Assignement2

项目链接：

组员：

17221294 杨洪源

17301066 张冰诚

项目要求：

RestApi

Oauth2 authentication

Providing Hateoas

Online Rest API document

RateLimiting for different type of users

Caching

RestApi:

在这里我们只展示一个restapi例子，其余一样请查看online document

Download ppt，pdf...:

Its resful api:

```
[
    {
        "fileName": "upload.pdf",
        "links": [
            {
                "rel": "download file",
                "href":       "http://localhost:9001/file/upload.pdf",
                "media": "GET",
                "title": "download file"
            },
            {
                "rel": "delete file",
                "href": "http://localhost:9001/file/upload.pdf",
                "media": "DELETE",
                "title": "delete file"
            }
        ]
    },
    {
        "fileName": "upload.pdf",
        "links": [
            {
                "rel": "download file",
                "href": "http://localhost:9001/file/upload.pdf",
                "media": "GET",
                "title": "download file"
            },
            {
                "rel": "delete file",
                "href": "http://localhost:9001/file/upload.pdf",
                "media": "DELETE",
                "title": "delete file"
            }
        ]
    }
]
```

上传文件的具体实现:

```
@Operation(summary = "Get All File Name", description = "", tags = { "file" })
@RequestMapping(value = "Allfile",method = RequestMethod.GET)
@ResponseBody
```

```
public List<FileZ> getAllFileZ() {
    List<FileZ> empsWithLinks = new ArrayList<>();
    List<FileZ> files = fileZService.getAllFileZ();
    if (!CollectionUtils.isEmpty(files)) {
        for (FileZ emp : files) {

            Link getEmplink =
WebMvcLinkBuilder.linkTo(FileUpDownLoadController.class).slash("file").slash(e
mp.getFileName()).withRel("download
file").withMedia("GET").withTitle("download file");
            Link delEmplink =
WebMvcLinkBuilder.linkTo(FileUpDownLoadController.class).slash("file")
                    .slash(emp.getFileName()).withRel("delete
file").withMedia("DELETE").withTitle("delete file");

            emp.add(getEmplink);
            emp.add(delEmplink);
            empsWithLinks.add(emp);
        }
    }
    return empsWithLinks;
}
```

Ouath2 autientication:

OAuth（Open Authorization，开放授权）是为用户资源的授权定义了一个安全、开放及简单的标准，第三方无需知道用户的账号及密码，就可获取到用户的授权信息

Ouath2具体实现:

配置权限服务器，用户密码处理器，判断从何处去读，从数据库还是内存中

```
import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.oauth2.config.annotation.configurers.ClientDetail
sServiceConfigurer;
import
org.springframework.security.oauth2.config.annotation.web.configuration.Author
izationServerConfigurerAdapter;
```

```
import
org.springframework.security.oauth2.config.annotation.web.configurers.Authoriz
ationServerEndpointsConfigurer;
import
org.springframework.security.oauth2.config.annotation.web.configurers.Authoriz
ationServerSecurityConfigurer;
import org.springframework.security.oauth2.provider.token.TokenStore;
import
org.springframework.security.oauth2.provider.token.store.JdbcTokenStore;

@Configuration
public class AuthServerConfig extends AuthorizationServerConfigurerAdapter {

  @Autowired
  DataSource ds;

  @Autowired
  AuthenticationManager authMgr;

  @Autowired
  private UserDetailsService usrSvc;

  @Bean
  public TokenStore tokenStore() {
    return new JdbcTokenStore(ds);
  }

  @Bean("clientPasswordEncoder")
  PasswordEncoder clientPasswordEncoder() {
    return new BCryptPasswordEncoder(4);
  }

  @Override
  public void configure(AuthorizationServerSecurityConfigurer cfg) throws
Exception {

    // This will enable /oauth/check_token access
    cfg.checkTokenAccess("permitAll");

    // BCryptPasswordEncoder(4) is used for oauth_client_details.user_secret
    cfg.passwordEncoder(clientPasswordEncoder());
  }

  @Override
  public void configure(ClientDetailsServiceConfigurer clients) throws
Exception {
    clients.jdbc(ds)
            .withClient("client_code")
            .secret(clientPasswordEncoder(
```

配置的用户密码服务

```
import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.BeanIds;
import
org.springframework.security.config.annotation.authentication.builders.Authent
icationManagerBuilder;
import
org.springframework.security.config.annotation.authentication.configurers.prov
isioning.JdbcUserDetailsManagerConfigurer;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCo
nfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
public class UserSecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  DataSource ds;

  @Override
  @Bean(BeanIds.USER_DETAILS_SERVICE)
  public UserDetailsService userDetailsServiceBean() throws Exception {
    return super.userDetailsServiceBean();
  }

  @Override
  @Bean(name = BeanIds.AUTHENTICATION_MANAGER)
  public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
  }

  @Bean("userPasswordEncoder")
  PasswordEncoder userPasswordEncoder() {
    return new BCryptPasswordEncoder(4);
  }

  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception
{
    // BCryptPasswordEncoder(4) is used for users.password column
    JdbcUserDetailsManagerConfigurer<AuthenticationManagerBuilder> cfg =
auth.jdbcAuthentication()
        .passwordEncoder(userPasswordEncoder()).dataSource(ds);

    cfg.getUserDetailsService().setEnableGroups(true);
    cfg.getUserDetailsService().setEnableAuthorities(false);
  }
}
```

运行截图:

```
http://localhost:9090/oauth/authorize?
client_id=appclient&response_type=code&scope=all&redirect_uri=http://www.baidu
.com
```

进入登入界面，取得code

```
https://www.baidu.com/?code=LWDYgW
```

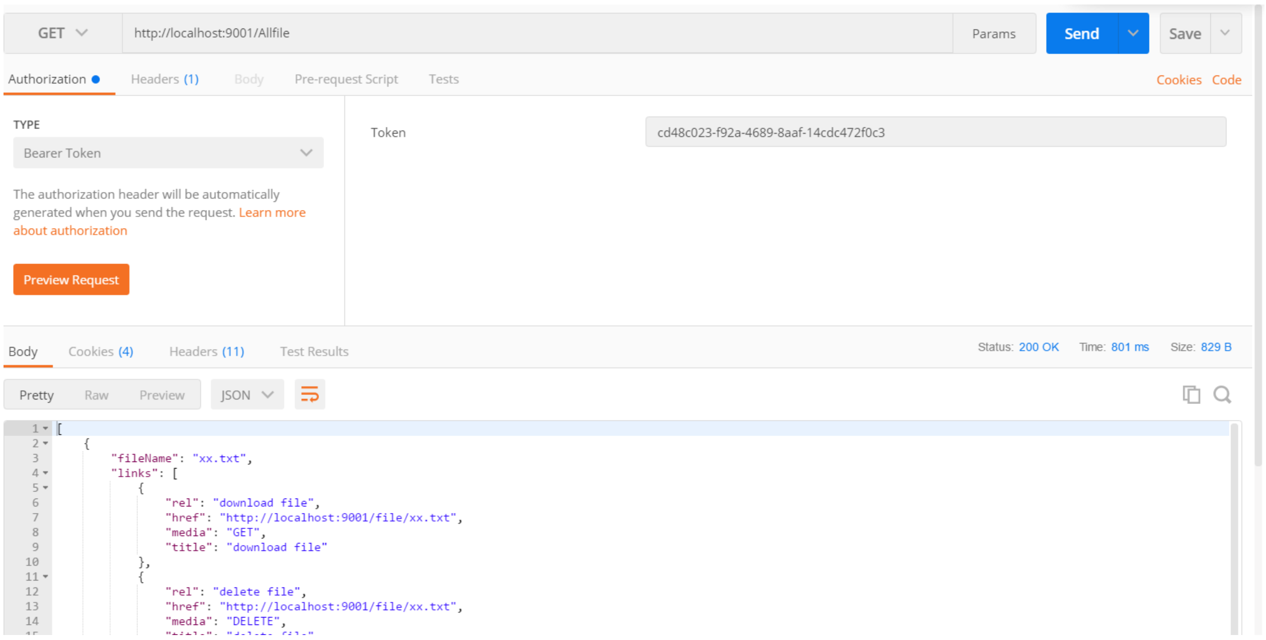
访问如下链接:

```
http://localhost:9090/oauth/token?
client_id=appclient&grant_type=authorization_code&redirect_uri=http://www.baid
u.com&client_secret=appclient@123&code=LWDYgW
```

Access token:

```
{
    "access_token": "cd48c023-f92a-4689-8aaf-14cdc472f0c3",
    "token_type": "bearer",
    "refresh_token": "d0144f0f-0757-4fb8-88bd-1f9df4a22a22",
    "expires_in": 113,
    "scope": "all"
}
```

最后出现授权码形式:


GET http://localhost:9001/Allfile

Authorization: TYPE Bearer Token

Token: cd48c023-f92a-4689-8aaf-14cdc472f0c3

The authorization header will be automatically generated when you send the request. Learn more about authorization

Status: 200 OK Time: 801 ms Size: 829 B

```
[
    {
        "fileName": "xx.txt",
        "links": [
            {
                "rel": "download file",
                "href": "http://localhost:9001/file/xx.txt",
                "media": "GET",
                "title": "download file"
            },
            {
                "rel": "delete file",
                "href": "http://localhost:9001/file/xx.txt",
                "media": "DELETE",
```

Providing Hateoas:

HATEOAS（*Hypermedia as the engine of application state*）*是 REST 架构风格中最复杂的约束，也是构建成熟 REST 服务的核心。它的重要性在于打破了客户端和服务器之间严格的契约，使得客户端可以更加智能和自适应，而 REST 服务本身的演化和更新也变得更加容易。*

Implementation:

这里的实现动作是不一样的

运行截图：

以资源的方式来展示：

```
[
    {
        "fileName": "upload.pdf",
        "links": [
            {
                "rel": "download file",
                "href":       "http://localhost:9001/file/upload.pdf",
                "media": "GET",
                "title": "download file"
            },
            {
                "rel": "delete file",
                "href": "http://localhost:9001/file/upload.pdf",
                "media": "DELETE",
                "title": "delete file"
            }
        ]
    },
    {
        "fileName": "upload.pdf",
        "links": [
            {
                "rel": "download file",
                "href": "http://localhost:9001/file/upload.pdf",
                "media": "GET",
                "title": "download file"
            },
            {
                "rel": "delete file",
                "href": "http://localhost:9001/file/upload.pdf",
                "media": "DELETE",
                "title": "delete file"
            }
        ]
    }
]
```

Online Rest Api document:

参照github文件目录下Swagger.html

RateLimiting for different type of users:

编写两个实现类进行限流处理

拦截器根据每个请求里的属性，判断用户，根据身份不同创建bucket，其中不同类型的bucket拥有的容量不同和创建令牌速率也不同，从而达到对不同用户产生不同的限流效果。

Perclientraitliming.java:拦截器

```java
package com.example.demo.raitLimit;

import java.time.Duration;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.TimeUnit;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.http.HttpStatus;
import org.springframework.web.servlet.HandlerInterceptor;

import io.github.bucket4j.Bandwidth;
import io.github.bucket4j.Bucket;
import io.github.bucket4j.Bucket4j;
import io.github.bucket4j.ConsumptionProbe;
import io.github.bucket4j.Refill;

public class PerClientRateLimitInterceptor implements HandlerInterceptor {

  private final Map<String, Bucket> buckets = new ConcurrentHashMap<>();

  private final Bucket freeBucket = Bucket4j.builder()
      .addLimit(Bandwidth.classic(10, Refill.intervally(10,
Duration.ofMinutes(1))))
      .build();

  @Override
  public boolean preHandle(HttpServletRequest request, HttpServletResponse
response,
      Object handler) throws Exception {

    Bucket requestBucket;

    String apiKey = request.getHeader("X-api-key");
    if (apiKey != null && !apiKey.trim().isEmpty()) {
      if (apiKey.startsWith("1")) {
        requestBucket = this.buckets.computeIfAbsent(apiKey, key ->
premiumBucket());
      }
      else {
        requestBucket = this.buckets.computeIfAbsent(apiKey, key ->
standardBucket());
      }
    }
    else {
      requestBucket = this.freeBucket;
    }

    ConsumptionProbe probe = requestBucket.tryConsumeAndReturnRemaining(1);
    if (probe.isConsumed()) {
      response.addHeader("X-Rate-Limit-Remaining",
          Long.toString(probe.getRemainingTokens()));
      return true;
    }

    response.setStatus(HttpStatus.TOO_MANY_REQUESTS.value()); // 429
    response.addHeader("X-Rate-Limit-Retry-After-Milliseconds",

 Long.toString(TimeUnit.NANOSECONDS.toMillis(probe.getNanosToWaitForRefill()))
);
```

```
    return false;
  }

  private static Bucket standardBucket() {
    return Bucket4j.builder()
        .addLimit(Bandwidth.classic(50, Refill.intervally(50,
Duration.ofMinutes(1))))
        .build();
  }

  private static Bucket premiumBucket() {
    return Bucket4j.builder()
        .addLimit(Bandwidth.classic(100, Refill.intervally(100,
Duration.ofMinutes(1))))
        .build();
  }

}
```

配置文件：

```
package com.example.demo.raitLimit;

import java.time.Duration;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

import io.github.bucket4j.Bandwidth;
import io.github.bucket4j.Bucket;
import io.github.bucket4j.Bucket4j;
import io.github.bucket4j.Refill;

@Configuration
public class RateLimitConfig implements WebMvcConfigurer {

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
      registry.addInterceptor(new PerClientRateLimitInterceptor())
          .addPathPatterns("/student/**");
      registry.addInterceptor(new PerClientRateLimitInterceptor())
        .addPathPatterns("/teacher/**");
      registry.addInterceptor(new PerClientRateLimitInterceptor())
          .addPathPatterns("/manager/**");

    }
```

```
  }
```

Caching:

application.properties:

```
#redis
redis.hostname = localhost
redis.port = 6379
redis.ttl.hours = 24
redis.timeout.secs= 15
redis.socket.timeout.secs= 15
```

RedisCacheConfig.java:

```
package com.example.demo.cache;

import java.time.Duration;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.cache.annotation.CachingConfigurer;
import org.springframework.cache.annotation.CachingConfigurerSupport;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.cache.interceptor.CacheErrorHandler;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.cache.RedisCacheConfiguration;
import org.springframework.data.redis.cache.RedisCacheManager;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import
org.springframework.data.redis.connection.lettuce.LettuceClientConfiguration;
import
org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.RedisSerializationContext;
import org.springframework.data.redis.serializer.RedisSerializer;

import io.lettuce.core.ClientOptions;
import io.lettuce.core.SocketOptions;

@Configuration
@EnableCaching
public class RedisCacheConfig extends CachingConfigurerSupport implements
CachingConfigurer {

  @Value("${redis.hostname:localhost}")
  private String redisHost;
  @Value("${redis.port:6379}")
  private int redisPort;

  @Value("${redis.timeout.secs:1}")
  private int redisTimeoutInSecs;

  @Value("${redis.socket.timeout.secs:1}")
  private int redisSocketTimeoutInSecs;

  @Value("${redis.ttl.hours:1}")
  private int redisDataTTL;

  @Bean
  public LettuceConnectionFactory redisConnectionFactory() {

    final SocketOptions socketOptions =
SocketOptions.builder().connectTimeout(Duration.ofSeconds(redisSocketTimeoutIn
Secs)).build();

    final ClientOptions clientOptions =
ClientOptions.builder().socketOptions(socketOptions).build();

    LettuceClientConfiguration clientConfig =
LettuceClientConfiguration.builder()

.commandTimeout(Duration.ofSeconds(redisTimeoutInSecs)).clientOptions(clientOp
tions).build();
    RedisStandaloneConfiguration serverConfig = new
RedisStandaloneConfiguration(redisHost, redisPort);

    final LettuceConnectionFactory lettuceConnectionFactory = new
LettuceConnectionFactory(serverConfig, clientConfig);
    lettuceConnectionFactory.setValidateConnection(true);
    return lettuceConnectionFactory;

  }

  @Bean
  public RedisTemplate<Object, Object> redisTemplate() {
    RedisTemplate<Object, Object> redisTemplate = new RedisTemplate<Object,
Object>();
    redisTemplate.setConnectionFactory(redisConnectionFactory());
    return redisTemplate;
  }

  @Bean
  public RedisCacheManager redisCacheManager(LettuceConnectionFactory
lettuceConnectionFactory) {
    RedisCacheConfiguration redisCacheConfiguration =
RedisCacheConfiguration.defaultCacheConfig().disableCachingNullValues()
        .entryTtl(Duration.ofHours(redisDataTTL))

.serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerialize
r(RedisSerializer.java()));

    redisCacheConfiguration.usePrefix();

    RedisCacheManager redisCacheManager =
RedisCacheManager.RedisCacheManagerBuilder.fromConnectionFactory(lettuceConnec
tionFactory)
        .cacheDefaults(redisCacheConfiguration).build();

    redisCacheManager.setTransactionAware(true);
    return redisCacheManager;
  }


  @Override
  public CacheErrorHandler errorHandler() {
    return null;
  }
}
```

将注解添加在方法头上：

```
@Cacheable(value= "classCache", key= "#p0")
    @RequestMapping("/student/home/time")//加载选择课程的时间，实现降序排列（成功）
    public JSONObject load_home_time(@RequestBody String json){
        System.out.println(json);       //ajax 对指定路有传参
        JSONObject jsonObject = JSONObject.parseObject(json);
        return
service.load_course_time(jsonObject.getString("id"),jsonObject.getString("type
"));
    }
```