函数，不需要每次轮询遍历fd；当fd就绪，会调用回调函数，把就绪的文件描述符和事件加入一个就绪链表，并拷贝到用户空间内存，应用程序不用亲自从内核拷贝。

iii. epoll_wait：监听epoll_ctl中注册的fd，在就绪链表中查看有没有就绪的fd，不用遍历fd。两种工作方式；

1. 水平触发：默认工作方式，epoll_wait检测到fd就绪，通知程序，不会立刻处理，下次epoll还会通知；

2. 边缘触发：epoll_wait通知会被立刻处理，下次不会通知；

8. 同步异步、阻塞非阻塞：

a. 同步异步：描述的是用户线程和内核的交互方式，同步是指线程发起io请求后需要等待或者轮询，内核io操作完成后才能继续执行；异步是指用户线程发起io请求后仍然继续执行，当内核io完成后会通知用户线程，或者调用用户线程注册的回调函数。

b. 阻塞非阻塞：描述用户线程调用内核io的方式：阻塞是指io操作彻底完成后才返回到用户空间；非阻塞指io操作被调用后，立刻返

回给用户一个状态值，无需等待io操作彻底完成。

9. io同步阻塞，同步非阻塞，多路复用：

a. 同步阻塞io：内核进行io操作时，用户线程阻塞；

b. 同步非阻塞：用户线程在发起io请求后立即返回，然后进行轮询，不断发起io请求，知道数据准备完成后，才正在进行io操作；

c. io多路复用：建立在select函数基础上，使用select函数避免同步非阻塞io的轮询等待过程；用户将需要io操作的socket添加到select中，线程阻塞等待select调用返回，当数据准备完成时，socket被激活，select函数返回，用户线程发起io请求，完成io操作。优势是用户可以在一个线程内同时处理多个socet的io请求。用户注册多个socket，不断调用select读取被激活的socket，同一线程同时处理多个io请求。

d. 异步io：内核读取数据，放在用户线程缓存区中，内核io完成后通知用户线程直接使用即可。

10. hashset和treeset：

a. hashSet是哈希表+红黑树，treeSet是红黑树；

b. hashSet允许空值，无序存储；自定义类实现treeSet，需要实现Comparable接口；

c. hashSet的add、remove、contains时间复杂度o(1)，treeSet是o(logn)；

11. 集合线程安全：vector、stack、hashtable、枚举；

12. 适用于查找的数据结构：

13. mysql高并发：

a. 代码中sql语句；

b. 数据库字段、索引；

c. 加缓存，redis/mencache；

d. 读写分离，主从复制；

e. 分区表；

f. 垂直拆分，解耦模块；

g. 水平切分；

14. 查找数据结构：

基于线性表的查找：

数组的顺序查找：O(1)

根据下标随机访问的时间复杂度为O(1);

二分查找：O(log2n)

分块查找：介于顺序查找和二分查找之间

跳跃链表：O(log2n)

基于树的查找：

二叉排序树：O(log2n)

平衡二叉树：O(log2n)

B-树：O(log2n)

B+树：O(log1.44n)

红黑树;

堆;

计算式查找：

哈希查找：O(1)

====2020/08/13====

1. 24点问题：
    a. what：4个数，加减乘除得到24，返回true，否则false。
2. 文本传输：
    a. 服务端：
        i. 创建服务器套接字并等待客户请求;
        ii. 收到请求并建立连接;
        iii. 按行读取客户端数据并写入到文件l;
        iv. 完成后向客户端发送响应。
    b. 客户端：
        i. 创建套接字;
        ii. 按行读取文本文件并发送;

====2020/08/12====

1. 适配器模式：

a. 定义：将一个类的接口转换成客户端需要的另一个接口，主要目的是**兼容性**，让原本接口不匹配的两个类协同工作。

b. 角色：目标接口，被适配者，适配器。

c. 通过适配器类，继承源角色，实现目标角色的接口，在适配器类中进行具体实现，达到适配的目的。

d. 类、对象、接口 适配器：

i. 类适配器：通过继承来实现，继承源角色，实现目标目标角色接口；

ii. 对象适配器：适配器拥有源角色实例，通过组合来实现适配功能，持有源角色，实现目标接口；

iii. 接口适配器：接口中有多个方法，用抽象类实现这个接口和方法，在创建子类时，只需要重写其中几个方法就行。

2. 装饰者模式：

a. 定义：以透明动态的方式来动态扩展对象的功能，是继承关系的一种代替方案。

b. 角色：抽象类，抽象装饰者，装饰者具体实现。

c. 一个抽象类有A方法，定义一个类作为抽象装饰者继承该抽象类，再创建具体装饰者类继承抽象装饰者类，并对其进行方法扩展，不用改变原来层次结构。

3. 适配器模式，装饰者模式，外观模式，区别：

a. 适配器模式将对象包装起来改变其接口；

b. 装饰者模式包装对象扩展其功能；

c. 外观模式保证对象简化其接口。

4. 代理模式：

a. what：**给某个对象提供一个代理对象，由代理对象控制该对象的引用。**

b. why：

i. 中介隔离作用，在客户类和委托对象之间，起到中介作用；

ii. 开闭原则，增加功能，对扩展开发，对修改封闭，给代理类增加新功能。

c. where：需要隐藏某个类，使用代理模式。

d. how：代理角色、目标角色、被代理角色，**静态代理，动态代理**；

i. 静态代理：

1. what：代理类创建实例并调用方法，在程序运行前，代理类已经创建好了；

2. why：

a. 优点：开闭原则，功能扩展；

b. 缺点：**接口发生改变，代理类也需要修改。**

3. where：需要代理某个类。

4. how：代理对象和被代理对象实现相同接口，通过调用代理对象的方法来调用目标对象。

ii. 动态代理：

1. what：程序运行时通过反射机制动态创建代理类；

2. why：

a. 优点：不需要继承父类，利用反射机制；

b. 缺点：目标对象需要实现接口。

3. where：代理某个类；

4. how：实现InvocationHandler接口，重写invoke方法，返回值时被代理接口的一个实现类。

iii. Cglib代理：

1. what：通过字节码创建子类，在子类中采用方法拦截来拦截父类的方法调用，织入横切逻辑，完成动态代理。

2. why：

a. 优点：不需要接口；

b. 缺点：对final无效。

3. where：不需要接口，代理。

a. SptingAOP中，加入容器的目标对象有接口，用动态代理；

b. 目标对象没有接口，用CGLib代理。

4. how：字节码。

5. 接口和抽象类：

a. 相同点：

i. 不能直接实例化；

ii. 包含抽象方法，则必须实现。

b. 不同点：

i. 继承extends只能支持一个类抽象类，实现implements可以实现多个接口；

ii. 接口不能为普通方法提供方法体，接口中普通方法默认为抽象方法；

iii. 接口中成员变量是public static final，抽象类任意；

iv. 接口不能包含构造器、初始化块。