# 短信登录

上节课我们已经成功搭建黑马点评项目的开发环境，现在我们开始完成功能！

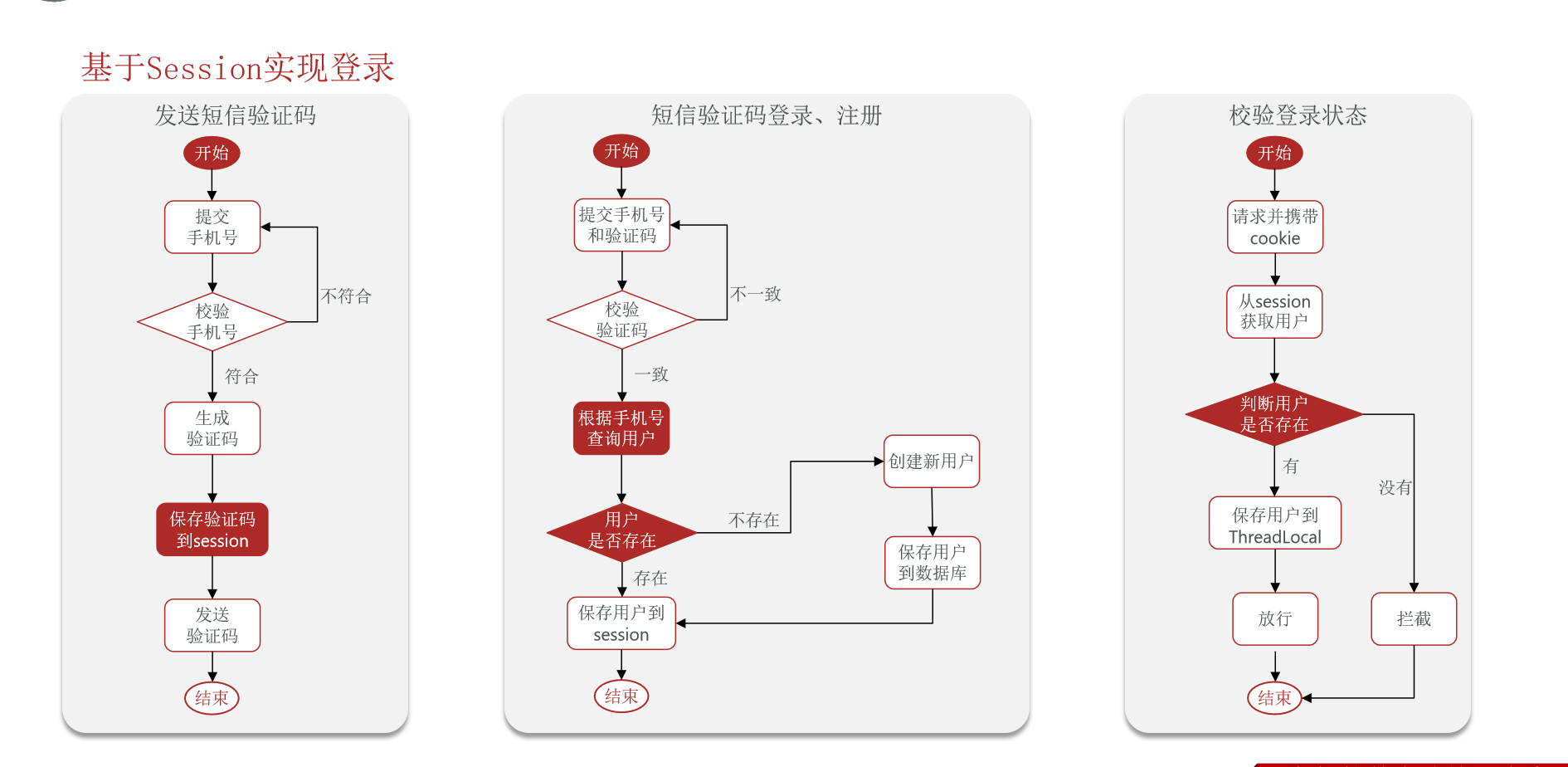
## 基于Session实现功能

登录流程分成三部分：

1：发送短信验证码；

2：基于短信验证码登录或注册；

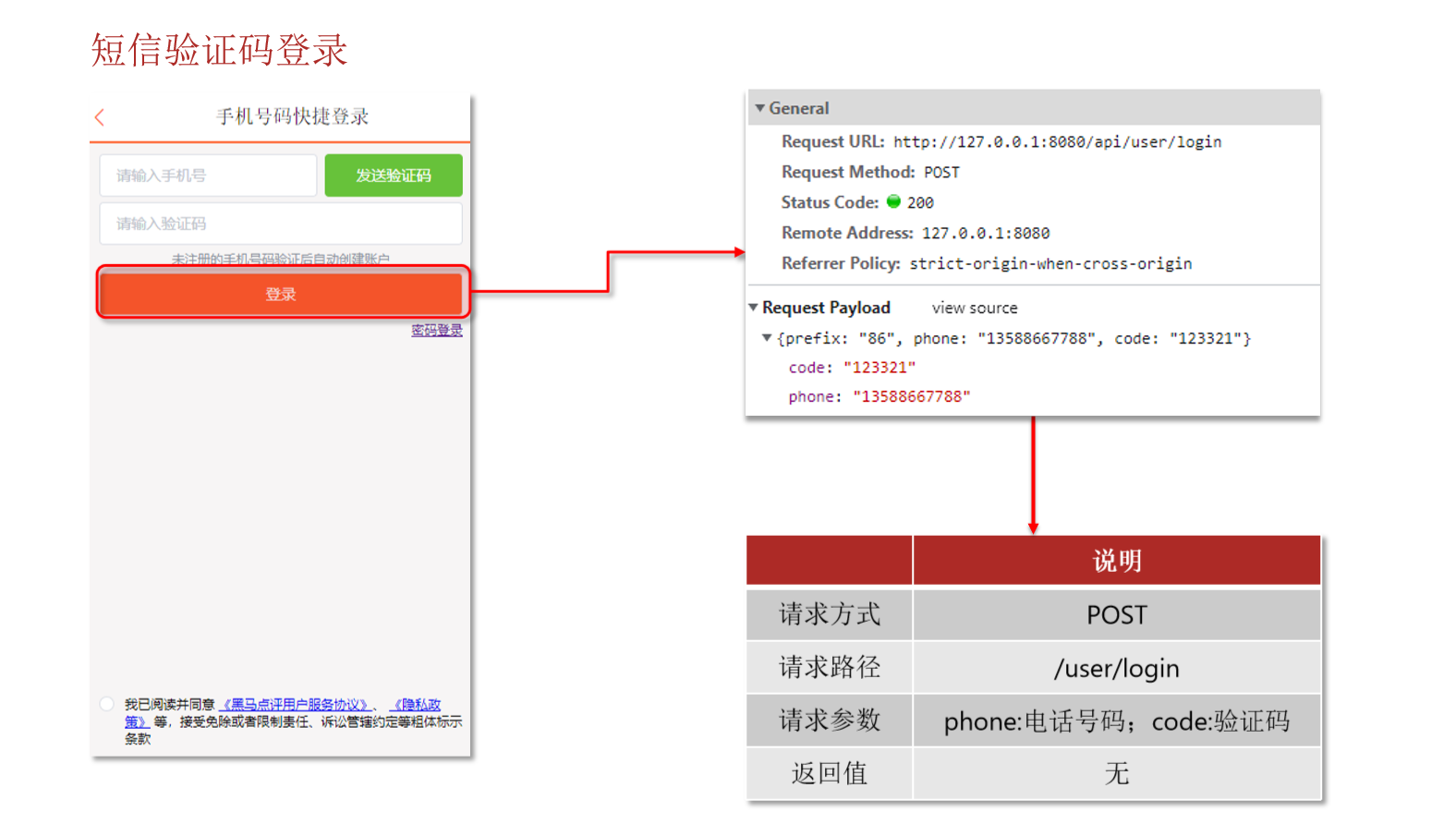
3：登陆状态的校验



### 1：发送短信验证码：



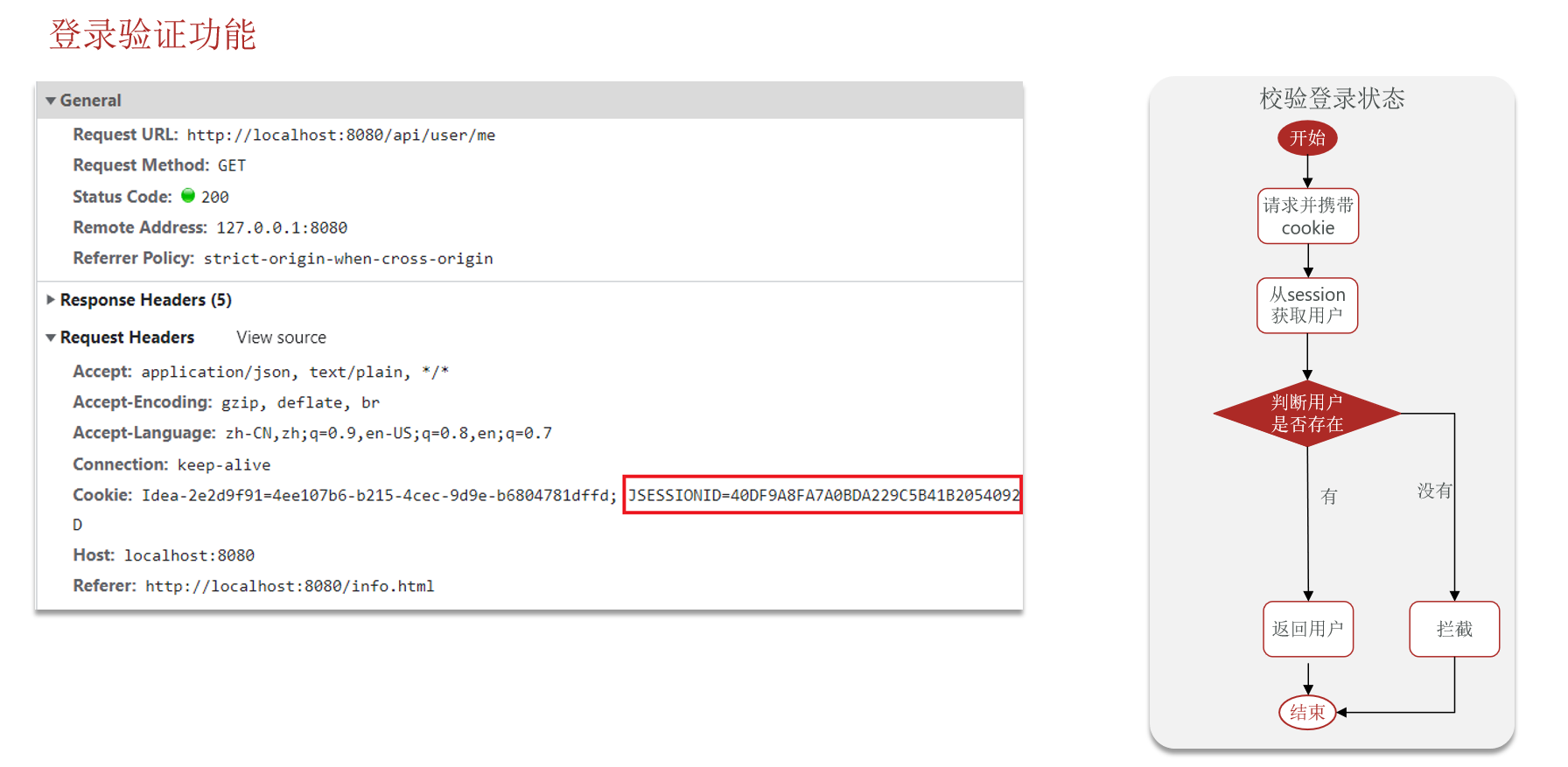
### 2：基于短信验证码登录或注册



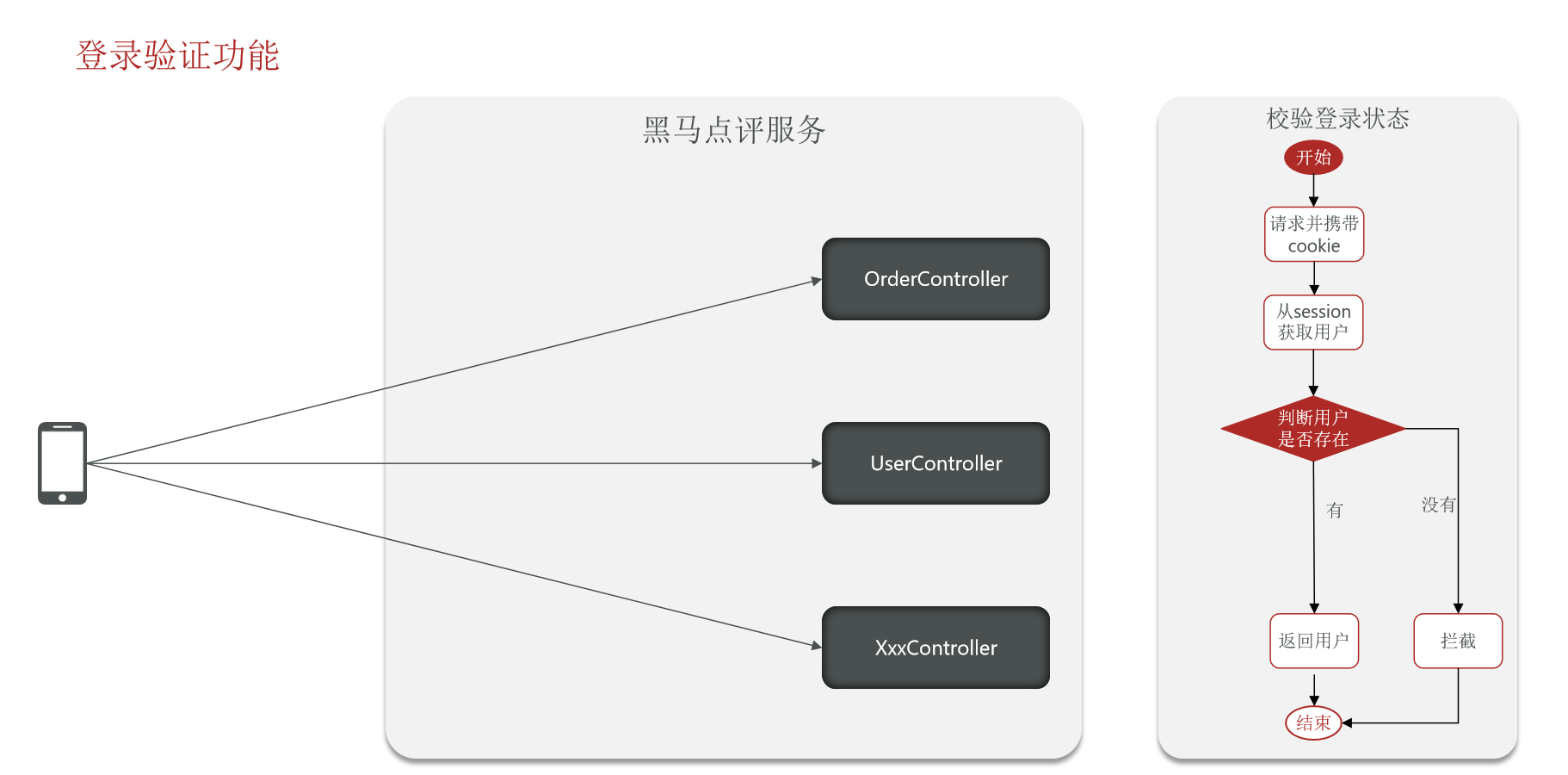
### 3：登陆状态的校验

上面已经实现发送短信验证码，以及基于短信验证码登录，不过可惜的是在前端页面依旧认为我们没有登录！为什么？

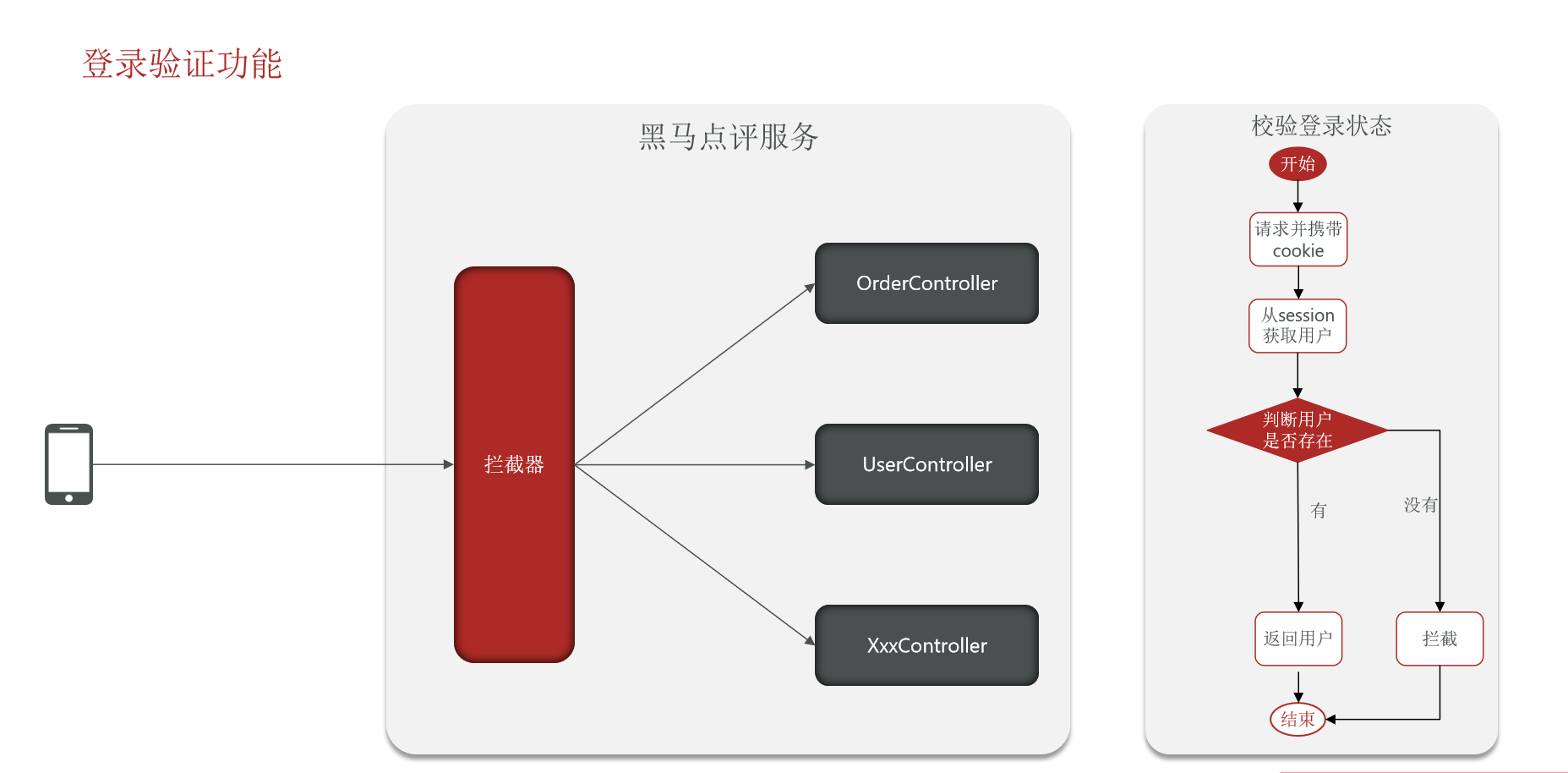
因为登录校验的功能没有做，即一些业务、资源只有登录用户才能访问！



项目中很多业务、资源都只有登录用户才能访问！拿到说你要在每一个请求都要写登录状态的校验逻辑吗？

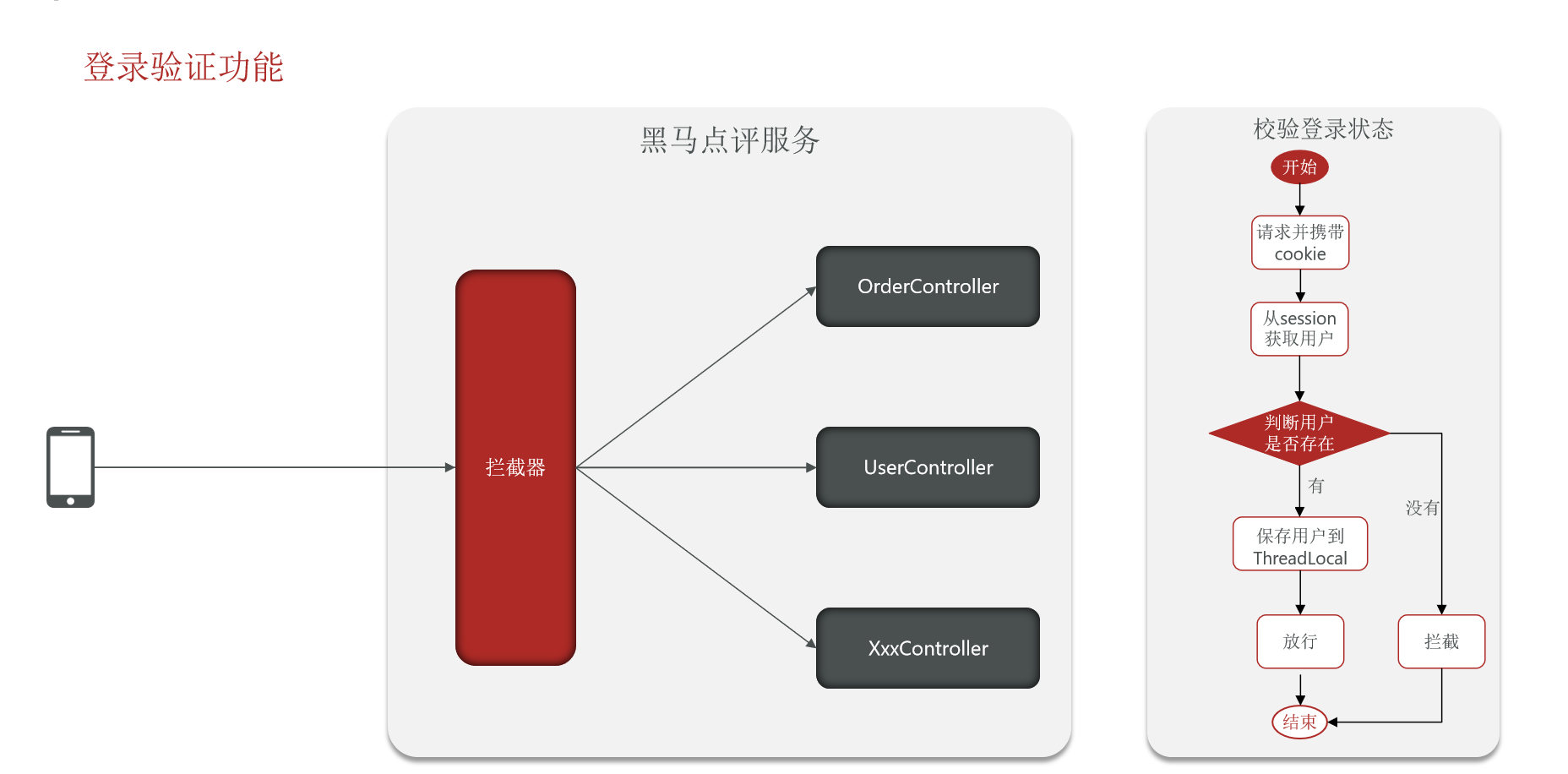


显然是不行的，太麻烦了吧！有更简单的方法，你想一下，在我们学习的SpringMVC知识中，有一个东西可以在所有的请求执行之去做！这个东西就是拦截器，有了拦截器以后，用户的请求就不再能够直接访问到Controller，所有请求都必须先进到拦截器，再由拦截器判断请求该不该放行，，让请求到达对应得Controller。有了拦截器，我们不就可以把用户登录状态校验的这一堆逻辑都放到拦截器里去做，这样一来，所有的Controller的请求都不用再写登录校验的逻辑，而全部都由拦截器来做！



但是上面还存在小小的问题，没错，拦截器确实可以帮助我们实现对于用户登录的校验，但是校验完了之后在后续的业务当中，人家是需要用户信息返回的！你在拦截器校验了，拿到了登录用户信息了，在Controller业务中怎么获取登录用户信息呢？所以我们需要一种方案把拦截器里拦截得到的用户信息传递到Controller里面去，而且在传递过程中需要注意线程安全问题。

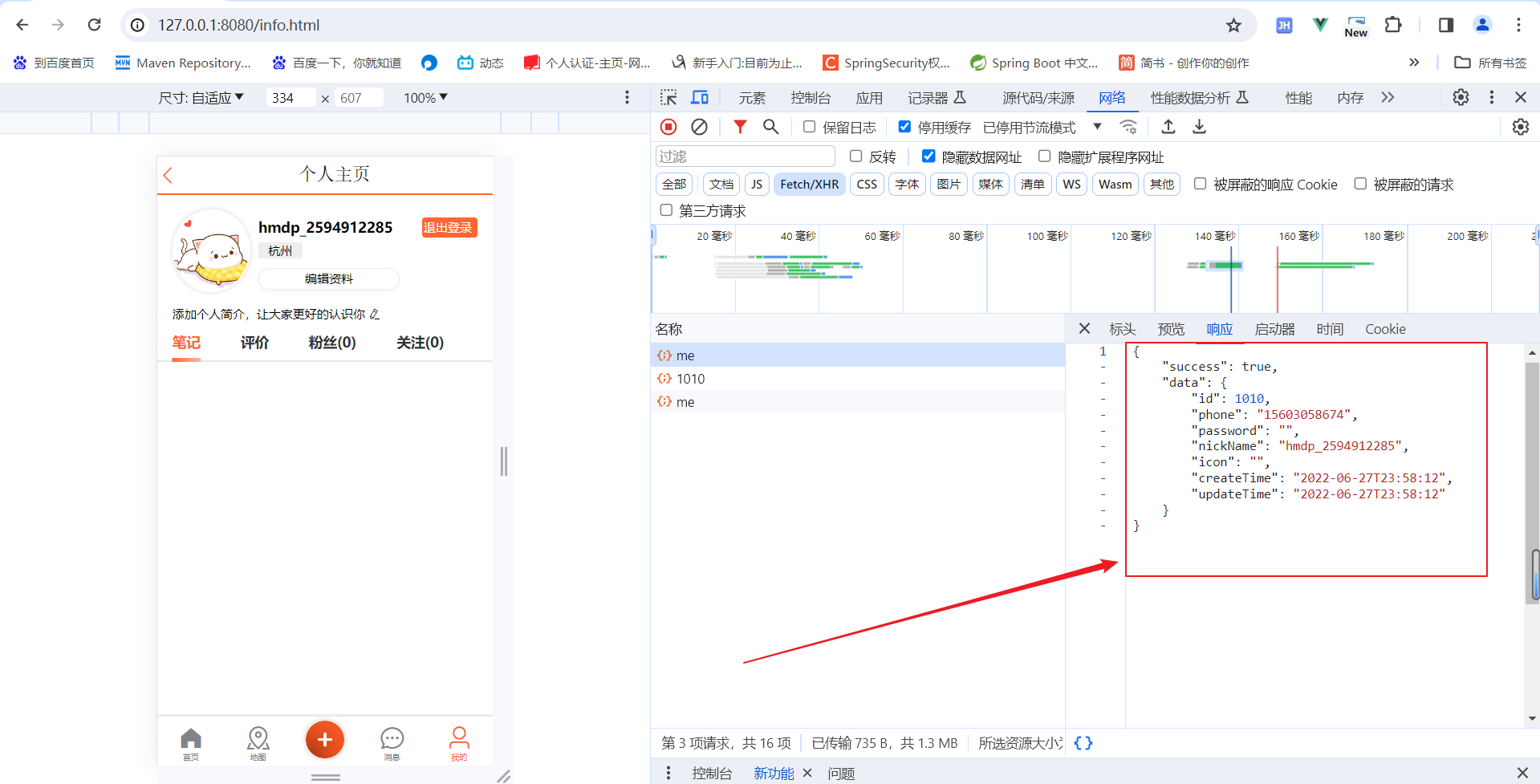
欸！我们可以用ThreadLocal解决！我们在拦截器里拦截得到的用户信息之后可以把用户信息保存到ThreadLocal里去，因为ThreadLocal是一个线程域对象，每一个进入Tomcat的请求都是一个独立的线程！那么将来ThreadLocal就会在线程内给你开辟一个内存空间去保存对应的用户，这样一来每一个线程相互不干扰。到了Controller之后，Controller再从ThreadLocal取出用户就OK 了！



### 4：隐藏用户敏感信息

同学们，上节课我们已经实现了登录拦截校验功能，不过这个功能有一点小问题，见下图，我们的登录校验功能返回的登录用户信息也太多了吧，把用户全部信息都返回回来了。

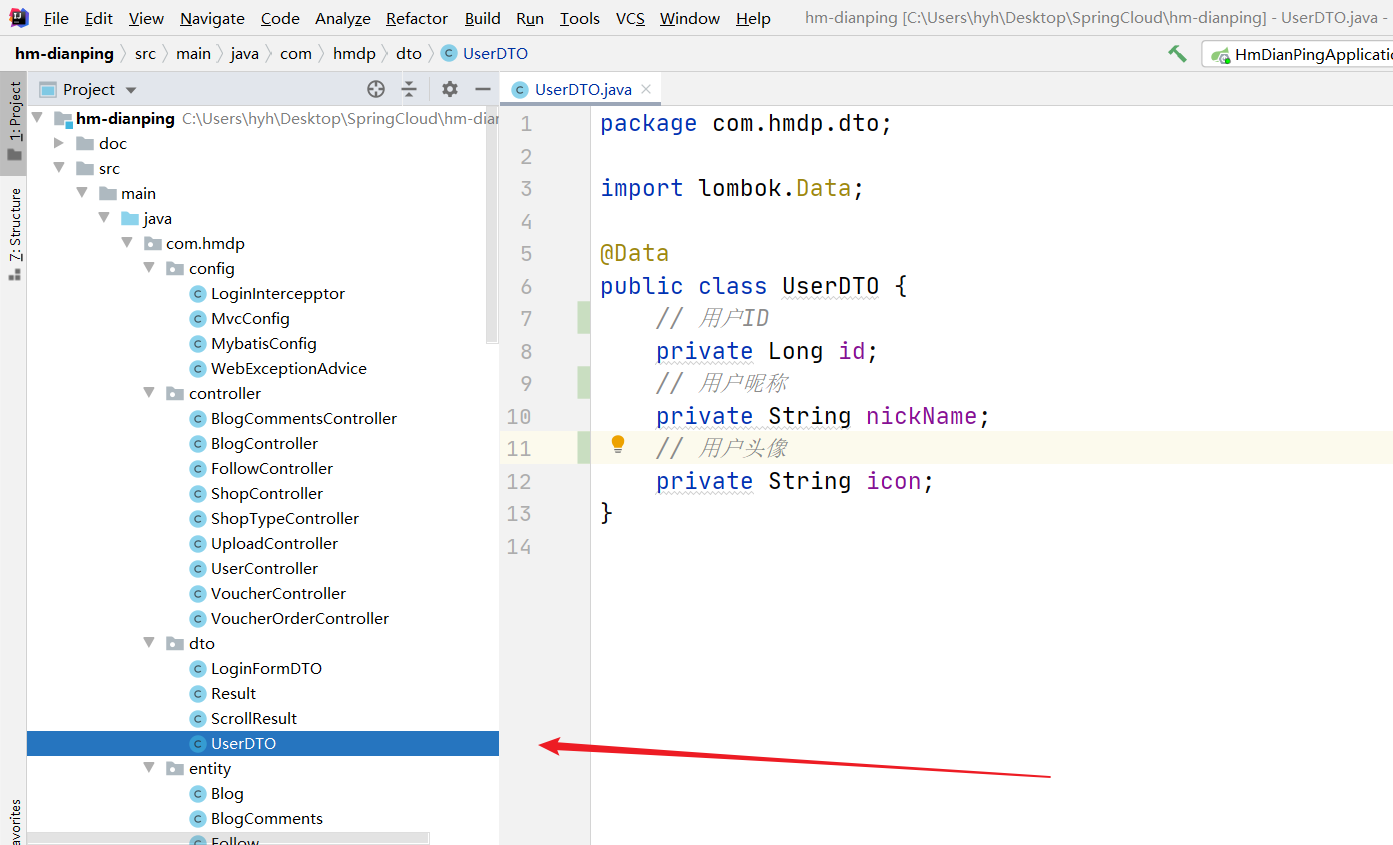
其实我们在查询当前登录用户的只要返回用户的id、头像、昵称就够了，像电话、密码等敏感信息是不应该返回的！有用户信息泄露风险！

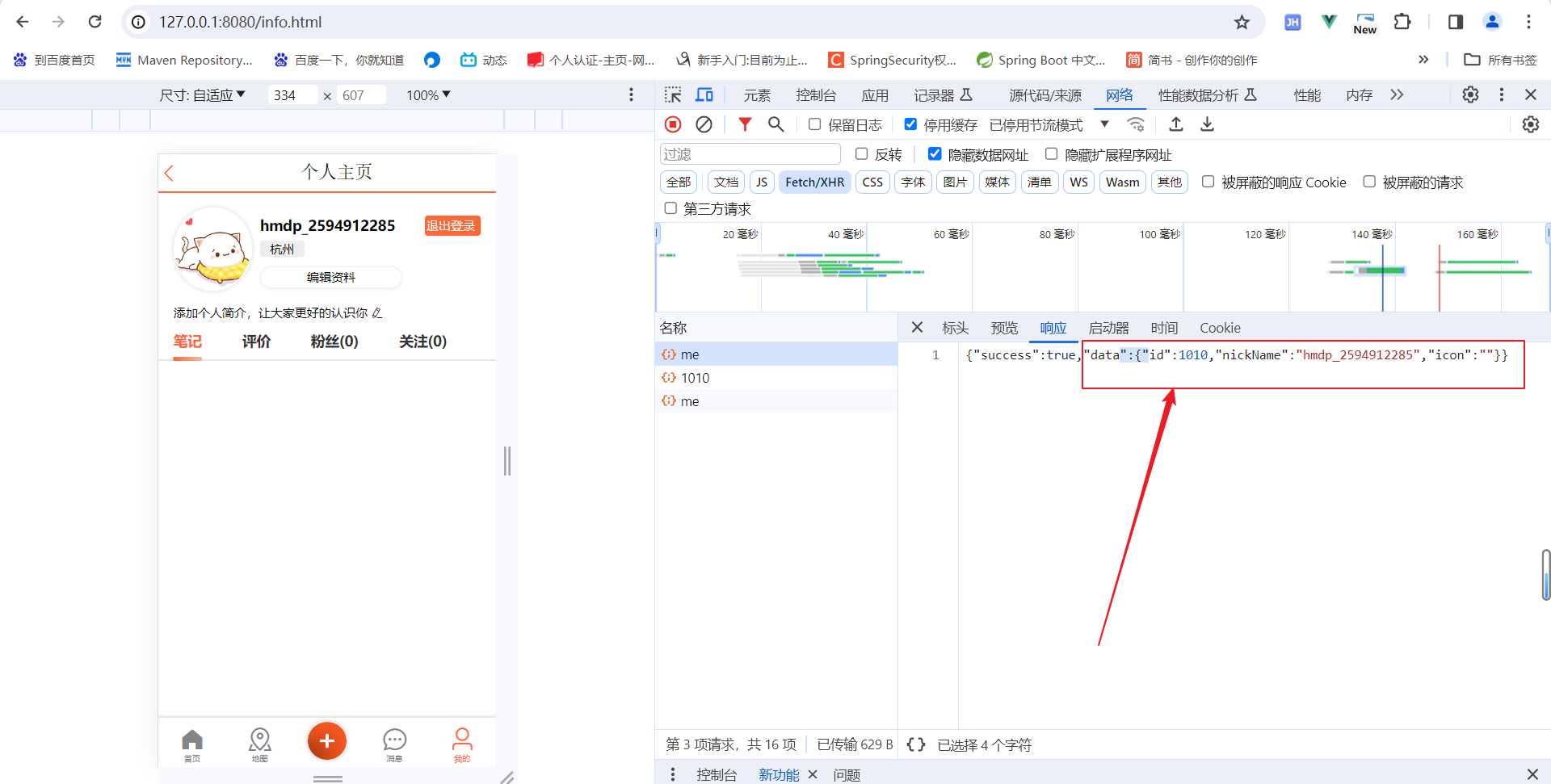


原因是我们在用户登录成功后存进Session的用户信息就是完整的用户信息，在拦截器中做用户登录拦截校验时存进ThreadLocal的用户信息就是Session中的用户信息，返回前端的登录用户信息是从ThreadLocal取出来的用户信息。自然就把登录用户的全部信息返回给前端了。

这样一有用户信息泄露的风险；二因为Session是Tomcat的内存空间，ThreadLocal也是内存空间，你在Session和ThreadLocal中存到东西越多越大，对整个服务来说压力就越大！所以说在Session和ThreadLocal存那么多不相关、不重要的信息是没必要的！

所以只在Session和ThreadLocal存必要的部分的用户信息就可以了，这就是存储的粒度的问题了！



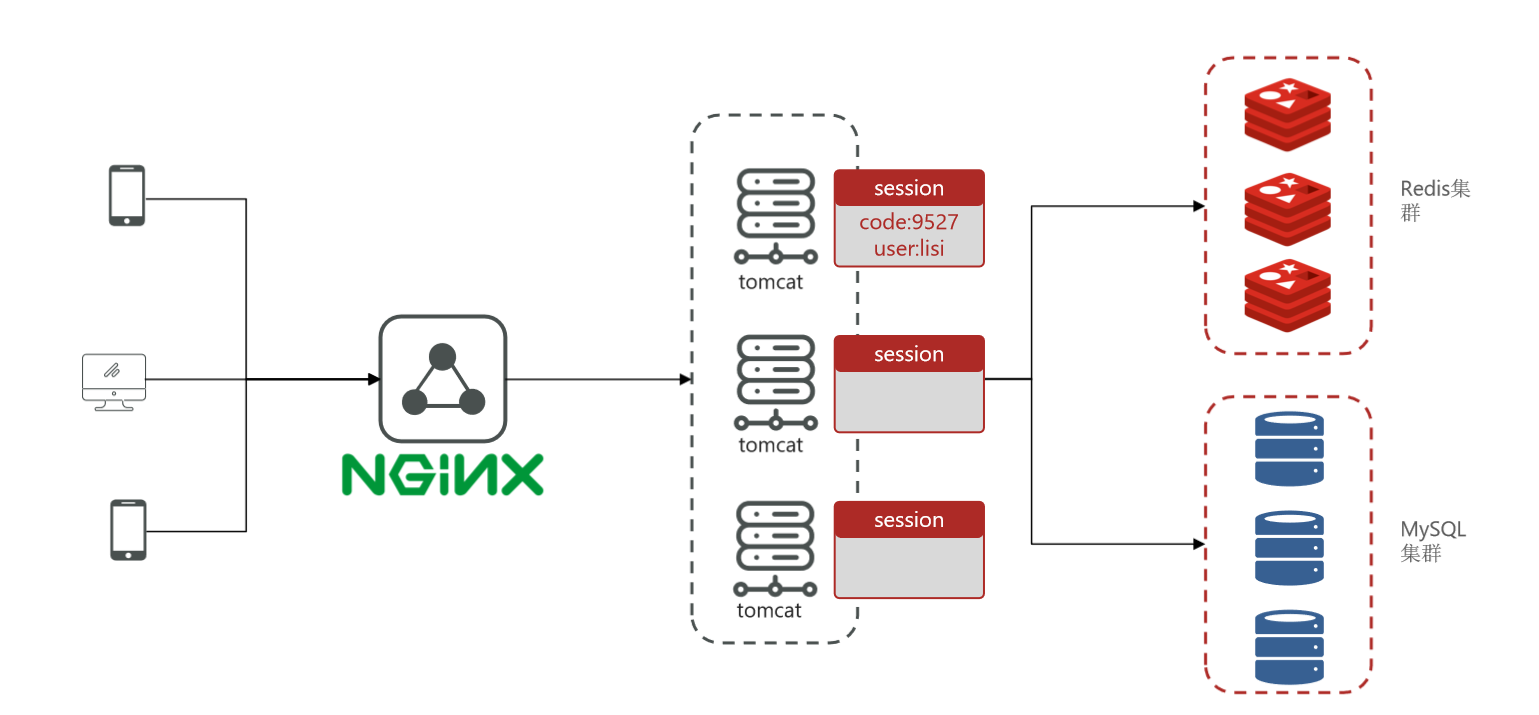


## 基于Session实现短息登录的问题

基于Session登录实现起来并不复杂，但是有一个致命问题，就是Session共享问题!

### Session共享问题

Session共享问题：多台Tomcat并不共享Session的存储空间，当请求切换到不同Tomcat服务时导致数据丢失的问题。



如图，上图是我们黑马点评的架构图，虽然我们的系统是一个单体式的架构，但是我们将来为了应对并发，是可以对单体项目做水平拓展，可以把系统部署多台Tomcat，配合Nginx形成负载均衡的集群！

一旦形成负载均衡的集群，这个时候当请求进入Nginx，Nginx会做一个负载均衡，在多台Tomcat之间做一个轮询，现在每一个Tomcat都会有自己的Session空间，这个时候假设说用户登录请求来了以后，第一次被Nginx负载均衡到了Tomcat1，登录用户的信息也好是保存在Tomcat1里的。紧接着用户第二次来做其他请求，比如获取登录用户信息的请求，如果被Nginx负载均衡到了Tomcat2，因为Tomcat2的Session是空的，于是就获取不到用户了。

这就是Session不共享带来的问题，这个时候你会告诉他用户未登录，这不扯呢吗？我前一秒才刚刚登录完！这个时候给用户带来的体验就非常地差！

### Redis-解决Session共享问题

那么怎么解决Session不共享的问题呢？很简单，就是让Session可以共享！那么Tomcat的Session可不可以共享？事实上在早期Tomcat为了解决这个Session不共享问题，确实提供了Session拷贝的功能！也就是说多台Tomcat之间只要做好一些配置，这些Tomcat之间是可以互相实现数据拷贝。欸！这样不就解决了Session不共享问题了吗！并不行！因为数据拷贝存在问题:

1：多台Tomcat保存相同的数据，大家互相拷贝是一种内存空间的浪费！

2：我们把数据拷贝的时候需要时间，是有延迟的，如果在这个延迟之内有人来访问没有数据的Tomcat依然有数据不一致的情况！

所以Tomcat这个数据拷贝方案推出之后并没有得到广泛的使用和认可！所以这种方案Pass！

因此我们必须要找到替代Session的东西！而这个替代Session的东西必须满足以下三个特点：

1：数据共享

2：内存存储

3：key、value结构

数据共享是最重要的，因为就是不共享才摒弃Session，数据共享了就可以让多台Tomcat都能看到相同的数据了！

内存存储，因为Session是基于内存的，所以读写效率高，像这种登录啊、校验啊。。。。。。这些的访问频率是非常高的，如果说你的读写性能差是无法满足高并发的需求的！

结构式key-value，因为session的存储是很简单的key-value，是非常方便简单的存取方案，我们的替代方案也应该如此！

欸！替代Session的答案就呼之欲出了：Redis！因为Redis完美符合以上特点！

## 基于Redis实现共享Session登录

使用Redis替代Session实现登录，并不是简单地说把数据存入Redis就行了，在业务流程上会有比较多的变化：

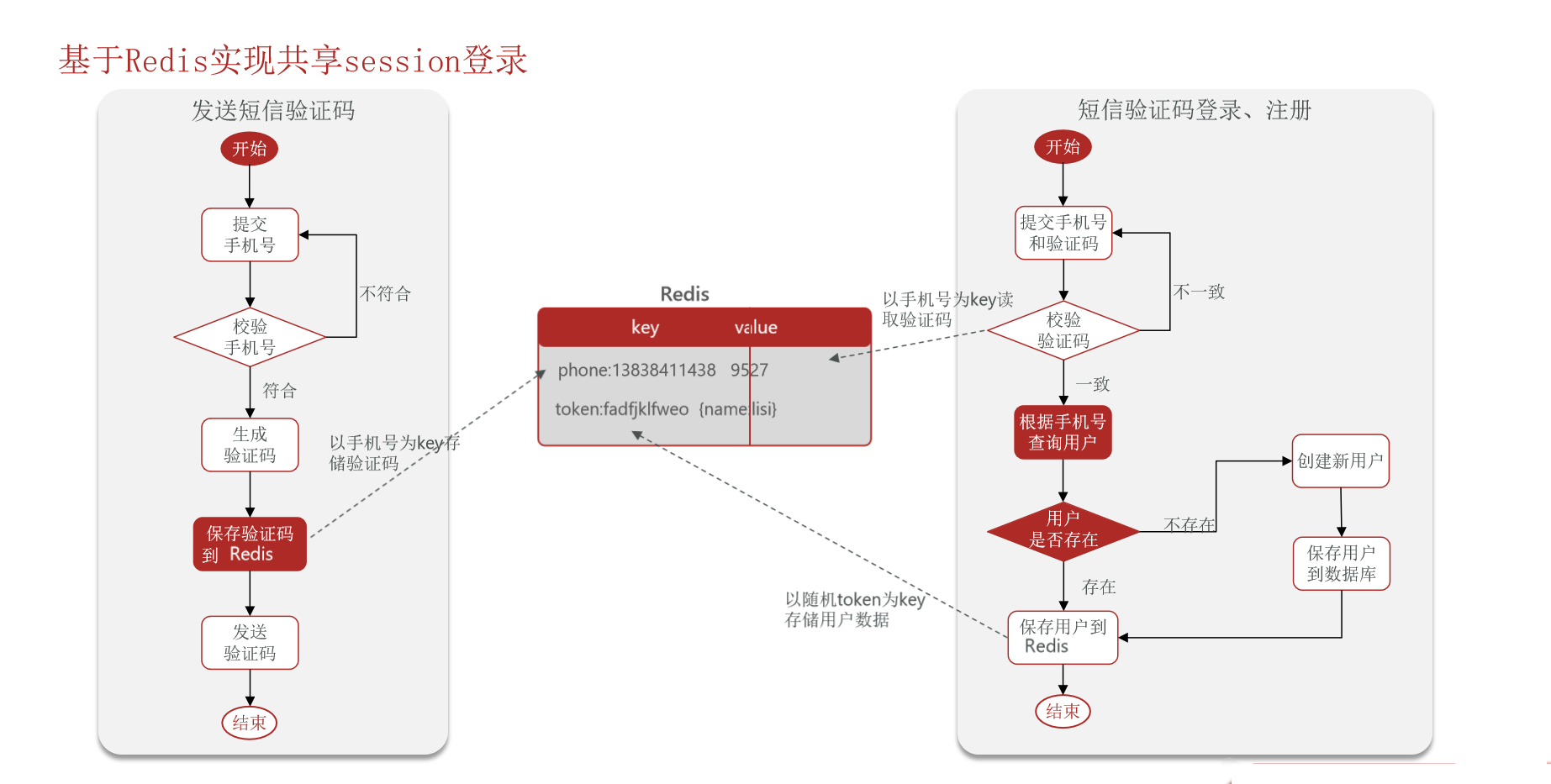
1：验证码存储到Redis时，key不能是Session方案的“code”。

因为Session的特点就是每一个浏览器和Tomcat“会话”的时候都有一个独立的Session！也就是说在Tomcat内部维护了很多很多的Session，不同浏览器在Tomcat服务的Session是独立的，每一个浏览器携带手机号来的时候都是自己独立的Session，那么Session用Code作为Key，但是互相不干扰，没有问题！

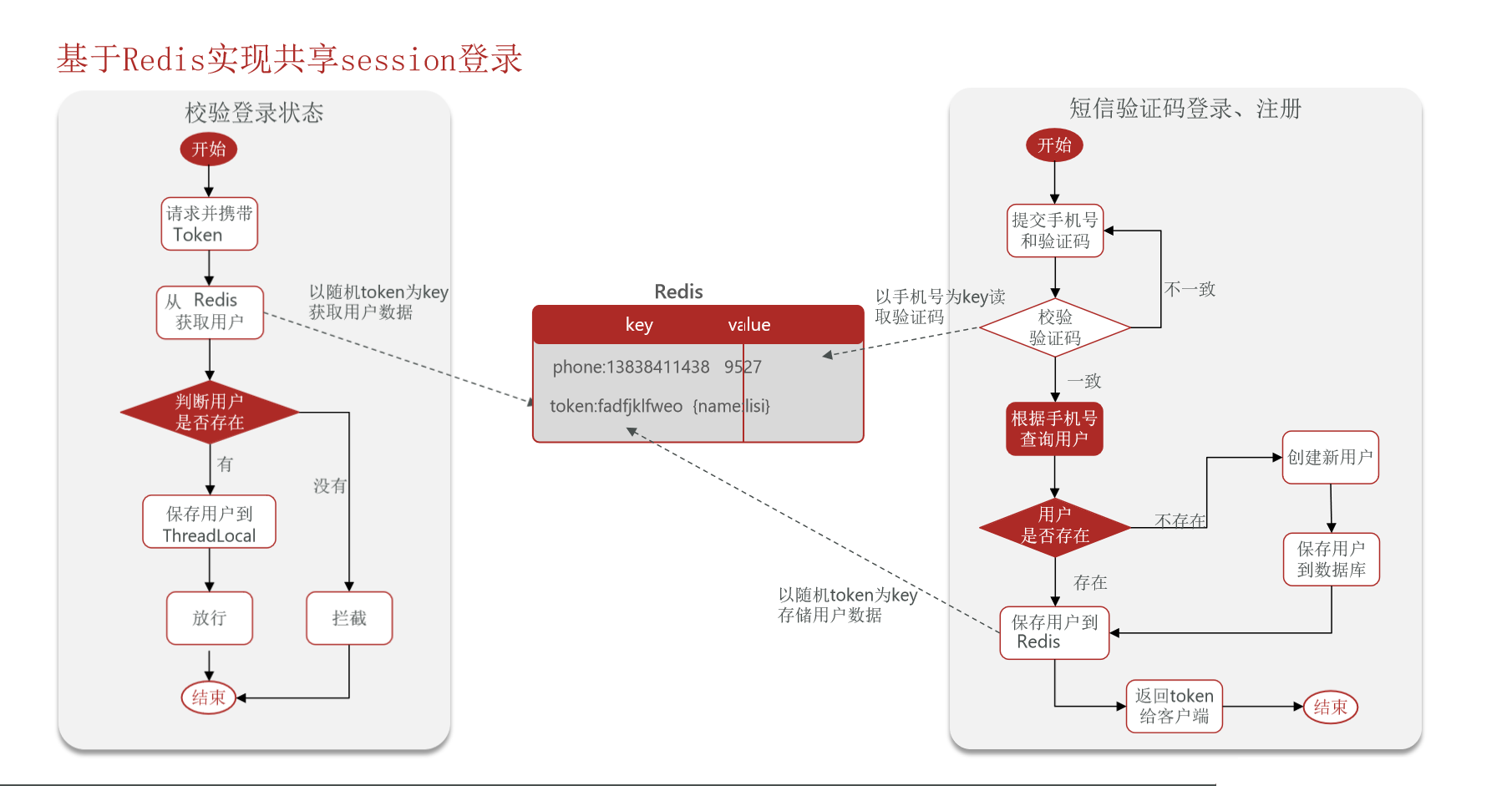
2：确保每一个不同的手机号来做验证的时候保存的Key都得是不一样的，key应该设计成：code:手机号。

3：为了用户信息安全不泄露，token应该是随机的，不能是手机号。

### 用户发送验证码和登录的流程



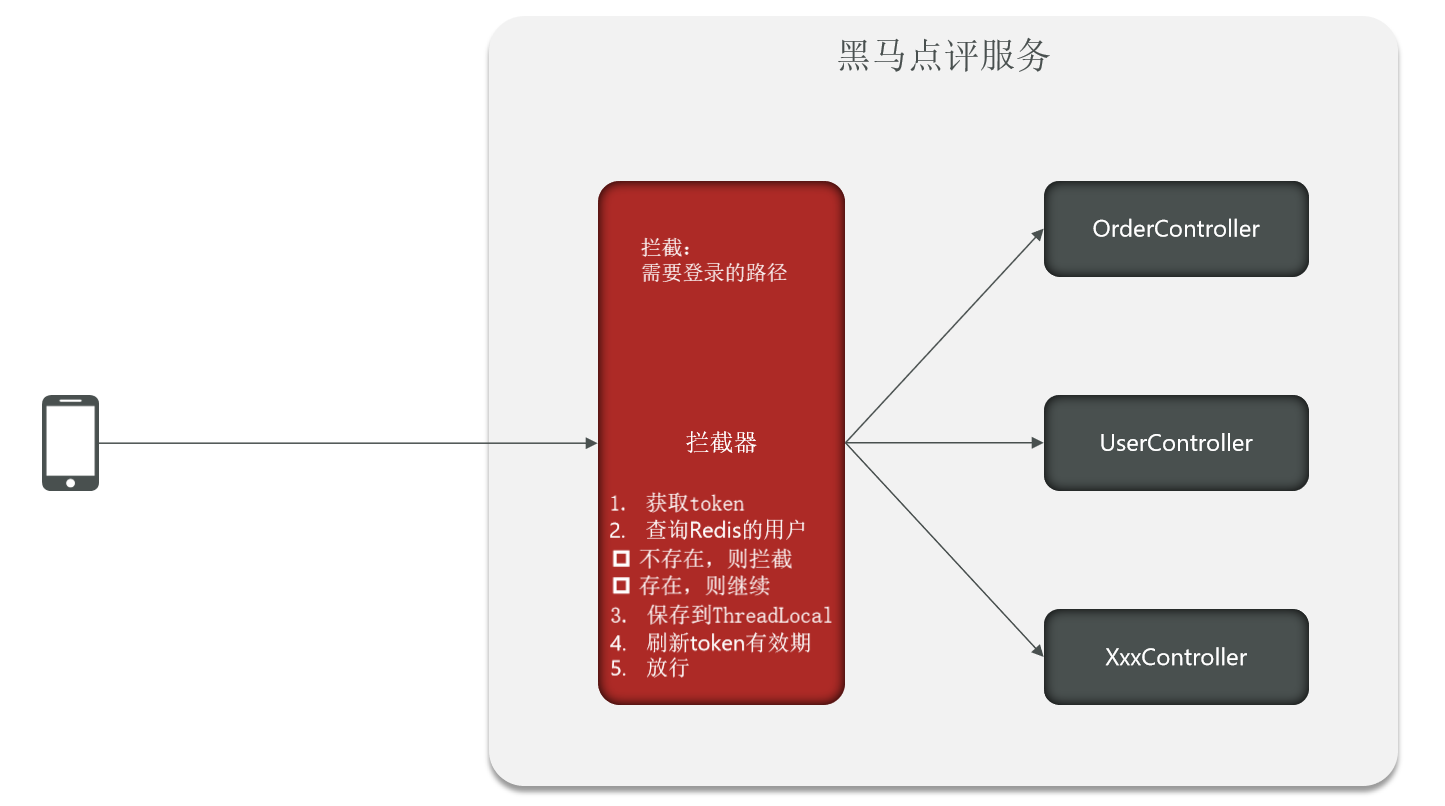
### 登录校验流程图



### 解决登录续期的问题

同学们，通过上面的学习我们同于实现了基于Redis短信登录功能，不过细心的你肯定发现了，我们这个登录还存在小小的问题！

我们的用户登录状态校验功能是基于拦截器做的，用户请求进入拦截器之后，我们会尝试去获取请求头中的token如果之前登录过，请求头中一定会有token，我们再去根据token到Redis中查询对应的用户信息，查到以后再对用户做判断，用户不存在拦截请求，用户存在就把用户保存到ThreadLocal当中，方便后续的Controller中的业务去使用，保存完还要刷新Token有效期，因为我们在Redis中保存的token的有效期是30分钟，如果不去做刷新，用户30分钟后就可能失去登录状态了，这样就对用户不太友好！所以我们要去做token刷新，每当用户来访问我们，我们都会去刷新一次，确保只要用户一直在操作，他这个token就不会消失。



当时我们现在的拦截器并不能真正地达成只要用户一直在访问就不会过期这个功能！因为我们的这个拦截器拦截的不是一切路径，只是拦截的是需要做登录校验的路径，即对用户信息有需求的路径！这就导致了如果我们的登录用户一直访问的是不需要登录校验的页面，即不需要登陆就能看的页面，这样这个拦截器就不生效，就不回去刷新token有效期，这样一来，30分钟以后，尽管用户一直在访问，但用户的登录状态也消失了！这绝对是不合理的地方！

我们可以怎么优化？我们可以在原有拦截器的基础上再加一个新的拦截器，这样用户请求就要先经过新拦截器在经过原有拦截器。我们在新拦截器中拦截一切请求，在这个新拦截器中做刷新Token有效期的动作！这样就可以确保一切请求都可以刷新token动作。

