第十一周作业

一、导致系统不可用的原因有哪些？保障系统稳定高可用的方案有哪些？请分别列举并简述

引起故障的原因可能有硬件故障，软件bug，系统发布，并发压力，网络攻击，外部灾害等等。可以从如下几个方面去保障高可用：

1. 解耦：高内聚，低耦合组件设计原则；面向对象基本设计原则；面向对象设计模式；领域驱动设计建模

2. 隔离：

    1. 业务和子系统隔离：比如卖家系统和买家系统隔离

    2. 微服务与中台架构：业务拆了以后部署到不同的集群里面去

    3. 生产者消费者隔离：使用分布式消息队列拆分服务

    4. 虚拟机与容器隔离

3. 异步：多线程编程；反应式编程；异步通信网络编程；事件驱动异步架构

4. 备份：集群设计；数据库复制（CAP原理）

5. 失效转移：数据库主主失效转移；负载均衡失效转移；设计无状态服务

6. 幂等：多次操作和一次操作结果一致，用于失效故障转移时的重试。

7. 事务补偿：针对分布式事务，业务逻辑上去实现事务补偿机制，即通过执行业务逻辑的逆操作，回滚到操作前的状态。

8. 重试：超时以后进行重试的行为。上游调用的超时时间要大于下游多级服务的调用时间之和

9. 熔断：如果某个服务出现故障，响应延迟或者失败率增加，会使得调用者线程堵塞，进而出现级连失效，这种情况下使用断路器阻断对故障服务的调用。

10. 限流

    1. 计数器（滑动窗口）算法：每个固定窗口时间内记录并限制访问数（这个容易出现流量突刺），因此通过窗口滑动的形式，使得每个子窗口的请求数都受限，这样相对来说更为平滑

    2. 令牌桶算法：以固定的速度往一个桶中发送令牌，每个请求进来会领一个令牌

    3. 漏桶算法：以固定的速度往桶里加令牌，同时桶本身以固定的速度流出令牌，相对来说比令牌桶算法更均匀一些（不过感觉会导致很多请求拿不到令牌，也有问题的样子）

    4. 自适应限流：实时采集系统的性能指标，并根据PID算法去控制限流值

11. 降级：关闭一些非核心功能，降低访问压力

12. 异地多活：高可用架构的最高形式，光纤挖断了也能高可用。关键问题是数据的一致性。

二、请用你熟悉的编程语言写一个用户密码验证函数，Boolean checkPW（String 用户 ID，String 密码明文，String 密码密文），返回密码是否正确 boolean 值，密码加密算法使用你认为合适的加密算法

代码如下文，先把明文加入userId和盐值后进行加密，再用密文和数据库中的密文进行比较。

```
package com.test;

import java.security.MessageDigest;
import java.util.Objects;

public class PasswdCheck {
    private final String salt = "dsafsfqwefghyj";

    public Boolean checkPw(String userId, String origin, String encripted) {
        String checkStr = userId + origin + salt;
        try {
            MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
            byte[] hash = messageDigest.digest(checkStr.getBytes("UTF-8"));
            String handledStr = byteToHex(hash);
            return Objects.equals(handledStr, encripted);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    private String byteToHex(byte[] bytes) {
        StringBuffer stringBuffer = new StringBuffer();
        String temp = null;
        for (int i = 0; i < bytes.length; i++) {
            temp = Integer.toHexString(bytes[i] & 0xFF);
            if (temp.length() == 1) {
```

```
                //1得到一位的进行补0操作
                stringBuffer.append("0");
            }
            stringBuffer.append(temp);
        }
        return stringBuffer.toString();
    }
}
```