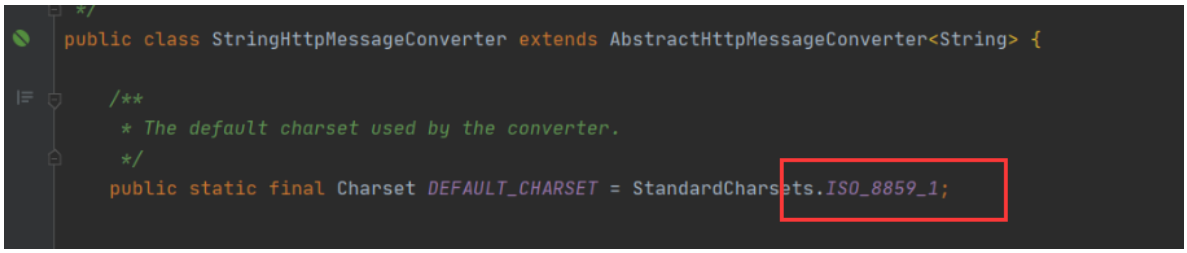
Spring MVC框架学习 ---- 传递参数

0、解决返回数据是乱码的问题

通过@ResponseBody注解的方式实现json格式传到页面的方法。首先查看源代码如下图，springmvc的默认编码是“ISO-8859-1”;



而我们通常编码都是使用UTF-8,所以我们需要在springmvc的注解配置中处理json格式的时候应该修改一下默认的编码格式。

springmvc配置文件中代码如下：

```
   <!--     开启mvc注解驱动,在注解的标签中加入  返回数据类型编码格式设置-->
<mvc:annotation-driven>
    <mvc:message-converters register-defaults="true">
     <bean
class="org.springframework.http.converter.StringHttpMessageConverter">
        <property name="supportedMediaTypes">
            <list>
                <value>text/html;charset=UTF-8</value>
                <value>application/json;charset=UTF-8</value>
            </list>
        </property>
    </bean>
    </mvc:message-converters>
</mvc:annotation-driven>
```

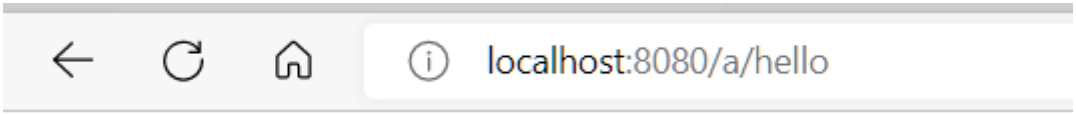
1、传递单个参数

传递单个参数，接收前端传递的参数，必须保证方法中的参数名与前端传递的key值保持一致

接收前端name的参数

```
    @RequestMapping(value = "/hello",produces =
"application/json;charset=utf8")
    @ResponseBody
    public String getParam(String name){
        return "注册成功："+name;
    }
```

发送get请求，传递name参数，返回结果



注册成功：null

不传递参数默认值为null，传递name参数返回后端处理过的结果。

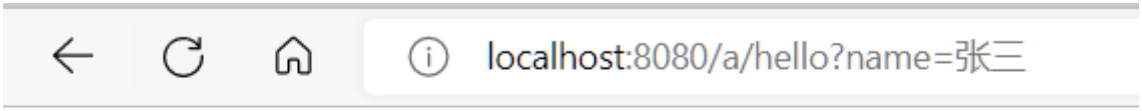
注册成功：张三

2、传递多个参数

接收前端的name 和 age参数

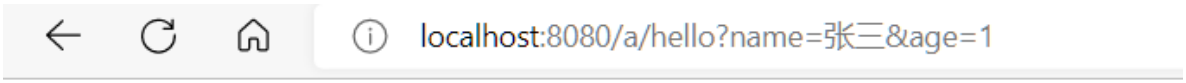
```
@RequestMapping(value = "/hello",produces = "application/json;charset=utf8")
    @ResponseBody
    public String getParam(String name,Integer age){
        return "注册成功："+name +" 年龄："+age;
    }
```

如果不传递age参数，返回age为null



注册成功：张三 年龄：null

传递name参数和age参数，后端会根据key值进行接收，处理后将数据返回给前端结果

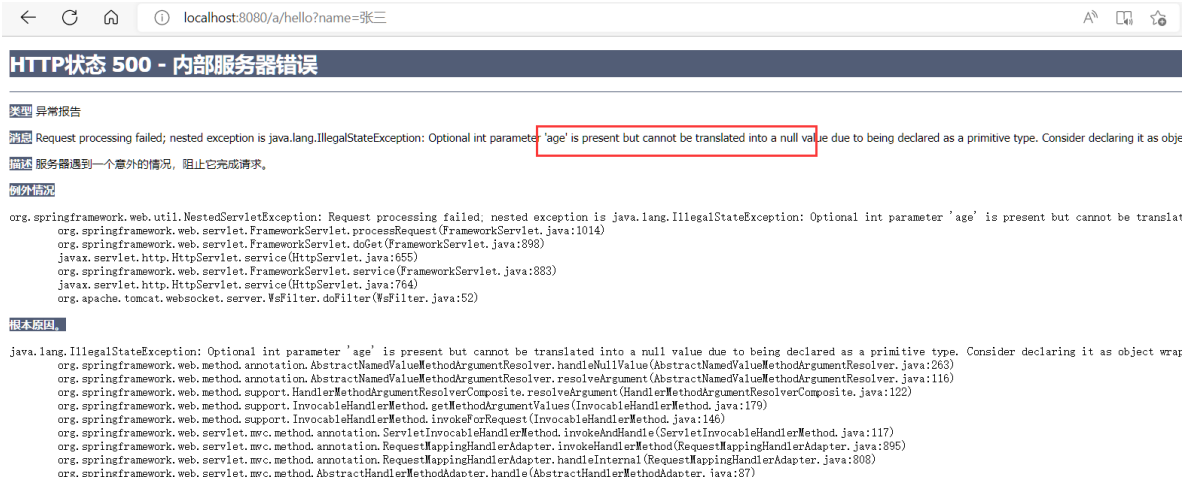


注册成功：张三 年龄：1

和上面的不一样，这次后端接收的age类型为 int

```
    @RequestMapping(value = "/hello",produces =
"application/json;charset=utf8")
    @ResponseBody
    public String getParam(String name,int age){
        return "注册成功："+name +"年龄："+age;
    }
```

如果我们不传递age的话，那么默认是一个null，那么类型就不匹配，报了500服务器内部错误的异常。



所以在这里一定要说明一点：

传递普通类型一定要传递包装类型，这样可以接收null值，不会报错。

3、接收对象类型

如果我们要接收一个对象类型的参数的话，那么我们先自己创建一个实体类对象，将这个对象类型作为参数传递到方法中。 前端传递参数的时候只需要 key值 和对象类型中的 属性名相同即可。

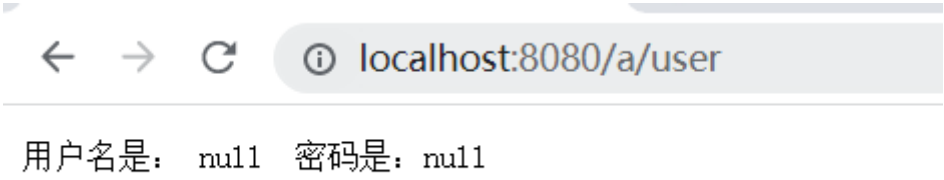
创建一个实体类对象，属性有userName、password

```
@Component
public class User {
    private String userName;
    private String password;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

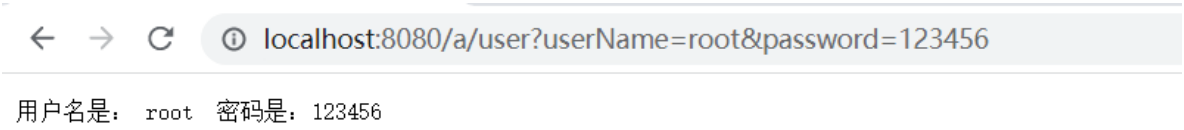
将对象作为参数传入方法中

```
    @RequestMapping("/user")
    @ResponseBody
    public String getUser(User user){
        return "用户名是：  "+user.getUserName() + "  密码是："
+user.getPassword();
    }
```

什么参数也不传递，返回结果都为null



前端将key值与对象的属性对应（必须一致，对大小写也敏感），传递参数



4、后端参数重命名

在某些特殊的情况下，前端传递的参数key与后端接收的参数名不一致，比如前端传递了一个time给后端，而后端又是用 createTime来接收的，这样就会出现参数接受不到的情况。我们可以使用 @RequestParam 来给前端参数的key重命名

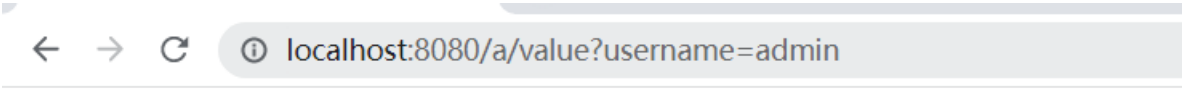
@Requestparam

这个注解放到对应参数的前面，里面填入对应前端参数的key，那么我们就完成了前端参数key与后端接收参数名的一个映射，即使双方key不一致也能够成功接收。

给后端参数重命名为username，使得前端传递的username 与后端的参数 name 形成映射关系，能够成功接收。

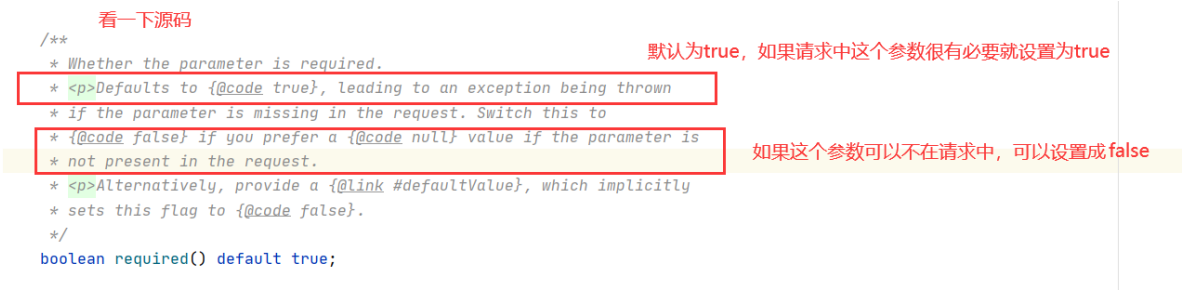
```
    @RequestMapping("/value")
    @ResponseBody
    public String getParam(@RequestParam("username") String name){
        return  "注册成功: "+name;
    }
```

前端传递参数 key为 username



注册成功：admin

@RequestParam 既能对前端参数重命名，也能保证该参数是否是 **必传参数**



默认为true，如果设置为true，那么这个参数是必传参数，如果在请求中没有传递，必会报错

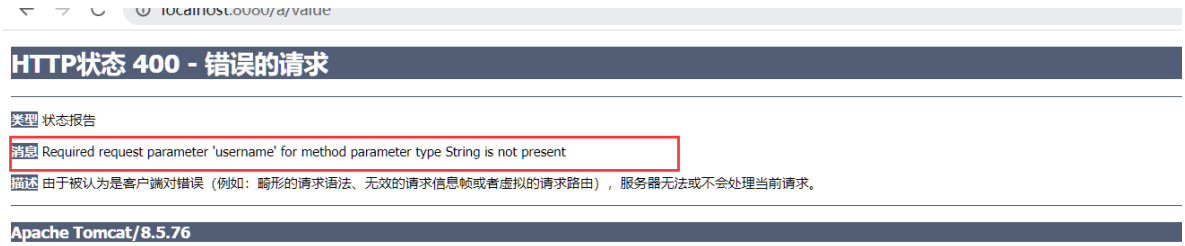
设置成false，那么这个参数可传可不传，不传的话默认为null

下面我们来看一下@RequestParam 的使用

设置required为true，（默认为true，可以省略），此时name为必传参数

```
1  @RequestMapping("/value")
2  @ResponseBody
3  public String getParam(@RequestParam(value = "username" ,required = true)
   String name){
4        return  "注册成功: "+name;
5  }
```

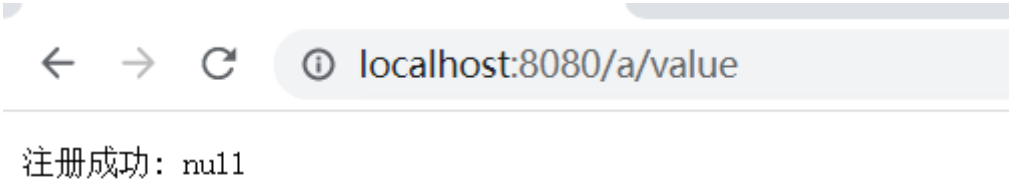
如果没有传递name参数，发生400错误，客户端错误，缺少必要的参数username



设置成非必传参数，必须显式设置 @RequestParam 的 required 属性为 false

```java

 @RequestMapping("/value")
 @ResponseBody
 public String getParam(@RequestParam(value = "username" ,required = true)
String name){
      return  "注册成功: "+name;
 }
```

没有传递参数，默认为null，没有发生异常，访问成功。



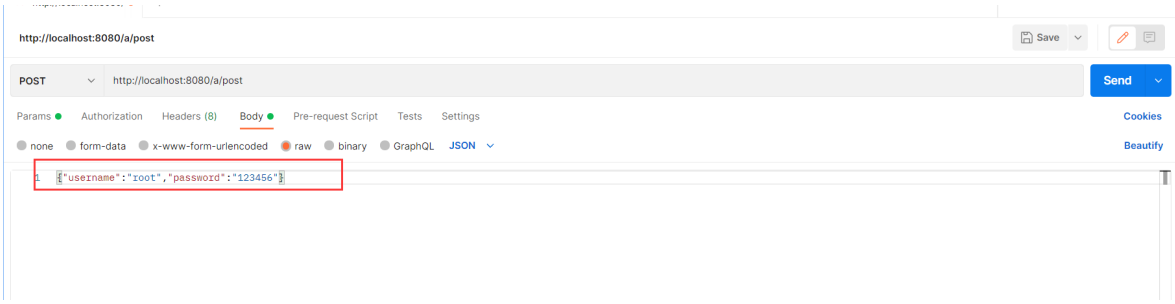
5、接收 JSON 类型

前端有可能会给后端传递一个 JSON格式类型的对象，那么后端如何接收呢?

（1）在pom.xml中引入 JSON相关依赖,否则无法接收JSON数据

```xml
<!--spring mvc-json依赖-->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.9.9</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-core</artifactId>
            <version>2.9.9</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-annotations</artifactId>
            <version>2.9.9</version>
        </dependency>
```

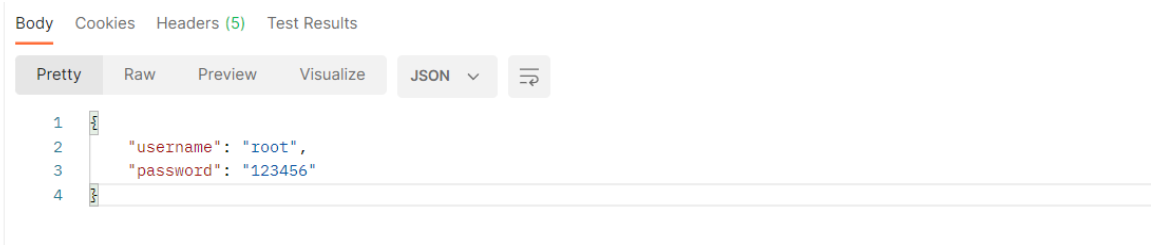
（2）构造POST请求，传递JSON格式的数据



（3）将JSON 数据 用实体对象来接收，保证key与属性名一致，同时必须加上 @RequestBody

```
    @PostMapping(value = "/post",produces = "application/json")
    @ResponseBody
    public Object getPost(@RequestBody User user){
        return user;
    }
```

（4）查看返回结果，成功接收与返回。



@RequestBody

只有当前注解使用在 JSON接收对象之前，当前接口才能成功的获取到前端的 JSON格式对象。

在前后端分离的使用中非常常见

6、RestFul 风格 API 的传参

什么是Restful 风格的API呢?

简单说一下，传参方式不一样

之前我们前端传递参数时 在映射路由后面？ 加上key值与value，通过&分割,如同以下类型的

> http://localhost:8080/a/value?username=root&password=123456

Restful 风格的 传参时 在接口后面直接使用/ + 值

> http://localhost:8080/a/value/root/123456

那么后端怎么接收这样的参数呢?

```
    @RequestMapping("/rest/{username}/{password}")
    @ResponseBody
    public String getRest( @PathVariable String username,@PathVariable String password){
        return "用户名是：  "+username + " 密码是：  "+password;
    }
```

- 在映射路由的时候，将后面参数的key值标识
- 在传参的时候，加上 @PathVariable 注解，自动将 上面注册路由的key与传递参数名所对应，必须一致

@PathVariable

在使用Restful风格的API 接口是，必须在参数前加上此注解。

如果什么属性也不写，那么属性为默认，会根据 后面的参数名 与 路由中的key值进行匹配，进而传递参数，同时和@RequestParam 属性一样，默认为required=true，为必传参数，如果不传递会发生500错误

最重要的是 values 和 required 两个属性，和 @RequestParam 属性的用法一样。

- value 可以将路由中的key值 与 后端的 形参变量名 进行映射，可以对前端 key 值重命名。
- required 默认为true,设置此参数为必传参数，设置为false，那么可传可不传，不传默认为null

7、传递文件参数

（1）在传递文件之前得配置文件相关的参数

在web.xml 中 servlet 标签中加入配置

设置传递文件的大小、传递的速度等等....

```
  <servlet>
    <servlet-name>springmvc</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>


    <multipart-config>
      <max-file-size>20848820</max-file-size>
      <max-request-size>418018841</max-request-size>
      <file-size-threshold>1048576</file-size-threshold>
    </multipart-config>


  </servlet>

  <servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
```

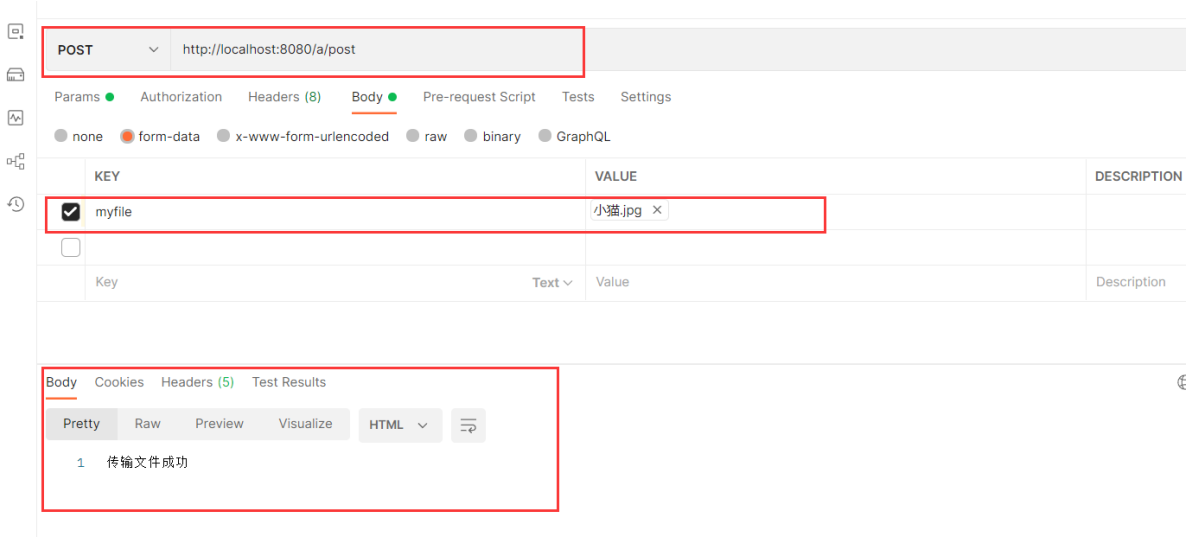
（2）传递文件使用 @RequestPart

必须标识 @RequestPart 使用在参数前面，参数的类型为 MultipartFile ,默认为必传参数，可以手动修改required=false,可以重命名后端形参名字。

```
  @PostMapping("/post")
  @ResponseBody
  public String getFile(@RequestPart("myfile") MultipartFile file) throws
IOException {
        file.transferTo(new File("C:\\Users\\rain7\\Desktop\\test.jpg"));
        return "传输文件成功";
  }
```

transferTo 方法 就是将 一个文件传送到一个路径当中，后面得加上自己决定的文件名。

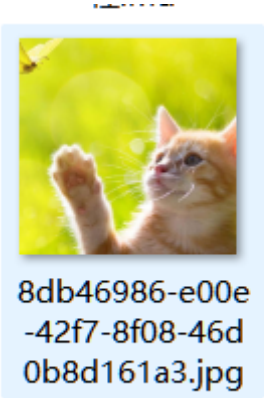
使用postman传递 文件参数，返回结果



我们来写一个更加规范的文件传输流程

```
    @PostMapping("/post")
    @ResponseBody
    public String getFile(@RequestPart(value = "myfile") MultipartFile
file) throws IOException {

        //1. 上传文件目录（发送到的目录位置）
        String upLoadPath = "C:\\Users\\rain7\\Desktop\\";

        //2. 获取文件后缀名，生成随机的文件名 （UUID）
        String fileName = UUID.randomUUID()+file.getOriginalFilename()
                .substring(file.getOriginalFilename().lastIndexOf("."));

        //3.拼接成一个完整的文件路径，进行参数文件传输
        file.transferTo(new File(upLoadPath+fileName));
        return "传输文件成功";
    }
```

传递成功，在传输目录中生成一个随即命名的文件



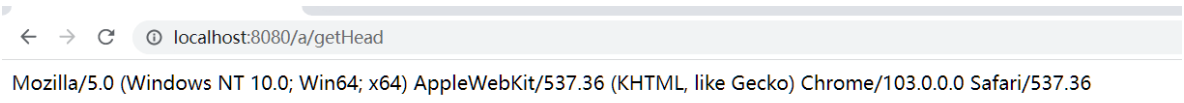
8、传递 Header 参数

在之前，我们使用 servlet 的时候，是使用 HttpServletRequest 来获取 Header 的，而 Spring MVC 底层也是调用的 servlet，所以完全可以靠之前 servlet 的方式读取 header、cookie、session.

（1）通过 HttpServletRequest 内置参数 进行获取 Header 参数

```
    @RequestMapping("/getHead")
    @ResponseBody
    public String getHead(HttpServletRequest request){
        String header = request.getHeader("User-Agent");
        return header;
    }
```

访问接口，成功拿到数据



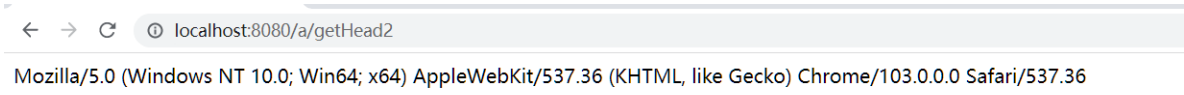
（2）我们在Spring MVC 中有一种更简单读取 Header 参数的方式

@RequestHeader

@RequestHeader 放在参数前面，value 为 想获取的 header 的 key 值，为了避免获取空值报错，required =false

```
    @RequestMapping("/getHead2")
    @ResponseBody
    public String getHead2(@RequestHeader(value = "User-Agent",required =
false) String UA){
        return UA;
    }
```

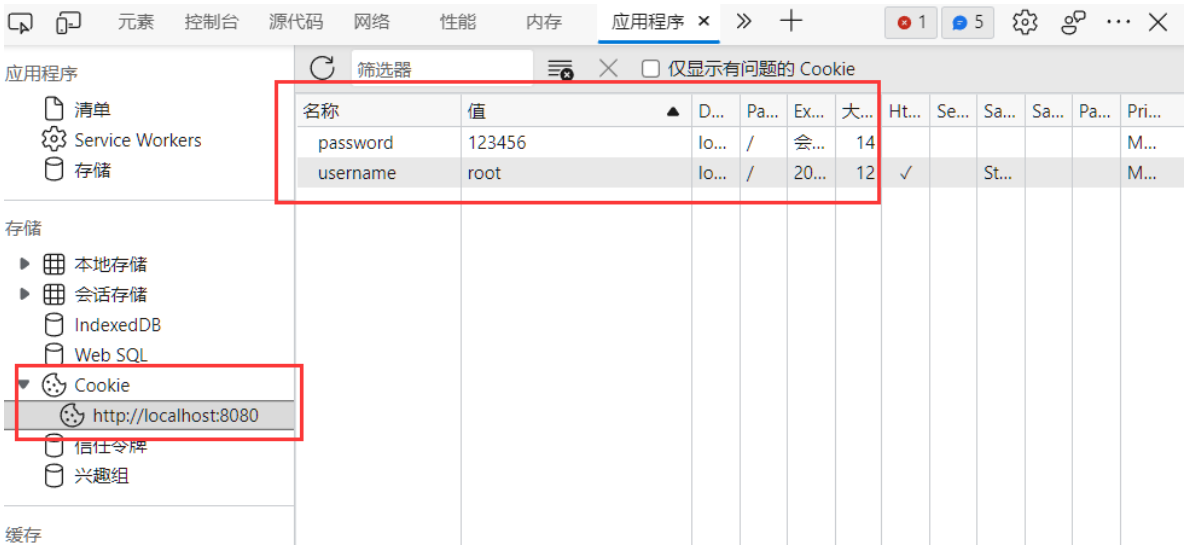
访问接口，获取Header成功



9、传递 Cookie 参数

和上面获取Header 参数一样，也有两种方式

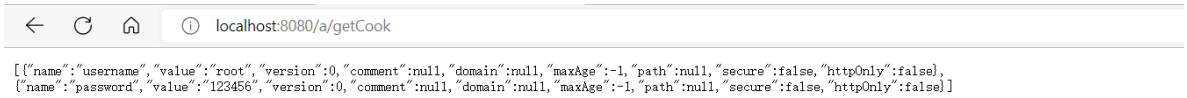
（0）首先在这个接口的地址，通过浏览器手动的设置一些cookie值



（1）通过 HttpServletRequest 获取Cookie参数

```java
    @RequestMapping("/getCook")
    @ResponseBody
    public Object getCook(HttpServletRequest request){
        Cookie[] str = request.getCookies();
        return str;
    }
```

访问接口，拿到 cookie 内容

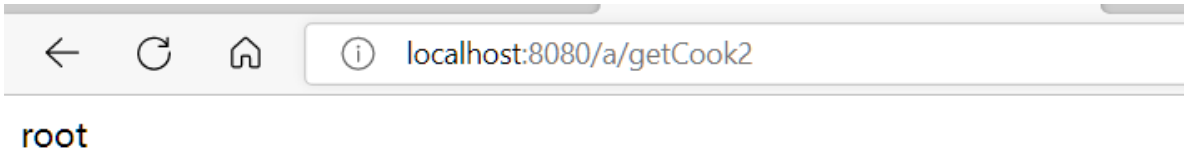


（2）通过注解 @CookieValue 的方式拿到 cookie参数

@CookieValue

```java
 @RequestMapping("/getCook2")
 @ResponseBody
 public String getCook2(@CookieValue(value = "username",required = false)
String str ){
     return str;
 }
```

通过 key 值拿到对应的 value，同时设置为 required = false,避免空值报错



10、传递 Session 参数

这个和之前两个一样，而且非常的常用。

在登陆的时候经常会用到 Session

（1）使用 servlet 的方式 传递 session 参数

登陆的时候，设置session

```
  @RequestMapping("/setSession")
    @ResponseBody
    public String setSession(HttpServletRequest request,String name){
        // 获取session，如果没有session的话，那么开启session
        HttpSession session = request.getSession(true);
        if(session!=null){
            session.setAttribute("username",name);// 将参数作为 session的内容传递进入
        }
         return "登陆成功";
    }
```

再次登陆的时候验证session

```
 @RequestMapping("/getSession")
    @ResponseBody
    public String getSession(HttpServletRequest request){
        // 如果不存在session，也不会再去创建session
        HttpSession session = request.getSession(false);

        //1、判断session 是否存在
        if(session==null){
            return "未登录";
        }

        //2、判断session中的 内容是否存在
        if(session.getAttribute("username")==null){
            return "未登录";
        }

        //3.如果前两层都通过，说明获取到了想要的session内容
        return "登陆成功!";
    }
```

（2）使用 @SessionAttribute 获取 Session中 的 参数

```
    @RequestMapping("/getSession")
    @ResponseBody
   public String getSession(@SessionAttribute(value = " username",required =
false) String username){
         if(username!=null){
             return "登录成功!";
         }
         return "登陆失败";
    }
```

总结

1、获取单个参数（多个参数）：在方法中写响应的参数即可实现

2、获取对象：在方法中直接写对象即可接收

3、获取JSON对象： @RequestBody 加到方法中的参数的前面

4、获取文件： 使用@RequestPart

5、获取Cookie/Session/Header: @CookieValue @SessionAttribute @RequestHeader