## 场景

Web应用服务器提供restful api服务，且这些接口API有权限保护。技术上：

* 采用shiro安全框架做身份认证和鉴权；
* 使用auth0 token代替session做无状态身份凭证；
* 利用redis的超时机制配合token本身的超时机制做token刷新；
* 使用https防止密码明文；
* 使用swagger2做接口文档和测试。

## 基本知识

1. Auth0的jwt

* Jwt由三部分组成header、payload和sign，前两部分是json对象并base64转为字符串，第三部分是前两部分的加密签名（需要secret）
* Jwt的payload可以存放任意信息，类似session，所以叫自包含。
* Jwt的关键在于签发token和验证token

1. Jwt token与session的区别

* Jwt token与单机版web app是有区别的，但与分布式的共享session区别不大，两者都是有信息存放在云数据库中（token的secret和session自身）。
* 在客户端上cookie存放了token自身和session的id。
* Jwt token自身存放信息量少且明文，适用于不需要状态的应用（restful API）。Session可存放任意状态信息，需放在服务器上。

## Jwt token的验证机制

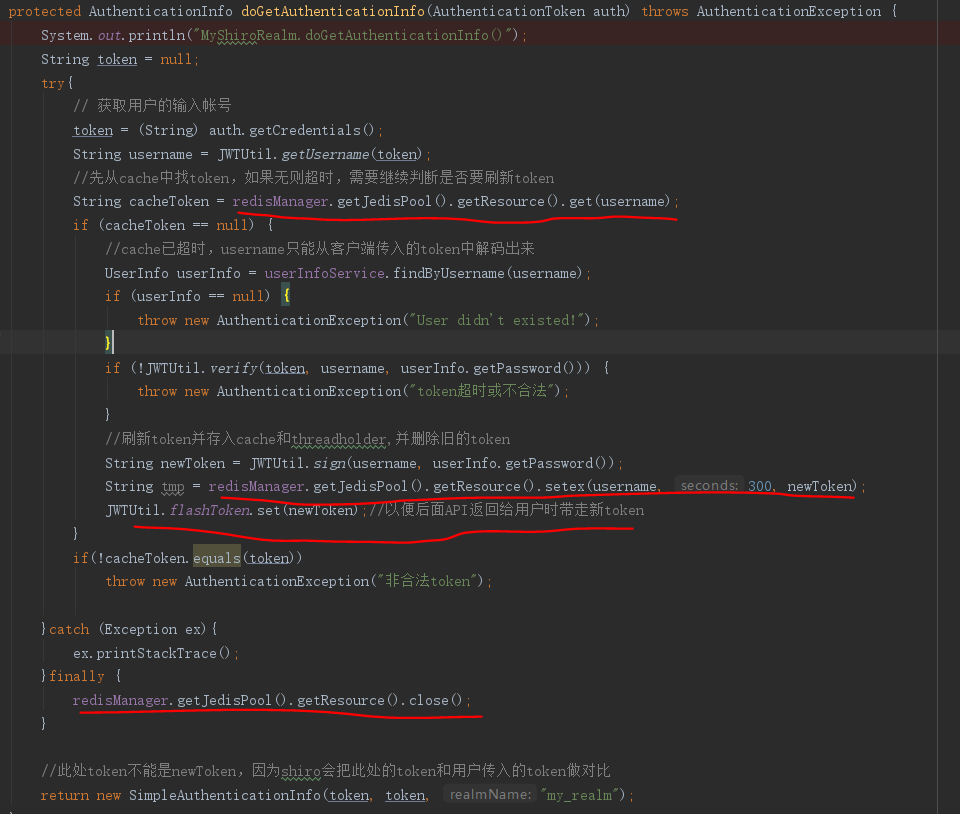
* 首先Jwt的验证逻辑与token签发逻辑是一样的。
* 由于jwt方案的目的是把以前放在服务器端的用户信息改放在客户端，这样服务器端的架构扩展就方便了。由此必须确保用户信息没有被客户端篡改。验证方法就是使用secret（保存在服务器端）加密jwt的前两部分，如果得到的签名与jwt第三部分的签名一样，那么就能证明jwt token没有被客户端篡改。
* 验证信息有两个方面：
* 一个是验证信息有无被篡改；
* 一个是验证信息是否达到服务器的要求，例如手机号码是否是服务器当前用户的手机号码（服务器端有需求是更改手机号码）。JWTVerifier的withClaim("username", username)。当发现用户手机号更新了，就要返回一个新的token给客户端。

## Auth0 token的超时处理方案

利用redis等的超时机制，把redis的超时时间比token的超时时间稍微短一点，流程如下，

1. Request到达服务器后，从redis中查找是否有相应的k-v。如果有则判断token值是否一样。
2. 如果无对应的k-v值，则表明token差不多超时或已超时了。如果是差不多超时则新建一个token返回给用户，如果是已超时多时则返回验证失败让用户再次登录。

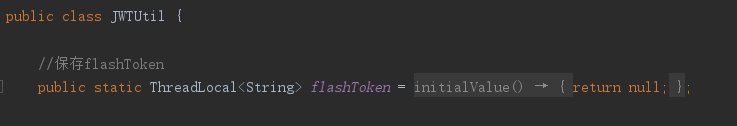
优点是实现简单，缺点是每次都要访问redis来验证是否超时。由于一个普通页面都可能多达几十次request请求，所以会增加redis服务器的负担（不能简单使用shiro的cacheManager缓存机制，这种机制是JVM缓存，因为这样利用不了redis的超时机制）



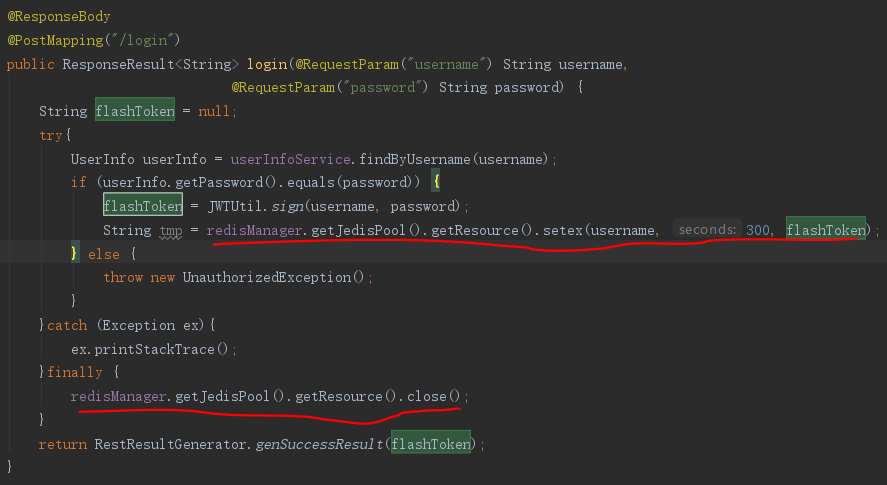
简易redis client配置。应该做成读配置文件



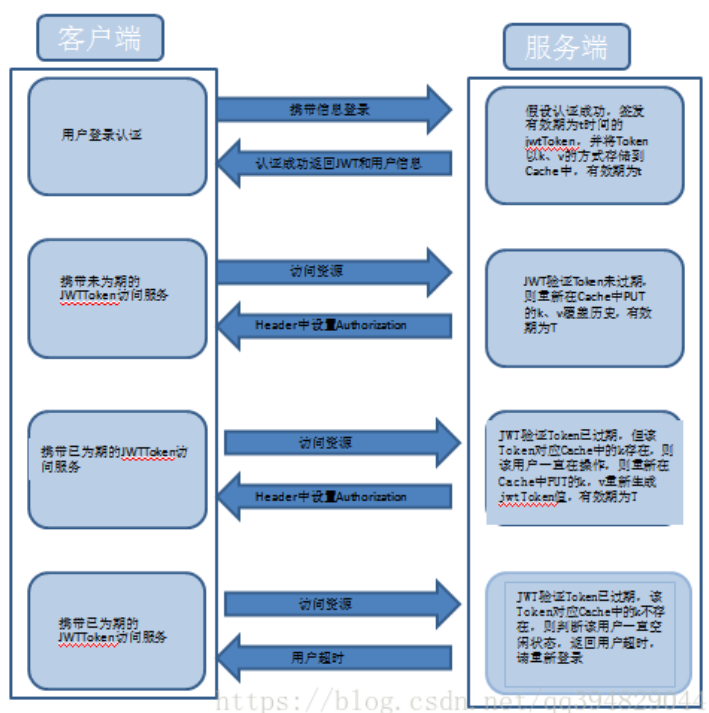
使用threadlocal做函数间变量传递



登录操作



注意：jedis每次用完后必须close，否则会链接溢出。



## 创建springboot2的restful api应用

1. 使用restful api风格创建一个对数据表的CRUD操作的接口应用，使用JPA

* 创建entity

使用了数据校验框架，JPA等技术。Mysql数据表会根据它自动生成

* 创建dao

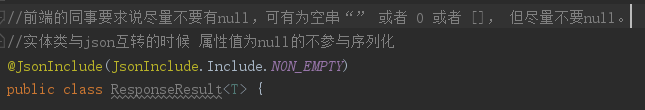
因为是使用JPA，所以最基本的CRUD实现都不需要写



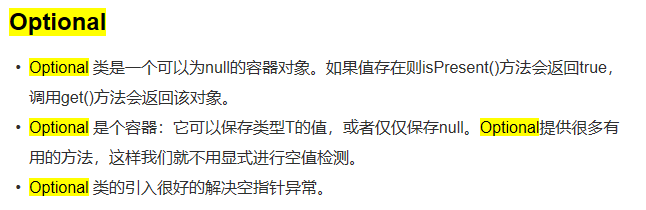
* 创建服务接口（service）
* 创建统一格式的响应返回

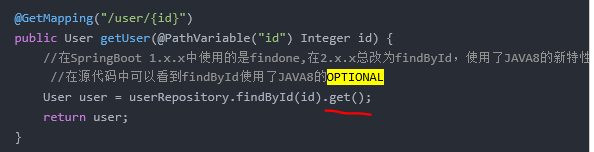
对于客户端开发或者网站开发而言，调用接口返回有统一的响应体，可以针对性的设计界面，代码结构更加清晰，层次也更加分明。内容可包括，

操作成功与否、数据本体、消息、错误代码（建议不要推给前端）



* 创建controller
* 所有api返回统一格式的响应。





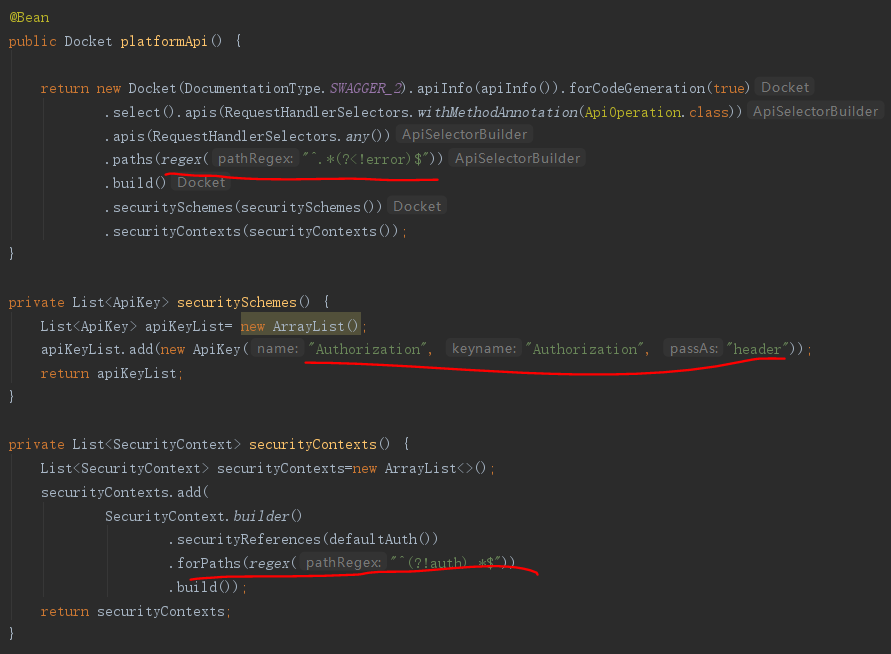
* 为一个domain对象创建6个基本的rest api操作，



* 创建统一异常处理

## 整合swagger2

1. 导入依赖
2. 创建swagger2的配置类
3. 为项目的controller层做文档和测试UI
4. 测试auth0守护的api时需要输入token，配置swagger2的token输入入口

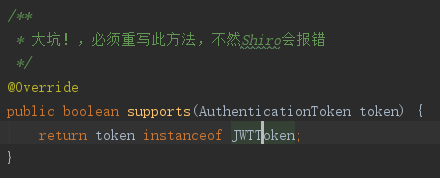


## 集成JWT Token



1. 生成jwt token签名的业务逻辑
2. 构建jwt token合法性验证逻辑，用户验证客户端提交的token是否合法
3. 构建jwt token的工具方法，例如获取token中的用户名信息
4. 实现JWTToken

JWTToken差不多就是Shiro用户名密码的载体。因为我们是前后端分离，服务器无需保存用户状态，所以不需要RememberMe这类功能，我们简单的实现下AuthenticationToken接口即可。因为token自己已经包含了用户名等信息，所以这里我就弄了一个字段。如果你喜欢钻研，可以看看官方的UsernamePasswordToken是如何实现的。

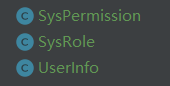
1. 选择jwt token作为客户端使用应用的身份凭据（非session id的方式）

## 整合shiro

使用shiro安全框架认证和鉴权客户端提交的用户名和密码（从http的request中提取），提交方式可以是form（web）或json（rest api），但都是明文。

1. 建立安全模块的业务领域模型和三层架构（DAO、SERVICE、MODEL）。

JPA会自动建表



1. 构建身份认证和鉴权业务逻辑模块

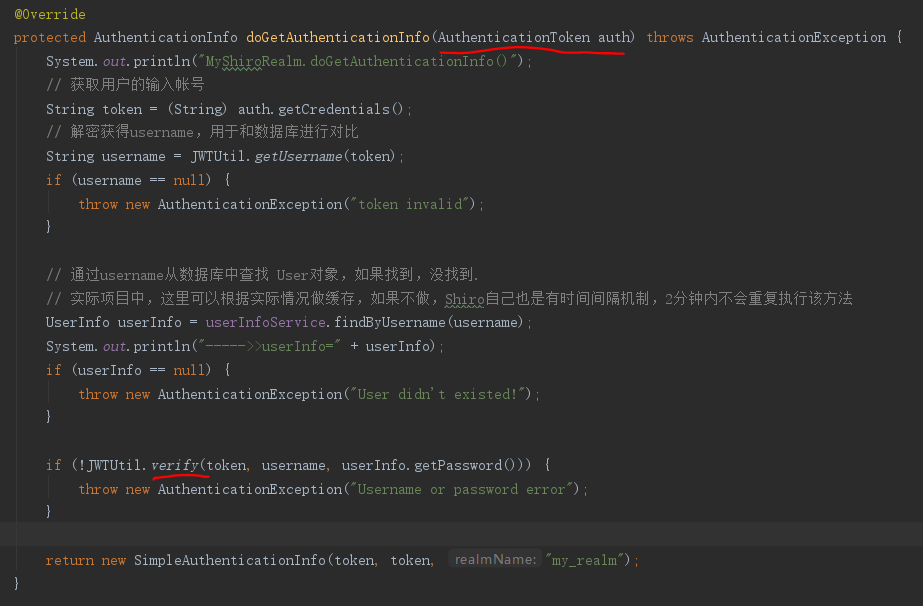
Realm：域，Shiro 从Realm获取安全数据（如用户、角色、权限），就是说SecurityManager要验证用户身份，那么它需要从Realm获取相应的用户进行比较以确定用户身份是否合法；

也需要从Realm得到用户相应的角色/权限进行验证用户是否能进行操作；可以把Realm看成DataSource ， 即安全数据源。

特殊场景，多realm同时存在，存在两张表user和admin，分别记录普通用户和管理员的信息。并且现在要实现普通用户和管理员的分开登录，即需要两个Realm——UserRealm和AdminRealm，分别处理普通用户和管理员的验证功能。



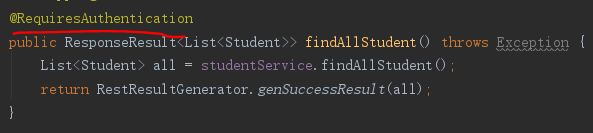
* 使用步骤1的安全业务模型做认证，从mysql中存储信息。



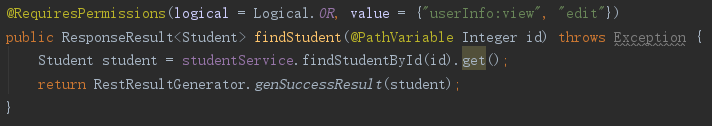
* 鉴权模块

## 使用shiro框架保护restful api接口

用户认证后才能使用的接口



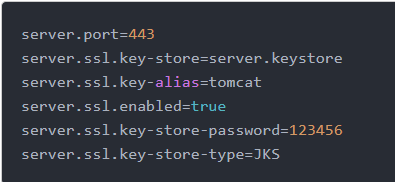
某两类角色才能使用的接口



某类权限才能使用的接口

## 使用https防止密码明文被盗

1. 使用jdk自带的keytool自建证书
2. 把keystone文件复制到项目的根目录并配置



1. 注入bean
2. Tomcat需要使用APR模式来实现SSL

由于最新版springboot2使用的内嵌tomcat是9.0.13，但其使用的native lib（libtcnative）是1.2.16。所以相应要使用1.2.18之后的版本，否则会出现UnsatisfiedLinkError错误。

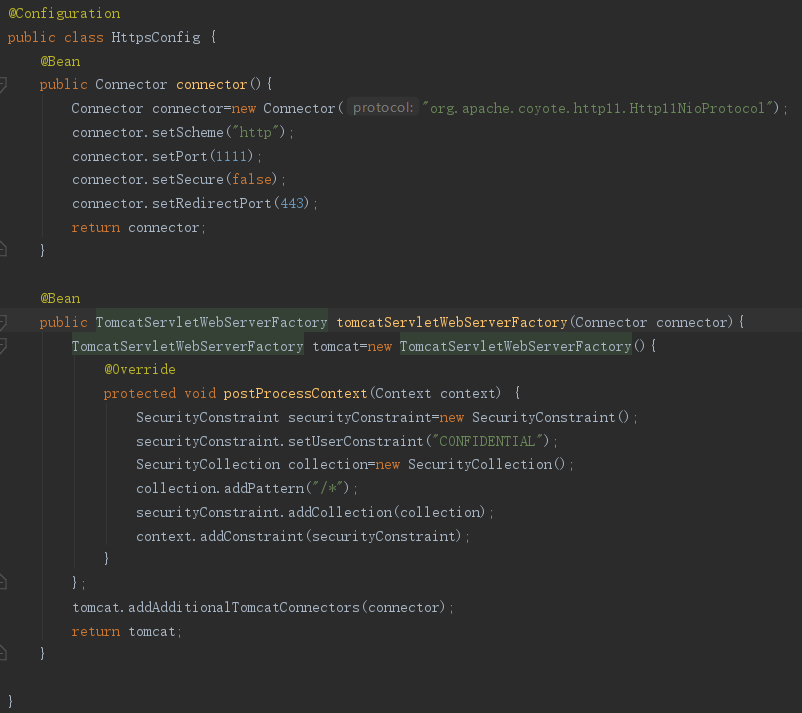


解决方法有，

降级使用9.0.12版本的tomcat



或自己编译最新的libtcnative版本。



## 总结

1. 本应用流程

* 使用shiro内置过滤器获取http request中authorization字段，如果没有该字段则是游客，可能需要登录
* 该authorization字段值是token，被截获后根据它生成jwt token对象以方便后面验证。
* 把token对象交给shiro的realm模块做身份认证，认证成功则放行。
* 应用的rest api接口由不同的权限注释守护，使用接口前会由realm做鉴权处理（获取当前用户的权限列表）。
* 游客可以使用login接口，并做登录业务，合法用户返回jwt token（签发）。

1. 需要进一步研究的点

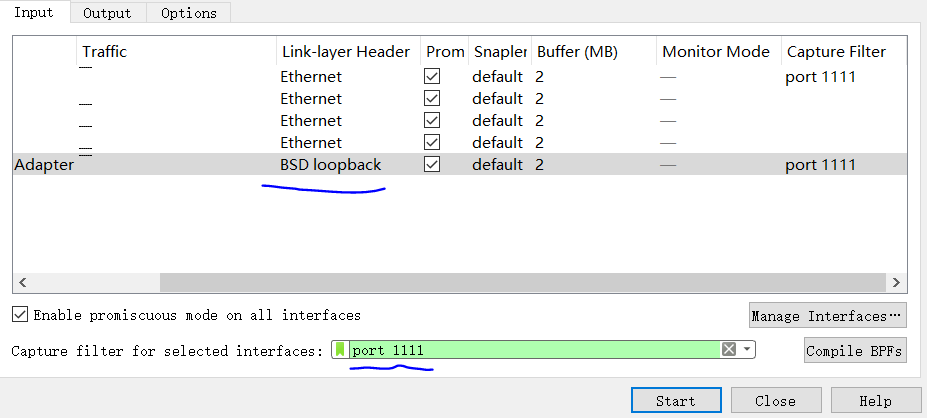
* Shiro-redis第三方包，它包装了一个session dao、redis manager、redis cache、集群和单点等组件

## 其他知识点

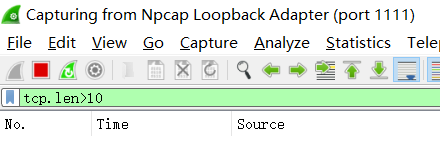
1. 使用wireshark检查swagger与服务器通讯情况

在swagger2 UI页面上测试restful API接口时发现没有应有的相应，使用wireshark检查原因。服务器端口号是1111，

* 设置wireshark capture filter，需要注意的是目标端口是在localhost上，所以要监听BSD loopback接口，否则捕捉不到包。



设置显示过滤器为tcp.len>10，这样才能把tcp keep alive等无用包过滤掉，



1. 客户端（浏览器或桌面应用）与web服务器的认证方式

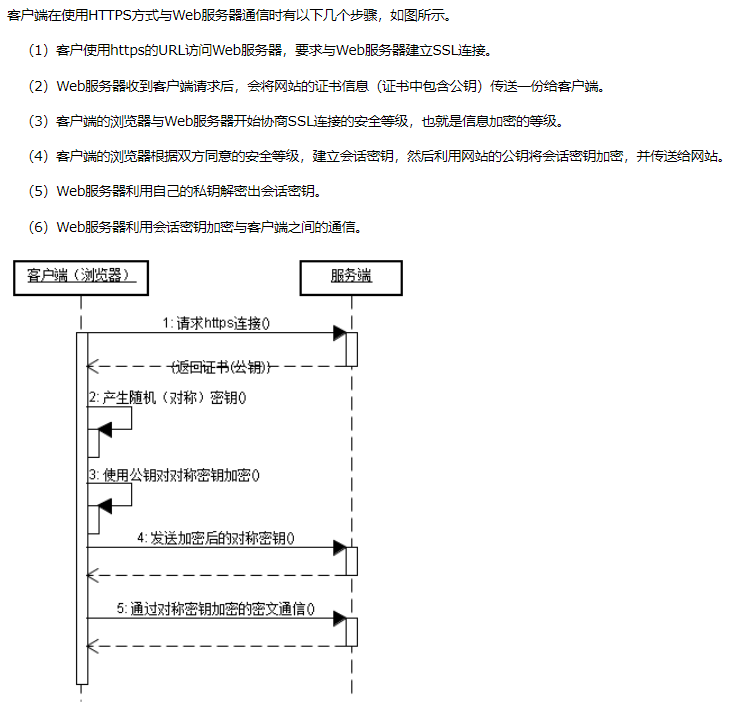
* HTTP基本认证

把用户名和密码用base64编码后放在http header的Authorization字段中发送给web服务器。

* HTTP OAuth认证

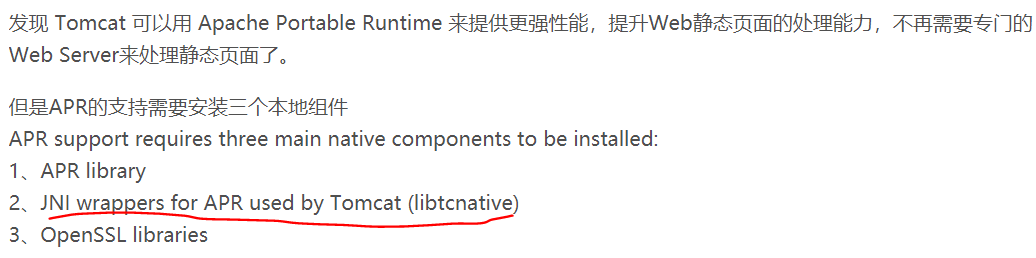
把token放在http header的Authorization字段中发送给web服务器。

1. https机制



1. Tomcat的三种模式

Tomcat有三种运营模式：bio、nio、apr，不同模式下Tomcat的运行效率差别比较大。



APR模式是从操作系统级别来解决异步的IO问题,大幅度的提高性能