**职业逻辑**

能够去规划自己的个人发展的人，通常都是有很多机会和可能性的人。

有很多机会和可能性的人，通常都是有Leadership，喜欢冒险的人。

有Leadership喜欢冒险的人，通常都是学习能力强，思维活跃，喜欢折腾，懂得“投资”的人。

学习能力强思维活跃的人，通常来说，都是喜欢看书，喜欢实践和新鲜事物，不怕艰难和挑战，用智力而不是使蛮力的人。

懂得“投资”的人，通常来说，他们更多的关注的是未来和长远的成长，而不是当下的KPI、奖金和晋升。

**能力、机遇、运气、方向、努力。**

能力：

机遇：

运气：

方向：

努力：

口才：沟通、表达能力还行，主要是不会主动交际，害羞、胆子小、放不开

为什么那么害怕与人沟通，主动搭讪？

1. 害羞、要面子、脸皮薄
2. 担心自己普通话说不好
3. 觉得自己能力不行，害怕

接下怎么做，怎么去突破这个问题？

1. 主动搭讪
2. 多观察别人搭讪 方法、表情、谈话内容
3. 每天要记录下当天自己哪些表现好、哪些表现不好

口才练习记录表（2020/1/14）

1、

**论语十则**

子曰：“学而时习之，不亦说乎？有朋自远方来，不亦乐乎？人不知而不愠，不亦君子乎？”  
曾子曰：“吾日三省吾身：为人谋而不忠乎？与朋友交而不信乎？传不习乎？”  
子曰：“温故而知新，可以为师矣。”  
子曰：“学而不思则罔，思而不学则殆。”  
子曰：“由，诲女知之乎！知之为知之，不知为不知，是知也。”  
子曰：“见贤思齐焉，见不贤而内自省也。”  
子曰：“三人行，必有我师焉。择其善者而从之，其不善者而改之。”  
曾子曰：“士不可以不弘毅，任重而道远。仁以为己任，不亦重乎？死而后已，不亦远乎？”  
子曰：“岁寒，然后知松柏之后凋也。”  
子贡问曰：“有一言而可以终身行之者乎？”子曰：“其恕乎！己所不欲，勿施于人。”

**基础知识**

操作系统、计算机网络、编译原理、数据结构与算法、数据库

**团员转接**

地址：<https://zhtj.youth.cn/zhtj/>

账号：431382199510090193 密码：hanchun0806

账号：360321199708066023 密码：lisha1009

**博客园账号**

风继续吹0806/han-chun1009

**码云账号**

[yeywj@163.com/ywj1129](mailto:yeywj@163.com/ywj1129)

**码云账号/GitHub**

[1243636085@qq.com/hanchun1009](mailto:1243636085@qq.com/hanchun1009)

**学习路线**

（开源项目基本使用 -> 基础知识 -> 框架源码 -> 架构和功能的设计 -> 算法和数据结构 -> 架构师）

**项目一般流程**

调研阶段===>立项阶段===>启动阶段===>需求阶段===>产品设计===>系统设计===>系统开发===>冒烟测试===>SIT测试===>UAT测试===>回归测试===>上线投产===>生产试运行

**学习方法（思维很重要）**

被动学习：如听讲、阅读、视听、演示、学习内容的平均留存率为5%、10%、20%和30%

- 主动学习：如通过讨论、实践、教授给他人，会将原来被动学习的内容留存率从5%提升到50%、75%和90%

所以，学习不是努力读更多的书，**盲目追求阅读的速度和数量**，**这会让人产生低层次的勤奋和成长的感觉**，这只是在使蛮力。**要思辨，要践行，要总结和归纳**，否则，你只是在机械地重复某件事，而不会有质的成长的。

**如何快速适应新工作**

1. 主动和同事们打成一片，尽早做相互了解。
2. 主动向领导要公司系统业务相关的资料，遇到疑问及时向领导、同事提问并记录答案。
3. 主动要求领导安排讲解公司业务和系统。
4. 在了解公司业务和系统的同时做输出操作，用process画图工具画出业务流程图和数据库表之间的关系图，用云笔记及时记录接触的业务词汇和解释。
5. 分析系统框架，阅读关键业务的源码
6. 开完会主动和领导汇报适应新工作的情况和想法
7. 和同事沟通交流中分享信息资源，告知自己的能力和能提供的参考资源和参考方案

**如何培养解决问题的意识**

**识别问题**：5w2h

What：问题是什么

Why：为什么会发生问题

Who：谁造成的问题

When：何时发生的问题

Where：哪里的问题

How：问题是怎样发生的

How much：问题发生的频率，影响程度如何

**分析问题：**需要依靠专业能力或者经验来找到所有可能的原因，然后可以通过**冰山分析法**、**鱼骨法**来分析问题的根本原因，这一点也特别重要，很多时候如果识别不出根本原因，那么只会是治标不治本，当然对于一些紧急事故，解决事故是紧急首要的，后续的问题管理则需要找出问题的根本原因，以防止后续问题的重复发生。

**解决问题：**需要根据分析出来的问题原因，给出解决方案，这个也需要专业能力和经验的支撑，如果有多个方案则可以使用理性决策的**比较矩阵**和**决策矩阵**支撑最终的方案的选择。

**总结：**就是掌握解决问题的方法论，带着Owner意识多去实践解决问题，多去模范学习别人如何解决问题，多去总结沉淀成自己的知识体系和方法论形成闭环。

**日常感触**

1. 不能为了挣钱而挣钱，要有长远的想法，不能被眼前的景物迷惑，
2. 合理规划财产、要开始记账。

**别限制了自己**

**做有价值的事。**这个世界对计算机人才的要求是供不应求的，所以，不要让自己为自己找各式各样的借口，让自己活在“玩玩具”、“搬砖”和“使蛮力加班”的境地。其实，我发现这世界上有能力的人并不少，但是有品味的人的确很少。所谓的有价值，就是，别人愿付高价的，高技术门槛的，有创造力的，有颠覆性的……

**扩大自己的眼界，开放自己的内心。**人要变得开放，千万不要做一个狭隘的民族主义者，做一个开放的人，把目光放在全人类这个维度，不断地把自己融入到世界上，而不是把自己封闭起来，这里，你的英文语言能力对你能不能融入世界是起决定性的作用。开放自己的心态，正视自己的缺点，你才可能往前迈进。你的视野决定了你的知不知道要去哪，你的开放决定了你想不想去。

**站在更高的维度。**面的维度会超过点的维点，空间的维度会超过面的维度，在更高维度上思考和学习，你会获得更多。整天在焦虑那些低维度的事（比如自己的薪水、工作的地点、稳不稳定、有没有户口……），只会让你变得越来越平庸，只要你站在更高的维度（比如： 眼界有没有扩大、可能性是不是更多、竞争力是不是更强、能不能解决更大更难的问题、能创造多大的价值……），时间会让你明白那些低维度的东西全都不是事儿。技术学习上也一样，站在学习编程语法特性的维度和站在学习编程范式、设计模式的维度是两种完全不一样的学习方式。

精于计算得失。很多人其实不是很懂计算。绝大多数人都是在算计自己会失去多少，而不会算会得到多少。而一般的人也总是在算短期内会失去什么，优秀则总是会算我投入后未来会有什么样的回报，前者在算计今天，目光短浅，而后者则是舍在今天，得在明天，计算的是未来。精于计算得失的，就懂得什么是投资，不懂的只会投机。对于赚钱，你可以投机，但是对于自己最好还是投资。

勇于跳出传统的束缚。有时候，跳出传统并不是一件很容易的事，因为大多数人都会对未知有恐惧的心理。比如：我看到很多人才都被大公司垄断了，其实，有能力的人都不需要加入大公司，有能力的人是少数，这些少数的人应该是所有的公司share着用的，这样一来，对于所有的人都是利益最大化的。这样的事现在也有，比如：律师、设计师……。但是，绝大多数有能力的技术人员是不敢走出这步。我在2015年到2016年实践过一年半，有过这些实践，做“鸡”的比“二奶”好多了，收入也好很多很多（不好意思开车了）……

真正的企业级架构就不一样了，其中并不仅仅只是RESTful API或RPC，还有各种**配套设施和控制系统**，比如：应用**网关，服务发现、配置中心、健康检查、服务监控、服务治理（熔断、限流、幂等、重试、隔离、事务补偿）、Tracing监控、SOA/ESB、CQRS、EDA**……这些东西在非Java的技术栈体系内，很难看到全貌，Java强大的生态环境，就是让你把注意力放到更高层次的架构和业务上来的。（**千万不要觉得，整几个服务RPC一下，加个缓存，加个队列，就能叫架构，那只是系统集成罢了**）

**针对校招**

Java研发的话需要准备：

【1. 最重要】Java语言基础、Java并发、Java虚拟机、数据结构与算法、操作系统、网络协议、MySQL【2. 次重要】Spring（IOC、AOP、事务等）、Redis、MQ、设计模式

【3. 加分】分布式（Dubbo、分布式事务、分布式锁）等等

**关于创业**

创业家要关系三件事：第一，要关心生命周期；第二，要关心产业变革；第三，要关心宏观波动。

**科技赋能及创新**

1. 创新是手段，不是目的；
2. 技术创新需要极大降低成本、或者极大提升效率；
3. 要全栈，不只做技术提供商；
4. 要有很强的产品化能力；
5. 供应链和成本控制极其重要；
6. To B创业公司CEO都是首席销售。

maven中scope属性

Dependency Scope

在POM 4中，<dependency>中还引入了<scope>，它主要管理依赖的部署。目前<scope>可以使用5个值：

\* compile，缺省值，适用于所有阶段，会随着项目一起发布。

\* provided，类似compile，期望JDK、容器或使用者会提供这个依赖。如servlet.jar。

\* runtime，只在运行时使用，如JDBC驱动，适用运行和测试阶段。

\* test，只在测试时使用，用于编译和运行测试代码。不会随项目发布。

\* system，类似provided，需要显式提供包含依赖的jar，Maven不会在Repository中查找它。

依赖范围控制哪些依赖在哪些classpath 中可用，哪些依赖包含在一个应用中。让我们详细看一下每一种范围：

compile （编译范围）

compile是默认的范围；如果没有提供一个范围，那该依赖的范围就是编译范围。编译范围依赖在所有的classpath 中可用，同时它们也会被打包。

provided （已提供范围）

provided 依赖只有在当JDK 或者一个容器已提供该依赖之后才使用。例如， 如果你开发了一个web 应用，你可能在编译 classpath 中需要可用的Servlet API 来编译一个servlet，但是你不会想要在打包好的WAR 中包含这个Servlet API；这个Servlet API JAR 由你的应用服务器或者servlet 容器提供。已提供范围的依赖在编译classpath （不是运行时）可用。它们不是传递性的，也不会被打包。

runtime （运行时范围）

runtime 依赖在运行和测试系统的时候需要，但在编译的时候不需要。比如，你可能在编译的时候只需要JDBC API JAR，而只有在运行的时候才需要JDBC

驱动实现。

test （测试范围）

test范围依赖 在一般的编译和运行时都不需要，它们只有在测试编译和测试运行阶段可用。

system （系统范围）

system范围依赖与provided 类似，但是你必须显式的提供一个对于本地系统中JAR 文件的路径。这么做是为了允许基于本地对象编译，而这些对象是系统类库的一部分。这样的构件应该是一直可用的，Maven 也不会在仓库中去寻找它。如果你将一个依赖范围设置成系统范围，你必须同时提供一个 systemPath 元素。注意该范围是不推荐使用的（你应该一直尽量去从公共或定制的 Maven 仓库中引用依赖）。

在xml中配置 把日志打印出来

<!-- 系统日志输出 -->

<appender name="file\_info" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>${LOG\_PATH}/sys-info.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<!-- 按天回滚 daily -->

<fileNamePattern>${LOG\_PATH}/sys-info.%d{yyyy-MM-dd}.log</fileNamePattern>

<!-- 日志最大的历史 60天 -->

<maxHistory>60</maxHistory>

</rollingPolicy>

<encoder>

<pattern>${log.pattern}</pattern>

</encoder>

<filter class="ch.qos.logback.classic.filter.LevelFilter">

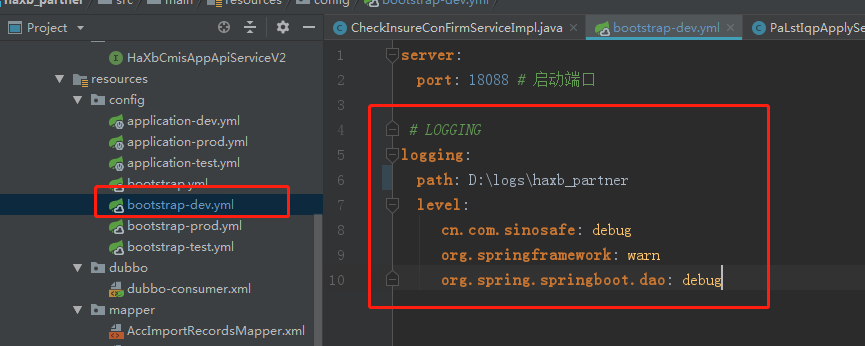
<level>INFO</level>

<onMatch>ACCEPT</onMatch>

<onMismatch>DENY</onMismatch>

</filter>

</appender>



基于token的鉴权机制

基于token的鉴权机制类似于http协议也是无状态的，它不需要在服务端去保留用户的认证信息或会话信息。这也就意味着机遇tokent认证机制的应用不需要去考虑用户在哪一台服务器登陆了，这就为应用的扩展提供了便利

流程

1、用户使用用户名密码请求服务器

2、服务器进行验证用户信息

3、服务器通过验证发送给用户一个token

4、客户端存储token，并在每次请求时附加这个token值

5、服务器验证token，并返回数据

这个token必须要在每次请求时发送给服务器，它应该保存在请求头中，另外，服务器要支持CORS（跨来源资源共享）策略，一般我们在服务端这么做就可以了 Access-Control-Allow-Origin：\*

JWT总结

优点：

1、因为json的通用性，所以JWT是可以跨语言支持的，像C#，JavaScript，NodeJS，PHP等许多语言都可以使用

2、因为由了payload部分，所以JWT可以在自身存储一些其它业务逻辑所必要的非敏感信息

3、便于传输，jwt的构成非常简单，字节占用很小，所以它是非常便于传输的

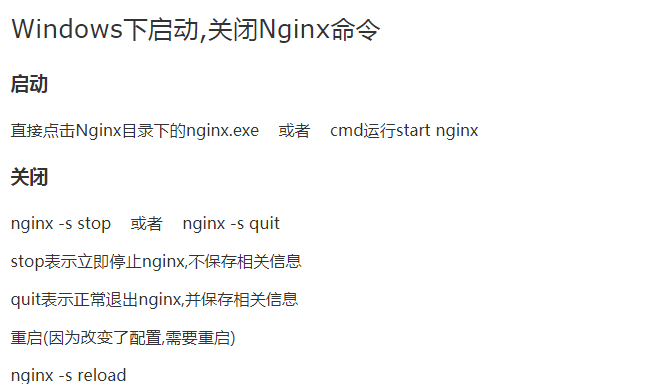
4、它不需要在服务端保存会话信息，所以它易于应用的扩展安全相关

5、不应该在jwt的payload部分存储敏感信息，因为该部分是客户端可解密的部分

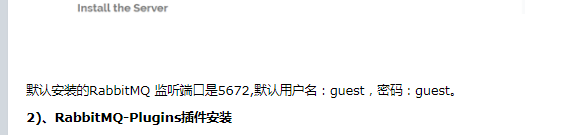
6、保护好secret私钥。该私钥非常重要

7、如果可以，请使用https协议

**Nginx笔记**



**Rabbitmq**



Windows 上关闭 打开 rabbitmq net stop RabbitMQ && net start RabbitMQ

**队列参数**

//声明队列

channel.QueueDeclare(

queue: QueueName, //队列名称

durable: false, //队列是否持久化.false:队列在内存中,服务器挂掉后,队列就没了;true:服务器重启后,队列将会重新生成.注意:只是队列持久化,不代表队列中的消息持久化!!!!

exclusive: false, //队列是否专属,专属的范围针对的是连接,也就是说,一个连接下面的多个信道是可见的.对于其他连接是不可见的.连接断开后,该队列会被删除.注意,不是信道断开,是连接断开.并且,就算设置成了持久化,也会删除.

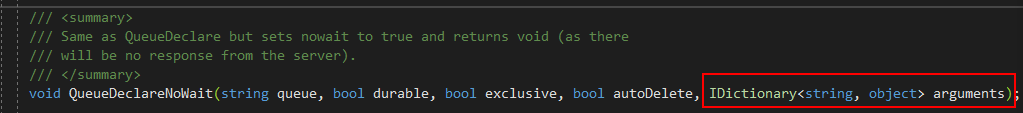
autoDelete: true, //如果所有消费者都断开连接了,是否自动删除.如果还没有消费者从该队列获取过消息或者监听该队列,那么该队列不会删除.只有在有消费者从该队列获取过消息后,该队列才有可能自动删除(当所有消费者都断开连接,不管消息是否获取完)

arguments: null //队列的配置

);

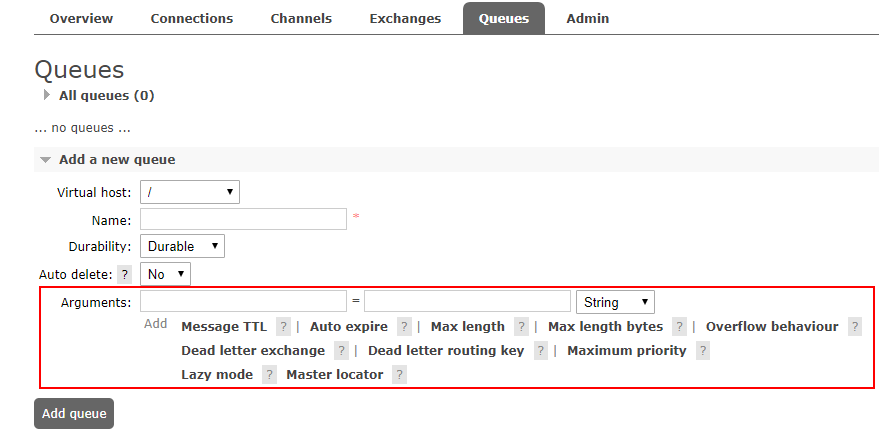
对于第5个参数: arguments ,

它的类型是一个键值对集合 :



它到底有哪些key呢?

我们可以通过 RabbitMQ 的管理页面看到:



一共10个:

* Message TTL : 消息生存期
* Auto expire : 队列生存期
* Max length : 队列可以容纳的消息的最大条数
* Max length bytes : 队列可以容纳的消息的最大字节数
* Overflow behaviour : 队列中的消息溢出后如何处理
* Dead letter exchange : 溢出的消息需要发送到绑定该死信交换机的队列
* Dead letter routing key : 溢出的消息需要发送到绑定该死信交换机,并且路由键匹配的队列
* Maximum priority : 最大优先级
* Lazy mode : 懒人模式
* Master locator :

Redis



常用的redis服务命令。

卸载服务：redis-server --service-uninstall

开启服务：redis-server --service-start

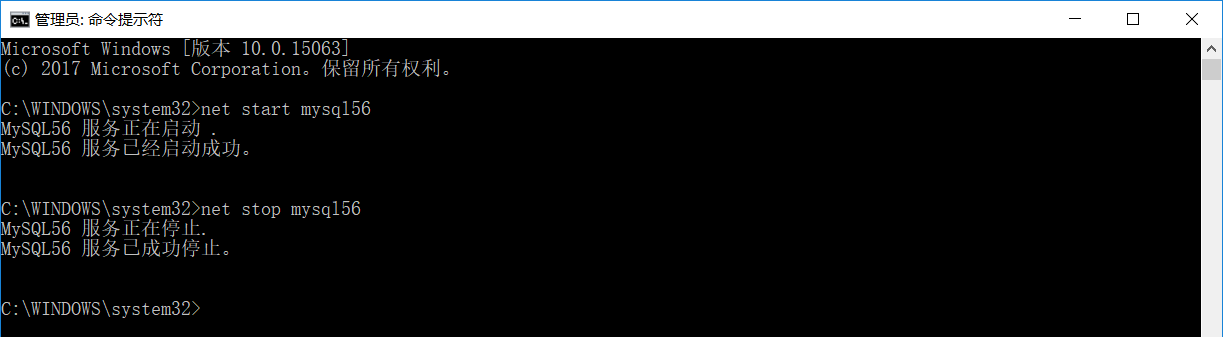
停止服务：redis-server --service-stop

**Mysql**

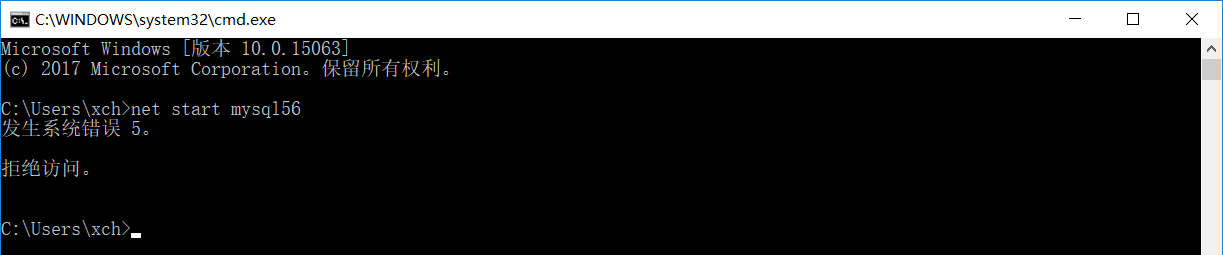
“Windows+R”-->输入“cmd”-->Enter，在DOS窗口命令提示符下输入：net start mysql56 启动MySQL服务器（mysql80是本机mysql的服务名）



输入：net stop mysql56 停止MySQL服务器



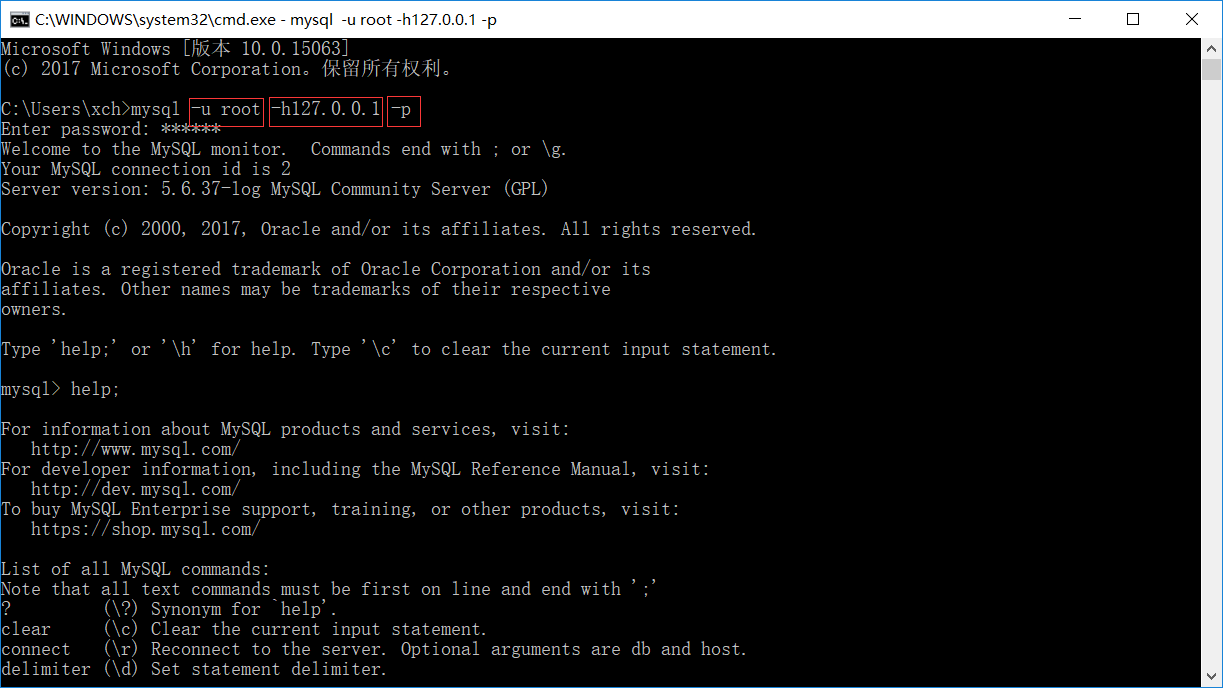
注：这里可能会遇到 “发生系统错误5 拒绝访问” 的错误，这是因为操作的权限不足，需要以管理员身份运行。



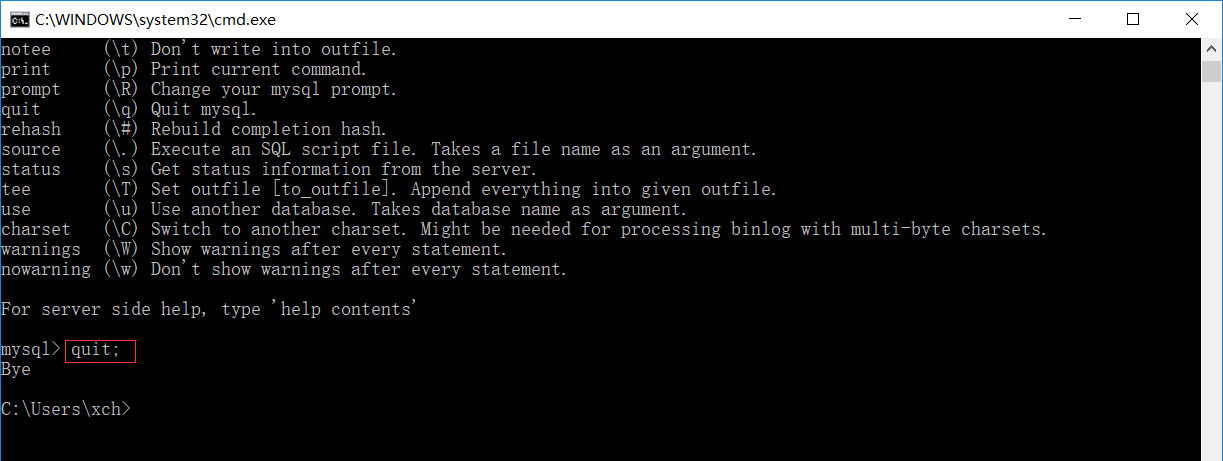
具体解决方案：http://www.cnblogs.com/xch-yang/p/7681554.html

连接和断开MySQL服务器

1. 连接MySQL服务器：输入mysql -u root -h127.0.0.1 -p，然后输入密码，Enter。（如下图，三个框依次是：用户名、MySQL服务器所在地址、用户密码）本机密码root



断开MySQL服务器：输入 exit; 或 quit;



**Node.js 中npm和 cnpm初步了解**

npm（node package manager）：nodejs的包管理器，用于node插件管理（包括安装、卸载、管理依赖等）

cnpm:因为npm安装插件是从国外服务器下载，受网络的影响比较大，可能会出现异常，如果npm的服务器在中国就好了，所以我们乐于分享的淘宝团队干了这事。来自官网：“这是一个完整 npmjs.org 镜像，你可以用此代替官方版本(只读)，同步频率目前为 10分钟 一次以保证尽量与官方服务同步。”

如果在cmd中输入命令，提示不是内部或外部命令，也不是可运行的程序或批处理文件，就要找到这个命令或者文件在哪个目录，然后输入cmd 再执行命令，或者把路径配置到环境变量的系统变量中。就可以直接运行。

**杀死进程**

Windows： ntsd -c q -p PID /taskkill /pid 1692 /f

Linux：查看端口占用netstat -aon|findstr "8088"，然后杀死进程taskkill /pid 4 -F

**Spring缓存注解@Cacheable、@CacheEvict、@CachePut使用**

 @Cacheable

可以标记在一个方法上，也可以标记在一个类上。当标记在一个方法上时表示该方法是支持缓存的，当标记在一个类上时则表示该类所有的方法都是支持缓存的。对于一个支持缓存的方法，Spring会在其被调用后将其返回值缓存起来，以保证下次利用同样的参数来执行该方法时可以直接从缓存中获取结果，而不需要再次执行该方法。

@CachePut

在支持Spring Cache的环境下，对于使用@Cacheable标注的方法，Spring在每次执行前都会检查Cache中是否存在相同key的缓存元素，如果存在就不再执行该方法，而是直接从缓存中获取结果进行返回，否则才会执行并将返回结果存入指定的缓存中。@CachePut也可以声明一个方法支持缓存功能。与@Cacheable不同的是使用@CachePut标注的方法在执行前不会去检查缓存中是否存在之前执行过的结果，而是每次都会执行该方法，并将执行结果以键值对的形式存入指定的缓存中。

@CacheEvict

是用来标注在需要清除缓存元素的方法或类上的。当标记在一个类上时表示其中所有的方法的执行都会触发缓存的清除操作。@CacheEvict可以指定的属性有value、key、condition、allEntries和beforeInvocation。其中value、key和condition的语义与@Cacheable对应的属性类似。即value表示清除操作是发生在哪些Cache上的（对应Cache的名称）；key表示需要清除的是哪个key，如未指定则会使用默认策略生成的key；condition表示清除操作发生的条件。下面我们来介绍一下新出现的两个属性allEntries和beforeInvocation。

@AutoConfigureBefore

@AutoConfigureBefore(Xxx.class)

此注解用在类名上，标识在加载Xxx类前加载该配置类

**lombok几个基本注解的使用**

@Data  
使用这个注解，就不用再去手写Getter,Setter,equals,canEqual,hasCode,toString等方法了，注解后在编译时会自动加进去。

@AllArgsConstructor  
使用后添加一个构造函数，该构造函数含有所有已声明字段属性参数

@NoArgsConstructor  
使用后创建一个无参构造函数

@Builder  
关于Builder较为复杂一些，Builder的作用之一是为了解决在某个类有很多构造函数的情况，也省去写很多构造函数的麻烦，在设计模式中的思想是：****用一个内部类去实例化一个对象，避免一个类出现过多构造函数，****

@NoArgsConstructor, @RequiredArgsConstructor, @AllArgsConstructor

注解在类上, 为类提供无参,有指定必须参数, 全参构造函数

@SneakThrows

提供抛出异常

@Synchronized

注解在方法上, 为方法提供同步锁

@Log

打印日志

@Log4j

解在类上, 为类提供一个属性名为 log 的 log4j 的日志对象

@Slf4j

注解在类上, 为类提供一个属性名为 log 的 log4j 的日志对象

@Accessors

Accessor的中文含义是存取器，@Accessors用于配置getter和setter方法的生成结果，下面介绍三个属性

fluent

fluent的中文含义是流畅的，设置为true，则getter和setter方法的方法名都是基础属性名，且setter方法返回当前对象。

@Data

@Accessors(fluent = true)

public class User {

private Long id;

private String name;

// 生成的getter和setter方法如下，方法体略

public Long id() {}

public User id(Long id) {}

public String name() {}

public User name(String name) {}

}

chain

chain的中文含义是链式的，设置为true，则setter方法返回当前对象。如下

@Data

@Accessors(chain = true)

public class User {

private Long id;

private String name;

// 生成的setter方法如下，方法体略

public User setId(Long id) {}

public User setName(String name) {}

}

prefix

prefix的中文含义是前缀，用于生成getter和setter方法的字段名会忽视指定前缀（遵守驼峰命名）。如下

@Data

@Accessors(prefix = "p")

class User {

private Long pId;

private String pName;

// 生成的getter和setter方法如下，方法体略

public Long getId() {}

public void setId(Long id) {}

public String getName() {}

public void setName(String name) {}

}

**构造函数**

简单地说构造函数是类函数，函数名与类名完全相同，无返回值  
构造函数属于类的成员函数，除了具有一般成员函数的特点外，还具有自己的特点：  
（1）是类的一个特殊的成员函数，函数名与类名相同；  
（2）访问属性应该是public；  
（3）功能是初始化对象，在构造函数中一般不作初始化以外的事情；  
（4）可以在类内定义，也可在类外定义；  
（5）无函数返回类型，注意：什么也不写，也不可写为void；  
（6）程序运行时，当对象被创建后，该对象所属的类的构造函数自动被调用，在该对象生存期中也只调用这一次；  
（7）可以重载

**Springboot注解记录**

**@Controller和@RestController的区别？**

1)如果只是使用@RestController注解Controller，则Controller中的方法无法返回jsp页面，配置的视图解析器InternalResourceViewResolver不起作用，返回的内容就是Return 里的内容。

例如：本来应该到success.jsp页面的，则其显示success.

@RestController注解相当于@ResponseBody ＋ @Controller合在一起的作用。

2)如果需要返回到指定页面，则需要用 @Controller配合视图解析器InternalResourceViewResolver才行。

3)如果需要返回JSON，XML或自定义mediaType内容到页面，则需要在对应的方法上加上@ResponseBody注解。

**@Configuration**

从Spring3.0，@Configuration用于定义配置类，可替换xml配置文件，被注解的类内部包含有一个或多个被@Bean注解的方法，这些方法将会被AnnotationConfigApplicationContext或AnnotationConfigWebApplicationContext类进行扫描，并用于构建bean定义，初始化Spring容器。

**@controller、@service、@repository、@component**

1、@controller 控制器（注入服务）

2、@service 服务（注入dao）

3、@repository dao（实现dao访问）

4、@component （把普通pojo实例化到spring容器中，相当于配置文件中的<bean id="" class=""/>）

　 @Component,@Service,@Controller,@Repository注解的类，并把这些类纳入进spring容器中管理。

下面写这个是引入component的扫描组件

<context:component-scan base-package=”com.mmnc”>

其中base-package为需要扫描的包（含所有子包）

1、@Service用于标注业务层组件

2、@Controller用于标注控制层组件(如struts中的action)

3、@Repository用于标注数据访问组件，即DAO组件.

4、@Component泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注。

**@ConfigurationProperties**

在编写项目代码时，我们要求更灵活的配置，更好的模块化整合。在 Spring Boot 项目中，为满足以上要求，我们将大量的参数配置在 application.properties 或 application.yml 文件中，通过 @ConfigurationProperties 注解，

**Spring注解之 @SuppressWarnings注解**

简介：java.lang.SuppressWarnings是J2SE5.0中标准的Annotation之一。可以标注在类、字段、方法、参数、构造方法，以及局部变量上。

作用：告诉编译器忽略指定的警告，不用在编译完成后出现警告信息。

使用：

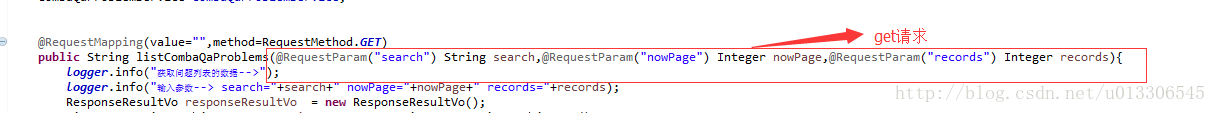
@SuppressWarnings(“”)

@SuppressWarnings({})

@SuppressWarnings(value={})

**@RequestParam和@RequestBody和@PathVariable区别**

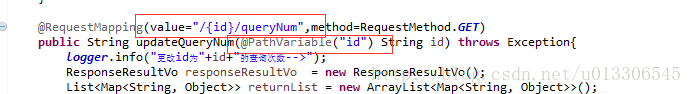
**get请求**



**post请求**

IMG_256

还有一种应用场景，接口规范为resultful风格时，举个例子：如果要获取某个id下此条问题答案的查询次数的话，则后台就需要动态获取参数，其注解为@PathVariable，并且requestMapping中的value应为value="/{id}/queryNum"，截图如下



springCloud注解记录

**idea插件**

1. Findbugs

**去提升看问题的高度(重点)**

如果以普通的视角去看，那么一颗树那也就只是一棵树而已，但是如果跳脱出目前的视角，站在更高的角度去看，它其实是森林的一部分。

把这个问题说得实际点：

假设今天你负责的是一个系统，那么你仅仅是把这个系统的基本原理搞懂了？还是可以把上下游有几个系统、每个系统之间如何调用、依赖方式都理顺？

假设今天你负责的是一块业务，那么你仅仅把自己负责的功能点弄清楚了？还是你可以从最上游开始，到你负责的系统，再到最下游，都思考得非常透彻？

眼界决定高度，多看、多想、多保持好奇心、多问几个为什么。

我认为大到项目、小到需求，如果在完成之后缺乏总结，那么某种程度上来说是失败的，可以总结的点非常多：

1、通过这个项目/需求，是否吃透了某一块业务，搞懂了来龙去脉。

2、通过这个项目/需求，是否充分理解了公司某个技术框架/基础组件的用法。

3、在整个项目的设计上，有哪些做的不好的地方。

4、在整个项目的开发（针对程序员而言），是否踩了坑，犯了低级的错误。

5、在整个项目的进度把控上、人员安排上、上下游协调上，是否存在不足之处。

6、经历了某次大促的值班，是否对可以熟练使用公司的监控工具，遇到突发事件，是否快速有效地进行了解决。

**SpringCould学习**

UriComponentsBuilder.fromUriString("http://HELLO-SERVICE/sayhello?name={name}").build().expand("王五").encode()参数“王五”加密处理,如下所示：

<http://HELLO-SERVICE/sayhello?name=%E7%8E%8B%E4%BA%94>，springcould会自己解密。

**负载均衡**

负载均衡有好几种实现策略，常见的有：

随机 (Random)

轮询 (RoundRobin)

一致性哈希 (ConsistentHash)

哈希 (Hash)

加权（Weighted）

**服务续约**

在注册完服务之后，服务提供者会维护一个心跳来不停的告诉Eureka Server：“我还在运行”，以防止Eureka Server将该服务实例从服务列表中剔除，这个动作称之为服务续约，和服务续约相关的属性有两个，如下：

eureka.instance.lease-expiration-duration-in-seconds=90

eureka.instance.lease-renewal-interval-in-seconds=30

第一个配置用来定义服务失效时间，默认为90秒，第二个用来定义服务续约的间隔时间，默认为30秒。

**Maven实战--dependencies与dependencyManagement的区别**

dependencies即使在子项目中不写该依赖项，那么子项目仍然会从父项目中继承该依赖项（全部继承）

dependencyManagement里只是声明依赖，并不实现引入，因此子项目需要显示的声明需要用的依赖。如果不在子项目中声明依赖，是不会从父项目中继承下来的；只有在子项目中写了该依赖项，并且没有指定具体版本，才会从父项目中继承该项，并且version和scope都读取自父pom;另外如果子项目中指定了版本号，那么会使用子项目中指定的jar版本。

**Git 命令**

Git branch查看所有分支

git branch <分支名>命令来创建一个分支

git checkout <分支名>来切换分支

如果觉得这样太麻烦，可以通过git checkout -b <分支名>来一步到位，创建并切换分支

git checkout -命令来切换回上一个分支

git merge --no-ff <分支名>命令将fa分支合并到master分支上。其中—no-ff表示强行关闭fast-forward方式，fast-forward方式表示当条件允许时，git直接把HEAD指针指向合并分支的头，完成合并，这种方式合并速度快，但是在整个过程中没有创建commit，所以如果当我们删除掉这个分支时就再也找不回来了，因此在这里我们将之关闭。

git log --graph命令来直观的查看分支的创建和合并等操作

**Git如何撤销merge操作**

方法一，reset 到 merge 前的版本，然后再重做接下来的操作，要求每个合作者都晓得怎么将本地的 HEAD 都回滚回去：

**$ git checkout 【行merge操作时所在的分支】**

**$ git reset --hard 【merge前的版本号】**

方法二，当 merge 以后还有别的操作和改动时，git 正好也有办法能撤销 merge，用

**git revert：$ git revert -m 【要撤销的那条merge线的编号，从1开始计算（怎么看哪条线是几啊？）】 【merge前的版本号】**

Finished one revert.

[master 88edd6d] Revert "Merge branch 'jk/post-checkout'"

1 files changed, 0 insertions(+), 2 deletions(-)

这样会创建新的 commit 来抵消对应的 merge 操作，而且以后 git merge 【那个编号所代表的分支】 会提示：

Already up-to-date.

因为使用方法二会让 git 误以为这个分支的东西都是咱们不想要的。

https://mp.weixin.qq.com/s?\_\_biz=MzI1NDY0MTkzNQ==&mid=2247484207&idx=1&sn=c1b162fd66a41cda4072b11c3750ae5a&scene=21#wechat\_redirect

**Lombok**

<!--Lombok--> 需要引入pom文件

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<scope>provided</scope>

</dependency>

**linux服务器查看项目日志命令**

1、tailf mywork.log | grep --line-buffered findUserList 实时跟踪日志，这里是只要findUserList 这个方法被运行，就会将它的日志打印出来，用于跟踪特定的日志运行。 --line-buffered 是一行的缓冲区，只要这一行的缓冲区满了就会打印出来，所以可以用于实时监控日志。

2、 tailf -n 500 mywork.log 打印最后500行日志，并且持续跟踪日志。

　　tail -n 2000 mywork.log | more 分页查看最后2000行日志。

3、 grep '调用远程服务运行结果是' mywork.log | more 将有关 '调用远程服务运行结果是' 字符串的结果都打印出来，并且是分页打印，用于日志太多的情况。用空格翻页。

4、 cat mywork.log | grep '查看前后100行' -C 100

5、 cat mywork.log | grep '查看前100行' -B 100

6、 cat mywork.log | grep '查看后100行' -A 100

7、按照行号查看日志

wc -l mainCms.log 显示文件一共有多少行。

cat -n mywork.log | tail -n +92|head -n 20 表示先查看92行之后的日志，然后在看这92行之后的日志的前20行。也就是查看92到112行之间的日志。

sed -n '5,10p' mywork.log 查看5行到10行的日志。

8、按日期查看日志

sed -n '/2018-05-26 17:07:00/,/2018-05-26 17:06:59/p' mywork.log

sed -n '/2018-08-16 18:/p' mainCms.log | less -mN

1. grep -E '123|abc' mywok.log 找出包含123或者abc的行。

**抓包工具**

抓包工具推荐:LINUX环境下用TCPDUMP,windows环境下用 wireshark

# 利用git标签回退至任意版本

通过git tag命令来查看当前有哪些tag

通过git show v20180107命令来查看对应的版本号

commit后面的就是对应的提交版本号，然后通过git reset --hard 5eb9cf18a命令回到想要的版本

命令 git tag <tagname> 用于新建一个标签，默认为HEAD，也可以指定一个commit id；

命令 git tag -a <tagname> -m "this is memeo" 可以指定标签信息；

命令 git tag 可以查看所有标签。

命令 git log --pretty=oneline --abbrev-commit 找到历史提交的commit id

命令 git tag -d <tagname> 删除标签

命令 git push origin :refs/tags/<tagname> 可以删除一个远程标签。

**sql如何获取相同字段的第一条数据（oracle）**

select \* from (select\* from tb\_apply\_loan\_checkup where C\_ORDER\_ID = #{OrderId} order by t\_crt\_tm desc) where rownum=1

**创建一个视图**

CREATE OR REPLACE VIEW INSERT\_ACCFEEMTDPLAN\_VIEW AS 后面加查询结果

**方法名加注释**

<https://blog.csdn.net/xiaoliulang0324/article/details/79030752>

**理解面向对象与面向过程**

1.用shell实现自动化脚本做的服务编排，一般都是面向过程，一步一步的。而k8s的编排却是面向对象的，因为它为这个顺序流抽象出了很多角色，将原本一步一步的顺序操作转变成了多个角色间的轮转和交互。

2.所谓为什么“充血模型”不流行，我认为不外呼两个。一，规范的领域模型对于底层基础架构来说并不友好（缺少setget），所以会导致规范的领域模型与现有基础架构不贴合，很难开发出完全贴合的基础架构，进而引深出合理的业务封装却阻碍关于复用通用抽象的矛盾。二，合理的业务封装，需要在战略上对业务先做合理的归类分割和抽象。而这个前置条件很少也不好达成。进而缺少前置设计封装出来的“充血模型”会有种四不像的味道，反而加剧了业务的复杂性，还不如“贫血模型”来得实用。事实上快节奏下，前置战略设计往往都是不足的，所以想构建优秀的“充血模型”架构，除了要对业务领域和领域设计有足够的认知，在重构手法和重构意愿上还要有一定讲究和追求，这样才能让项目以“充血模型”持续且良性的迭代。

3.“充血模型”相对于“贫血模型”有什么好处？从我的经验来看，可读性其实可能“贫血模型”还好一点，这也可能有思维惯性的原因在里面。但从灵活和扩展性来说“充血模型”会优秀很多，因为好的“充血模型”往往意味着边界清晰（耦合低），功能内敛（高内聚）。

MVC这种框架模式本质上与面向对象并不冲突。当我们在讨论面向对象的时候，我们究竟应该怎样去定义一个对象，究竟什么才能被我们看成是对象，是不是只有像某种物体，比如说一只鸟或者一只狗我们才能去把他定义为对象？我认为，MVC里面的三个部分Model 、Controller 、View 我们都能把他们单独的看成一个对象，比如说Model，本来它是数据单元，但是如果我们把他看做一个对象的话，里面存储的数据不就是我们对象里的属性么，而对于数据的二次加工处理等等操作不就是对象里的方法么？同理，对于View而言，里面小的view组件或者是其他的view不就是我们对象里面的属性，而对于不同的view组件或其他view的组合或者其他的处理操作不就是对象里面的方法么？所以说，不必死抠定义，数据就一定要和业务逻辑组成一个类云云。。。我们最后写出来的代码的目的就是，1.要解决问题；2.代码有可扩展性，可读性；3，代码解耦。

**Docker理解**

**更高效的利用系统资源：**由于容器不需要进行硬件虚拟以及运行完整操作系统等额外开销，Docker 对系统资源的利用率更高。无论是应用执行速度、内存损耗或者文件存储速度，都要比传统虚拟机技术更高效。因此，相比虚拟机技术，一个相同配置的主机，往往可以运行更多数量的应用。

**更快速的启动时间：**传统的虚拟机技术启动应用服务往往需要数分钟，而Docker 容器应用，由于直接运行于宿主内核，无需启动完整的操作系统，因此可以做到秒级、甚至毫秒级的启动时间。大大的节约了开发、测试、部署的时间。

**一致的运行环境：**开发过程中一个常见的问题是环境一致性问题。由于开发环境、测试环境、生产环境不一致，导致有些bug 并未在开发过程中被发现。而Docker 的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会再出现「这段代码在我机器上没问题啊」这类问题。

**持续交付和部署：**Docker是build once，run everywhere. 使用Docker 可以通过定制应用镜像来实现持续集成、持续交付、部署。开发人员可以通过Dockerfile 来进行镜像构建，并结合持续集成(Continuous Integration) 系统进行集成测试，而运维人员则可以直接在生产环境中快速部署该镜像，甚至结合持续部署(Continuous Delivery/Deployment) 系统进行自动部署。

**更轻松的迁移：**Docker 使用的分层存储以及镜像的技术，使得应用重复部分的复用更为容易，也使得应用的维护更新更加简单，基于基础镜像进一步扩展镜像也变得非常简单。此外，Docker 团队同各个开源项目团队一起维护了一大批高质量的官方镜像，既可以直接在生产环境使用，又可以作为基础进一步定制，大大的降低了应用服务的镜像制作成本。使用Dockerfile 使镜像构建透明化，不仅仅开发团队可以理解应用运行环境，也方便运维团队理解应用运行所需条件，帮助更好的生产环境中部署该镜像。

**Docker中基本概念**

**镜像(Image)**

镜像，从认识上简单的来说，就是面向对象中的类，相当于一个模板。从本质上来说，镜像相当于一个文件系统。Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

**容器(Container)**

容器，从认识上来说，就是类创建的实例，就是依据镜像这个模板创建出来的实体。容器的实质是进程，但与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的命名空间。因此容器可以拥有自己的root 文件系统、自己的网络配置、自己的进程空间，甚至自己的用户ID 空间。容器内的进程是运行在一个隔离的环境里，使用起来，就好像是在一个独立于宿主的系统下操作一样。这种特性使得容器封装的应用比直接在宿主运行更加安全。

**仓库(Repository)**

仓库，从认识上来说，就好像软件包上传下载站，有各种软件的不同版本被上传供用户下载。镜像构建完成后，可以很容易的在当前宿主机上运行，但是，如果需要在其它服务器上使用这个镜像，我们就需要一个集中的存储、分发镜像的服务，Docker Registry 就是这样的服务。

**系统安全**

安全是一个不断演进的目标，追求一个全面的、系统级别的安全方案是非常重要的。站在安全领域的角度，我们鼓励分层的概念，每一层都只管理自己职责范围内的安全问题，每一层的安全机制越严格，我们的应用就越健壮、越安全。

1、**在最底层，**我们需要处理传输层安全和系统识别，从而避免中间人(man-in-the-middle )攻击。

2、接着我们会使**用防火墙**，可能会联合VPN或者IP安全机制来保证只有被授权的系统才能进行连接。

3、在企业环境中，我们需要部署一个**DMZ**( demilitarized zone )服务来隔离对外提供访问的接口的服务器与内部数据库和应用服务器。

4、我们的操作系统也扮演了安全中的一环，例如使用不具有特定权限的用户运行进程，限制用户最大可以操作的文件数量等。**操作系统通常也会配置自己的防火墙**。

5、我们可能还会尝试使阻止**DDOS( Distributed DenialofService)分布式拒绝服务**和**暴力破解攻击**（brute force attacks ）。一个入侵检测系统对于攻击的监控和响应是非常有用的，可以帮助我们实时的拒绝某些TPC/IP地址的访问。

6、从更高的层面即**JVM的层面**来说，我们可以通过配置最小化一个Java类可以具有的权限(译者注：通过JAVA\_HOME/jre/lib/security/java.policy文件进行配置)

7、最后我们在**应用层面**添加一些**领域特定的安全配置**。

**Spring Security 模块**

核心： authentication（认证） 和authorization（授权）

**Java instanceof用法详解**

instanceof 是 Java 的保留关键字。它的作用是测试它左边的对象是否是它右边的类的实例，返回 boolean 的数据类型。

**高内聚低耦合**

从模块粒度来看：高内聚--尽可能类的每个成员方法只完成一件事（最大限度的聚合）；低耦合：减少类内部，一个成员方法调用另一个成员方法。

从类角度来看：高内聚低耦合--减少类内部，对其他类的调用；

从功能块来看：高内聚低耦合--减少模块之间的交互复杂度（接口数量，参数数据）即横向：类与类之间、模块与模块之间；纵向：层次之间；尽可能，内容内聚，数据耦合。

**定义类与类之间的交互关系**

泛化（Generalization）可以简单理解为继承关系。具体到 Java 代码就是下面这样：

public class A { ... }

public class B extends A { ... }

实现（Realization）一般是指接口和实现类之间的关系。具体到 Java 代码就是下面这样：

public interface A {...}

public class B implements A { ... }

聚合（Aggregation）是一种包含关系，A 类对象包含 B 类对象，B 类对象的生命周期可以不依赖 A 类对象的生命周期，也就是说可以单独销毁 A 类对象而不影响 B 对象，比如课程与学生之间的关系。具体到 Java 代码就是下面这样：

public class A {

private B b;

public A(B b) {

this.b = b;

}

}

组合（Composition）也是一种包含关系。A 类对象包含 B 类对象，B 类对象的生命周期跟依赖 A 类对象的生命周期，B 类对象不可单独存在，比如鸟与翅膀之间的关系。具体到 Java 代码就是下面这样：

public class A {

private B b;

public A() {

this.b = new B();

}

}

关联（Association）是一种非常弱的关系，包含聚合、组合两种关系。具体到代码层面，如果 B 类对象是 A 类的成员变量，那 B 类和 A 类就是关联关系。具体到 Java 代码就是下面这样：

public class A {

private B b;

public A(B b) {

this.b = b;

}

}

或者

public class A {

private B b;

public A() {

this.b = new B();

}

}

依赖（Dependency）是一种比关联关系更加弱的关系，包含关联关系。不管是 B 类对象是 A 类对象的成员变量，还是 A 类的方法使用 B 类对象作为参数或者返回值、局部变量，只要 B 类对象和 A 类对象有任何使用关系，我们都称它们有依赖关系。具体到 Java 代码就是下面这样：

public class A {

private B b;

public A(B b) {

this.b = b;

}

}

或者

public class A {

private B b;

public A() {

this.b = new B();

}

}

或者

public class A {

public void func(B b) { ... }

}

**Sql学习**

**常用的文本处理函数：**

LEFT()（或使用子字符串函数） 返回字符串左边的字符

LENGTH()（也使用DATALENGTH()或LEN()） 返回字符串的长度

LOWER()（Access使用LCASE()） 将字符串转换为小写

LTRIM() 去掉字符串左边的空格

RIGHT()（或使用子字符串函数） 返回字符串右边的字符

RTRIM() 去掉字符串右边的空格

SOUNDEX() 返回字符串的SOUNDEX值 SOUNDEX是一个将任何文本串转换为描述其语音表示的字母数字模式的算法。

UPPER()（Access使用UCASE()） 将字符串转换为大写

**数值处理函数：**

ABS() 返回一个数的绝对值

COS() 返回一个角度的余弦

EXP() 返回一个数的指数值

PI() 返回圆周率

SIN() 返回一个角度的正弦

SQRT() 返回一个数的平方根

TAN() 返回一个角度的正切

**在使用GROUP BY子句前，需要知道一些重要的规定：**

GROUP BY子句可以包含任意数目的列，因而可以对分组进行嵌套，更细致地进行数据分组。

如果在GROUP BY子句中嵌套了分组，数据将在最后指定的分组上进行汇总。换句话说，在建立分组时，指定的所有列都一起计算（所以 不能从个别的列取回数据）。

GROUP BY子句中列出的每一列都必须是检索列或有效的表达式（但不能是聚集函数）。如果在SELECT使用表达式，则必须在GROUP BY子句中指定相同的表达式。不能使用别名。 大多数SQL实现不允许GROUP BY列带有长度可变的数据类型（如文本或备注型字段）。

除聚集计算语句外，SELECT语句中的每一列都必须在GROUP BY子句中给出。

如果分组列中包含具有NULL值的行，则NULL将作为一个分组返回。如果列中有多行NULL值，它们将分为一组。

GROUP BY子句必须出现在WHERE子句之后，ORDER BY子句之前。

**获得系统日期：**

DBMS 函数/变量

Access NOW()

DB2 CURRENT\_DATE

MySQL CURRENT\_DATE()

Oracle SYSDATE

PostgreSQL CURRENT\_DATE

SQL Server GETDATE()

SQLite date('now')

**视图的一些常见应用：**

1、重用SQL语句。

2、简化复杂的SQL操作。在编写查询后，可以方便地重用它而不必知道其基本查询细节。

使用表的一部分而不是整个表。

3、保护数据。可以授予用户访问表的特定部分的权限，而不是整个表的访问权限。

4、更改数据格式和表示。视图可返回与底层表的表示和格式不同的数据

**在开始创建索引前，应该记住以下内容：**

索引改善检索操作的性能，但降低了数据插入、修改和删除的性能。在执行这些操作时，DBMS必须动态地更新索引。

索引数据可能要占用大量的存储空间。

并非所有数据都适合做索引。取值不多的数据（如州）不如具有更多可能值的数据（如姓或名），能通过索引得到那么多的好处。

索引用于数据过滤和数据排序。如果你经常以某种特定的顺序排序数据，则该数据可能适合做索引。

可以在索引中定义多个列（例如，州加上城市）。这样的索引仅在以州加城市的顺序排序时有用。如果想按城市排序，则这种索引没有用处。

没有严格的规则要求什么应该索引，何时索引。大多数DBMS提供了可用来确定索引效率的实用程序，应该经常使用这些实用程序

**数据结构与算法**

1、数据结构和算法学习的精髓-复杂度分析

2、最常用的、最基础的数据结构：数组、链表、栈、队列、散列表、二叉树、堆、跳表、图、Trie 树

3、最常用的算法：递归、排序、二分查找、搜索、哈希算法、贪心算法、分治算法、回溯算法、动态规划、字符串匹配算法

**事务管理**

因为Spring的默认的事务规则是遇到**运行异常（RuntimeException）**和**程序错误（Error）**才会回滚。如果想针对非检测异常进行事务回滚，可以在@Transactional 注解里使用 rollbackFor 属性明确指定异常。

**Transactional注解的常用属性表：**

**属性**  **说明**

propagation 事务的传播行为，默认值为 REQUIRED。

isolation 事务的隔离度，默认值采用 DEFAULT

timeout 事务的超时时间，默认值为-1，不超时。如果设置了超时时间(单位秒)，那么如果超过该时间限制了但事务还没有完成，则自动回滚事务。

read-only 指定事务是否为只读事务，默认值为 false；为了忽略那些不需要事务的方法，比如读取数据，可以设置 read-only 为 true。

rollbackFor 用于指定能够触发事务回滚的异常类型，如果有多个异常类型需要指定，各类型之间可以通过逗号分隔。{xxx1.class, xxx2.class,……}

noRollbackFor 抛出 no-rollback-for 指定的异常类型，不回滚事务。{xxx1.class, xxx2.class,……}

**提示**

事务的传播机制是指如果在开始当前事务之前，一个事务上下文已经存在，此时有若干选项可以指定一个事务性方法的执行行为。 即:在执行一个@Transactinal注解标注的方法时，开启了事务；当该方法还在执行中时，另一个人也触发了该方法；那么此时怎么算事务呢，这时就可以通过事务的传播机制来指定处理方式。

**TransactionDefinition传播行为的常量：**

**常量**  **含义**

TransactionDefinition.PROPAGATION\_REQUIRED 如果当前存在事务，则加入该事务；如果当前没有事务，则创建一个新的事务。这是默认值。

TransactionDefinition.PROPAGATION\_REQUIRES\_NEW 创建一个新的事务，如果当前存在事务，则把当前事务挂起。

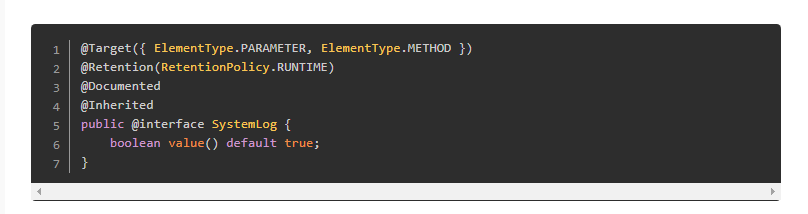
TransactionDefinition.PROPAGATION\_SUPPORTS 如果当前存在事务，则加入该事务；如果当前没有事务，则以非事务的方式继续运行。

TransactionDefinition.PROPAGATION\_NOT\_SUPPORTED 以非事务方式运行，如果当前存在事务，则把当前事务挂起。

TransactionDefinition.PROPAGATION\_NEVER 以非事务方式运行，如果当前存在事务，则抛出异常。

TransactionDefinition.PROPAGATION\_MANDATORY 如果当前存在事务，则加入该事务；如果当前没有事务，则抛出异常。

TransactionDefinition.PROPAGATION\_NESTED 如果当前存在事务，则创建一个事务作为当前事务的嵌套事务来运行；如果当前没有事务，则该取值等价于TransactionDefinition.PROPAGATION\_REQ



**注解大全**

**Retention注解**

Retention(保留)注解说明,这种类型的注解会被保留到那个阶段. 有三个值:

1.RetentionPolicy.SOURCE —— 这种类型的Annotations只在源代码级别保留,编译时就会被忽略

2.RetentionPolicy.CLASS —— 这种类型的Annotations编译时被保留,在class文件中存在,但JVM将会忽略

3.RetentionPolicy.RUNTIME —— 这种类型的Annotations将被JVM保留,所以他们能在运行时被JVM或其他使用反射机制的代码所读取和使用.

上面示例中, @Retention(RetentionPolicy.RUNTIME)注解表明 Test\_Retention注解将会由虚拟机保留,以便它可以在运行时通过反射读取.

**Target注解**

@Target说明了Annotation所修饰的对象范围：Annotation可被用于 packages、types（类、接口、枚举、Annotation类型）、类型成员（方法、构造方法、成员变量、枚举值）、方法参数和本地变量（如循环变量、catch参数）。在Annotation类型的声明中使用了target可更加明晰其修饰的目标。

作用：用于描述注解的使用范围（即：被描述的注解可以用在什么地方）

取值(ElementType)有：

1.CONSTRUCTOR:用于描述构造器

2.FIELD:用于描述域

3.LOCAL\_VARIABLE:用于描述局部变量

4.METHOD:用于描述方法

5.PACKAGE:用于描述包

6.PARAMETER:用于描述参数

7.TYPE:用于描述类、接口(包括注解类型) 或enum声明

**基本算法思想**

贪心算法：

分治算法：

动态规划：

回溯算法：

枚举算法：