第十一周总结

课堂笔记：

11.1 安全架构：web攻击与防护

1. 最常见的攻击：XSS攻击，CSRF跨站脚本跨站点请求伪造，SQL注入攻击

    1. XSS攻击：攻击用户电脑，通过登陆用户的浏览器去目标站点发布恶意内容，其他点开这个恶意内容的用户也会执行这个恶意脚本，由此进行扩散。

        1. 防护手段：对一些字符标签进行转义，不允许上传脚本内容等。

        2. 用户可以提交内容的网站都需要进行XSS的消毒处理

    2. SQL注入攻击：用户提交的内容里面包含了一个SQL命令

        1. 获取数据库表结构信息的手段：

            1. 开源：如果服务器用了开源代码，这样攻击者可能很容易知道底层的设计和结构

            2. 错误回显：如果网站开启了错误回显，攻击者可以构造非法参数，并利用报错信息提取信息

            3. 盲注：不停地执行各种脚本，根据返回结果来猜，这种方式难度比较大

        2. 防护手段：插入各种不可见字符，使之不具有工具性（消毒）；SQL预编译；

    3. CSRF：跨站点请求伪造。用户登陆网站A -> 用户访问黑客网站B -> B的返回中包含了一个请求，去访问A

        1. 防护手段：

            1. 表单token：表单提交的参数，包含一个token，并且这个token只有目标用户的服务器才能构造出来。实际应用中还会拿用户的指纹，以及各种其他信息一起去计算加密token

            2. 要求输入验证码

            3. referer check：用得不多，效果有限

2. 其他需要注意的点：

    1. 关闭错误回显，error code等

    2. HTML注释（注释会显示在浏览器，响应中的HTML不建议加注释）

    3. 文件上传：防止用户上传可执行文件（文件类型控制，访问类型控制）

    4. 路径遍历：URL里面不要展示文件，目录等的路径

3. Web应用防火墙：通用，开源等

4. 网站安全漏洞扫描：自己攻击自己看看，这个也有工具

11.2 安全架构：加密与解密

1. 敏感信息加密可以在网站被攻破时起到保护信息的作用，黑客知道攻破了也得不到什么时，可能就不会去攻击了

2. 加密可分为三类：单项散列加密，对称加密，非对称加密。

1. 单项散列加密：单项加密，不可解密。用于不需要知道密码实际上是什么的情况，典型的应用是登陆的过程，只需要比较密文即可。使用中，需要加一点盐，防止黑客通过查表破解获取明文。

2. 对称加密：有一个密钥，明文可以通过它加密得到秘文，秘文也可以通过它解密得到铭文。

3. 非对称加密：和对称加密类似，但是加密和解密的密钥不同，并且算法复杂度貌似更高

1. 通过公钥加密，私钥解密，可以用来传递信息，比如HTTPS的验证过程，用户用公钥加密一个随机对称密钥，服务器用私钥解密得到对称密钥，之后就用对称密钥进行通信。

2. 通过私钥加密，公钥解密，可以起到签名的作用，比如A用私钥加密一段信息说A要转给B1个比特币，那么所有服务器都可以通过公钥解密得到A转给B一个比特币这个信息，并且确认它一定是A发起的，因为只有A有私钥

3. 密钥保护：把密钥分成许多片，放在不同的存储里面，每一种存储的管理员都不同

11.3 安全架构：反垃圾与风控

1. 反垃圾邮件：居然是分类模型。。。人工智能算法模型都用上了。。有点拼

2. 分类模型使用，教程中用了贝叶斯算法作为例子。手动标记分类 -> 训练 -> 对待分类邮件进行分类

3. 布隆过滤器：记录发送垃圾邮件的邮箱地址进行发垃圾，由于垃圾邮件量级及其巨大，所以用布隆过滤器进行粗筛，节省资源

4. 风控规则引擎：用于批量管理配置各种规则，用于识别各种有风险的行为。

5. 机器学习：通过模型去识别各种高风险的操作，交易等

11.4 高可用：可用性度量

1. 可用性分为三个部分：可用性的度量，可用性的方案，以及可用性的运维

2. 可用性用9的个数来评判，常见的qq可用性是4个9，即年度不可用时间小于53分钟，国内大部分牛逼的互联网应用差不多都是这个级别，金融类可能达到99.995%，99.997%，推特早期只有98%

3. 故障分级管理：不同服务权重不同，比如网站整体不可用权重100，核心功能不可用或不顺畅20，B类故障6，C类故障1

4. 故障分管理：故障换算成分数到每个人头上，并计入年终绩效考核

5. 引起故障的原因：硬件故障，软件bug，系统发布，并发压力，网络攻击，外部灾害

11.5 高可用：提升系统可用性的架构方案

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。

11.6 高可用：架构运维方案

1. 发布：启动新服务器 -> 注册负载均衡 -> 负载均衡中去掉旧服务器 -> 等一定时间后关闭旧服务器

2. 自动化测试：需要大量脚本和case，初期成本高，后期成本相对降低

3. 自动化部署：持续集成，持续交付，持续部署

4. 预发布验证：集群中存在一台特殊的服务器，不接入负载均衡，只有工程师可以访问到

5. 代码版本控制：保证系统可以正确回滚

6. 自动化发布、灰度发布

7. 网站运行监控：及时报警以及提供信息

8. 高可用价值观

    1. 系统简单，使问题易于发现，快速解决

    2. 目标明确，解决特定场景问题

    3. 价值回归，成本收益要合理

11.7 高可用故障案例分析

1. 故障1: 日志打爆了磁盘；可在日志配置文件中根据包名对不同的依赖进行日志级别的配置

2. 故障2: 首页请求打爆mysql；首页数据应该从缓存拿

3. 故障3: 多个方法（含不常用远程方法同时使用了synchronized关键字），导致程序时不时来一波超时。使用锁还是要谨慎的

4. 故障4: 缓存的疏于管理导致数据库被打爆

5. 故障5: apache+jboss架构，jboss没启动好，apache先启动了，导致请求打到未启动好的jboss，服务崩溃

6. 故障6: 不同大小的文件混用存储导致用户上传小文件时无限超时

7. 故障7: 压测占用带宽导致网络崩溃

8. 故障8: 注释掉读取缓存的代码进行测试，误发线上打爆数据库

9. 故障9: 新用户没有判空导致的长时间NPE

个人总结：

和题目一样，第十一周主要讲了安全和稳定这两个话题。前三节着重于安全话题，主要是关于web攻击和防护的各种手段，加解密的各种方式以及反垃圾，风控的各种方法。个人感觉讲得比较总括性，每个话题深入进去其实都是一个很深的话题。后四节着重讲可用性，分性能指标，提高性能的架构方案以及高可用的运维方案几个方面。这个相对来说也比较泛，对

照我个人平时忙碌的工作的内容，多多少少可以和自己所使用的一些东西对照，并进行归类，不过总感觉像这样简单过一遍还是无法真正做到运用自如啊。。