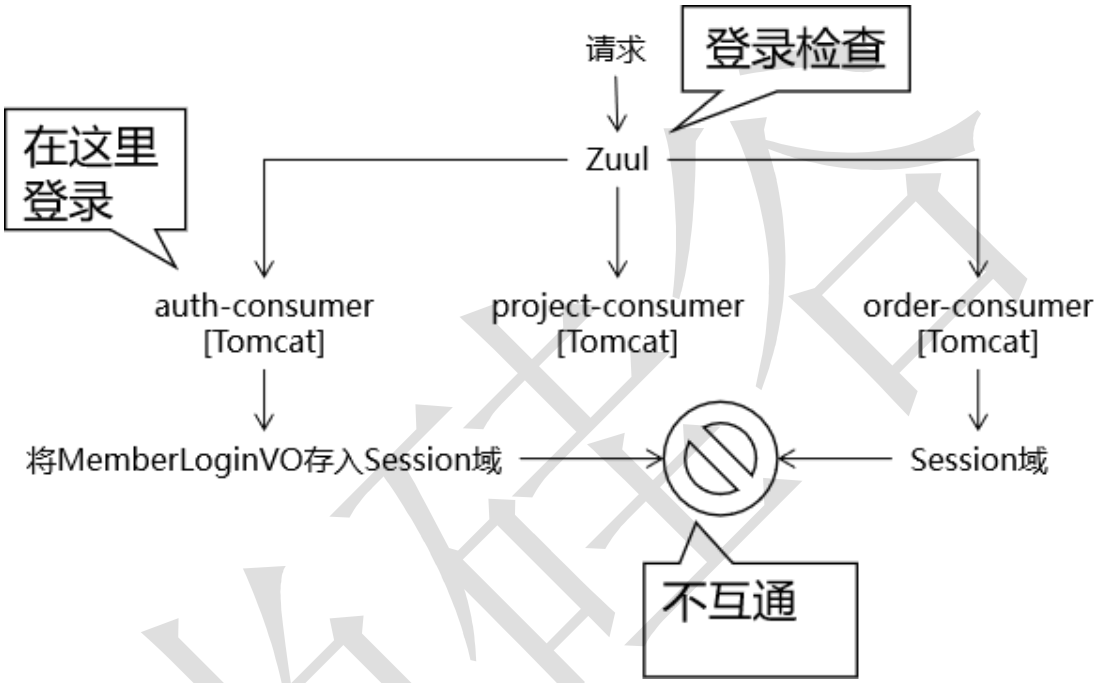
尚筹网

[16-尚硅谷-尚筹网-前台-会员登录]

1 会员登录功能延伸



新目标：使用 Session 共享技术解决 Session 不互通问题。

2 会话控制回顾

2.1 Cookie 的工作机制

服务器端返回 Cookie 信息给浏览器

Java 代码：response.addCookie(cookie 对象);

HTTP 响应消息头：Set-Cookie: Cookie 的名字=Cookie 的值

浏览器接收到服务器端返回的 Cookie，以后的每一次请求都会把 Cookie 带上

HTTP 请求消息头：Cookie： Cookie 的名字=Cookie 的值

2.2 Session 的工作机制

获取 Session 对象：request.getSession()

检查当前请求是否携带了 JSESSIONID 这个 Cookie

带了：根据这个 JSESSIONID 在服务器端查找对应的 Session 对象

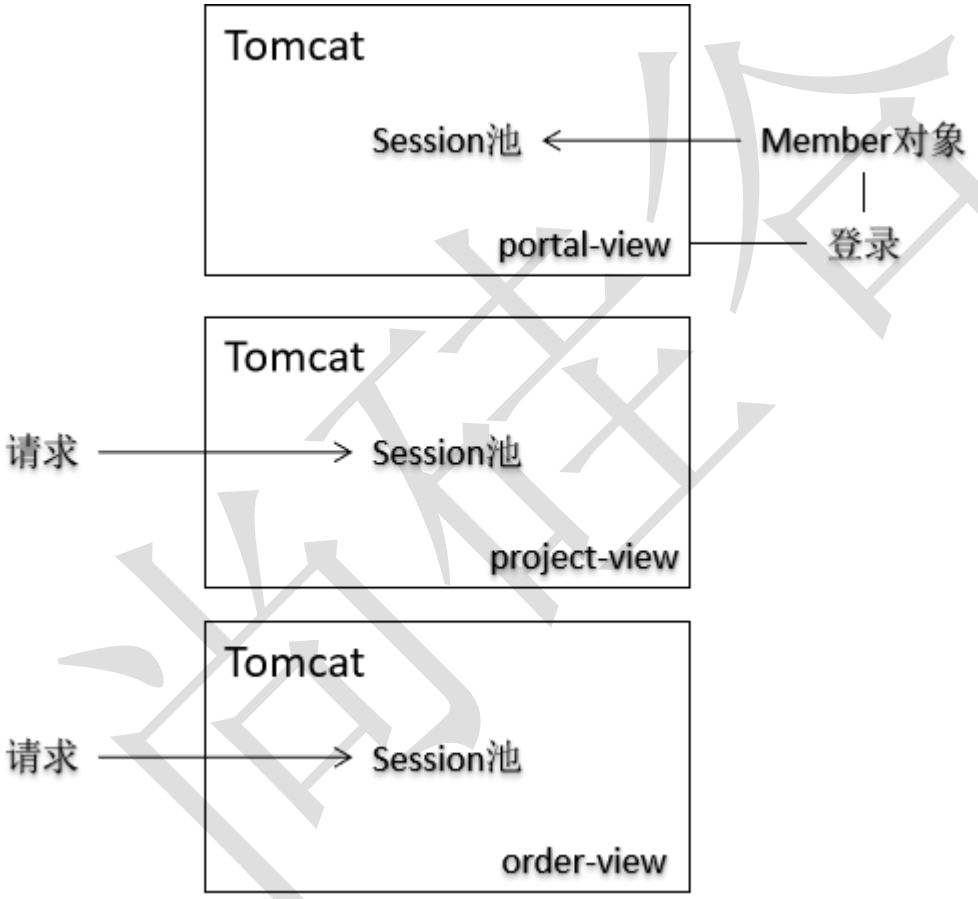
能找到：就把找到的 Session 对象返回

没找到：新建 Session 对象返回，同时返回 JSESSIONID 的 Cookie

没带：新建 Session 对象返回，同时返回 JSESSIONID 的 Cookie

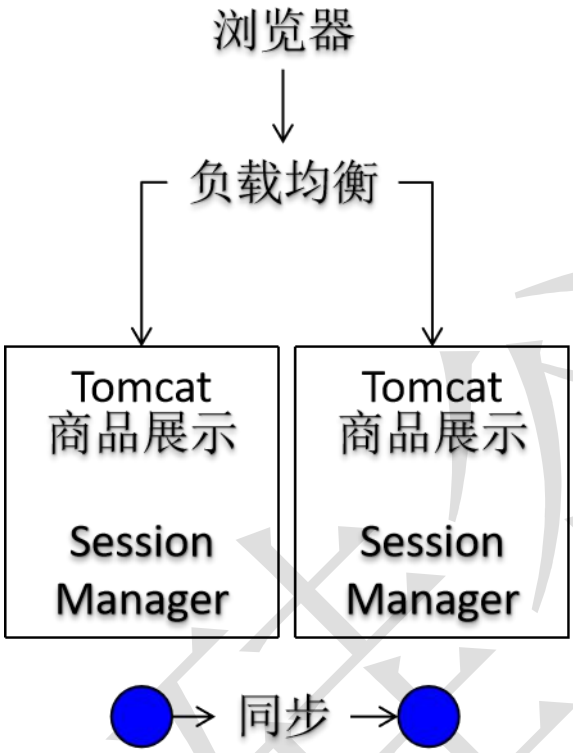
3 Session 共享

在分布式和集群环境下，每个具体模块运行在单独的 Tomcat 上，而 Session 是被不同 Tomcat 所“区隔”的，所以不能互通，会导致程序运行时，用户会话数据发生错误。有的服务器上有，有的服务器上没有。



3.1 解决方案探索

3.1.1 Session 同步



问题 1：造成 Session 在各个服务器上“同量”保存。TomcatA 保存了 1G 的 Session 数据，TomcatB 也需要保存 1G 的 Session 数据。数据量太大的会导致 Tomcat 性能下降。

问题 2：数据同步对性能有一定影响。

3.1.2 将 Session 数据存储在 Cookie 中

做法：所有会话数据在浏览器端使用 Cookie 保存，服务器端不存储任何会话数据。

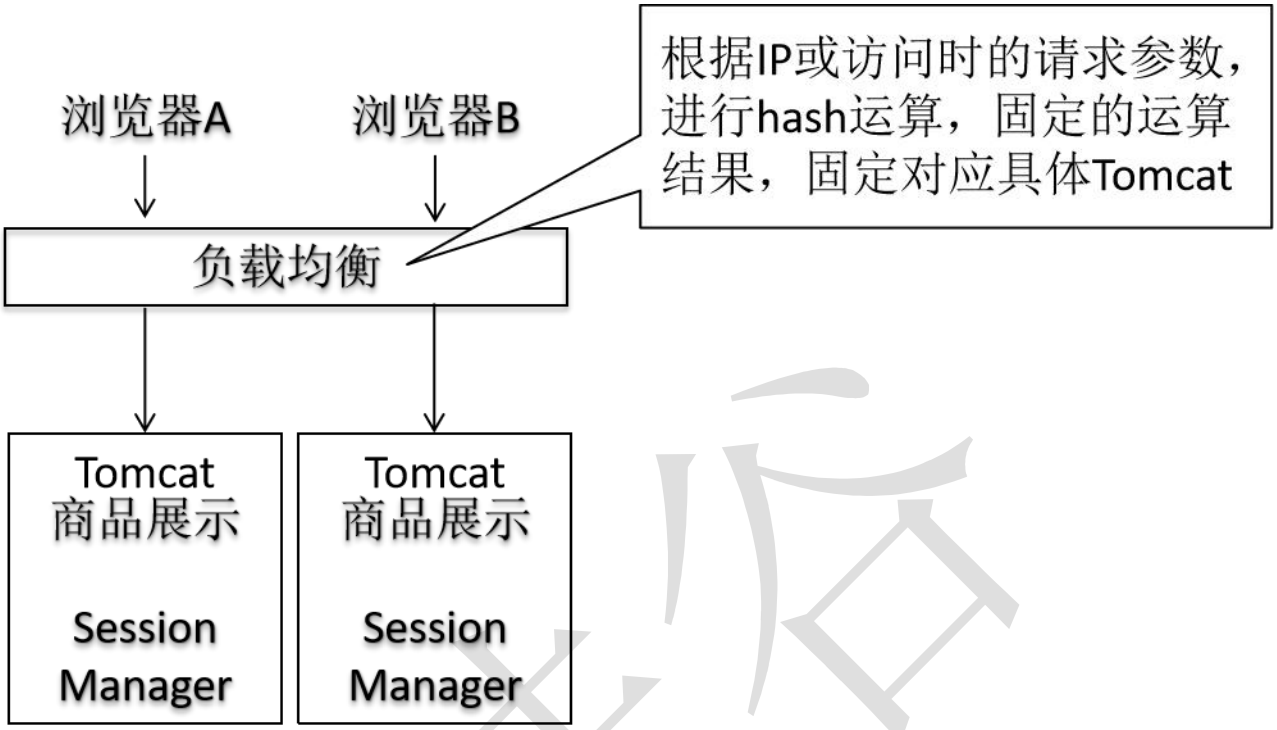
好处：服务器端大大减轻了数据存储的压力。不会有 Session 不一致问题

缺点：

Cookie 能够存储的数据非常有限。一般是 4KB。不能存储丰富的数据。

Cookie 数据在浏览器端存储，很大程度上不受服务器端控制，如果浏览器端清理 Cookie，相关数据会丢失。

3.1.3 反向代理 hash 一致性



问题 1：具体一个浏览器，专门访问某一个具体服务器，如果服务器宕机，会丢失数据。存在单点故障风险。

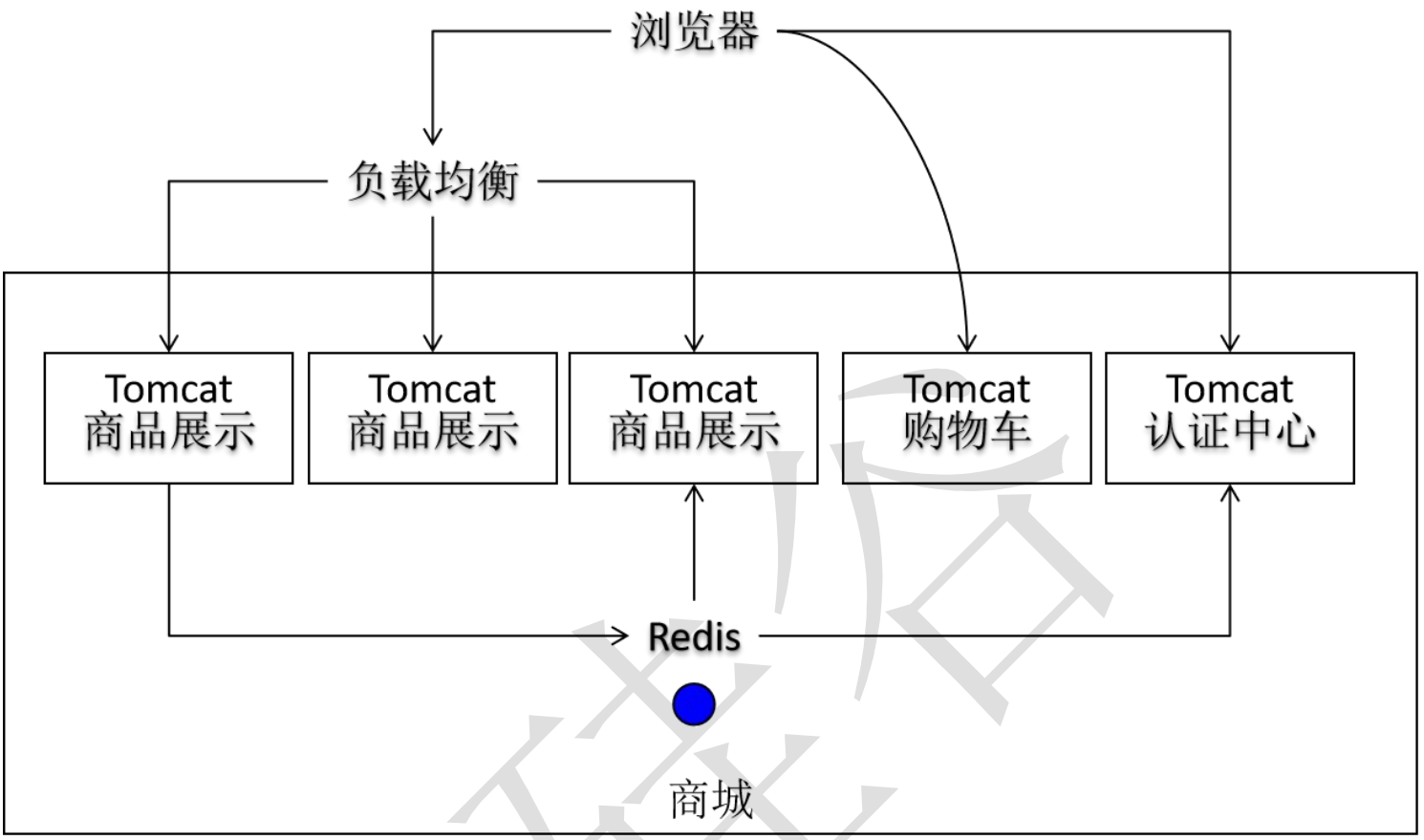
问题 2：仅仅适用于集群范围内，超出集群范围，负载均衡服务器无效。

3.1.4 后端统一存储 Session 数据

后端存储 Session 数据时，一般需要使用 Redis 这样的内存数据库，而一般不采用 MySQL 这样的关系型数据库。原因如下：

Session 数据存取比较频繁。内存访问速度快。

Session 有过期时间，Redis 这样的内存数据库能够比较方便实现过期释放。



优点

访问速度比较快。虽然需要经过网络访问，但是现在硬件条件已经能够达到网络访问比硬盘访问还要快。

硬盘访问速度：200M/s

网络访问速度：1G/s

Redis 可以配置主从复制集群，不担心单点故障。

3.2 SpringSession 使用

以下文档针对在 SpringBoot 环境下使用

3.2.1 引入依赖

```
<!-- 引入 springboot&redis 整合场景 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<!-- 引入 springboot&springsession 整合场景 -->
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-data-redis</artifactId>
```

```
</dependency>
```

3.2.2 编写配置

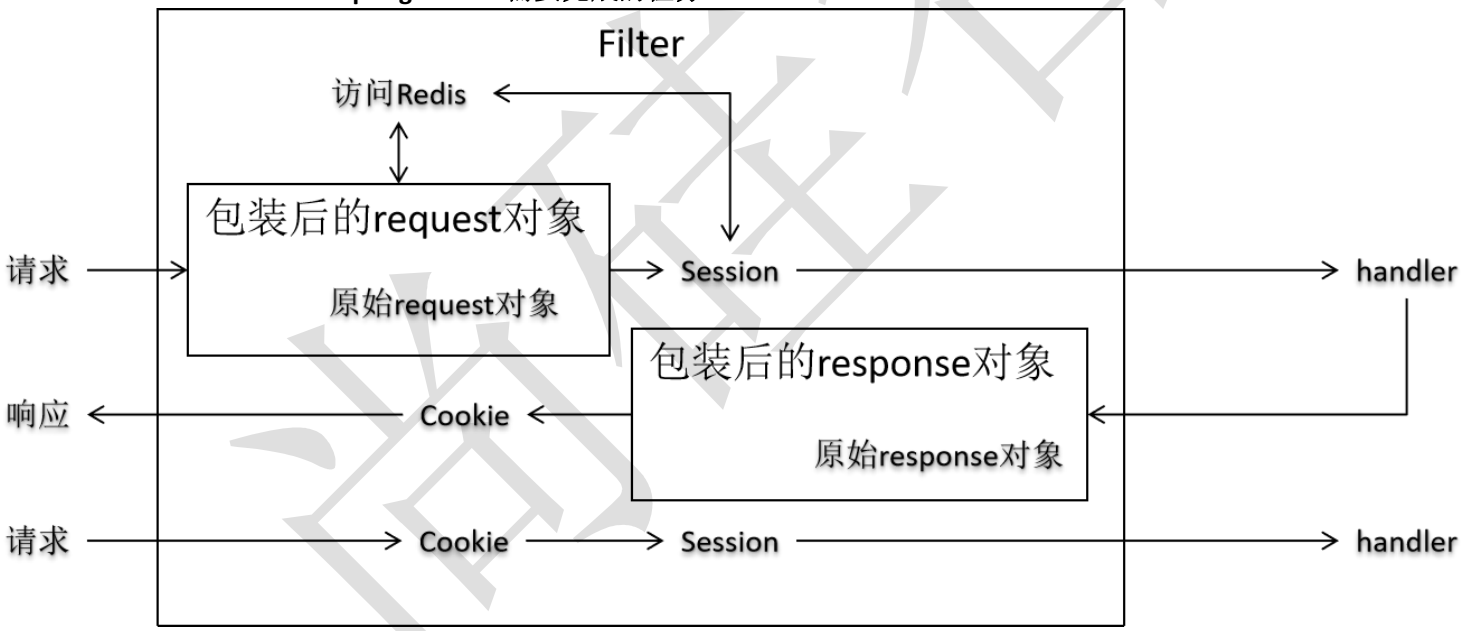
```
# redis 配置
spring.redis.host=192.168.56.100
spring.redis.jedis.pool.max-idle=100

# springsession 配置
spring.session.store-type=redis
```

※注意：存入 Session 域的实体类对象需要支持序列化!!!

3.3 SpringSession 基本原理

概括：SpringSession 从底层全方位“接管”了 Tomcat 对 Session 的管理。

3.3.1 SpringSession 需要完成的任务



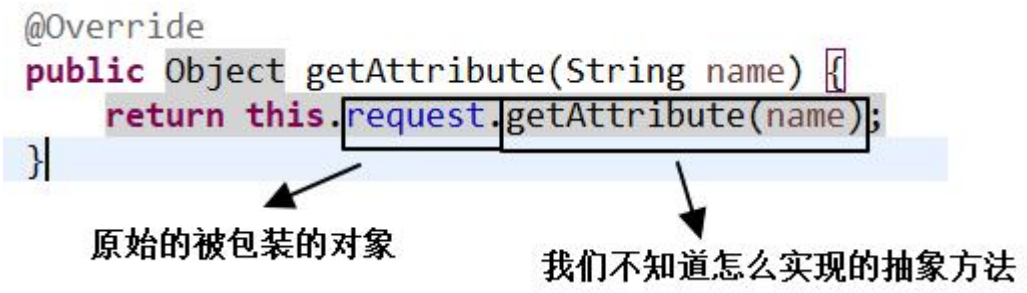
3.3.2 SessionRepositoryFilter

利用 Filter 原理，在每次请求到达目标方法之前，将原生 HttpServletRequest/HttpServletResponse 对象包装为 SessionRepositoryRequest/ResponseWrapper。

包装 request 对象时要做到：包装后和包装前类型兼容。所谓类型兼容：“包装得到的对象 instanceof 包装前类型”返回 true。

只有做到了类型的兼容，后面使用包装过的对象才能够保持使用方法不变。包装过的对象类型兼容、使用方法不变，才能实现“偷梁换柱”。

但是如果直接实现 HttpServletRequest 接口，我们又不知道如何实现各个抽象方法。这个问题可以借助原始被包装的对象来解决。

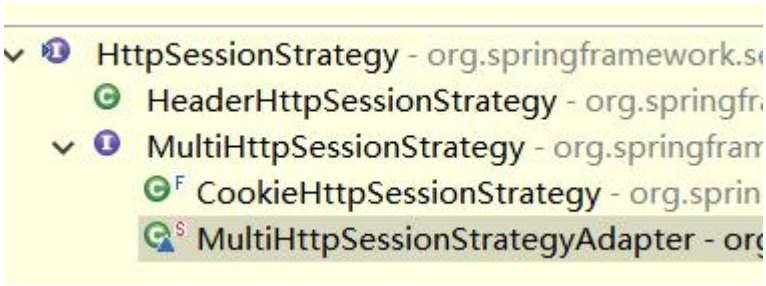


想要修改行为的方法，按照我们自己的需求编写代码即可。

```
@Override
protected void doFilterInternal(HttpServletRequest request,
        HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {
    request.setAttribute(SESSION_REPOSITORY_ATTR, this.sessionRepository);

    // 包装对象并不是凭空创建一个相同类型的对象，而是借助原生对象的主体功能，修改有特殊需要的功能。
    SessionRepositoryRequestWrapper wrappedRequest = new SessionRepositoryRequestWrapper(
            request, response, this.servletContext);
    SessionRepositoryResponseWrapper wrappedResponse = new SessionRepositoryResponseWrapper(
            wrappedRequest, response);

    HttpServletRequest strategyRequest = this.httpSessionStrategy
            .wrapRequest(wrappedRequest, wrappedResponse);
    HttpServletResponse strategyResponse = this.httpSessionStrategy
            .wrapResponse(wrappedRequest, wrappedResponse);

    try {
        filterChain.doFilter(strategyRequest, strategyResponse);
    }
    finally {
        wrappedRequest.commitSession();
    }
}
```

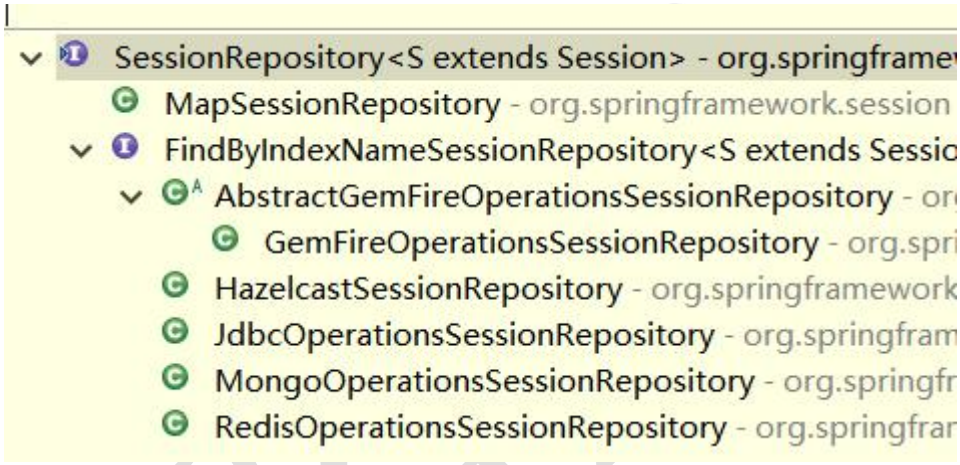
3.3.3 HttpSessionStrategy

封装 Session 的存取策略；cookie 还是 http headers 等方式；



3.3.4 SessionRepository

指定存取/删除/过期 session 操作的 repository



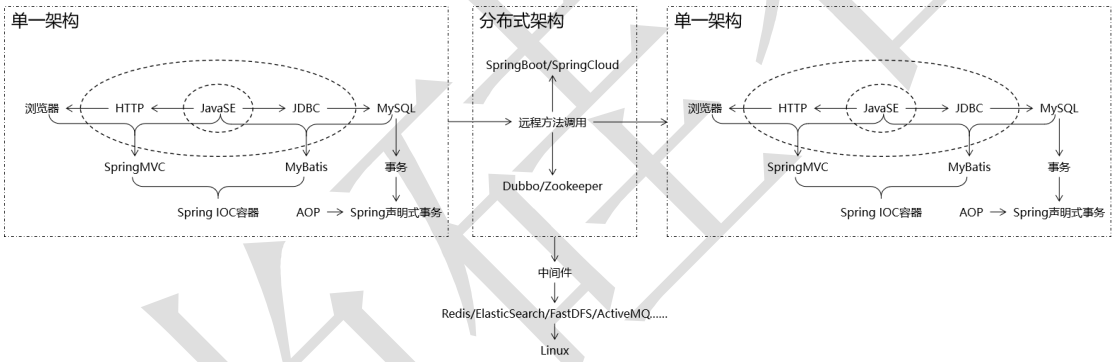
3.3.5 RedisOperationsSessionRepository

使用 Redis 将 session 保存维护起来

```
public RedisOperationsSessionRepository(
        RedisOperations<Object, Object> sessionRedisOperations) {
    Assert.notNull(sessionRedisOperations, "sessionRedisOperations cannot be
    this.sessionRedisOperations = sessionRedisOperations;
    this.expirationPolicy = new RedisSessionExpirationPolicy(sessionRedisOpe
            this);
}
```

```
public void save(RedisSession session) {
    session.saveDelta();
    if (session.isNew()) {
        String sessionCreatedKey = getSessionCreatedChannel(session.getId());
        this.sessionRedisOperations.convertAndSend(sessionCreatedKey, session
        session.setNew(false);
    }
}

@Scheduled(cron = "${spring.session.cleanup.cron.expression:0 * * * * *}")
public void cleanupExpiredSessions() {
    this.expirationPolicy.cleanExpiredSessions();
}
```

4 星图

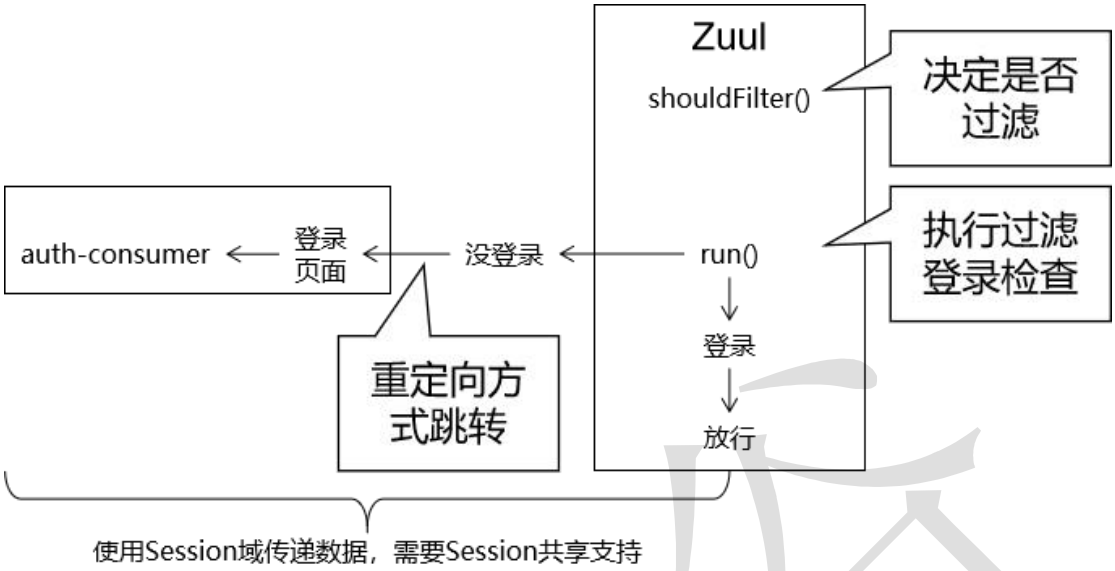


5 登录检查

5.1 目标

把项目中必须登录才能访问的功能保护起来，如果没有登录就访问则跳转到登录页面。

5.2 思路



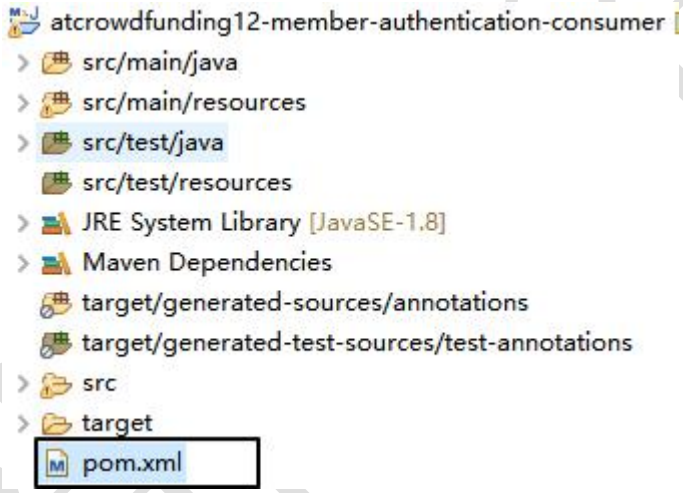
5.3 代码：设置 Session 共享

5.3.1 zuul 工程


```
<!-- 引入 springboot&redis 整合场景 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<!-- 引入 springboot&springsession 整合场景 -->
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-data-redis</artifactId>
</dependency>
```


```
spring:
  application:
    name: atguigu-crowd-zuul
  redis:
    host: 192.168.201.100
  session:
    store-type: redis
```

5.3.2 auth-consumer 工程

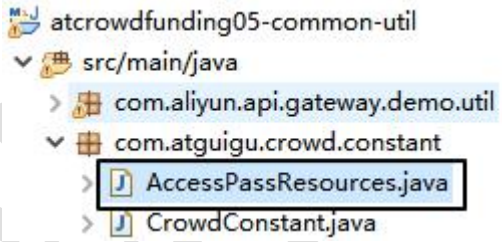



```
<!-- 引入 springboot&redis 整合场景 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<!-- 引入 springboot&springsession 整合场景 -->
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-data-redis</artifactId>
</dependency>
```


```
spring:
  application:
    name: atguigu-crowd-auth
  thymeleaf:
    prefix: classpath:/templates/
    suffix: .html
  redis:
    host: 192.168.201.100
  session:
    store-type: redis
```

5.4 代码：准备不需要登录检查的资源

5.4.1 准备好可以放行的资源




```
public static final Set<String> PASS_RES_SET = new HashSet<>();

static {
        PASS_RES_SET.add("/");
        PASS_RES_SET.add("/auth/member/to/reg/page");
        PASS_RES_SET.add("/auth/member/to/login/page");
        PASS_RES_SET.add("/auth/member/logout");
        PASS_RES_SET.add("/auth/member/do/login");
        PASS_RES_SET.add("/auth/do/member/register");
        PASS_RES_SET.add("/auth/member/send/short/message.json");
}

public static final Set<String> STATIC_RES_SET = new HashSet<>();

static {
```

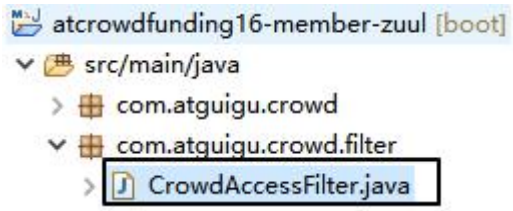
```
        STATIC_RES_SET.add("bootstrap");
        STATIC_RES_SET.add("css");
        STATIC_RES_SET.add("fonts");
        STATIC_RES_SET.add("img");
        STATIC_RES_SET.add("jquery");
        STATIC_RES_SET.add("layer");
        STATIC_RES_SET.add("script");
        STATIC_RES_SET.add("ztree");
}
```

5.4.2 判断当前请求是否为静态资源

```
/**
 * 用于判断某个 ServletPath 值是否对应一个静态资源
 * @param servletPath
 * @return
 *         true：是静态资源
 *         false：不是静态资源
 */
public static boolean judgeCurrentServletPathWetherStaticResource(String servletPath) {

    // 1.排除字符串无效的情况
    if(servletPath == null || servletPath.length() == 0) {
        throw new RuntimeException(CrowdConstant.MESSAGE_STRING_INVALIDATE);
    }

    // 2.根据“/”拆分 ServletPath 字符串
    String[] split = servletPath.split("/");

    // 3.考虑到第一个斜杠左边经过拆分后得到一个空字符串是数组的第一个元素，所以需要使用下标 1 取第二个元素
    String firstLevelPath = split[1];

    // 4.判断是否在集合中
    return STATIC_RES_SET.contains(firstLevelPath);
}
```

5.5 代码：ZuulFilter

5.5.1 创建 ZuulFilter 类




```
@Component
public class CrowdAccessFilter extends ZuulFilter {
```

5.5.2 shouldFilter()方法

```
@Override
public boolean shouldFilter() {

    // 1.获取 RequestContext 对象
    RequestContext requestContext = RequestContext.getCurrentContext();

    // 2.通过 RequestContext 对象获取当前请求对象（框架底层是借助 ThreadLocal 从当前线程上获取事先绑定的 Request 对象）
    HttpServletRequest request = requestContext.getRequest();

    // 3.获取 servletPath 值
    String servletPath = request.getServletPath();

    // 4.根据 servletPath 判断当前请求是否对应可以直接放行的特定功能
    boolean containsResult = AccessPassResources.PASS_RES_SET.contains(servletPath);

    if(containsResult) {

        // 5.如果当前请求是可以直接放行的特定功能请求则返回 false 放行
        return false;
    }

    // 5.判断当前请求是否为静态资源
    // 工具方法返回 true：说明当前请求是静态资源请求，取反为 false 表示放行不做登录检查
    // 工具方法返回 false：说明当前请求不是可以放行的特定请求也不是静态资源，取反为 true 表示需要做登录检查
    return !AccessPassResources.judgeCurrentServletPathWetherStaticResource(servletPath);
```

```
}
```

5.5.3 run()方法

```
@Override
public Object run() throws ZuulException {

    // 1.获取当前请求对象
    RequestContext requestContext = RequestContext.getCurrentContext();
    HttpServletRequest request = requestContext.getRequest();

    // 2.获取当前 Session 对象
    HttpSession session = request.getSession();

    // 3.尝试从 Session 对象中获取已登录的用户
    Object loginMember = session.getAttribute(CrowdConstant.ATTR_NAME_LOGIN_MEMBER);

    // 4.判断 loginMember 是否为空
    if(loginMember == null) {

        // 5.从 requestContext 对象中获取 Response 对象
        HttpServletResponse response = requestContext.getResponse();

        // 6.将提示消息存入 Session 域
        session.setAttribute(CrowdConstant.ATTR_NAME_MESSAGE,
CrowdConstant.MESSAGE_ACCESS_FORBIDEN);

        // 7.重定向到 auth-consumer 工程中的登录页面
        try {
            response.sendRedirect("/auth/member/to/login/page");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    return null;
}
```

5.5.4 filterType()方法

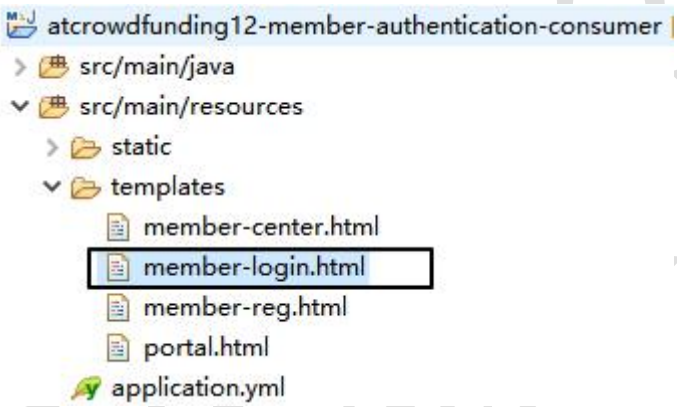
```
@Override
public String filterType() {
```

```
    // 这里返回“pre”意思是在目标微服务前执行过滤
    return "pre";
}
```

5.5.5 filterOrder()

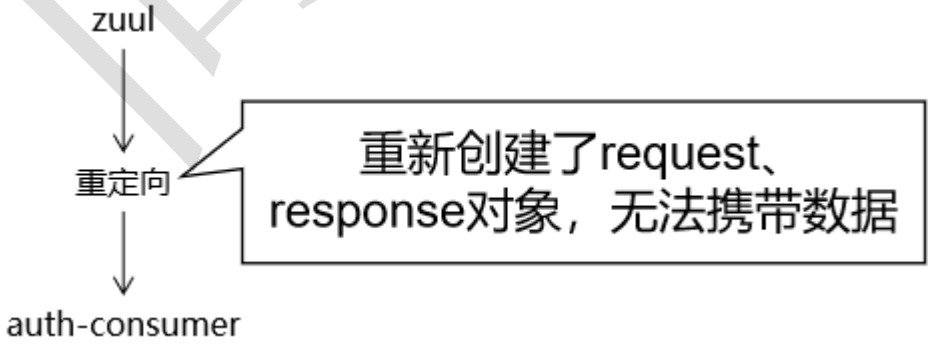
```
    @Override
    public int filterOrder() {
        return 0;
    }
```

5.6 代码：登录页面读取 Session 域




```
<p th:text="${session.message}">这里登录检查后发现不允许访问时的提示消息</p>
```

5.7 代码：Zuul 中的特殊设置



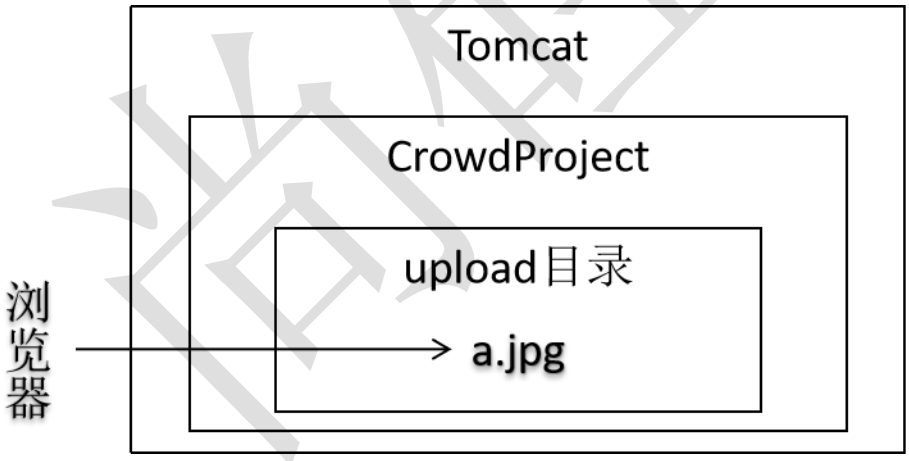
为了能够让整个过程中保持 Session 工作正常，需要加入配置：


```
zuul:
  ignored-services: "*"
  sensitive-headers: "*"   # 在 Zuul 向其他微服务重定向时保持原本头信息（请求头、响应头）
  routes:
    crowd-portal:
      service-id: atguigu-crowd-auth
      path: /** # 这里一定要使用两个“*”号，不然“/”路径后面的多层路径将无法访问
```

6 阿里云的 OSS 对象存储

6.1 提出问题

6.1.1 以前上传文件时保存位置

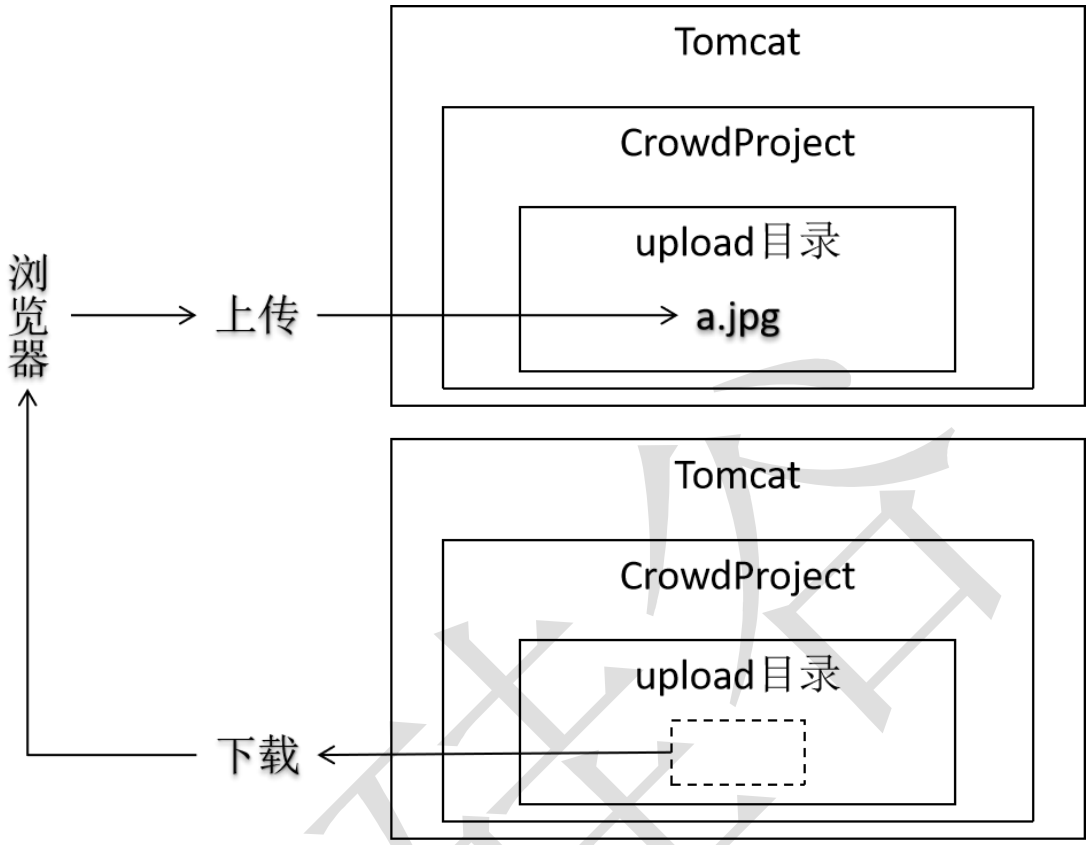


6.1.2 问题 1：Web 应用重新部署导致文件丢失

重新部署 Web 应用时，卸载（删除）旧的 Web 应用，连同用户上传的文件一起删除。重新加载新的 Web 应用后以前用户上传的文件不会自动恢复。

危害总结：Web 应用重新部署会导致用户上传的文件丢失。

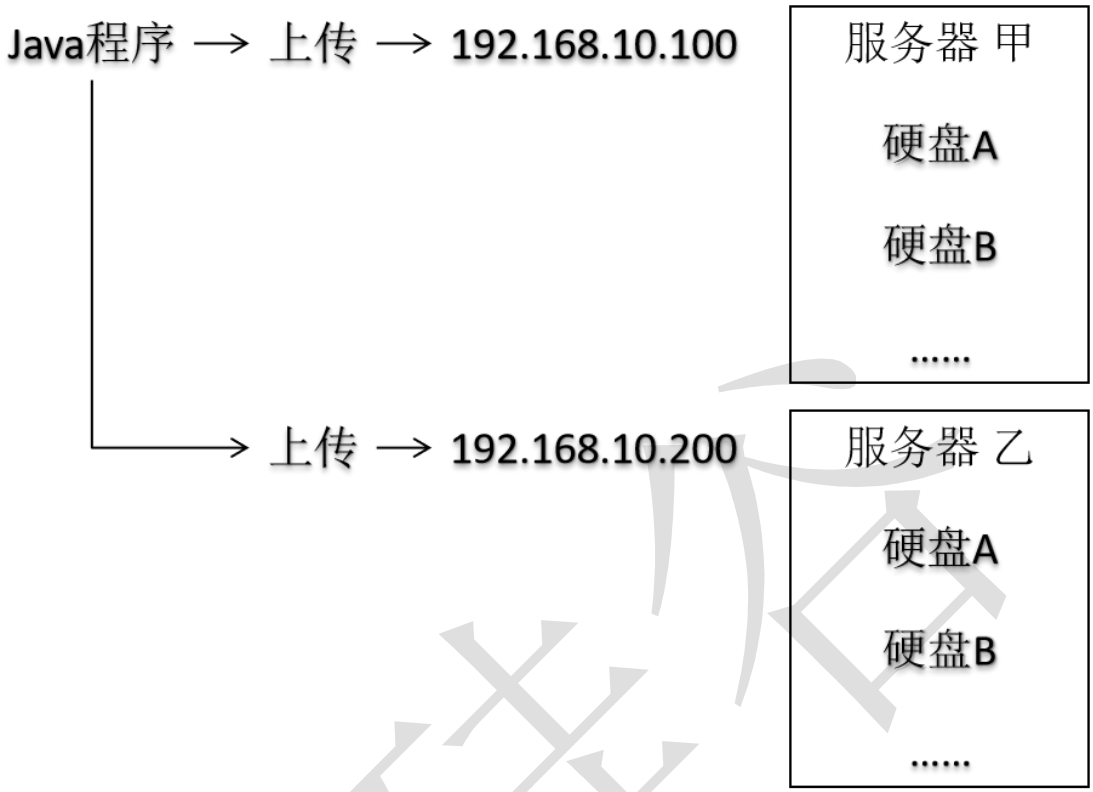
6.1.3 问题 2：集群环境下文件难以同步



6.1.4 问题 3：Tomcat 被拖垮

用户上传的文件如果数据量膨胀到了一个非常庞大的体积，那么就会严重影响 Tomcat 的运行效率。

6.1.5 问题 4：服务器存储自动扩容问题



危害总结：手动对服务器进行扩容，有可能导致项目中其他地方需要进行连带修改。

6.2 解决方案介绍

6.2.1 自己搭建文件服务器

举例：FastDFS

好处：服务器可以自己维护、自己定制。

缺点：需要投入的人力、物力更多。

适用：规模比较大的项目，要存储海量的文件。

6.2.2 使用第三方云服务

举例：阿里云提供的 OSS 对象存储服务。

好处：不必自己维护服务器的软硬件资源。直接调用相关 API 即可操作，更加轻量级。

缺点：数据不在自己手里。服务器不由自己维护。

适用：较小规模的应用，文件数据不是绝对私密。

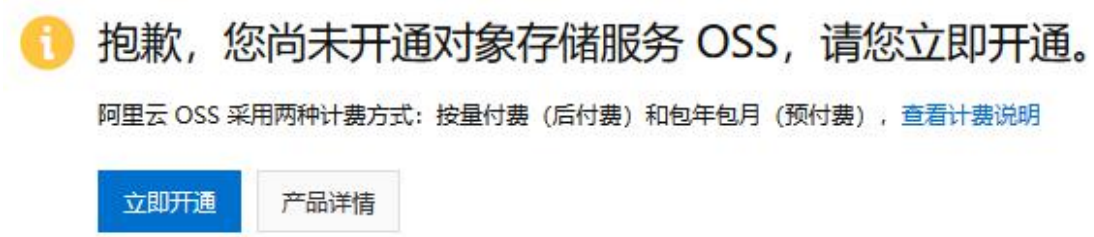
6.3 开通 OSS 服务步骤

注册阿里云账号

完成实名认证

登录后在左侧边栏找到对象存储 OSS

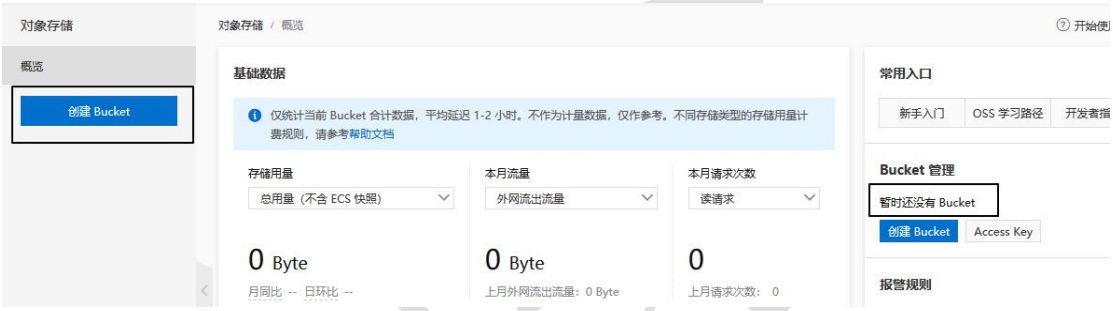
点击立即开通



勾选后点击立即开通

开通成功

打开 OSS 控制台



6.4 OSS 使用

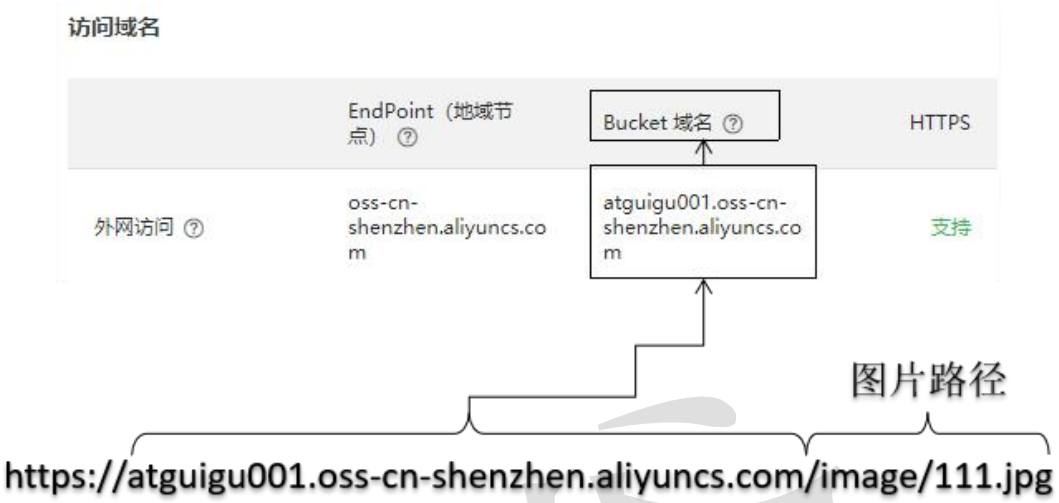
创建 Bucket

在 bucket 中创建目录

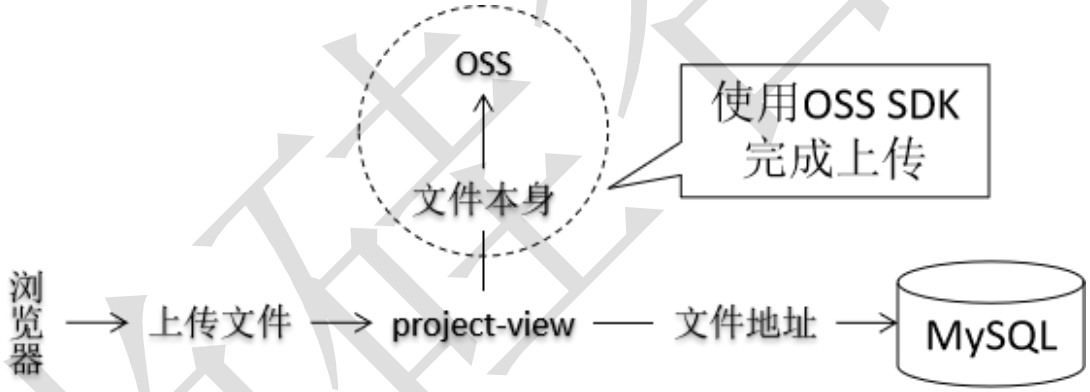
上传文件

上传成功

浏览器访问路径组成



6.5 Java 程序调用 OSS 服务接口



6.5.1 参考文档地址

https://help.aliyun.com/product/31815.html?spm=a2c4g.11186623.6.540.5e9a58d5ZnuSyZ

6.5.2 官方介绍

阿里云对象存储服务（Object Storage Service，简称 OSS），是阿里云提供的海量、安全、低成本、高可靠的云存储服务。您可以通过调用 API，在任何应用、任何时间、任何地点上传和下载数据，也可以通过 Web 控制台对数据进行简单的管理。OSS 适合存放任意类型的文件，适合各种网站、开发企业及开发者使用。按实际容量付费真正使您专注于核心业务。

6.5.3 创建 AccessKey

[1] 介绍

阿里云账号、密码→登录后在网页上操作

AccessKey→Java 程序登录 OSS 进行操作

访问密钥 AccessKey（AK）相当于登录密码，只是使用场景不同。AccessKey 用于程序方式调用云服务 API，而登录密码用于登录控制台。如果您不需要调用 API，那么就不需要创建 AccessKey。

您可以使用 AccessKey 构造一个 API 请求（或者使用云服务 SDK）来操作资源。AccessKey 包括 AccessKeyId 和 AccessKeySecret。

AccessKeyId 用于标识用户，相当于账号。
AccessKeySecret 是用来验证用户的密钥。AccessKeySecret 必须保密。

警告 禁止使用主账号 AK，因为主账号 AK 泄露会威胁您所有资源的安全。请使用子账号（RAM 用户）AK 进行操作，可有效降低 AK 泄露的风险。

[2] 创建子账号 AK 的操作步骤

1.使用主账号登录 RAM 管理控制台。

2.如果未创建 RAM 用户，在左侧导航栏，单击用户管理，然后单击新建用户，创建 RAM 用户。如果已创建 RAM 用户，跳过此步骤。

3.在左侧导航栏，单击用户管理，然后单击需要创建 AccessKey 的用户名，进入用户详情页面。

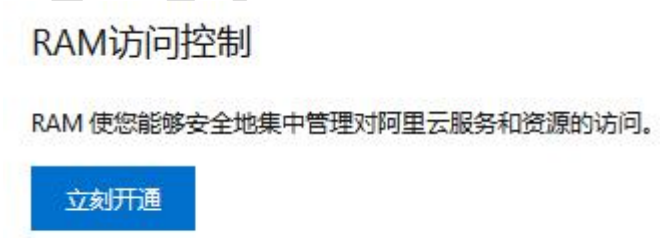
4.在用户 AccessKey 区域，单击创建 AccessKey。

5.完成手机验证后，在新建用户 AccessKey 页面，展开 AccessKey 详情，查看 AcessKeyId 和 AccessKeySecret。然后单击保存 AK 信息，下载 AccessKey 信息。

注意 AccessKey 创建后，无法再通过控制台查看。请您妥善保存 AccessKey，谨防泄露。

6.单击该 RAM 用户对应的授权，给 RAM 用户授予相关权限，例如 AliyunOSSFullAccess 将给 RAM 用户授予 OSS 的管理权限。

[3] 操作步骤截图



[4] 创建结果

用户登录名称 atguigu190830oss@1581777492970661.onaliyun.com

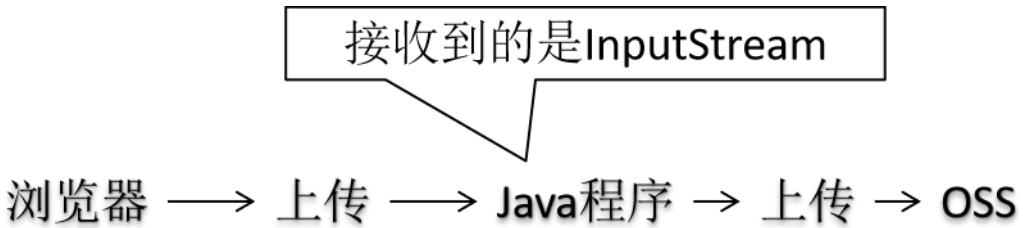
AccessKey ID LTAI4FtyK7WcqxaiE9ttukNi

AccessKeySecret 43OFOgMghjsitNsNlTGqcpPGgyzqgU

[5] SDK 参考

JDK：Java Development Kit

SDK：Software Development Kit

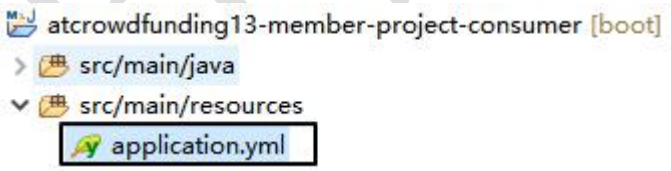


6.6 将 OSS 引入项目

6.6.1 准备 OSSProperties 类


```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Component
@ConfigurationProperties(prefix = "aliyun.oss")
public class OSSProperties {

    private String endPoint;
    private String bucketName;
    private String accessKeyId;
    private String accessKeySecret;
    private String bucketDomain;

}
```

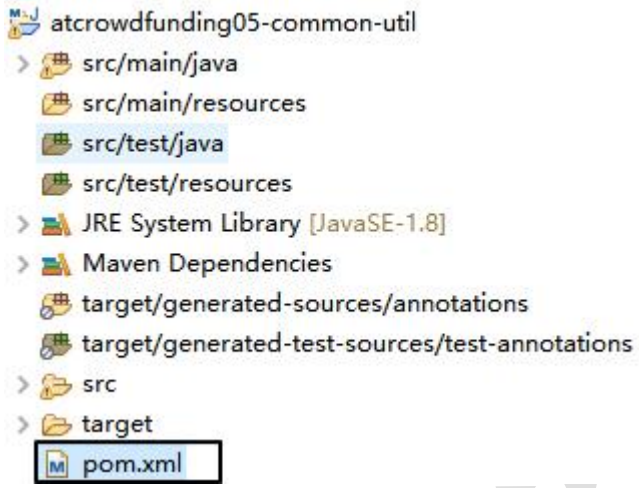
6.6.2 将 OSS 代码中用到的属性存入 yml 配置文件




```
aliyun:
  oss:
    access-key-id: LTAI4FtyK7WcqxaiE9ttukNi
    access-key-secret: 43OFOgMghjsitNsNlTGqcpPGgyzqgU
    bucket-domain: http://atguigu190830.oss-cn-shenzhen.aliyuncs.com
    bucket-name: atguigu190830
    end-point: http://oss-cn-shenzhen.aliyuncs.com
```

6.6.3 工具方法

加入依赖




```
<!-- OSS 客户端 SDK -->
<dependency>
    <groupId>com.aliyun.oss</groupId>
    <artifactId>aliyun-sdk-oss</artifactId>
    <version>3.5.0</version>
</dependency>
```


```
/**
 * 专门负责上传文件到 OSS 服务器的工具方法
 * @param endpoint          OSS 参数
 * @param accessKeyId       OSS 参数
 * @param accessKeySecret   OSS 参数
 * @param inputStream       要上传的文件的输入流
 * @param bucketName        OSS 参数
 * @param bucketDomain      OSS 参数
 * @param originalName      要上传的文件的原始文件名
 * @return    包含上传结果以及上传的文件在 OSS 上的访问路径
 */
public static ResultEntity<String> uploadFileToOss(
        String endpoint,
        String accessKeyId,
        String accessKeySecret,
```

```
            InputStream inputStream,
            String bucketName,
            String bucketDomain,
            String originalName) {

    // 创建 OSSClient 实例。
    OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId, accessKeySecret);

    // 生成上传文件的目录
    String folderName = new SimpleDateFormat("yyyyMMdd").format(new Date());

    // 生成上传文件在 OSS 服务器上保存时的文件名
    // 原始文件名：beautfulgirl.jpg
    // 生成文件名：wer234234efwer235346457dfswet346235.jpg
    // 使用 UUID 生成文件主体名称
    String fileMainName = UUID.randomUUID().toString().replace("-", "");

    // 从原始文件名中获取文件扩展名
    String extensionName = originalName.substring(originalName.lastIndexOf("."));

    // 使用目录、文件主体名称、文件扩展名称拼接得到对象名称
    String objectName = folderName + "/" + fileMainName + extensionName;

    try {
        // 调用 OSS 客户端对象的方法上传文件并获取响应结果数据
        PutObjectResult putObjectResult = ossClient.putObject(bucketName, objectName,
inputStream);

        // 从响应结果中获取具体响应消息
        ResponseMessage responseMessage = putObjectResult.getResponse();

        // 根据响应状态码判断请求是否成功
        if(responseMessage == null) {

            // 拼接访问刚刚上传的文件的路径
            String ossFileAccessPath = bucketDomain + "/" + objectName;

            // 当前方法返回成功
            return ResultEntity.successWithData(ossFileAccessPath);
        } else {
            // 获取响应状态码
            int statusCode = responseMessage.getStatusCode();
```

```
                // 如果请求没有成功，获取错误消息
                String errorMessage = responseMessage.getErrorResponseAsString();

                // 当前方法返回失败
                return ResultEntity.failed("当前响应状态码="+statusCode+" 错误消息="+errorMessage);
            }
        } catch (Exception e) {
            e.printStackTrace();

            // 当前方法返回失败
            return ResultEntity.failed(e.getMessage());
        } finally {

            if(ossClient != null) {

                // 关闭 OSSClient。
                ossClient.shutdown();
            }
        }

}
```

7 今后项目中重定向的问题

7.1 描述问题

http://localhost:4000

http://loclahost:80

是两个不同网站，浏览器工作时不会使用相同的 Cookie。

7.2 解决问题

以后重定向的地址都按照通过 Zuul 访问的方式写地址。

```
redirect:http://www.crowd.com/auth/member/to/center/page
```

8 Zuul 需要依赖 entity 工程

8.1 问题描述

通过 Zuul 访问所有工程，在成功登录之后，要前往会员中心页面。

这时，在 ZuulFilter 中需要从 Session 域读取 MemberLoginVO 对象。SpringSession 会从 Redis 中加载相关信息。

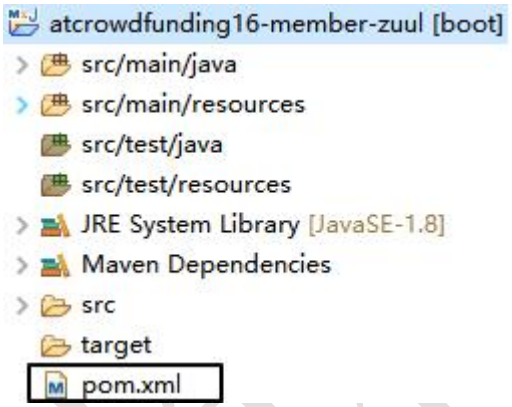
相关信息中包含了 MemberLoginVO 的全类名。

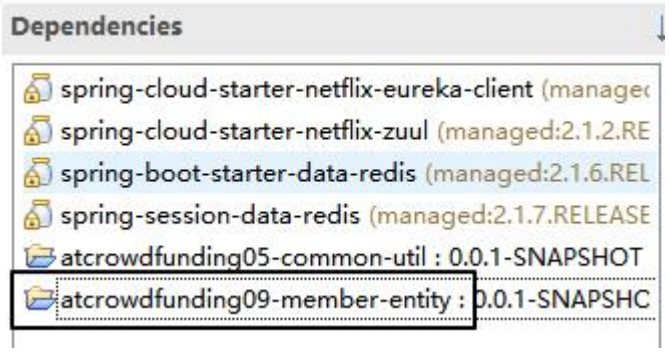
需要根据这个全类名找到 MemberLoginVO 类，用来反序列化。

可是我们之前没有让 Zuul 工程依赖 entity 工程，所以找不到 MemberLoginVO 类。抛找不到类异常。

8.2 解决

让 Zuul 工程依赖 entity 工程。

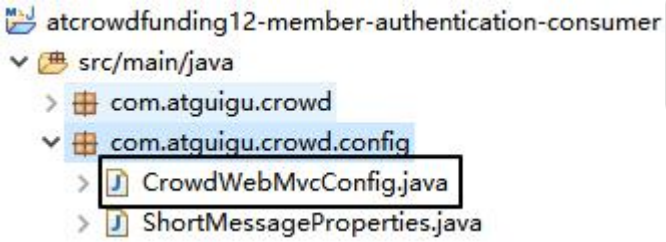




9 从个人中心跳转到发起项目的表单页面

9.1 第一步点击“我的众筹”


```
<a th:href="@{/member/my/crowd}">我的众筹</a>
```




```
registry.addViewController("/member/my/crowd").setViewName("member-crowd");
```

准备 member-crowd.html 页面

9.2