# 1. spring boot返回json数据

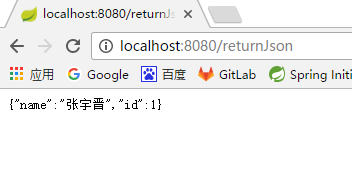
## 注意.Jackson对象和json做转换，一定要使方法空构造

## 1.编写controller

@RestController  
**public class** ReturnJsonController {  
 *//Spring boot 默认使用的json 解析框架是Jackjson* @RequestMapping(value = **"/returnJson"** ,method = RequestMethod.***GET***)  
 **public** User returnJson(){  
 User user = **new** User();  
 user.setId(1);  
 user.setName(**"张宇晋"**);  
 **return** user; *//直接返回就是一个json* }

## 2.浏览器测试

<http://localhost:8080/returnJson>



# 2. fastJson解析JSON数据

## 1.配置相关依赖

<**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 <**version**>1.2.15</**version**>  
</**dependency**>

## 2. fastjson 使用的时候需要让启动类继承WebMvcConfigurerAdapter

**public class** StudyvedioApplication **extends** WebMvcConfigurerAdapter{

## 3.重写WebMvcConfigurerAdapter的内部类configureMessageConverters

**public void** configureMessageConverters(List<HttpMessageConverter<?>> converters) {  
 **super**.configureMessageConverters(converters);  
 *//1需要先定义个convert转消息的对象* FastJsonHttpMessageConverter fastConverter = **new** FastJsonHttpMessageConverter();  
 *//2.添加fastjson的配置信息，比如是否要格式化返回的json数据* FastJsonConfig fastJsonConfig = **new** FastJsonConfig();  
 fastJsonConfig.setSerializerFeatures(  
 SerializerFeature.***PrettyFormat*** );  
 *//3.在convert中添加配置信息* fastConverter.setFastJsonConfig(fastJsonConfig);  
 *//4.将convert添加到converters中* converters.add(fastConverter);  
}

## 4.User 类中添加字段Date 日期，设置格式，用来验证是不是我们的fastjson起了作用

**int age**;  
String **name**;  
*//测试fastjson的时候用到  
//这个注释在在fastjson的包里面的 com.alibaba.fastjson.annotation.JSONField;*@JSONField(format = **"yyyyMMddhhmmss"**)  
Date **createTime**;

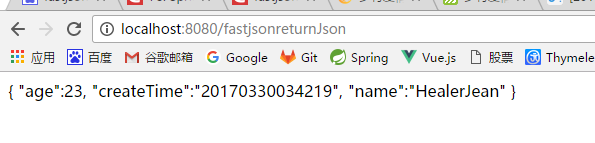
## 5.controller 中设置date的值

@RequestMapping(value = **"/ fastjsonreturnJson**

**"** ,method = RequestMethod.***GET***)  
**public** User returnJson(){  
 User user = **new** User();  
 user.setAge(23);  
 user.setName(**"HealerJean"**);  
 user.setCreateTime(**new** Date());  
 **return** user; *//直接返回就是一个json*}

## 6.测试成功<http://localhost:8080/fastjsonreturnJson>

### 说明 下面的日期格式变了



# 3. fastjson解析第二种的方法。启动类不需要继承，只要注入一个bean就行，其实和上面的第一种大致是一样的

@SpringBootApplication  
**public class** StudyvedioApplication{  
 @Bean  
 **public** HttpMessageConverters fastJsonHttpMessageConverter() {  
 *//1需要先定义个convert转消息的对象* FastJsonHttpMessageConverter fastConverter = **new** FastJsonHttpMessageConverter();  
 *//2.添加fastjson的配置信息，比如是否要格式化返回的json数据* FastJsonConfig fastJsonConfig = **new** FastJsonConfig();  
 fastJsonConfig.setSerializerFeatures(  
 SerializerFeature.***PrettyFormat*** );  
 *//3.在convert中添加配置信息* fastConverter.setFastJsonConfig(fastJsonConfig);  
 HttpMessageConverter<?> converter = fastConverter;  
 *//4.将convert添加到converters中* **return new** HttpMessageConverters(converter);  
 }

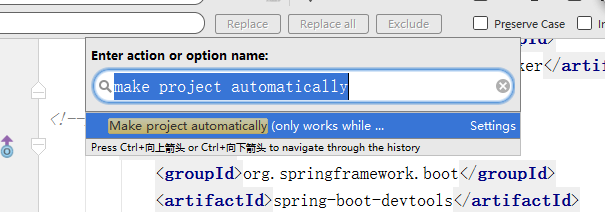
# 4.热部署

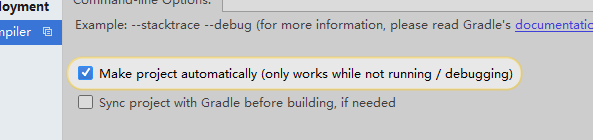
## 1.采用devtools 热部署

<http://jisonami.iteye.com/blog/2319112>

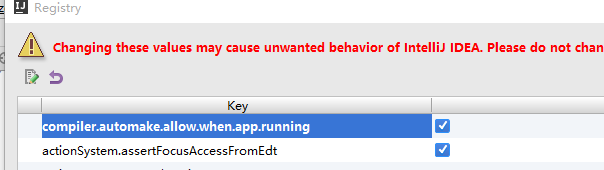
### 1.设置intellij idea软件

#### 1、CTRL + SHIFT + A --> 查找make project automatically --> 选中





#### 2、CTRL + SHIFT + A --> 查找Registry --> 找到并勾选compiler.automake.allow.when.app.running



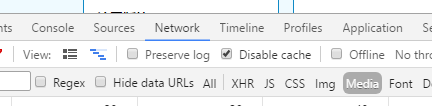
### 2.添加依赖

*<!--热部署 devtools-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-devtools</**artifactId**>  
 <**optional**>true</**optional**>  
 </**dependency**>

<**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 <**configuration**>  
 <**fork**>true</**fork**> *<!--热部署需要-->* </**configuration**>  
 </**plugin**>  
 </**plugins**>  
</**build**>

### 3. Chrome禁用缓存

#### 方法：F12（或Ctrl+Shift+J或Ctrl+Shift+I）--> NetWork --> Disable Cache(while DevTools is open



### 4.经测试，成功

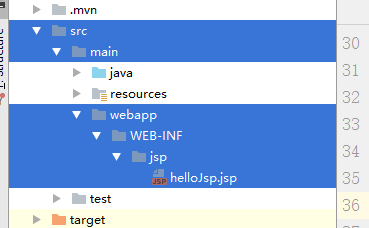
# 5.添加jsp支持

## 1. 添加依赖

*<!--jsp支持依赖-->* <**dependency**>  
 <**groupId**>org.apache.tomcat.embed</**groupId**>  
 <**artifactId**>tomcat-embed-jasper</**artifactId**>  
 </**dependency**>

## 2. 添加web资源

### 1.找到路径



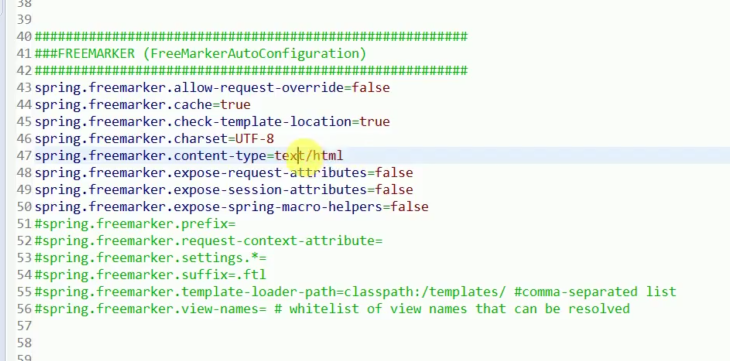
## 3.配置application.properties 添加前缀和后缀

*# 页面默认前缀目录***spring.mvc.view.prefix**=**/WEB-INF/jsp/***# 响应页面默认后缀***spring.mvc.view.suffix**=**.jsp***# 自定义属性，可以在Controller中读取***application.hello**=**Hello GOD**

## 4.编写controller

@Controller  
  
**public class** HelloController {  
 *// 从 application.properties 中读取配置，如取不到默认值为Hello* @Value(**"${application.hello:Hello}"**)  
 **private** String **hello**;  
 @RequestMapping(**"/helloJsp"**)  
 **public** String helloJsp(Map<String, Object> map) {  
 System.***out***.println(**"HelloController.helloJsp().hello="** + **hello**);  
 map.put(**"hello"**, **hello**);  
 **return "helloJsp"**;  
  
 }  
  
}

# 6. freemarker模板（ftl文件格式，可以使用el表达式${name}） 可以和thymeleaf并存使用，



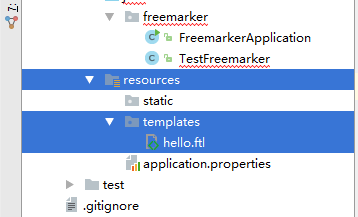
## 1.添加依赖

*<!--freemarker依赖-->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-freemarker</**artifactId**>  
</**dependency**>

## 2. 编写application.properties 添加freemarker 信息

**spring.freemarker.cache**=**false  
spring.freemarker.charset**=**UTF-8  
spring.freemarker.check-template-location**=**true  
spring.freemarker.content-type**=**text/html  
spring.freemarker.expose-request-attributes**=**true  
spring.freemarker.expose-session-attributes**=**true  
spring.freemarker.request-context-attribute**=**request**

## 3.模板资源位置



<!DOCTYPE **html**>  
<**html class="no-js" lang="zh-CN" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="utf-8"**/>  
 <**title**></**title**>  
</**head**>  
<**body**>  
freemarker 模板页面 **${user}**

</**body**>  
</**html**>

### 4.编写controller

@Controller  
**public class** TestFreemarker {  
 @RequestMapping(value = **"/freemarker"** ,method = RequestMethod.***GET***)  
 **public** String returnJson(Model model){  
 model.addAttribute(**"user"**,**"HealerJean"**);  
 **return "freemarker"**; *//直接返回就是一个json* }  
}

## 5.测试成功 <http://localhost:8080/freemarker>

