**Scattered But Useful Programming Knowledge**

**---- VICTORY LOVES PREPARATION**

**@AUTHOR Count Monte Cristo**

**@CREATE 2015年09月11日 00:00**

**@DESCRIPTION Record some useful programming knowledge in solving problems， waiting to be classified.**

**@VERSION 2016年08月28日 11:24**

**2016/05/20 10:42**

**2016/05/19 09:59**

**2016/05/17 15:50**

**2016/05/04 19:10**

**2016/04/29 14:23**

**2016年04月08日 09:44:48**

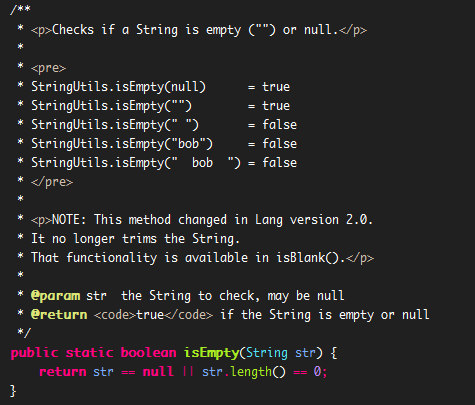
**v1.0.5.20160223\_145615\_alpha**

**~~D20150924~~ ~~D20151022~~ ~~D20151125~~ ~~D20151203~~ ~~D20160216~~**

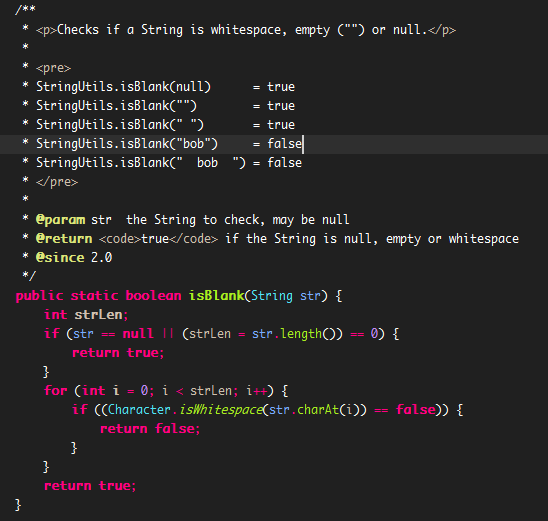
**@CRITERIA 凯旋基诺**

1. **关于MyBatis和Hibernate的事务处理**
   1. MyBatis和Hibernate都是支持事务的持久层框架（ORM），也就说是它们在进行数据库操作的时候都是先把CRUD的操作写进内存（缓存）中，然后通过事务进行提交（调用commit()方法，先使用flush()清除缓存然后写入数据库）
2. **开发、测试和生产环境**
   1. **具体解析**
      1. 开发环境
         1. 开发环境是程序员们专门用于开发的服务器，配置可以比较随意，为了开发调试方便，一般打开全部错误报告；
      2. 测试环境
         1. 一般是克隆一份生产环境的配置，一个程序在测试环境工作不正常，那么肯定不能把它发布到生产机上；
      3. 生产环境
         1. 是指正式提供对外服务的，一般会关掉错误报告，打开错误日志；
   2. **注意**
      1. 三个环境也可以说是系统开发的三个阶段：开发->测试->上线，其中生产环境也就是通常说的真实环境

1. **项目原型图（原型设计）**
   1. 就是项目成果的一个粗略图
2. **StringUtils.isEmpty()和StringUtils.isBlank()**
   1. **StringUtils.isEmpty()**



* 1. **StringUtils.isBlank()**



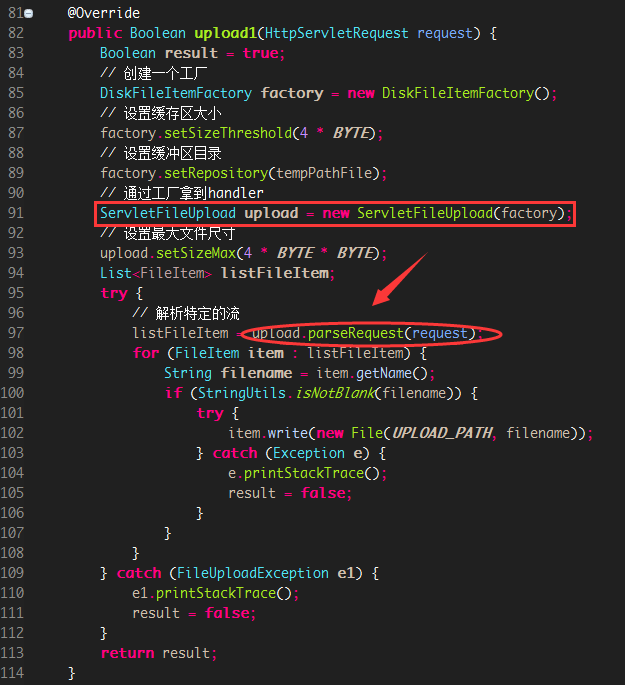
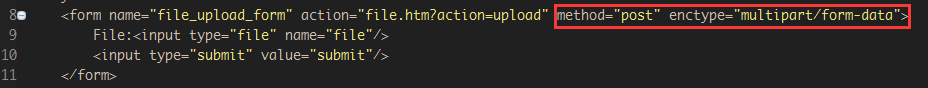
1. **Tomcat的默认编码 @version D20151125**
   1. Tomcat8之前默认使用的是**ISO-8859-1**编码，可以在**Tomcat**的**server.xml**文件中通过设置**URIEncoding**属性进行编码的配置，Tomcat8之后默认使用**UTF-8**
2. **关于缓存 @version D20151127**
   1. **@reference 缓存@百度百科**
   2. **简要说明**
      1. 缓存就是数据交换的缓冲区（称作Cache），一般是指运行速度在内存和CPU之间的缓存，**但是也可以指在硬盘、网络上的缓存**
      2. 常用缓存
         1. WEB缓存
            1. 可以缓存在客户端、代理服务器和远程服务器上
         2. 浏览器缓存
            1. 可以缓存在内存或者硬盘上

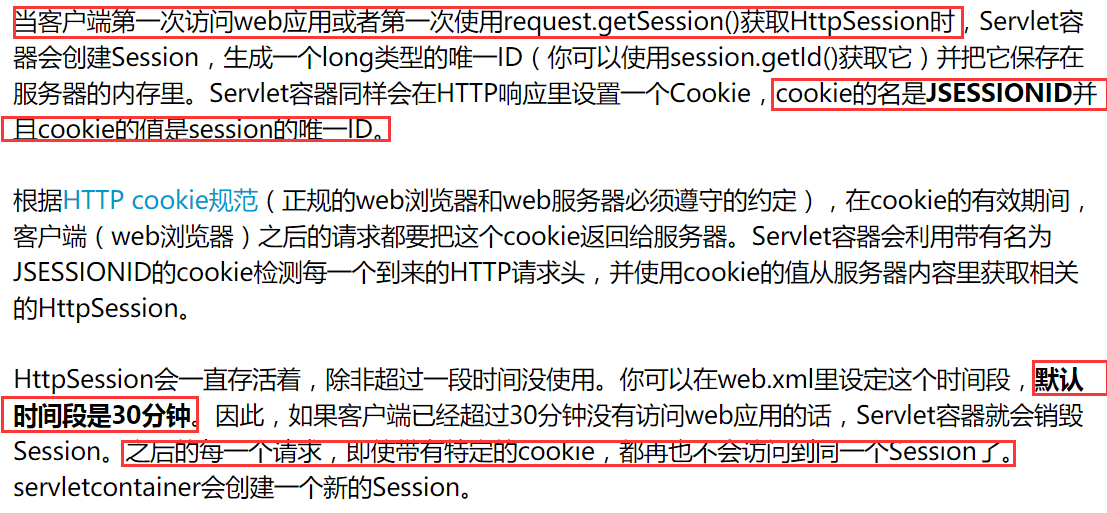
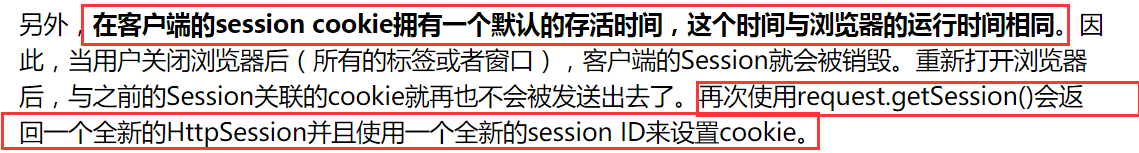
**8. 通配符 @version D20151130**

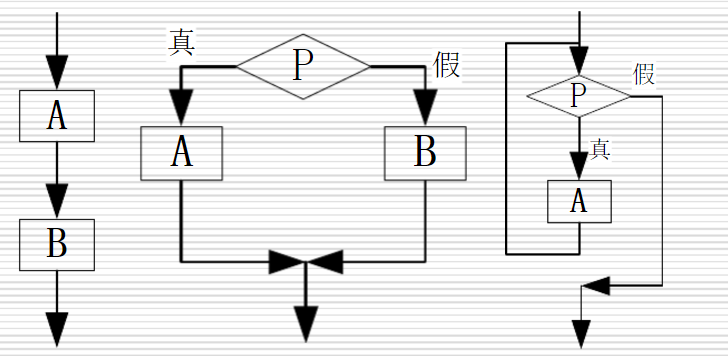
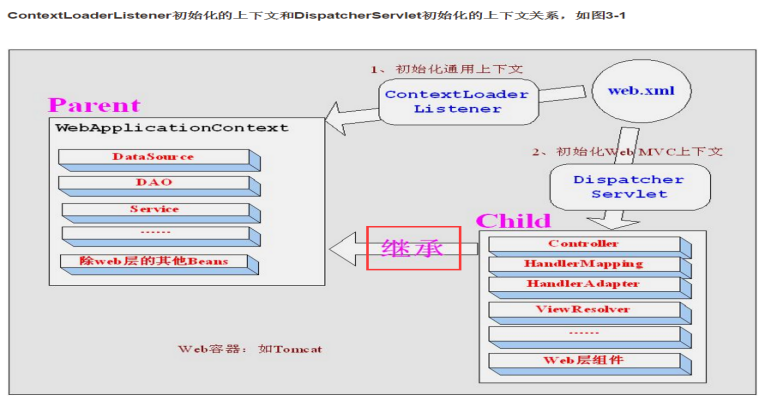
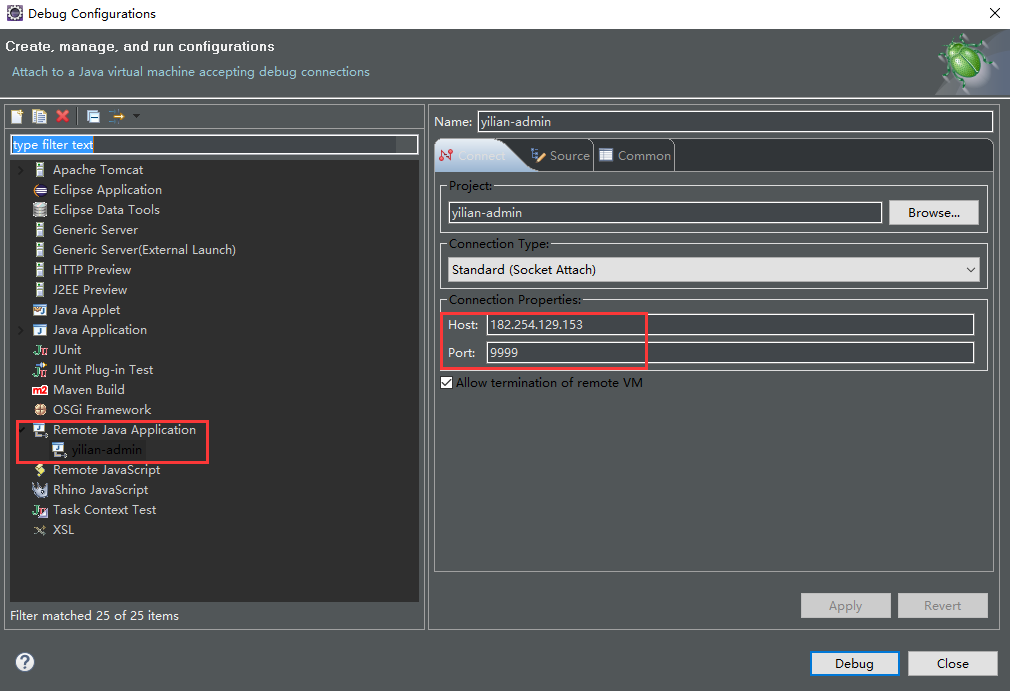
* 1. 通：全部；配：符合；符：符号。即全部符合的符号
  2. 分类：
     1. \*：代替0个、1个、2个或者多个字符
     2. ？：代替1个字符
     3. 注意，还有其他的通配符，但是应用场景比较局限
  3. 应用场景
     1. 搜索（匹配）

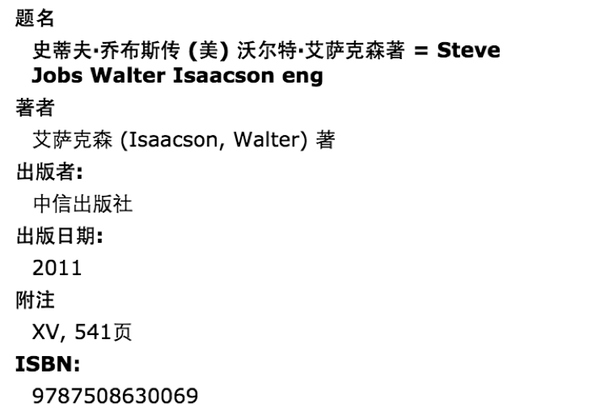
**9. 关于SQL中的select、insert、update以及delete的返回值**

* 1. 正常情况（无异常）下，成功则返回受影响的行数、失败则为0
  2. SQL不对则会抱出异常（此时当然什么都没有返回了）

1. **脚本语言 @version D20151203**
   1. 脚本语言又被称为扩建的语言，或者动态语言（**是指程序在运行时可以改变其结构：新的函数可以被引进，已有的函数可以被删除等在结构上的变化，因为只有在被调用时进行解释或者编译**），是一种编程语言，用来控制软件应用程序，脚本通常以文本（如ASCII)保存，**只在被调用时进行解释或编译**。
2. **图片上传** **@version D20151215**
   1. **实例（view和service）**
   2. 注意
      1. 普通的form表单不支持图片的传递，需要指定enctype=”multipart/form-data”，同时method必须为post方式
      2. 箭头所指的地方是ServletFileUpload解析HttpServletRequest请求，获取图片“片段”集合（List<FileItem>）
3. **Session和Cookie**
   1. **Session**
      1. 存放在服务器内存当中
   2. **Cookie**
      1. 放置在HTTP响应中
   3. **详情**



1. **SQL可以通过字符串比较时间**
2. **SVN**
   1. **mark as merged**
      1. 把本地分支的代码合并到主分支上（**注意：如果冲突没有解决的话会覆盖掉远程的代码**）
   2. **冲突解决**
      1. 先update然后解决冲突（mine、==、remote分隔的代码），然后mark as merged，最后commit
3. **JS数据类型**
   1. **JS拥有动态类型，这意味着相同的变量可以有不同的类型**，也就是说JS虽然是弱类型语言（变量只有var类型，实际上把var细分还有整型、字符串型等，只是JS统一使用var类型，所以是弱类型），但是其可以有整型、字符串型等其他数据类型，关键在于数据的表现形式是什么样的就怎么解析
4. **程序的三种语句结构**
   1. 图示
   2. 顺序结构
   3. 选择结构（if、switch etc.）
   4. 循环结构（for、while、do while etc.）
5. **编码（**形式或者格式的转变**）问题**
   1. ISO8859-1是单字节编码，向下兼容ASCII，支持部分欧洲使用的语言
   2. 有的编码单字节编码、有的是双字节编码、还有的是多字节编码
   3. **编码之间有的能够相互转换，有的不能，视具体使用情况决定**
   4. UTF-8编码可以用GBK和ISO8859-1解码后
   5. GBK编码后只能用ISO8859-1解码后编码回去；
   6. UNICODE（Java中的变量都是以该编码存储）不兼容ISO8859-1，兼容GBK和UTF-8
   7. FAQ
      1. Web项目乱码
   8. Answer
      1. 在调用getParameter()之前通过request.setCharacterEncoding设置字符编码，
      2. 调用new String(str.getBytes("ISO8859-1"), "UTF-8")编码后解码（除了Unicode不兼容ISO8859-1以外其他的编码应该都兼容 @notice 未验证，但是可以这样使用）
   9. 注意
6. **Service中使用@Autowired注解问题**
   1. 此时需要<context:component-scan base-package="" />标签进行组件扫描才可以使该注解生效，即注入需要的bean
   2. **注意：此时如果service已经通过xml的方式注入到了Spring容器中，该service还是要加上组件扫描的，也就是说“Spring管理的Bean和组件扫描没有关系，即组件扫描不在Spring容器管理Bean的职能范围，Bean加上组件扫描功能才能进行注解扫描”，！！！对啊，Spring容器管理Bean和组件扫描有毛关系啊，还是没有深入理解Spring。**
7. **Spring MVC ModelMap和HttpServletRequest的关系**
   1. **@reference <http://blog.csdn.net/flashflight/article/details/43018333>**
   2. Spring会把ModelMap中的数据放到HttpServletRequest，然后HttpServletRequest传给下一个Servlet（无论是Handler还是处理页面的servlet，页面解析元素在servlet中把request中的值取出来放到具体的页面中，然后把页面返回给浏览器渲染）。
8. **Context理解及实例**
   1. Context是代表系统的上下文（亦即环境），通常是使用Map存放数据
   2. 实例
      1. ServletContext
         1. **@reference JDK**
         2. ServletContext是用来让servlet和其容器（存在数据）沟通的桥梁
         3. 对于ServletContext（这个Context，上下文），一个JVM一个WEB应用有且只有一个，分布式应用需要外部资源（如，数据库）存储context以便共享之
      2. WebApplicationContext
         1. **@Reference Spring**
         2. 存放Beans等
      3. 关系简易图
         1. 
         2. 注意点
            1. “继承”说的不是很确切，应该是WebApplicationContext共享于ServletContext（反之则不可以），所以如果给应用添加事务，（对于Spring+Spring MVC的Web Application）（如果需要在Controller和其他beans中使用事务）则有两个上下文（应该尽量把Service和Dao以及其他配置文件配置在WebApplicationContext中，而controller配置在ServletContext中）需要配置
9. **回调函数**
   1. 回调函数就是一个通过函数指针调用的函数。如果你把函数的指针（地址）作为参数传递给另一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行响应。
   2. 意义
      1. 通过函数指针（地址）调用的函数
   3. 作用
      1. 对特定的时间或者条件进行相应
   4. 特点
      1. 不是由实现方法直接调用，而是作为调用函数的参数进行调用
10. **MD5盐值**
    1. 实际使用
       1. 为了使相同的密码拥有不同的hash值（当然此时的盐值也是不同的）
    2. （名词）盐化
       1. 相同的字段拥有不同的hash值
       2. 例如
          1. hash(a + salt1) = x
          2. Hash(a + salt2) = y
          3. 其中salt1和salt2就是盐值，让相同的数据（a）拥有了不同的hash值（当然前提是hash算法相同）
11. **Hash化/散列（通过散列算法把输入的任意长度的数据压缩成固定长度）**
    1. Hash，一般翻译做“散列”，也有直接音译为“哈希”的，就是把任意长度的输入（又叫做预映射， pre-image），通过散列算法，变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，所以不可能从散列值来唯一的确定输入值。简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。
12. **数字签名**
    1. 数字签名，就是只有信息的发送者才能产生的别人无法伪造的一段数字串，这段数字串同时也是对信息的发送者发送信息真实性的一个有效证明
    2. 特点
       1. 保证**信息传输的完整性**、**发送者的身份认证**，防止交易中的抵赖发生。
    3. （**关键词：报文正文、哈希函数、报文文摘和数字签名（公钥、私钥）**）
       1. 发送报文时，发送方用一个哈希函数从报文文本中生成报文摘要，然后用自己的私人密钥对这个摘要进行加密，这个加密后的摘要将作为报文的数字签名和报文一起发送给接收方，接收方首先用与发送方一样的哈希函数从接收到的原始报文中计算出报文摘要，接着再用发送方的公用密钥来对报文附加的数字签名进行解密，如果这两个摘要相同、那么接收方就能确认该数字签名是发送方的
13. **Struts2中action为什么必须是 prototype的，不能是singleton的？**
    1. 因为Webwork（Struts2）的action不是线程安全的，singleton指的是对某个对象的完全共享，包括代码空间和内存空间，所以此种情况下Struts2的action必须是prototype的，这样就不会在action非线程安全的情况下导致数据错误
14. **UA（User-Agent）**
    1. User Agent中文名为用户代理，是Http协议中的一部分，属于头域的组成部分，User Agent也简称UA。它是一个特殊字符串头，是一种向访问网站提供你所使用的浏览器类型及版本、操作系统及版本、浏览器内核、等信息的标识。通过这个标识，用户所访问的网站可以显示不同的排版从而为用户提供更好的体验或者进行信息统计；例如用手机访问谷歌和电脑访问是不一样的，这些是谷歌根据访问者的UA来判断的。UA可以进行伪装。浏览器的UA字串的标准格式：浏览器标识 (操作系统标识; 加密等级标识; 浏览器语言) 渲染引擎标识版本信息。但各个浏览器有所不同。
15. **表单提交**
    1. 如果GET请求的表单action属性中已经包含参数，浏览器会直接将其过滤掉，再附加form表单数据
16. **长连接和短连接**
    1. TCP进行传输数据的时候，端到端存在建立连接和关闭的操作，这样的话就是会耗费时间的。短连接是在一次请求操作的时候就有连接和关闭的操作，下次请求的时候继续重复建立连接和关闭连接操作；长连接这是建立一次连接之后，进行多次请求操作，确定请求结束就关闭连接，这样节省了（n-1）次请求建立连接和关闭连接的时间。长连接的维持是通过在服务器端进行keep-alive操作，也就说客户端没有数据包发送的时候，长连接辅助程序就会发送心跳包来维持这个连接。
17. **JS解析JSON字符串为JS对象**
    1. 通过eval("("+data+")")引用
    2. 注意
       1. 如果data里面还有一个json字符串，那么此时被解析为JS对象（仍然是json字符串），因为注意看eval中多了个（），原因：由于json是以”{}”的方式来开始以及结束的，在JS中，它会被当成一个语句块来处理，所以必须强制性的将它转换成一种表达式。
18. 转义
    1. 含义：所谓转义，就是把要转义的对象转化成并不是它**本来意义**（即转义对象所在语言环境中的意义）的一种行为操作
    2. 例如
       1. 在C语言中n就是代表字母n本身，但是加上C语言的转义符“\“变成”\n“，那么就代表了C语言的换行的作用
       2. 在HTML中双引号”代表着其语法特征，例如<input value=”aa”>，但是如果””中还有双引号的话，如value=”aa”bb”，那么此时value的值就是aa，for HTML就是这样处理的，所以需要转义双引号（让其不具备在HTML语言中本来的含义），HTML使用&quot进行双引号的转义
19. 关于Java和JS中的字符串问题
    1. 两者都是对象
20. 关于JS中的JSON字符串和JSON对象问题
    1. **关于这段话的理解，因为JSON字符串有可能是数组，所以此时JSON字符串转化为JSON对象的就是字符串转换为数组了**
21. **轮询**
    1. **JS版本**
       1. 
       2. 通过setTimeout(function(){}, millisecond)调用
    2. **Java版本**
22. **关于Servlet如何同时处理多个请求的问题**
    1. Servlet通过维护线程池方式，采用多线程的来处理多个请求的【**单例（相对于多例而言节省了内存空间），多线程（操作系统机制）】**
    2. 线程：程序执行流中的最小单位
    3. 实际情况
       1. 一个请求来到服务器，服务器分配一个处在等待状态的线程来**处理**该请求，当下一个请求来到服务器的时候，服务器又分配一个处在等待状态的线程来**处理**该请求。
       2. 特别注意处理过程
          1. 代码放在内存上的，服务器分配线程处理请求的时候，肯定知道了对应对象的内存地址，然后给线程，然后线程去从对象的内存开始地址读取，然后执行，这样就把内存空间省下来，不过加重了操作系统的负担，但是考虑到操作系统的速度远远大于内存的速度，在内存的存储方面和CPU的处理速度方面，还是后者有较大的“空间”，所以选择给多线程，给CPU处理
23. **Transient【变量的修饰词，不加入序列化过程】**
    1. 变量的修饰词，被该关键词修饰的变量不参加序列化的过程
24. **数据字典【数据的定义和描述】**
    1. 数据字典是指对数据（包含数据项、数据结构、数据流、数据存储、处理逻辑、外部实体等）的定义和描述，其目的是对数据流程图中的各个元素做出详细的说明
25. **Spring测试【加载Spring容器】**
    1. 实例
       1. 
    2. 利用SpringJUnit4ClassRunner类加载Spring的配置文件，在测试类中生成Spring的环境（Spring的环境一般配置在web服务器中）
    3. 一般测试service而不测试dao（service在dao的外层，这样可以省去测试dao层 @version 2016/06/07 14:20）
26. **跨域请求【只要协议、域名、端口有任何一个不同，都被当作是不同的域】**
    1. 
    2. **注意**
       1. 域名分为多级，每一级都要相同才行...
27. URLEncode **@version 2016/08/11 14:32**
    1. 把URL转换成application/x-www-form-urlencoded格式
    2. 注意
       1. 有的时候后台map(“url”,“/user/action=add?name=aaa&age=18”),然后前台传给后台url，这样在获取url的时候就会把&后面的参数漏掉，所以要使用java.net.URLEncoder.encode()进行转换格式为x-www-form-urlencoded，该格式确保该url对应的字符串在使用的时候为一个整体
28. **Java中||和|、&&和& @version 2016/08/15 17:20**
    1. &&和||分别都是“短路”逻辑运算符，也就是说对于&&来说，如果第一个不成立，表达式必定为false，此时就不用判断后面的条件了；对于||来说，如果第一个成立，表达式必定为true，此时就不用判断后面的条件了
    2. &、|则都需要进行比较
29. **Eclipse远程调试 @version 2016/08/16 12:30**
    1. **@reference <http://blog.csdn.net/afgasdg/article/details/9236877>**
    2. **两步走**
       1. 服务器设置远程访问信息
          1. 默认已经设置好，端口8000【可自定义】
          2. 可以通过cat catalina.out|grep dt\_socket检查远程调试端口信息
       2. 本地eclipse设置debug远程服务器信息
          1. 
    3. **注意事项**
       1. CATALINA\_OPTS="-Xdebug -Xrunjdwp:transport=dt\_socket,address=9999,server=y,suspend=n"（重新定义的了远程调试信息），其中CATALINA\_OPTS的设置需要注意一下定义位置，因为catalina.sh原来就有CATALINA\_OPTS内容设置，如果新的设置位置放在了旧的前面，此时显然CATALINA\_OPTS被重新赋值，新的不起作用了=\_\_\_=
30. **Java集合框架总结**
31. **SQL IN()操作符效率低以及解决方案**
32. **SQL Where字句条件顺序效率问题 @version 2016/08/22 @notice 网上大部分说执行顺序从左往右，但是myself实测从右往左** 
    1. 结果集越小的越往右放（从右往左执行），这样比反之速度要快
33. **Eclipse反编译**
    1. JadClipse
34. **元数据【关于数据的数据】 @VERSION 2016年08月24日 11:31** 
    1. 元数据（Metadata）是关于数据的**组织**、**数据域**及其**关系**的信息，简言之，元数据就是关于数据的数据；
    2. 例图



1. 为什么重载的时候类型不同（其他内容相同）的方法会判断为duplicate @VERSION 2016年09月21日 16:14
   1. @GUESS 因为如果其他内容相同，也就意味着获取的内容相同，仅仅是返回内容的类型不同，而且Java本身有类型转换功能，所以为了避免Java“鼓励”构建重复代码的缺点，所以不允许此情况，如果有需要可以通过类型转换得到自己所需