# 猜想

总的来说所有的技术都是可以用mvc来大概分类的，如redis就是一存储工具🡪m

# 技巧

**类的使用：**jdk文档中有最佳实践、使用时注意的错误等。如ReentrantLock可重入锁类。以后可以去借鉴下。

# Mybatis：

SqlSessionFactoryBuilder从获取mybatis配置信息🡪生成SqlsessionFactory，SqlsessionFactory生成一个sqlSession🡪sqlSession获取mapperStatement对象，运行其映射的sql语句🡪执行后获得结果resultSet，resultSet有个处理器ResultSetHandler，对结果进行处理，包装成java对象 SqlSessionFactory：

生成sqlsession对象的工厂类，封装了执行sql的所有方法（selectList、selectOne等，所以可以自己获取该对象执行sql）。

Mapper类，其实就是接口。

命名空间：

全限定名调用（如<mapper namespace="org.mybatis.example.BlogMapper">）

，不会出现重名问题，ide也可以补全，可以在其他地方调用mapper。

猜测：

mybatis根据sqlsession操作sql，根据mapper获取sql语句。那要集成mybatis到项目，需要配置的东西：1、sqlSession用到的信息如数据库连接信息，驱动等，也许还有事务，2、mapper文件和xml文件的映射。

原理：

1、通过反射转换java类和sql语句。

如参数对象映射到sql中，resultSet对象映射到结果类。

# 时间复杂度：

耗时与数量增长的关系。如数量增长2n，O(n)的时间增长2n，即2倍；

**二分法**，查询时，每次能排除一半数据，log2N所以时间复杂度O(logn)。

# CAS 操作：

（类似乐观锁原理）保证线程安全。compare and swap的缩写，中文翻译成比较并交换。操作原来是：1、若其他线程正在修改，则操作失败。2、我认为该内存位置的值是A，若看到该值真的是A，就进行修改，否则就一直循环，直到看到值为A就修改。乐观锁原理，认为在它之前没线程操作该值。

缺点：

ABA问题：若开始是A，中间改成B,又改回A，这种情况无法预计。增加版本号才行。

若不成功则占用cpu，开销大。可优化一下。

只能同时保证一个共享变量的原子性。

# volatile关键字：

java告诉处理器，该变量经常变，强制将修改的值放入主内存。主存的更新使缓存中的值失效）。

# 注解的原理：

注解被编译后的本质就是一个继承Annotation接口（可以标记代码）的接口，java通过动态代理生成它的一个对象，之后就能操作它的属性了。

印象：注解其实就是一个接口。

# Shiro：

印象：shiro不应该是无权限就跳转登录页，否则人家登录了没权限也会跳转登录页面，应该是1、一开始访问登录页面，获取token。2、以后带上token访问即可。

# Slf4j日志：

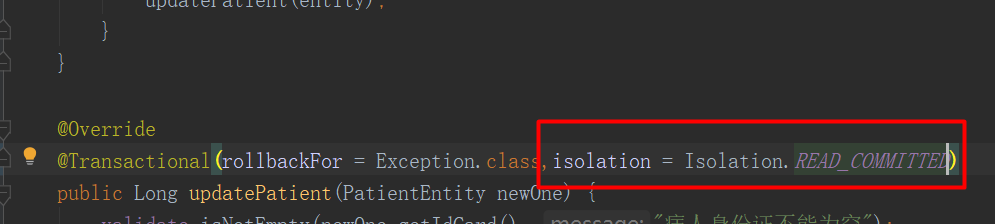
对项目输出日志的管理，控制其显示和保存的位置等。日志是知道系统状态的很重要的东西。

# Zookeeper：

1. 类似文件系统的结构，保存各个服务的信息
2. 给消费者提供服务的地址信息，消费者就可以访问了。
3. 投票机制保持数据一致性：
   1. Leader接收到修改一个值的提议，分发到每个服务中，每个服务进行投票，
   2. 超过半数同意，就通过，让所有服务按提议进行修改，保证一致性。
   3. 若leader挂了，则投票选举leader，为避免出现同票数导致zk挂掉，要求有奇数台机器进行投票。
4. Zk在选举后进行数据同步，集群越多同步的数据越多，同步失败又会进行选举，或者io压力大系统会挂掉。

# 数据库并发事务：

1. 存在的问题：
   1. 读未提交的数据🡪脏读
   2. 一个事务内读取两次，在两次中间被改掉了值，两次读的不一样🡪不可重复读
   3. 一个事务内，多次查询的记录行数不一样，多了或少了🡪幻读
2. 解决方法，事务隔离级别：
   1. DEFAULT，默认，一般是READ\_COMMITTED
   2. READ\_UNCOMMITED，可以读未提交的。不能解决上面的事务问题
   3. READ\_COMMITED，只能读已经提交的。解决脏读问题
   4. REPEATABLE\_READ，保证重复读取的结果一样，解决脏读，不可重复读问题。貌似会忽略事务执行过程中，记录的变化（如增加、删除行）。
   5. SERIALIZABLE，可序列化，事务挨个进行，解决上述所有事务问题，但是性能差。
3. 事务隔离级别的落地：
   1. Springboot中



* 1. 直接操作数据库连接对象



Mysql引擎：

1. 只有innodb支持事务。Myisam不支持事务。

# Mysql的innodb引擎主键索引：

主键索引是b+树结构，叶节点保存了所有的表数据。即索引和表数据不是分开的，表数据保存在索引叶节点中。

# mysql B+树做索引数据结构的落地形式:

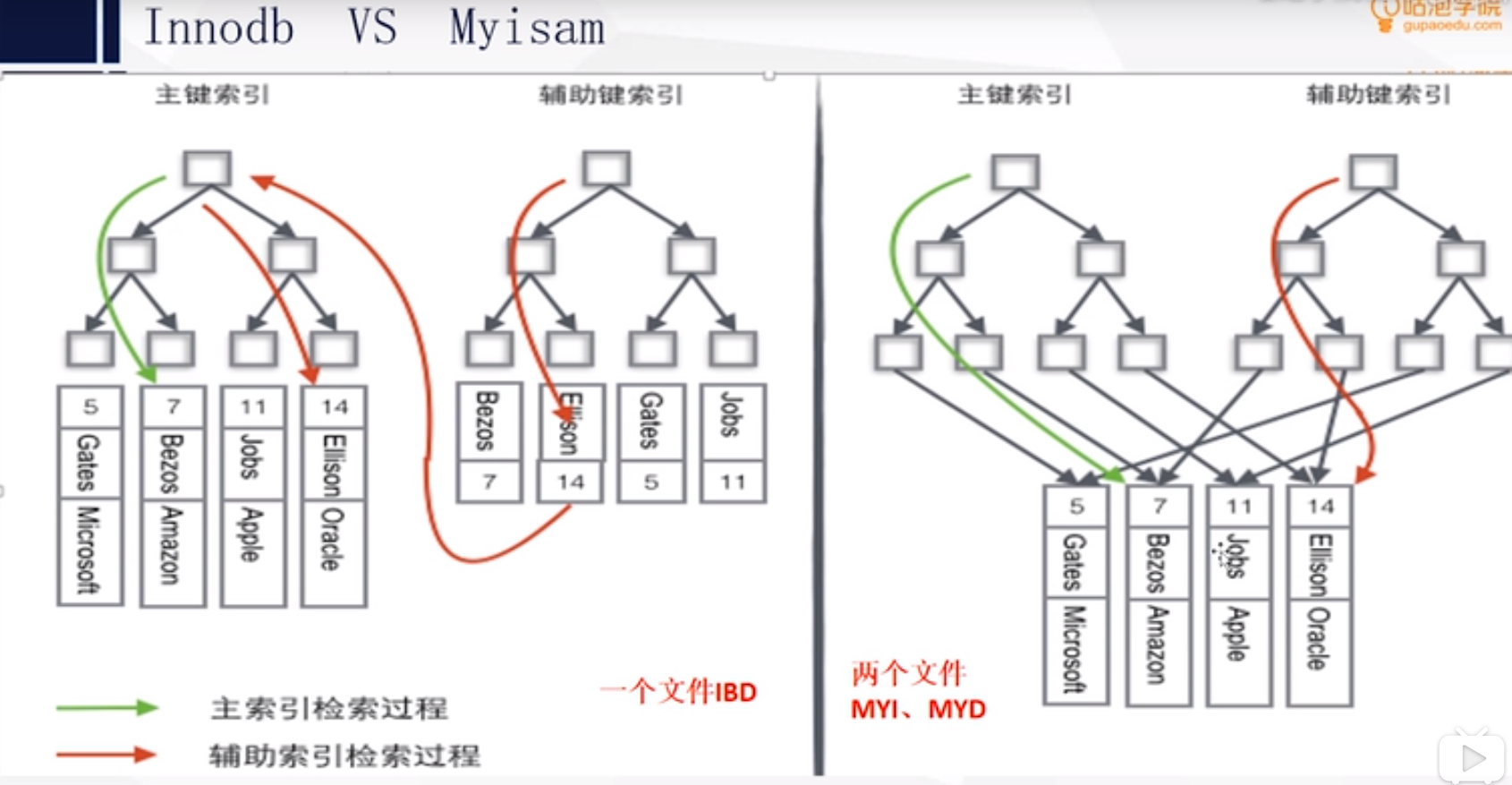
**myisam引擎：**

1. 索引文件为一个MYI文件，数据文件MYD文件
2. 主键索引与其他索引相同，无主次。

**innodb引擎：**

1. 主键索引属于聚集索引（物理地址顺序与树的叶节点的指针顺序一致）。
2. 其他索引都是辅助索引，叶节点存主键的值
   1. 不管主键地址怎么变化，不需要改变

3、只有一个IBD文件



B树结构的东西，离散性差的话，整个树都是一个值，不如不优化，所以重复值多的列不建索引。

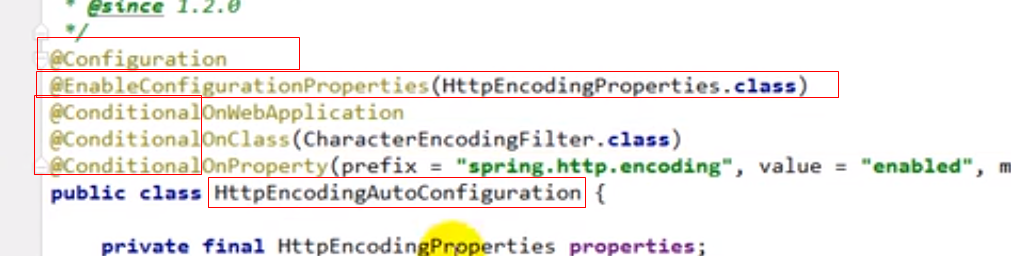
最左匹配原则：从左往右找。索引的值根据指定的编码规则转换，由小到大匹配。

联合索引：其实就是索引关键字为多个。如单索引：a；联合索引：acb；

思考：索引为acb，那搜索条件为dbc的话，第一个d没命中索引，所以要按照索引值顺序查询。

# SpringBoot自动配置：

1. 自动配置类一般类名是xxxAutoConfiguration;
2. 该类中的注解：
   1. @Configuration表明是配置类
   2. @EnbleConfigurationProperties指定配置文件类
   3. @ConditionalXXX表明是条件
   4. 我们能配置的属性，从配置文件类中拿，接上@ConditionalOnProperty指定的前缀，放入主配置文件中即可。



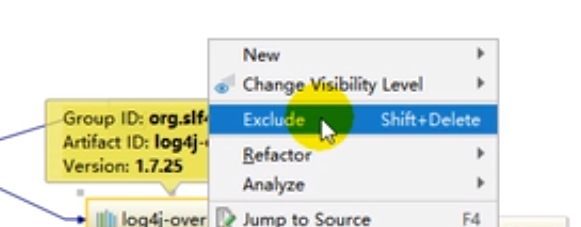
# SLF4J日志：

1、本身只是提供抽象层，针对不同的日志，导不同的日志实现包，以及设置各自日志实现对应的配置。

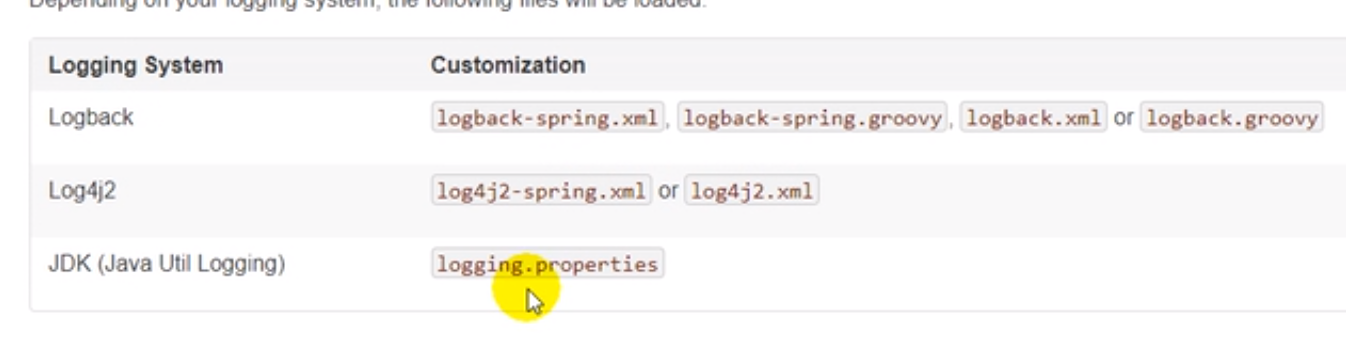
2、springboot要统一自己的日志框架，要排除掉springboot自己的日志框架，排除jar包，用日志适配包替换成自己的日志框架。

a) 排除方法：pom.xml中右键🡪Diagrams🡪show dependencies…🡪找到要排除的jar🡪右键🡪Excluds

b) 再在pom.xml中引入自己的jar即可



3、放自己的日志框架配置文件，简单，放指定名字的配置文件到类路径下即可



4、若使用的配置文件是XXX-spring.xml，可以使用springboot的高级功能，主配置文件用spring.profiles.active=dev，可以让他根据xml中的<springProfiles name=”dev”>标签选择输出格式。

# quartz定时任务框架：

1. 就是一个scheduler来管理job和trigger对象的东西。有各种定时方式。
2. 可持久化到数据库

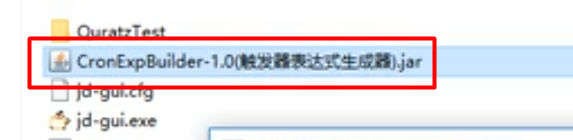
常用的触发器：

1、 simpletrigger，一个简单的触发器，适用于一定时间间隔重复执行这种简单任务。

2、 crontriggerr，根据日历触发，功能强大，如可以设置每个星期五执行等等，有自己的表达式实现复杂时间的设置。

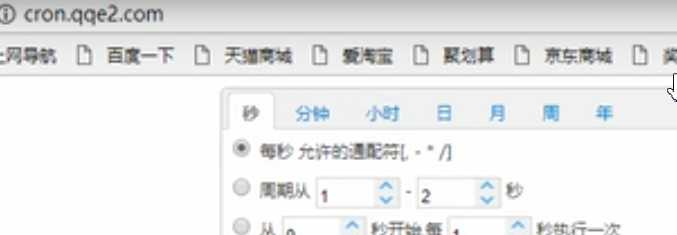
a) 该表达式有工具可以生成，无须自己拼

工具的jar包：





网上也有工具网站：



# 反射

在运行时操作类。（生成对象、调用方法等）

# 静态代理：

New class()等，会直接生成对象。

# OOP:

Object Oriented Programming,面向对象程序设计。

# sql和nosql区别

非关系型数据库，MongDB、 Redis、Memcache

关系型数据库和非关系型区别：

nosql：

优点

1. 简单易部署
2. 数据放内存中，读写快。
3. 支持key-value，文件等，key-value类型，数据无耦合，无须层层连接，查询快，易扩展。

缺点：

不支持事务。

sql：

优点：

1. 支持事务，保持数据一致性
2. 标准化
3. join等复杂查询

缺点：

1. 大量数据写入慢。

2、字段固定，扩展性差

# Redis：

Redis单线程，若数据库有几千万的数据，如果用了 keys \*会很消耗资源，导致其他服务失败，服务宕机，所以要禁用keys命令（在配置文件中的rename-mand，将keys命令重命名成其他单词等）

五种数据结构：

String（Map<String,String>）、list(Map<String,List<Stirng>>)、set(Map<String,Set<String>>)、sort\_set、hash(Map<String,Map<String,String>>)

在分库分表中，redis可以做id的自增长

SpringBoot提供了redis客户端：RedisTemplate

**RedisTemplate和StringRedisTemplate区别：**

RedisTemplate使用的是JdkSerializationRedisSerializer，存入数据（key和value）会先序列化成字节数组再存入Redis数据库，可读性非常差。但是如果你的数据是复杂的对象类型，而取出的时候又不想做任何的数据转换，直接从Redis里面取出一个对象，那么使用RedisTemplate是更好的选择。

 StringRedisTemplate（或者RedisTemplate<String, String>）使用的是StringRedisSerializer，序列化成Stirng再存入如Redis数据库，可读性好。

# SpringBoot：

简化spring使用

常用注解：

@PathVaribale 获取url中的数据

@RequestParam 获取请求参数的值

@GetMapping 组合注解，是@RequestMapping(method = RequestMethod.GET)的缩写

@RestController是@ResponseBody和@Controller的组合注解。

# SpringCloud：

基于http方式调用其他服务（dubbo是rpc方式），对调用的技术无限制，调用灵活。

**Eureka注册中心**

双层map方式存储服务信息，维护服务的地址信息，给客户端提供服务地址信息。

可以配置

**Hystrix豪猪**

解决雪崩问题（某服务访问量太多而卡主，导致系统崩溃）

服务降级：给非核心服务配置一定线程数，线程用完则等待，超时返回失败信息，可以配置一个方法在失败后访问。

服务熔断：类似电路漏电跳闸。某任务能判断是否有服务超时，超时则主动停止整个任务。但是过一段时间后hytrix会让它再次访问，超时再停止，循环。

# 遇见过的难点

1、我当时刚维护销售报表系统，就要用java重写一个销售报表，旧的是ruby写的，后来根据返回的数据猜它的实现，再跟进代码看细节，重写出来。

2、最常碰到的问题是，接手没有文档的项目，怎么梳理业务逻辑关系，代码关系？

起起服务，从页面上跑一条业务，同时后台跟踪走过的代码，跟踪数据库，跟踪数据走向，跟踪页面。

梳理整理业务逻辑，整理代码走向，整理数据走向，表间关系。

# ==和equals的区别

**==：**

引用类型比较首地址，基本数据类型比较值

**equals：**

比较两对象长得是否一样，默认用==实现。

每个类都有该方法，继承自object类，通常自己重写该方法，来比较对象是否相同。如String类就重写了equals方法。

与hashcode的联系：一般重写了equals方法，都要重写hashcode方法，以保证相同的对象有相同的哈希码，这是java标准。

# SpringMvc流程

DispatcherServlet( 前端控制器)拦截到请求，解析其地址，🡪解析后用handlermapping（处理器映射器，映射，即两元素的对应关系）找到对应的controller（处理器），然后调用handlerAdapter（处理器适配器，适配器设计模式），执行controller🡪执行后返回modelandview对象给前端控制器，前端控制器再用视图处理器处理后返回页面。

# Spring如何管理各个bean：

通过反射生成对象（先获取到依赖的其他对象存入一个数据结构中，依次生成，再获取到对象的生命周期方法，进行对象的初始化、实例化、销毁等。）🡪放入ioc容器🡪通过getBean（）等方法获取对象。

# 一个类Test的实例t中，两个方法A、B都加了锁，A调用B时有影响么：

调用类中synchronized修饰的方法时，锁对象是实例t，线程要获取到的是t的锁。而AB方法都需要获取到t的锁才能执行，所以A获取锁后调用B是没影响的。

# 线程安全，锁：

**同步代码块：**

Synchronized（锁对象）{代码。。。}

Synchronized（任意）{代码。。。}

锁对象：1、可以是任意对象。2、同一锁对象的线程会同步，反之不同步。

原理：貌似是使用监视器锁（monitor，其实现依赖互斥锁），在要同步的代码块前后加类似开始（monitorenter）和结束（monitorexit）的标记。jvm试着用无锁实现，不行再一级级直到重锁（互斥锁）实现。

**同步方法：**

Synchronized修饰方法

Synchronized（this）{代码。。。}

锁对象：this（就是new Test（）对象）

**静态同步方法**

Synchronized修饰静态方法

Synchronized（Test.class）{代码。。。}

静态方法在程序运行时就初始化了，比this对象早，所以其锁对象不是this。

锁对象：class文件对象

**锁机制：**

Jdk1.5出现的lock接口 （java.util.concurrent.locks）

使用方法：

1、创建一个lock 的实现类对象。2、在线程安全的代码前获取锁（t.lock()方法）.3、在需要线程安全的代码后调用，一般是try/catch的finally中，调用unlock（）方法释放锁。

# Wait和sleep的区别：

我的理解：

wait（2000），原理是释放锁（wait需要用到monitor,只能在synchronized中使用），让其他线程执行,需要让其他人唤醒自己。

Sleep是Thread对象本身要休眠，不涉及锁的释放。

1、wait是所有对象的方法，但是sleep是Thread的方法。---基于xxx对象的方法

2、sleep不是基于monitor而wait基于monitor（wait需要在synchronized中执行）。--用在何处

3、sleep不会释放锁（monitor），但是wait会释放锁且将锁加入到monitor等待队列—原理

4、sleep不需要唤醒，但是wait需要。--其他

异常捕获：

使用try（）catch{ e.printStack }

# 分布式id生成方案：

Twitter公司的雪花算法工具类

uuid

# Restful风格：

面向资源编程（数据表就是一个资源，如文章表article就是一个资源。），一种表示资源的方式，一种规则把。

如：



# 实现单例模式：

饿汉式：开始就创建。

懒汉式：用到再创建。

多线程下可能有多个实例

Synchronized修饰：性能不够

# 遇见过的难点

1、我当时刚维护销售报表系统，就要用java重写一个销售报表，旧的是ruby写的，后来根据返回的数据猜它的实现，再跟进代码看细节，重写出来。

2、最常碰到的问题是，接手没有文档的项目，怎么梳理业务逻辑关系，代码关系？

起起服务，从页面上跑一条业务，同时后台跟踪走过的代码，跟踪数据库，跟踪数据走向，跟踪页面。

梳理整理业务逻辑，整理代码走向，整理数据走向，表间关系。

# mysql乐观锁悲观锁实现：

**乐观锁：**

实现：update的时候条件加上版本号

update stock set count = count -1,version = version + 1 where version = " + stock.getVersion()

性能好，并发好。

不能防止脏读。

**悲观锁：**

事务加上版本号

实现：

（1）开启事务，即connection.setAutoCommit(false)（设置为手动提交）;。

（2）mybatis中使用Transaction注解：

@Transactional(propagation=Propagation.NESTED,isolation=Isolation.DEFAULT,readOnly = false,rollbackFor=RestException.class)

手动提交，手动回滚。这样，若事务中的sql语句用了索引列限定某些行时，会自动用行级锁锁定他们，没限定则用表锁。

# 集群、分布式和负载均衡：

集群：同一服务多个实例

分布式：不同服务不同实例

负载均衡：在集群中按照算法选择实例

nginx负载均衡算法：

1、轮询（默认）

2、weight权重

3、ip\_hash按ip的hash值选服务（这样用户会固定一台服务器，可解决session一致性）

4、fair响应时间短优先分配

5、url\_hash按url的hash值选服务（适合缓存放在后端服务器中）

# io与nio线程模型

# sql与nosql特性

sql特性，acid

atomicity 原子性（对于成功和失败来说 ，全部成功则成功，失败一个则全部失败）

consistency 一致性

isolation 隔离性

durability 持久性

nosql特性，cap（没有nosql能同时满足这三点）

consistency 强一致性

availability 可用性

partition tolerance 分区容错性