

1. Spring
2. 父子容器

在一个项目中引入Spring和SpringMVC这两个框架，那么它其实就是两个容器，Spring是父容器，SpringMVC是其子容器，并且在Spring父容器中注册的Bean对于SpringMVC容器中是可见的，而在SpringMVC容器中注册的Bean对于Spring父容器中是不可见的，也就是子容器可以看见父容器中的注册的Bean，反之就不行。

1. 异常拦截器

a. 使用Spring MVC提供的SimpleMappingExceptionResolver

b . 实现Spring的异常处理接口HandlerExceptionResolver 自定义自己的异常处理器

c. 使用@ExceptionHandler注解实现异常处理: @ControllerAdvice

1. 日志拦截器
2. 拦截器，实现MethodInterceptor

重写：invoke(MethodInvocation invocation) 方法

1. AOP : ProceedingJoinPoint
2. 配置文件加载：

util:properties和context:property-placeholder标签都可以用来获取外部配置文件中的内容

1. PropertyPlaceholderConfigurer

或者

<context:property-placeholder location="classpath:jdbc.properties"/>

1. <util: 方式，这种方式加载配置文件的时机更早：

加载文件：

<util:properties id="apihttp" location="classpath:server.properties"/>

获取参数：

<property name="url" value="#{apihttp['apihttp.car.url']}"/>

1. 定时任务 ：

@Scheduled(cron = "${carLeagueDaily.schedule.cron}")

carLeagueDaily.schedule.cron=0 0 9 \* \* ?

1. 传集合对象或复杂对象：

RespMsg addEmployeeList(@RequestBody List<EmployeeDTO> employeeList)

前端需要以json格式传对象

a. @RequestMapping：@RequestMapping 是一个用来处理请求地址映射的注解，可用于类或方法上。

b. @ResponseBody：@Responsebody 注解表示该方法的返回的结果直接写入 HTTP 响应正文（ResponseBody）中，一般在异步获取数据时使用，通常是在使用 @RequestMapping 后，

返回值通常解析为跳转路径，加上 @Responsebody 后返回结果不会被解析为跳转路径，而是直接写入HTTP 响应正文中。

该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区。

c. @RequestParam:Post和Get方法都可以使用，解析 请求头和请求体中数据，实质是将Request.getParameter() 中的Key-Value参数Map利用Spring的转化机制ConversionService配置，转化成参数接收对象或字段。

用来处理Content-Type: 为 application/x-www-form-urlencoded编码的内容。（Http协议中，如果不指定Content-Type，则默认传递的参数就是application/x-www-form-urlencoded类型）

RequestParam可以接受简单类型的属性，也可以接受对象类型。

在Content-Type: application/x-www-form-urlencoded的请求中，get 方式中queryString的值，和post方式中 body data的值都会被Servlet接受到并转化到Request.getParameter()

参数集中，所以@RequestParam可以获取的到。

d.@RequestBody:只能POST方法中使用，解析请求体中的数据(流？)

@RequestBody 注解则是将 HTTP 请求正文插入方法中，使用适合的 HttpMessageConverter 将请求体写入某个对象。

处理HttpEntity传递过来的数据，一般用来处理非Content-Type: application/x-www-form-urlencoded编码格式的数据。

GET请求中，因为没有HttpEntity，所以@RequestBody并不适用。

POST请求中，通过HttpEntity传递的参数，必须要在请求头中声明数据的类型Content-Type，SpringMVC通过使用HandlerAdapter 配置的HttpMessageConverters来解析HttpEntity中的

数据，然后绑定到相应的bean上。

总结:

在GET请求中，不能使用@RequestBody。

在POST请求，可以使用@RequestBody和@RequestParam，但是如果使用@RequestBody，对于参数转化的配置必须统一。

<http://blog.csdn.net/xinluke/article/details/52710706>

http://blog.csdn.net/ff906317011/article/details/78552426

1. Mybatis
2. 接口扫描：MapperScannerConfigurer

|  |
| --- |
| <!-- 配置Mybatis的ReadMapperScanner -->  <bean id=*"mapperScan"* class=*"org.mybatis.spring.mapper.MapperScannerConfigurer"*>  <property name=*"basePackage"* value=*"com.mljr.car.league.erp.mapper"* />  <property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactory"* />  </bean> |

1. 文件扫描：

|  |
| --- |
| <!-- 配置Mybatis的ReadSqlSessionFactory -->  <bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>  <property name=*"configLocation"* value=*"classpath:config/mybatis.xml"* />  <property name=*"dataSource"* ref=*"dataSource"* />  <property name=*"mapperLocations"*>  <list>  <value>classpath\*:mapper/\*\*/\*Mapper.xml</value>  </list>  </property>  </bean> |

1. 新增时返回自动生成的主键：

|  |
| --- |
| <insert id=*"add"* useGeneratedKeys=*"true"* keyProperty=*"vo.id"*>  INSERT INTO employee ( telephone,name,password)  VALUES ( #{vo.telephone}, #{vo.name}, #{vo.password})  </insert> |

1. 更新数据时，注意不要把未传的字段更新成null
2. 尽量避免关联查询
3. If判断 和 for 循环(**务必注意，下面这种形式idList为空时，会查询所有数据**)

|  |
| --- |
| <if test=*"idList!=null and idList.size>0"*>  AND a.id in  <foreach collection=*"idList"* item=*"id"* index=*"index"*  open=*"("* close=*")"* separator=*","*>  #{id}  </foreach>  </if> |

1. 批量插入

|  |
| --- |
| <insert id =*"addBatch"* parameterType=*"java.util.List"* >  <selectKey resultType =*"java.lang.Long"* keyProperty= *"id"*  order= *"AFTER"*>  SELECT LAST\_INSERT\_ID()  </selectKey >  insert into employee  ( telephone,name,password )  values  <foreach collection =*"list"* item=*"vo"* index= *"index"* separator =*","*>  ( #{vo.telephone}, #{vo.name}, #{vo.password} )  </foreach >  </insert > |

1. 特殊字符处理：
2. 纯文本标记：<![CDATA[<]]>
3. 使用转移字符：&lt;
4. 公共服务
5. 分页：PageHelper
6. 日志：日志拦截器
7. 异常：异常统一处理
8. Redis缓存

缓存不变的数据(城市主键等)；

缓存变的数据(门店名称)；

缓存登录信息；

防止重复提交

1. 缓存更新策略：
2. 失效：应用程序先从cache取数据，没有得到，则从数据库中取数据，成功后，放到缓存中。
3. 命中：应用程序从cache中取数据，取到后返回。
4. 更新：先把数据存到数据库中，成功后，再让缓存失效。
5. 数据库不存在：缓存中存放null值，避免继续请求数据库，设置较短的过期时间
6. 过期时间：每次缓存设置过期时间，避免意外的脏数据
7. 缓存穿透：就是指某个key，先查cache没查到，再查db也没有查到。所有的请求都会查数据库。

所谓“缓存穿透“，就是指某个key，先查cache没查到，再查db也没有查到。

这种key的存在，会导致cache一直没办法命中，压力一直打在db上面。如果访问很高频，可能会压垮DB。

解决办法其实也很简单：当查询DB没查到时，往缓存中写入一个空值（缺省值），这样第2次再查，就不会打到DB上了。

1. 缓存雪崩：所谓“缓存雪崩“，是指缓存的机器挂了，或者数据未加载到缓存中，或者缓存同一时间大面积的失效，从而导致所有请求都去查数据库，导致数据库CPU和内存负载过高，甚至宕机。

这种问题的解决策略，一般有以下2个方面：

* 1. 提高缓存的HA。比如缓存的主从复制。
  2. 对DB的访问实行限流、降级。
  3. 缓存过期时间。以redis为例，将过期设置放到1数据库，真实数据放到0数据库，key值相同，假设都为key1。

应用程序首先判断1库这条数据是否失效，当1库标记数据库数据失效或过期，在1库中设置新的过期时间。然后从数据库取数据更新0数据库中的数据。如果判断1数据库未失效，从0数据库取出数据返回。

1. 防止重复提交：

|  |
| --- |
| String lockKey = LockConstants.***LOCK\_KEY\_ACTIVE\_LEAGUE\_PREFIX*** + activateCarUnionDTO.getDealerId();  **if** (!redissonLockService.tryLock(lockKey)) {  **return** RespMsg.*fail*("重复请求");  //直接返回  } |

|  |
| --- |
| @Override  **public** **boolean** tryLock(String key,**long** waitTime, **long** leaseTime, TimeUnit unit) {  **try** {  RLock lock = redisson.getLock(key);  **boolean** res = lock.tryLock(waitTime, leaseTime, unit);  **return** res;  } **catch** (InterruptedException e) {  RedissonLockServiceImpl.***log***.error("获取redis锁异常, key is {}, timeout is {},leaseTime is {}, unit is {}.", key, waitTime,leaseTime, unit,  e);  **throw** **new** LockException(RespStatusEnum.***FAIL***,"缓存锁定失败");  }  } |

1. MQ消息队列

短信、推送消息发送；

数据最终一致性，分布式事务；

1. 削峰，顺序化请求，堆积能力
2. 分布式事务：数据最终一致性
3. 数据库增加一张日志记录表，利用本地事务，保证同时成功。然后发送消息，消费发送成功后，修改数据库日志状态为消费成功。
4. 数据库事务提交成功后，发送消息
5. 消费端保证幂等性
6. 人工补偿机制
7. 提供响应效率，不重要的请求不影响主流程。例如：短信，推送消息等
8. 提高系统高可用性：集群，持久化等
9. 订阅服务
10. 消息可靠性：消息重试
11. 消息过滤
12. 消息有序
13. 消息优先级
14. 回溯消息
15. 定时消息
16. Zookeeper/Dubbo
17. 生产者在注册中心注册服务
18. 消费者从注册中心获取生产者地址列表
19. 消费者调用生产者
20. Mysql
21. 主从热备
22. 业务分库：erp， merchant，car-center
23. 数据备份：每天
24. 特殊字段命名：valid，手型号，姓名，密码
25. 数据库优化
26. 索引
27. 汇总表
28. 选择合适的字段类型：tinyInt,datetime,
29. 适当的冗余字段
30. 查询优化：
31. 查询尽量少的字段，减少select \*查询
32. 利用索引查询
33. 避免关联查询
34. Where语句后面，过滤数据量大的放最后面
35. 使用预编译的SQL：可以使用缓存，避免注入问题
36. 数据库连接池
37. 微服务部署思想
38. 基础服务
39. 数据库分库
40. 熔断
41. 负载
42. 异常监控
43. 异常日志监控(邮件)
44. 服务器监控（短信）
45. 日志分析

1）ELK + HDFS + Spark + mysql

1. 行为统计
2. 友盟
3. 埋点
4. 数据仓库
5. fineReport
6. 报表
7. 数据分析
8. 单元测试

1）单元测试

2）压力测试

3) postman

1. 自动化服务
2. jenkins自动部署
3. docker容器
4. 部署方案
5. App 、WEB、ERP、merhcant、center、埋点 集群
6. 注册中心集群
7. Mysql 主从
8. Redis主从
9. MQ主从
10. 定时任务单机
11. 项目管理
12. worktitle
13. 早会
14. 文档
15. 把任务分配出去
16. 怎么避免相同的问题再次出现
17. 跟进客户问题
18. 慢查询跟进
19. 人员管理
20. 梯队培养
21. ONE2ONE
22. 代码review
23. 做事
24. 邮件确认
25. 安全：
26. https
27. 加密
28. 实现过滤器统一拦截
29. 对参数排序，拼接成字符串
30. 加密算法：MD5(秘钥+参数字符串+秘钥+时间戳)
31. 数据库存储密码时加密
32. 项目优化：
33. Lombok: @Data, @Slf4j
34. Guava:
35. 阿里编码规约扫描：**Alibaba Java Coding Guidelines**
36. GoogleStyle：File – Editor - Code style
37. 泛型工具类，统一处理返回结果：

|  |
| --- |
| **public** **static** <T>T resultWithCheck(RespDTO<T> respDTO, String serviceName) {  **if** (respDTO.getStatus() != RespStatusEnum.***SUCCESS***.getStatus()) {  *logger*.error("调用{}服务异常，返回结果，status:{},msg:{}",serviceName, respDTO.getStatus(), respDTO.getMsg());  **throw** **new** ServiceException(respDTO.getMsg());  }  **return** respDTO.getData();  } |

1. 自定义异常
2. 枚举
3. 字段唯一性设置
4. Jrebal 热部署
5. 项目中遇到的问题：
6. 引入本地jar
7. 幂等性
8. 内存分析
9. Jvisualvm
10. Linux 常用命令：
11. netstat –anp|grep port
12. ps –ef|grep 进程名
13. more fileName，然后查询
14. du –sh / df –h