

27 | 初识认证机制：认证机制能解决哪些问题？

2023-06-23 Barry 来自北京

《Python实战 · 从0到1搭建直播视频平台》



你好，我是 Barry。今天是端午假期，祝你节日快乐，给假期仍然在学习的你点赞。

在 [🔗 上节课](#)，我们利用 Flask-Restful 实现了接口规范化开发，相信你已经可以根据功能需求独立开发接口了。

但是在项目开发中，我们单独考虑开发功能还不够，还要在用户使用系统期间，保护用户隐私和数据安全。例如在系统中需要验证用户的身份，保护用户的个人信息和隐私。这就引出了我们这节课的核心—— Flask 认证机制。

认证机制能解决什么问题

接下来，我们主要从四个维度来看看，在系统中认证机制都能够解决哪些问题。

首先是**保护用户隐私和访问权限**。如果一个系统没有身份认证机制，用户提交的个人信息和敏感数据等就会暴露在公共网络上。这样操作会存在大量风险，例如黑客可能会利用这些信息进行诈骗、恶意攻击以及其他非法活动。此外，系统如果没有足够的访问控制，也可能导致用户无法访问他们需要的资源或服务。

其次，要**防止异常访问和攻击**。在系统中有了认证机制，只有经过认证的用户才可以访问特定的资源，从而保护应用程序免受未经授权的访问和攻击。比较典型的案例就是防止恶意的 CSRF 攻击，保护系统免受盗用用户身份的攻击等。

第三，就是**建立用户对系统的信任**。认证机制可以为系统用户提供更安全的使用体验。对用户的各类数据做好严密保护，这有助于我们建立系统和用户之间的信任关系，否则将无法留存用户。

第四是**优化程序流程**。一个有效的认证机制，可以让用户和应用程序之间的交互过程更加规范，这有助于提高应用程序的可靠性、稳定性和可扩展性。此外，认证机制还可以简化代码和减少开发难度，提高我们的开发效率。

通过前面这四个维度的分析，我们认识到系统中认证机制的重要性。我们这就来了解一下在 Flask 中都有哪些认证方式。

认证方式详解

在 Flask 中，我们可以根据具体应用需求选择合适的认证方式，常用的认证方式包括 Cookie、Session 和 Token。那么三者之间在使用上有什么区别呢？只有充分了解了这三种方式，我们才能选出最适合在线视频项目的方案。

Cookie 是存储在客户端浏览器中的小段文本。它的作用是保存客户端的状态信息，最主要的功能就是存储用户的认证信息。有了 Cookie 的帮助，请求响应结束后，服务器仍然能保存一些信息，例如记录用户登录状态等。

但是，因为在浏览器中 Cookie 可以轻易被修改，将明文认证信息存储在 Cookie 中可能存在安全风险。所以，在 Flask 中通常使用 Session 对象来加密存储 Cookie 数据。当然，

Session 对象也可以用于存储认证信息，它提供了一种轻量级的会话管理机制。同时，Session 数据存储在服务器端，无法被客户端直接修改。

Session 的使用当然也会存在风险，因为 Session 机制是基于 Cookie 的，Cookie 被拦截时，用户容易被跨站攻击。Session 一般存储在内存中，每个用户通过认证之后，都会将 Session 数据保存在服务器的内存中，所以用户量达到一定程度时就会给服务器造成负担。

了解了 Cookie 和 Session 的认证方式，接下来我们再看看 Token 的认证方式。基于 Token 的认证机制工作原理是将认证信息返回给客户端，由客户端自己保存，等待访问服务器请求资源的同时，带上 Token 认证信息就可以。

如果我们采用这样的方式，服务器端就不需要保留用户的认证或者是会话等信息。对用户而言，不需要指定请求存储了用户认证信息的某个特定服务器，可以在多个服务器之间使用 APP，在增加更多的用户或者更多的功能时，就更容易扩展应用程序。

综合前面的分析，我们的项目中最终选择 Token 的认证方式。

认识 Token

在 Flask 中，Token 可以分为两种类型，分别是 JWT (JSON Web Token) 和自定义 Token。

JWT 是一种开放标准，它提供了一种轻量级且安全的身份验证方法。JWT 分成三个部分，分别是 header (头部)、payload (载荷) 和 signature (签名)。具体含义和用途你可以参考后面的表格。

shikey.com转载分享

参数名	用途
header	包含有关 Token 的元数据，如 Algorithm（加密算法）和 Type（Token 类型）
payload	包含有效载荷数据，如用户 ID、角色、过期时间等
signature	用于验证 Token 的完整性和真实性，确保 Token 未被篡改



在项目中使用 JWT 完成 Token 认证的好处是：它可以在不同系统之间传递，并且不需要依赖数据库或其他身份验证机制。

在了解了 JWT 之后，我们紧接着来认识一下自定义 Token，对比一下二者的区别。自定义 Token 是我们根据应用程序需求自己设计的 Token。它通常包含用户 ID、过期时间、角色等信息，同时还可以包括用于验证 Token 完整性和真实性的签名。当然了，我们也可以结合自己的需求自定义其他参数进行传递。

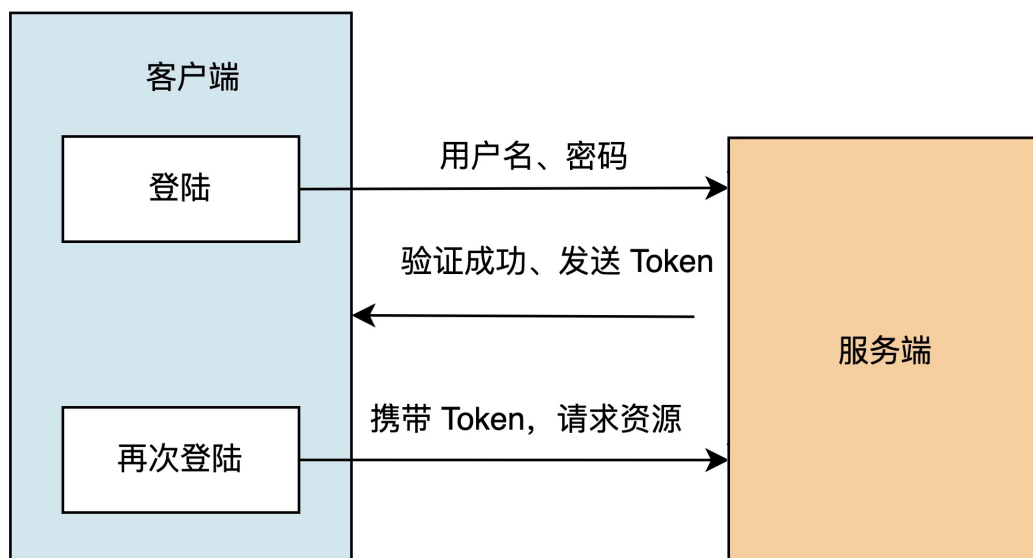
我们之所以使用自定义 Token，是因为它可以根据项目需求去灵活设计 Token 的组成结构和验证机制。我们在使用的过程中还需要注意，**自定义 Token 需要实现完整的 Token 验证逻辑，其中包括签名验证、过期时间检查、防止 Token 泄露等安全措施。**

说完了两种 Token 形式之后，接下来，我们看看如何通过 Token 实现认证机制。

Token 的认证流程

在 Flask 中，Token 认证机制可以用于保护项目的敏感资源，限制用户只有经过身份验证后才能访问与其相关的资源。

Token 认证是一种基于令牌的身份验证方法，它在客户端和服务器之间传递一个加密的令牌，以此来实现用户信息的认证。为了帮你直观理解认证流程。我为你准备了后面这张认证机制流程图。



首先我们需要**生成 Token**。在用户成功登录系统后，后端会生成相应的 Token，并将 Token 存储在服务器端，这里可以选择数据库或内存缓存来存储。

紧接着，在用户访问需要认证的资源时，服务端将 **Token 发送给客户端**。这一步传送的实现方式就是将 Token 存储在 Cookie 或 HTTP Header 中。在客户端发送请求时，服务端可以通过解析 Token、检查过期时间、验证签名等方式来**验证 Token 的完整性和真实性**，从而建立起认证机制。

当然我们为了确保 Token 的安全性，服务端需要**实现 Token 的过期时间和失效机制**。当 Token 过期或被篡改时，应用程序需要更新或删除相应的 Token，从而保证认证的时效性。

这里还需要注意的一个地方就是，Token 认证中有两个必要组件——日志和内存，它们分别用于记录认证过程、存储令牌及用户信息。两者有助于保证系统的安全性，让系统正常运行。接下来我们就一起了解一下日志和内存的工作原理。

日志和内存的工作原理

日志主要用来记录认证过程中的一些事件，例如系统中的用户访问受保护的资源事件、用户信息认证成功或失败等等。通过记录这些信息，可以让我们更好地了解应用程序的安全性和可能

潜在的攻击问题。当发生异常或攻击时，日志还可以帮助开发人员快速定位问题并采取相应的措施。

实现日志记录

那么如何在项目中实现日志的记录呢？在 Flask 中我们可以使用 Python 的内置 logging 模块来实现记录日志。你可以对照表格看看具体的日志级别分类，从上往下级别依次升高，最高的是 CRITICAL。

级别	记录详情
DEBUG	详细的调试信息，用于诊断和排错。
INFO	用于记录一般的运行信息，通常用于监视应用程序的运行状态。
WARNING	用于记录可能出现问题或潜在危险的情况，例如配置错误或功能限制等。
ERROR	用于记录应用程序出现的错误情况，例如 IO 操作失败、无法连接数据库等。
CRITICAL	用于记录最严重的错误情况，这类错误通常导致应用程序无法继续运行，例如内存耗尽、系统崩溃等。



话不多说，我们进入实操环节。首先，我们需要安装 logging 的扩展，执行后面的命令即可。

shikey.com转载分享

复制代码

```
1 pip install logging
```


shikey.com转载分享

完成安装之后，要在 config.config 配置文件中配置日志级别，这里我们配置 INFO 等级。

复制代码

```
1 LEVEL_LOG = logging.INFO
```


还有一步不能忘记，就是给生产环境的 config 也设置日志级别。

 复制代码

```
1 class ProConfig(Config):
2     LEVEL_LOG = logging.ERROR
3     DEBUG = False
4     SQLALCHEMY_DATABASE_URI = "mysql+pymysql://root:flask_project@127.0.0.1:3306/
```

紧接着，我们在 api/utils 目录下，新建 log_utils.py 文件。在文件中定义三种不同的日志格式和输出方式，用来满足不同的日志记录需求。


三种输出模式都是什么作用呢？setup_log 函数将日志记录到文件中，setup_logger 函数同时记录日志到文件和控制台上，而 json_log 函数则是将日志以 json 格式记录到文件中。

接下来。我们再来看看它们是如何将日志文件写入文件中的。这里主要借助 RotatingFileHandle 函数，它是 Python 的 logging 模块中的一个处理器，可以把日志信息写到文件中，并且支持日志滚动，也就是当文件大小达到一定阈值时，会自动创建一个新的日志文件，避免单个日志文件过大造成的存储问题。

对于日志文件过多而导致存储空间被占用的这个问题，通过日志的滚动就能解决。日志滚动是指将旧的日志文件删除，并将新的日志文件添加到日志滚动组中。

通过指定文件的最大量、备份数量等参数，就可以控制日志的滚动。具体的日志应用代码案例如下所示。

shikey.com转载分享


 复制代码

```
1 def setup_log(logger_name=None, log_file='logs/log', level=logging.INFO):
2     """根据创建app时的配置环境，加载日志等级"""
3     # 设置日志的记录等级
4     logging.basicConfig(level=level) # 调试debug级
5     # 创建日志记录器，指明日志保存的路径、每个日志文件的最大量、保存的日志文件个数上限
6     file_log_handler = RotatingFileHandler(log_file, maxBytes=1024 * 1024 * 100,
7     # 创建日志记录的格式      日志等级      输入日志信息的文件名 行数      日志信息
8     formatter = logging.Formatter('%(asctime)s - %(levelname)s %(filename)s:%(lin
9     # 为刚创建的日志记录器设置日志记录格式
10    file_log_handler.setFormatter(formatter)
```

```
11 # 为全局的日志工具对象（flask app使用的）添加日志记录器
12 logging.getLogger(logger_name).addHandler(file_log_handler)
```

在上面代码中，我们首先要设置日志的记录等级为 INFO。随后再创建一个 `RotatingFileHandler` 对象来记录日志，里面需要设置日志文件的路径、每个日志文件的最大量为 100 MB，最多保存 10 个文件。随后是创建日志记录的格式，这里采用字符串的格式。最后，将日志记录器添加到全局的日志工具对象中，这样我们就完成了日志应用的配置代码。

那么如何同时记录日志到文件和控制台上呢？这时我们可以使用 `logging` 模块中的 `FileHandler` 和 `StreamHandle`，将日志同时写入文件和控制台，具体的代码实现是后面这样。

 复制代码

```
1 def setup_logger(logger_name, log_file, level=logging.INFO):
2     """
3     %(asctime)s 即日志记录时间，精确到毫秒
4     %(levelname)s 即此条日志级别
5     %(filename)s 即触发日志记录的python文件名
6     %(funcName)s 即触发日志记录的函数名
7     %(lineno)s 即触发日志记录代码的行号
8     %(message)s 这项即调用如app.logger.info('info log')中的参数，即message
9     :param logger_name:
10    :param log_file:
11    :param level:
12    :return:
13    """
14    log = logging.getLogger(logger_name)
15    # 创建日志对象
16    formatter = logging.Formatter('%(asctime)s : %(message)s')
17    # 设置格式, '% (asctime) s: %(message) s', 即日志记录时间和日志内容
18    file_handler = logging.FileHandler(log_file, mode='w')
19    # 创建一个文件处理器，将日志内容输出到名为log_file的文件中。
20    file_handler.setFormatter(formatter)
21    # 将日志格式设置为文件处理器的格式
22    stream_handler = logging.StreamHandler()
23    # 创建一个流处理器，将日志内容输出到控制台
24    stream_handler.setFormatter(formatter)
25    # 将日志格式设置为流处理器的格式
26    log.setLevel(level)
27    # 将日志级别设置为传入的参数level
28    log.addHandler(file_handler)
```



```
29     log.addHandler(stream_handler)
30     # 将文件处理器和流处理器添加至日志对象中
```

代码整体的流程和将日志文件写入文件是类似的，`setup_logger` 函数会把日志记录器的名称、记录文件的名称和记录级别作为参数。我们先创建日志对象，然后再设置日志格式。紧接着就是创建文件处理器和流处理器，最后将两个处理器添加到日志对象中。

我们将日志内容输出到文件时，设置了参数 `mode= 'w'`，这表示以写入方式打开文件，这种模式会先清空文件中的内容，然后写入新的内容。如果写入的时候发现文件不存在，系统就会创建一个新文件。

当然，`mode` 的值除了是 `'W'` 以外还包括其他的类型，例如后面这些。

1.`mode= 'a'` 表示以追加模式打开日志文件，新写入的日志会添加到文件末尾。

2.`mode= 'x'` 表示以排他模式打开日志文件，如果文件已经存在就会抛出 `FileExistsError` 异常。

3.`mode= 'b'` 表示以二进制模式打开日志文件，适用于 Windows 文件系统。

Redis 存储

清楚了日志的作用具体用法，我们继续来看 Token 认证中的内存。


Token 认证通常使用一个内存存储来保存生成的 Token。项目中对内存部分的操作，我们借助选用 Redis 来实现。使用 Redis 可以用来存储用户 Token 和相关的用户信息。

我们先来了解一下它的工作原理，当用户登录成功后，服务器会生成一个 Token 并将其存储到 Redis 中，同时将 Token 返回给客户端。客户端在后续的请求中，需要在请求头中携带该 Token，服务器在接收到请求后，会检查该 Token 是否在 Redis 中存在，并获取相关的用户信息。

如果 Token 验证成功，即认为该请求是来自已登录的用户，服务器会返回相应的数据；否则服务器就要返回相应的错误信息。


使用 Redis 可以简单而高效地实现 Token 认证操作，避免在每个接口请求中都需要去查询数据库的开销，这样会大大提高系统的性能。

我们这就来看看具体的配置实现。第一步就是在 config.config 配置文件中设置。后续我们还会使用到 Session 来存储客户端与服务器交互时的一些信息，这里我们一并配置完成。具体的代码是后面这样。

 复制代码

```
1 REDIS_HOST = '127.0.0.1'
2 # Redis服务器的IP地址和端口号
3 REDIS_PORT = 6379
4
5 # 指定session使用什么来存储
6 SESSION_TYPE = 'redis'
7 # 指定session数据存储在后端的位置
8 SESSION_REDIS = StrictRedis(host=REDIS_HOST, port=REDIS_PORT)
9 # 是否使用secret_key签名你的session
10 SESSION_USE_SIGNER = True
11 # 设置过期时间，要求'SESSION_PERMANENT'，True。而默认就是31天
12 PERMANENT_SESSION_LIFETIME = 60 * 60 * 24 # 一天有效期
```

第二步，我们需要在 api/init.py 文件中，设置创建 Redis 的连接对象。

 复制代码

```
1 redis_store = None
2 global redis_store
3 # 创建redis的连接对象
4 redis_store = StrictRedis(host=config.REDIS_HOST, port=config.REDIS_PORT, decode_
```

到这里，我们就完成了使用 Token 进行认证的前置工作。通过上面的学习，你已经掌握了 Token 认证的两个核心配置应用。日志用来记录认证过程中的一些事件，而 Redis 用来存储用户 Token 和相关的用户信息。

总结

又到了课程的尾声，我们来总结一下这节课重点。

认证机制在项目中不可或缺，它极大地保护了用户在平台内的安全性。Flask 里常用的认证方式包括使用 Cookie、Session 和 Token。

Session 和 Cookie 的方式下，服务器必须存储用户的认证信息，这样用户量增加时服务器负荷就会增加。而 Token 很好地解决了这一问题，只在用户登录时，将 Token 临时存放在内存中，用于快速验证。把 Token 发给各个客户端，解决了应用程序的扩展问题。

随后，我们学习了 Token 里的三个重要部分，主要信息存储在 payload 中，我们借助 JWT 中的 encode 函数并且使用密钥来生成 Token。解码时则要借助函数 decode，这个过程同样需要使用到密钥。

日志和 Redis 内存是 Token 认证中的重点。日志方面，你需要重点掌握编写日志的几种模式对应的代码。内存方面，使用 redis_store 扩展管理内存的过程里，我们重点要关注如何在 config 配置文件中做好设置，建议你自己在课后也尝试练习一下。

思考题

既然我们要通过 Token 进行用户认证，请你思考一下在调用接口的时候怎样实现统一化管理呢？这样我们不用每次请求都需要在代码中写一遍 Token 参数，然后再传递给服务端了。

欢迎你在留言区和我交流互动，如果这节课对你有启发，也推荐你把这节课分享给更多朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

shikey.com转载分享

精选留言 (2)



coderHOW

2023-06-23 来自广东

思考题：使用请求拦截，每次请求接口可以带上token等信息

老师，我这边有几点不太明白

- 1、使用token服务端不需要保存用户认证和会话，但是又需要内存存token，在过期时间之前都会保存这个token吗，理论上如果用户量太多内存也会有压力吧
- 2、如果发给客户端的token保存在cookie中，可能也会被其他人拿到伪造用户登录行为，这个如何避免呢(我想到就是可以在荷载数据中加入ADDR、UA这样的信息来识别)
- 3、前端发来的token如果还没到过期时间，但是redis又查不到，可不可以通过解密token然后根据荷载数据的某些信息通过mysql去查数据返回呢

作者回复: 1、是的，当用户量增加时，服务端需要分配更多的内存来存储所有的token。然而，可以通过一些技术来减轻内存压力，当然可以通过一些方法来进行优化：可以通过设置较短的过期时间可以减少服务端内存的使用。

2、可以使用HTTPOnly和Secure属性来防止token被恶意获取。HTTPOnly属性可以防止JavaScript访问cookie，而Secure属性可以确保只有在安全情况下才能发送cookie。当然UA的方式也是OK的

3、通过这样的方式你需要确保Redis中保存的token是经过加密的，以确保其安全性，如果Redis和MySQL之间的数据同步出现故障或错误，可能会导致安全漏洞和其他问题。所以这种方式还是存在一定的风险。



1



peter

2023-06-23 来自北京

请教老师两个问题：

Q1: OAuth2会用到吗？

Q2: http header中保存token，是保存在哪个字段？

另外，token保存在cookie中，session也是基于cookie，token与session有什么区别？

作者回复: 1、在使用 Flask 框架构建认证机制时，OAuth2 是一个常见的选择，但它并不是唯一的选择。OAuth2 是一种开放授权标准，它允许用户授权第三方应用程序访问其账户信息，而无需共享其用户名和密码。这个你可以尝试进行应用，项目中主要还是通过token来实现。

2、在 HTTP header 中，通常会将token保存在一个特殊的字段中，这个字段被称为 "Authorization" 字段。当使用 OAuth2 等认证协议时，Authorization 字段通常会被用来传递访问令牌，以便客户端可以向服务器证明其身份并获取访问权限。

3、对于Token和session的区别，Token 通常是基于 JSON Web Token (JWT) 标准的，它包含了用户的标识符和其他相关信息，以及签名来确保令牌的安全性。Session（会话）是一种服务器端的身份验证机制，服务器为每个用户创建一个会话对象，该对象包含了用户的身份验证信息和会话状态。这是二者的区别



shikey.com转载分享

shikey.com转载分享