基于网络爬虫的数据取证平台

Electronic Data Forensics Platform
based on Web Crawler

答辩人：李想

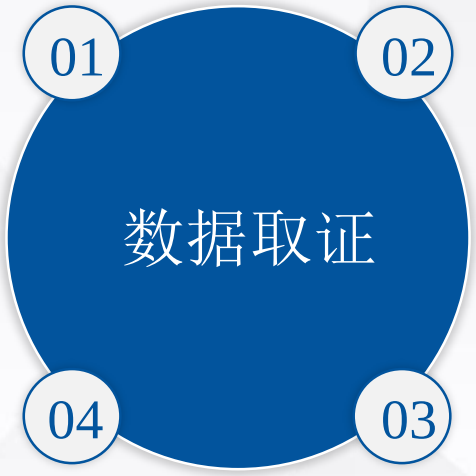
专业：信息工程

Contents

PART 01

研究背景

案例分析



自己辛辛苦苦写的文章未经允许被他人肆意转载！怎么办？

发现某网页上的内容涉及黄赌毒！怎么办？

最近视觉中国陷入图片版权门！想留证据，怎么办？

最近某明星在某视频网站上被黑的很惨，想拿到证据给对方发传票！怎么办？

研究目的

不用想怎么办了！快登录“爬虫取证平台”进行取证吧！

Attention:不要996！不要9.9！一律开源！免费！

微信公众号	一键、批量截图；下载原始图片、HTML文件
任意网页	批量、自动截图
视频网站	视频基本信息采集；视频下载

研究意义

高效	效率远远高于手工取证
权威	不存在篡改、伪造现象
及时	远程取证，随时随地无拘束
完整	取证内容丰富且完整
定制	可根据需求二次开发，拓展性强

PART 02

系统原理

技术选型

框架类型	框架名称	选择理由
系统架构	B/S 架构	B/S 架构仅仅要求用户有一台安装浏览器的可联网设备，就可以随时随地进行访问使用，无需安装本地应用软件
后端语言	Java	Java 语言具有与平台无关的特点，通过 Java 虚拟机技术，可以方便的把项目搭建到不同的操作系统之上，具有良好的跨平台特性
后端开发框架	SSM 框架（Spring+Spring MVC+MyBatis）	当前流行的进行开发。可以为系统提供良好的高并发性、可拓展性、可维护性。并且可大大提高开发效率，利于团队协作开发，简化开发流程，加快产品落地速度。
采集模块	Python	对爬虫十分擅长，并且可以方便的与 Java 后端服务进行交互，利于团队写作开发。
前端界面渲染框架	Vue.js	由华人开发的、当前最火的前端框架，易于上手，采用自底向上增量开发的设计。易于实现响应的数据绑定和组合的视图组件。
文件存储框架	FastDFS	开源的轻量级分布式文件系统，解决了大容量存储和负载均衡的问题，特别适合以文件为载体的在线服务。

SSM框架

框架（Framework）

应用程序的骨架。规范程序开发；方便拓展维护。

SSM框架（Spring、Spring MVC、MyBatis）

企业级
轻量
主流
高拓展性
易维护
规范性

SSM-Spring

概念

容器。融合现有技术；良好的编程习惯。

特性

IoC（控制反转）：将原本通过代码来手动创建对象的控制权反转，交给Spring容器来管理

AoP（面向切面编程）：将业务中常用的特殊操作如日志、事务等提炼出来，并在业务中切入这些操作，这些操作也称之为“横切逻辑”或切面

SSM-Spring MVC

概念

Spring中用于Web应用开发的优秀框架，延续经典的MVC思想

特性

MVC设计模式：分离出了Model(模型），View(视图)，Controller(控制器)

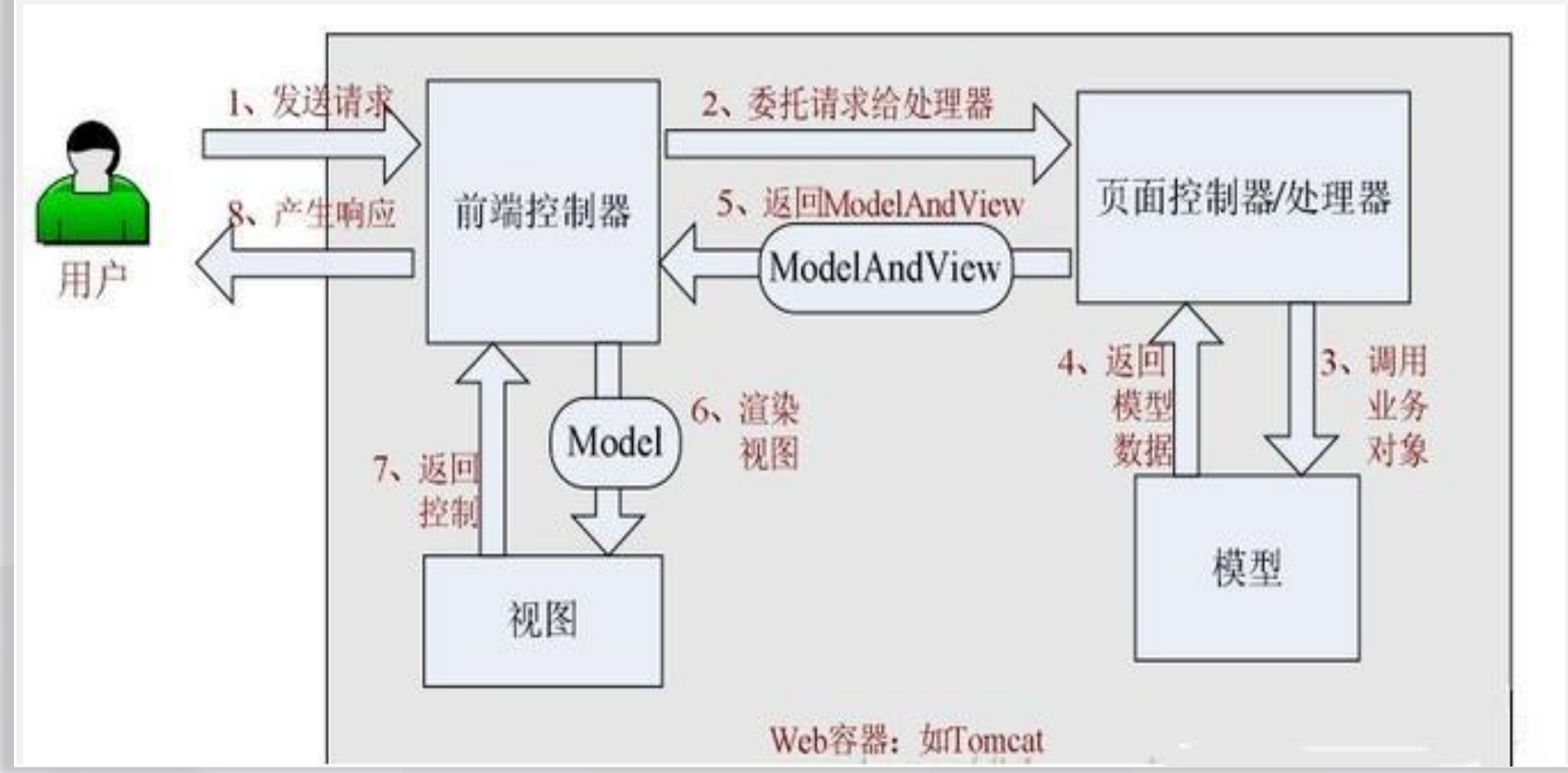
松耦合可拔插：灵活、高拓展性

原生支持Spring特性

REST风格

SSM-Spring MVC

Spring MVC请求处理流程



SSM-MyBatis

概念

数据持久层框架，它内部封装了JDBC访问数据库的操作

特性

ORM（对象/关系映射）：在对象模型和关系型数据库之间建立了映射关系

半自动化ORM：较好的性能、灵活性

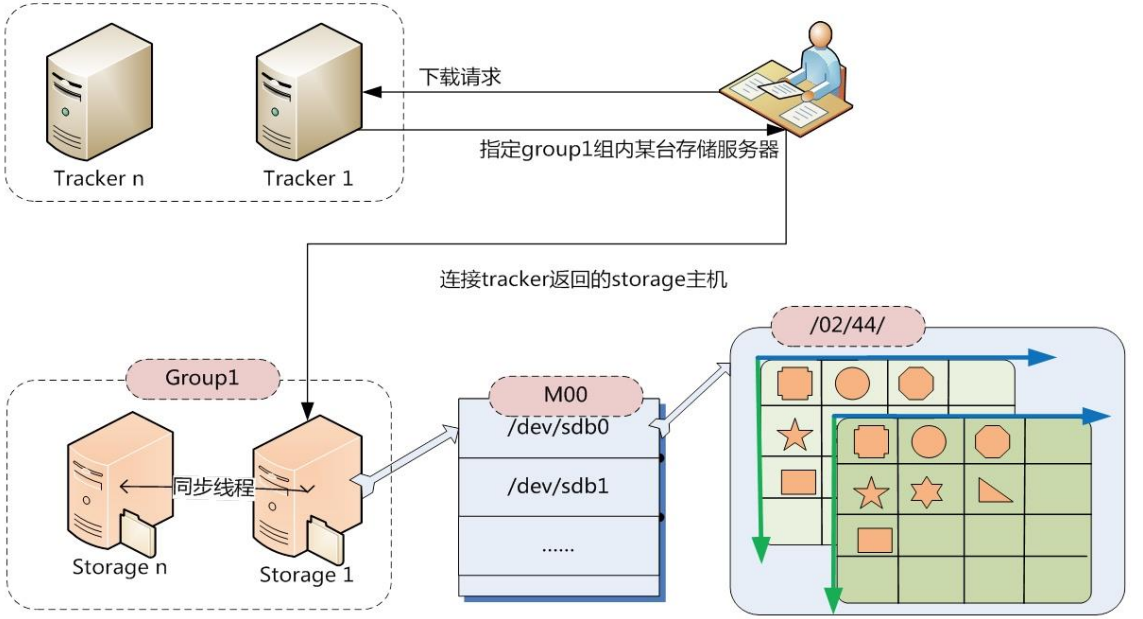
动态SQL

文件存储系统

FastDFS概念及特性

开源的轻量级分布式文件系统，具有冗余备份、负载均衡、高性能等特性。

原理



PART 03

设计实现

工作安排

分工

服务器后端
前端
数据库
文件存储模块

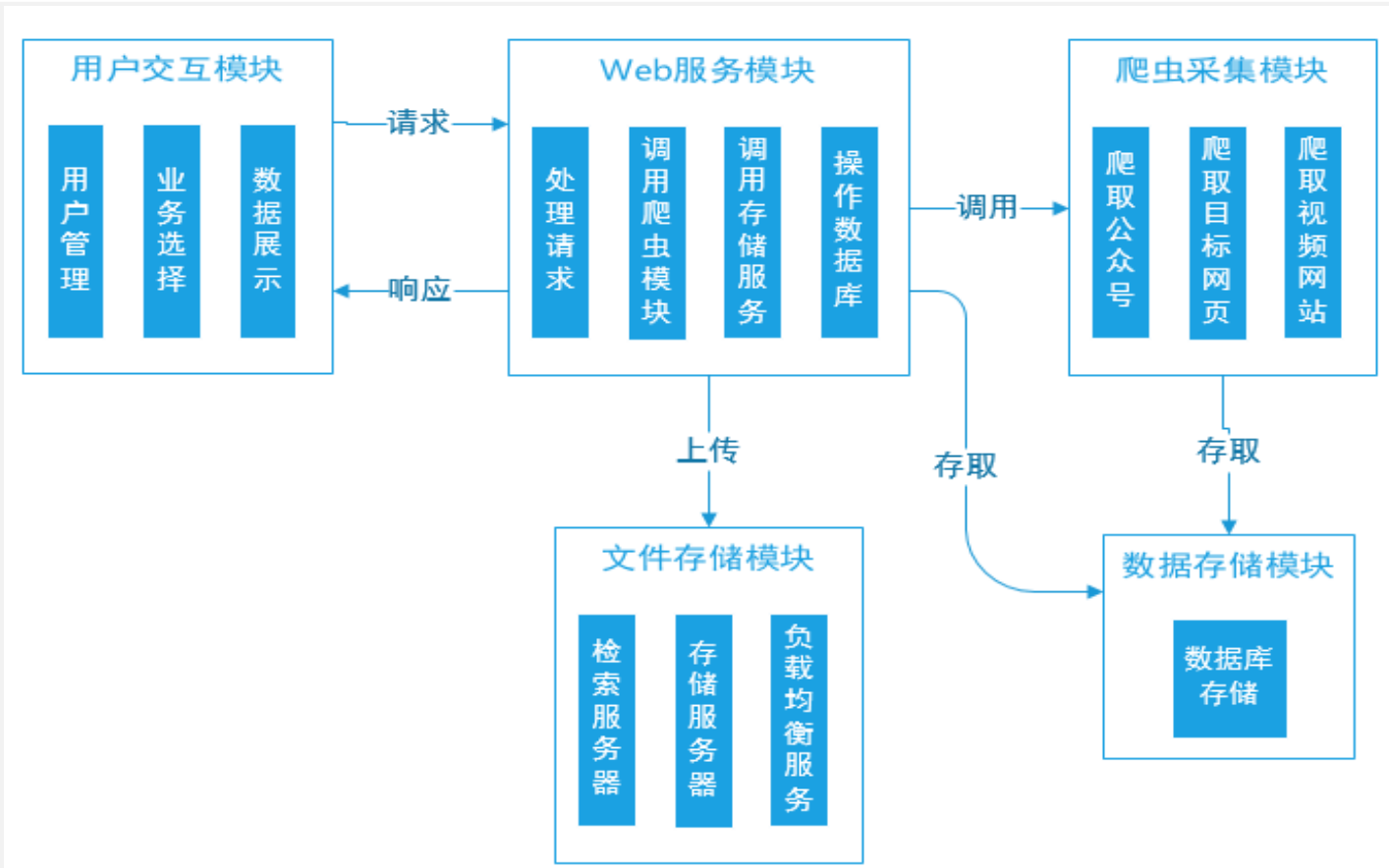
工作量

文件数多达： 53+
代码量高达： 7500+
阅读文献： 30+
阅读书籍： 3+
观看视频： 40+
管理服务器： 3

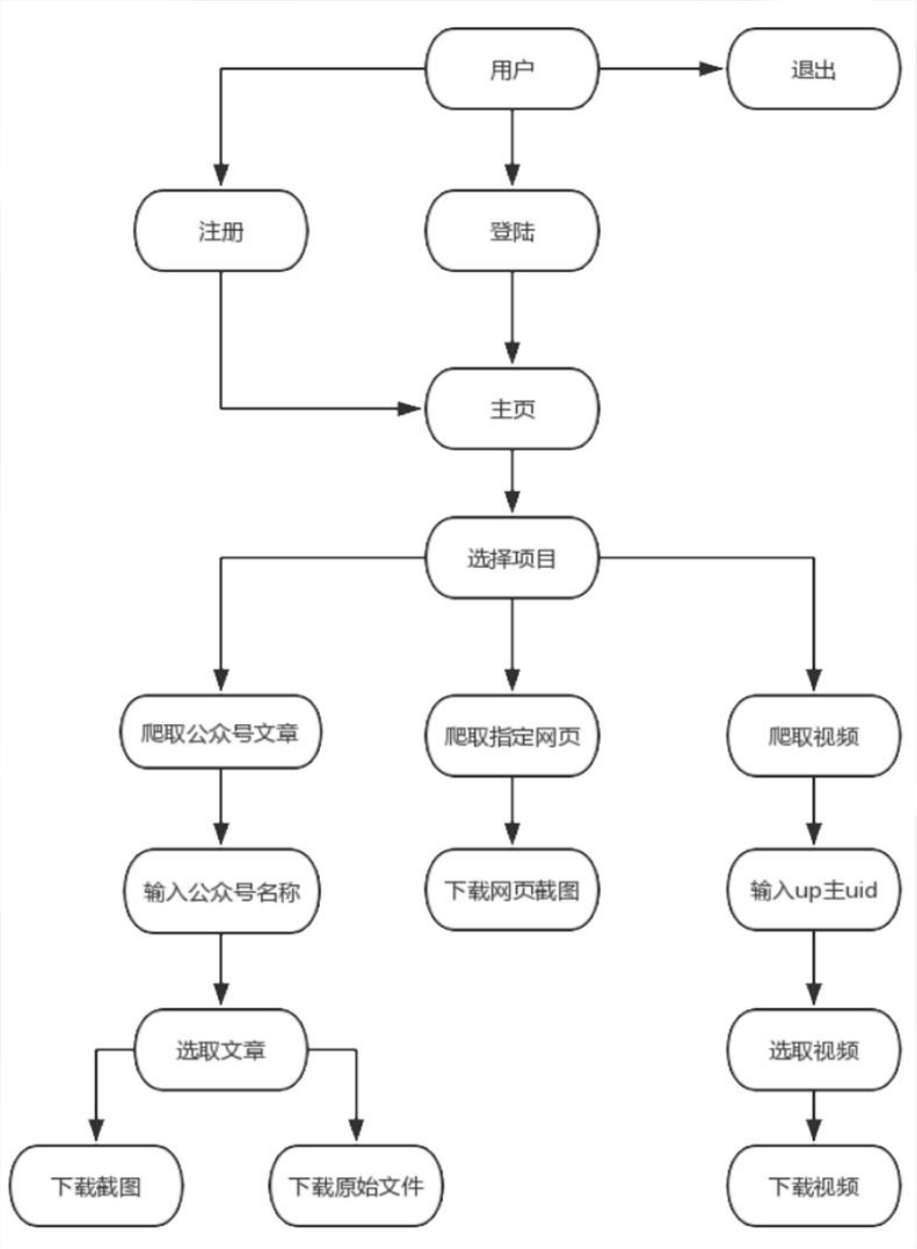
1. 阅读相关文献，小组讨论项目需求、框架、技术
2. 通过书籍、视频学习SSM框架、文件服务器知识
3. 继续学习相关知识，业务详细分析，编写代码完成后端业务逻辑
4. 学习前端框架完成前端界面；系统融合、优化、测试
5. 论文撰写完善

框架设计

系统框图

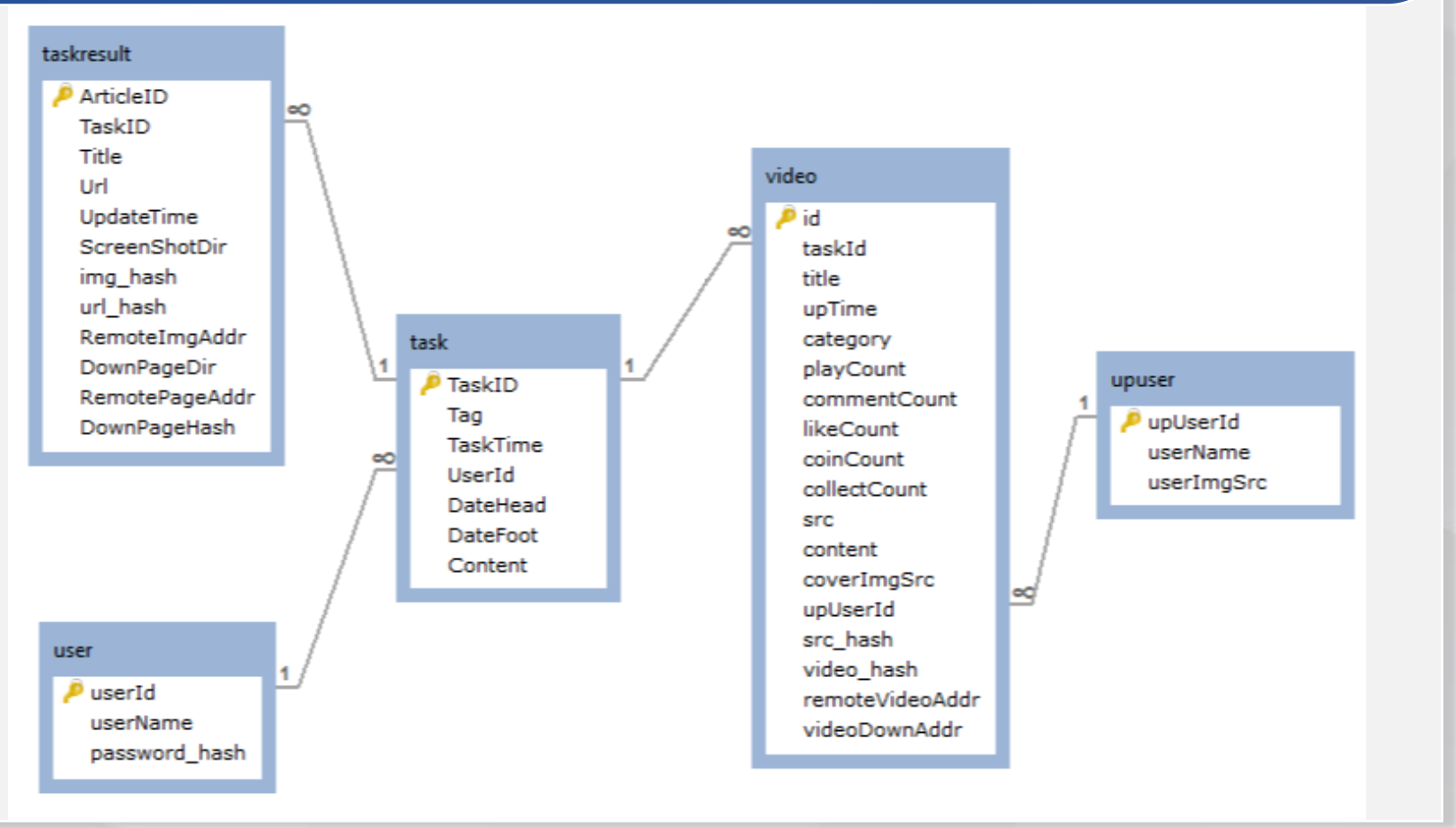

用户交互模块
用户管理
业务选择
数据展示
请求
响应
Web服务模块
处理请求
调用爬虫模块
调用存储服务
操作数据库
调用
爬虫采集模块
爬取公众号
爬取目标网页
爬取视频网站
上传
存取
存取
文件存储模块
检索服务器
存储服务器
负载均衡服务
数据存储模块
数据库存储

业务流程

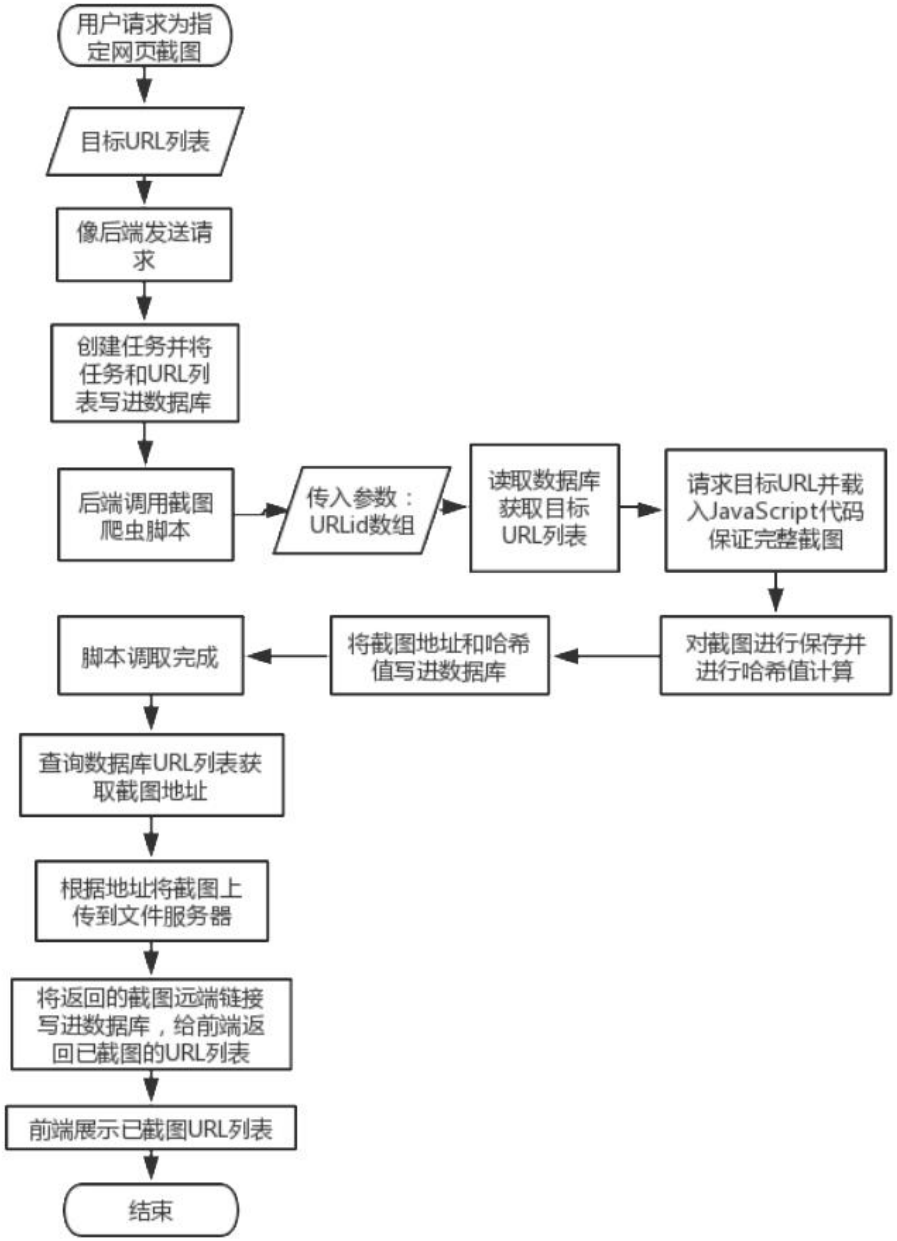

用户
退出
注册
登陆
主页
选择项目
爬取公众号文章
爬取指定网页
爬取视频
输入公众号名称
下载网页截图
输入up主uid
选取文章
选取视频
下载截图
下载原始文件
下载视频

数据层设计

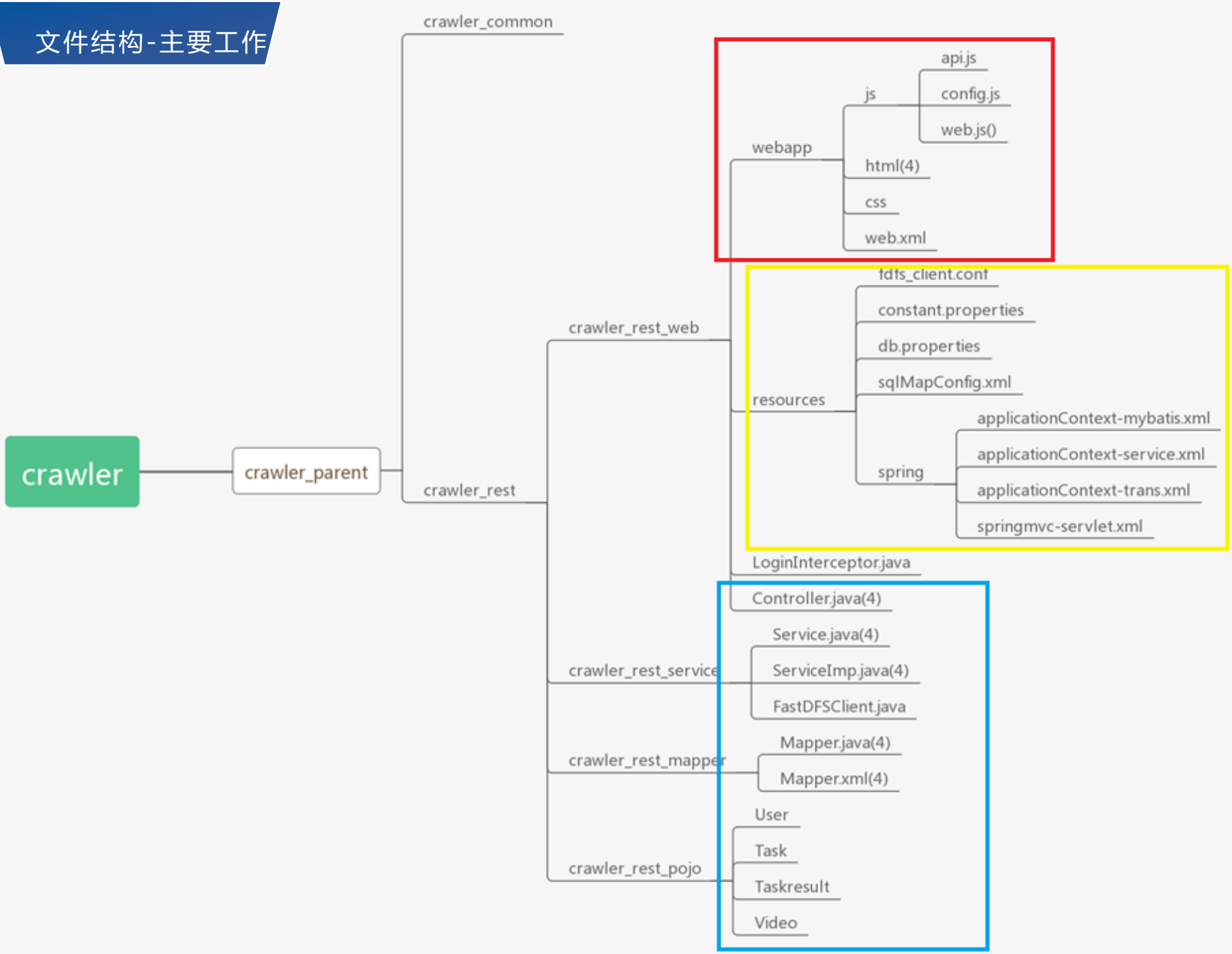
数据库关系视图



爬取网页算法流程


用户请求为指定网页截图
目标URL列表
像后端发送请求
创建任务并将任务和URL列表写进数据库
后端调用截图爬虫脚本
传入参数：URLid数组
读取数据库获取目标URL列表
请求目标URL并载入JavaScript代码保证完整截图
对截图进行保存并进行哈希值计算
将截图地址和哈希值写进数据库
脚本调取完成
查询数据库URL列表获取截图地址
根据地址将截图上传到文件服务器
将返回的截图远端链接写进数据库，给前端返回已截图的URL列表
前端展示已截图URL列表
结束

文件结构-主要工作


crawler
crawler_parent
crawler_common
crawler_rest
crawler_rest_web
webapp
api.js
js
config.js
web.js()
html(4)
css
web.xml
resources
fdfs_client.conf
constant.properties
db.properties
sqlMapConfig.xml
spring
applicationContext-mybatis.xml
applicationContext-service.xml
applicationContext-trans.xml
springmvc-servlet.xml
LoginInterceptor.java
Controller.java(4)
crawler_rest_service
Service.java(4)
ServiceImp.java(4)
FastDFSClient.java
crawler_rest_mapper
Mapper.java(4)
Mapper.xml(4)
crawler_rest_pojo
User
Task
Taskresult
Video

爬取网页代码实现

1. 前端请求
2. 后端Controller拦截
3. Controller调用Service接口
4. ServiceImp进行业务处理，调用Mapper进行数据库操作
5. Mapper通过XML构造动态SQL
6. 返回JSON数据给前端
7. 前端数据解析渲染

```
function urlscreen(params) {
    return service({
        url: '/task/url',
        method: 'post',
        data: JSON.stringify(params)
    })
}
```

爬取网页代码实现

1. 前端请求
2. 后端Controller拦截
3. Controller调用Service接口
4. ServiceImp进行业务处理，调用Mapper进行数据库操作
5. Mapper通过XML构造动态SQL
6. 返回JSON数据给前端
7. 前端数据解析渲染

```
 //截获请求
@RequestMapping("/url")
//使返回结果自动转化为json格式
@ResponseBody
//@RequestBody注解可以从请求body中接收json类型的参数并利用同名自动赋值给实体类
public List<Taskresult>  showUrl(HttpServletRequest request,
@RequestBody List<String> urllist){
    //从session中获取用户id
     HttpSession session = request.getSession();
     String userId = session.getAttribute("LOGIN_USER").toString();
     //调取Service层接口处理业务逻辑
     List<Taskresult> taskresultList = taskService.showUrlTaskresult
     (userId, urllist);
     //返回文章列表并转化为json格式
     return taskresultList;

}
```

爬取网页代码实现

1. 前端请求
2. 后端Controller拦截
3. Controller调用Service接口
4. ServiceImp进行业务处理，调用Mapper进行数据库操作
5. Mapper通过XML构造动态SQL
6. 返回JSON数据给前端
7. 前端数据解析渲染

```
 //截获请求
@RequestMapping("/url")
//使返回结果自动转化为json格式
@ResponseBody
//@RequestBody注解可以从请求body中接收json类型的参数并利用同名自动赋值给实体类
public List<Taskresult>  showUrl(HttpServletRequest request,
@RequestBody List<String> urllist){
    //从session中获取用户id
     HttpSession session = request.getSession();
     String userId = session.getAttribute("LOGIN_USER").toString();
     //调取Service层接口处理业务逻辑
     List<Taskresult> taskresultList = taskService.showUrlTaskresult
     (userId, urllist);
     //返回文章列表并转化为json格式
     return taskresultList;

}
```

爬取网页代码实现

1. 前端请求
2. 后端Controller拦截
3. Controller调用Service接口
4. ServiceImp进行业务处理，调用Mapper进行数据库操作
5. Mapper通过XML构造动态SQL
6. 返回JSON数据给前端
7. 前端数据解析渲染

```
TaskresultExample example = new TaskresultExample();
TaskresultExample.Criteria criteria = example.createCriteria()
;
criteria.andTaskidEqualTo(taskid);
taskresultList.clear();
//调用Mapper
taskresultList = taskresultMapper.selectByExample(example);
List<Integer> articleidlist = taskresultList.stream().map
(Taskresult::getArticleid).collect(Collectors.toList());
List<String> articleidls = articleidlist.stream().map
(a->a.toString()).collect(Collectors.toList());

//传articleid 列表给python,让其截图并将地址写入数据库

//将本地的图片传到fastdfs，并返回图片链接，并写进数据库，返回
resultlist
taskresultList.clear();
taskresultList = showScreen(articleidls);
return taskresultList;
```

爬取网页代码实现

1. 前端请求
2. 后端Controller拦截
3. Controller调用Service接口
4. ServiceImp进行业务处理，调用Mapper进行数据库操作
5. Mapper通过XML构造动态SQL
6. 返回JSON数据给前端
7. 前端数据解析渲染

```
<select id="selectByPrimaryKey" parameterType="java.lang.Integer"
resultMap="BaseResultMap">
  select
  <include refid="Base_Column_List" />
  from taskresult
  where ArticleID = #{articleid,jdbcType=INTEGER}
</select>
```

爬取网页代码实现

1. 前端请求
2. 后端Controller拦截
3. Controller调用Service接口
4. ServiceImp进行业务处理，调用Mapper进行数据库操作
5. Mapper通过XML构造动态SQL
6. 返回JSON数据给前端
7. 前端数据解析渲染

```
 //截获请求
@RequestMapping("/url")
//使返回结果自动转化为json格式
@ResponseBody
//@RequestBody注解可以从请求body中接收json类型的参数并利用同名自动赋值给实体类
public List<Taskresult>  showUrl(HttpServletRequest request,
@RequestBody List<String> urllist){
    //从session中获取用户id
     HttpSession session = request.getSession();
     String userId = session.getAttribute("LOGIN_USER").toString();
     //调取Service层接口处理业务逻辑
     List<Taskresult> taskresultList = taskService.showUrlTaskresult
     (userId, urllist);
     //返回文章列表并转化为json格式
     return taskresultList;

}
```

爬取网页代码实现

1. 前端请求
2. 后端Controller拦截
3. Controller调用Service接口
4. ServiceImp进行业务处理，调用Mapper进行数据库操作
5. Mapper通过XML构造动态SQL
6. 返回JSON数据给前端
7. 前端数据解析渲染

```
let article;
let articles = [];
for (let i = 0; i < this.rawdata.length; i++) {
    article = {
        articleid: this.rawdata[i]["articleid"],
        title: this.rawdata[i]["title"],
        url: this.rawdata[i]["url"],
        updatetime: new Date(this.rawdata[i]["updatetime"]
         * 1000).toLocaleDateString(),
        imgHash: this.rawdata[i]["imgHash"],
        remoteimgaddr: "http://185.22.152.75:8888/" +
        this.rawdata[i]["remoteimgaddr"],
    }
    articles.push(article);
}
```

PART 04

总结展望

成果展示

2min彩蛋

系统特点

企业级	定位：真正投入市场使用
分布式	证据爬取、数据存储、文件存储三管齐下
高拓展	可定制；易维护；易拓展
低耦合	前后端分离、各模块分离、各司其职
高可信	第三方存证、不可逆哈希校验、云备份

平台对比

与市场上的平台对比

	Crawler Forensics	其他取证平台
费用	免费	较高
门槛限制	较低、面向任何需要固定证据的用户，可以及时进行证据固定，以免证据被销毁或篡改。	较高，需要联系相关客服说明情况，然后缴费后才能使用，不利于及时取证。
开源	是	否
架构	B/S	多为 C/S
方便性	无需下载和安装	多数需要下载并安装
自由度	自由度高，可根据业务需要进行自由定制。	自由度低，需要联系公司相关人员进行讨论，确定需求。

难点亮点

脚本调取	传参问题；数据流读取问题
数据库优化	索引；批量添加数据；动态SQL
拦截器	登录状态验证
文件管理	抽离常量；抽离配置信息；抽离API请求

不足与展望

不足与展望

爬取过程等待时间还不够短。加强并发机制加快处理速度

与法律法规的融合度还不够高。加强与法律人员的合作与交流

爬取速度对网络速度比较依赖。改善服务器带宽条件。

THANKS

别走开
彩蛋来了