JVT 技术分享记录

# PPT内容概要

## 功能演示

1. 登录
2. 注册
3. 找回密码
4. 图片/文件上传
5. 调度管理
6. 可视化配置应用

## 软件架构及功能设计

系统架构图

功能体系架构图

数据库详细设计图

## 技术点应用分析

1. **框架体系学习**
2. Spring Boot + MyBatis
3. Vue + Express +SASS
4. Node + JADE + SASS
5. **技术点学习**
6. Redis缓存：
7. 唯一性验证：
8. 输出实体控制：
9. 登录
10. 注册
11. 找回密码
12. 图片/文件上传
13. 调度管理
14. 可视化应用管理

## 小结与提问

# 功能点原理笔记记录

1. **Redis缓存**

**过期删除策略：定时删除、惰性删除、定期删除**

**定时删除：**

**【优点】** 对内存是最友好的:通过使用定时器，定时删除策略可以保证“过期键”会尽可能快地被删除，并释放“过期键”所占用的内存。

**【缺点】**对CPU时间是最不友好的:在“过期键”比较多的情况下，删除 过期键 这一行为可能会占用相当一部分CPU 时间，在内存不紧张但是CPU 时间非常紧张的情况下.将CPU 时间用在删除和当前任务无关的过期键上，无疑会对服务器的响应时间和吞吐量造成影响。

**惰性删除：**

**【优点】**对CPU 时间来说是最友好的:程序只会在取出键时才对键进行过期检查，这可以保证删除过期键的操作只会在非做不可的情况下进行， 并且删除的目标仅限于当前处理的键，这个策略不会在删除其他无关的过期键上花费任何CPU时间。

**【缺点】**对内存是最不友好的: 如果一个键已过期，而这个键又仍然保留在数据库中，那么只要这个过期键不被删除，它所占用的内存就不会释放。在使用惰性删除策略时，如果数据库中有非常多的过期键，而这些过期键又恰好没有被访问到的话，那么它们也许永远也不会被删除(除非用户手动执行FLUSHDB),我们甚至可以将这种情况看作是一种内存泄漏一一无用的垃圾数据占用了大量的内存，而服务器却不会自己去释放它们，这对于运行状态非常依赖于内存的Redis服务器来说，肯定不是一个好消息。

**定期删除：**

定期删除策略每隔一段时间执行一次删除过期键操作，并通过限制删除操作执行的时长和频率来减少删除操作对CPU 时间的影响。除此之外，通过定期删除过期键，定期删除策略有效地减少了因为过期键而带来的内存浪费。定期删除策略的难点是确定删除操作执行的时长和频率:如果删除操作执行得太频繁，或者执行的时间太长，定期删除策略就会退化成定时删除策略，以至于将C P U 时间过多地消耗在删除过期键上面。如果删除操作执行得太少，或者执行的时间太短，定期删除策略又会和惰性删除策略一样，出现浪费内存的情况。

因此，如果采用定期删除策略的话，服务器必须根据情况，合理地设置删除操作的执行时长和执行频率。

Reidis 过期删除策略

Redis服务器实际使用的是惰性删除和定期删除两种策略: 通过配合使用这两种删除策略，服务器可以很好地在合理使用CPU时间和避免浪费内存空间之间取得平衡。

1. **唯一性验证**

**【传统方案】**

业务字段唯一性验证，主要包含新增时字段唯一性验证和修改时字段唯一性验证；

在传统应用中，对于唯一性验证采用的是具体业务具体验证方式。

**【JVT项目采用方案】**

本项目采用的是配置式唯一性验证

在“过滤器”中将请求参数读取并获取reqeust参数并设置request唯一性验证的属性attribute,然后在“拦截器”中判断controller的“注解”及注解的相关信息，如果有验唯一性，获取验证唯一性相关信息和request 唯一性验证信息，结合两者进行相关逻辑的验证。

1. **输出实体字段控制**

**【区域项目采用方案】**

目前区域对于前端字段输出控制有三种方式：

1. 全部输出不做任何控制
2. 根据需要重新定义实体
3. 查询 字段 处理，输出空值方式

**【JVT项目采用方案】**

利用“注解”在ResponseBodyAdvice进行控制

1. **登录**

OAUTH2.0 授权登录

1. **注册**

密码加密：加盐加密

注册码管理：注册码时效性管理+注册码校验机制

1. **找回密码**

找回密码URL管理：找回密码URL管理机制

1. **图片/文件上传**

上传进度控制：

上传安全性管理：

1. **调度管理**
2. **可视化应用管理**