从零开始学Spring Boot

作者: 林祥纤 http://412887952-qq-com.iteye.com

从零开始学Spring Boot 通过实战让你快速入门Spring Boot。

本书由ITeye提供的电子书DIY功能自动生成于 2016-05-21

目 录

1. 从零开始学Spring Boot

1.1 （0）前言【从零开始学Spring Boot】

发表时间: 2016-04-15 关键字: 从零开始学Spring Boot, Spring Boot

在此对整体技术简单说明下。

开发工具和开发环境：

Win7 64位操作

Eclipse-jee-mars-2-win32-x86_64

Jdk 1.8

Maven管理项目

如果要学习Spring Boot那么至少是需要一个开发工具的，至于是什么IDE这个就随意了，在此基础上，可以安装一些插件，有利于开发。插件列表(非必须)：

Spring Tool Suite（spring tool suite 是一个基于eclipseIDE开发环境中的用于开发spring应用程序的工具。提供了开箱即用的环境用于实现，调试和部署你的spring应用，包括为关键的的服务器和云计算，Git,Maven,AspectJ,和最新的Eclipse版本提供整合支持。）

Propedit （在做国际化编辑一些简体中文、繁体中文等Unicode资源文件时，总是需要使用native2ascii编码，那么在Eclipse安装此插件就可以轻松搞定了）

好了，如果一切准备就绪，那么就可以开始我们的编程之旅了。

1.2 （0）资料官网【从零开始学Spring Boot】

发表时间: 2016-04-15 关键字: 从零开始学Spring Boot, Spring Boot

Spring Boot官网：http://projects.spring.io/spring-boot/

Eclipse官网：http://www.eclipse.org/

Maven官网：http://maven.apache.org/

JDK下载地址：http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html

Spring Tool Suite地址：http://spring.io/tools/sts/all

1.3 （1）spring boot起步之Hello World【从零开始学Spring Boot】

发表时间: 2016-04-15 关键字: 从零开始学Spring Boot, Spring Boot, spring boot起步之Hello World

1.1 介绍

自从structs2出现上次的漏洞以后，对spring的关注度开始越来越浓。

以前spring开发需要配置一大堆的xml,后台spring加入了annotaion，使得xml配置简化了很多，当然还是有些配置需要使用xml,比如申明component scan等。

前段时间发现了spring开了一个新的model spring boot,主要思想是降低spring的入门，使得新手可以以最快的速度让程序在spring框架下跑起来。

那么如何写Hello world呢？

Hello之步骤:

(1)新建一个Maven Java 工程

(2)在pom.xml文件中添加Spring Boot Maven依赖

(3)编写启动类

(4)运行程序

1.2 Hello之New

这个步骤很简单，相比大家都会，小编在此为了文档的完整性，稍作简单说明：

首先使用IDE（Eclipse,MyEclipse）工具新建一个Maven工程，可以是Maven Java Project,也可以是Maven Web Project,随便取一个工程名称。我使用的是MyEclipse，工程名是spring-boot-hello1。

1.3 Hello之Maven

第二步，在pom.xml中引入spring-boot-start-parent,spring官方的解释叫什么stater poms,它可以提供dependency management,也就是说依赖管理，引入以后在申明其它dependency的时候就不需要version了，后面可以看到。

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.3.3.RELEASE</version>
 </parent>
```

1.4 Hello之maven web

第三步，因为我们开发的是web工程，所以需要在pom.xml中引入spring-boot-starter-web,spring官方解释说spring-boot-start-web包含了spring webmvc和tomcat等web开发的特性。

```
<dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>
```

1.5 Hello之Maven Run Application

如果我们要直接Main启动spring，那么以下plugin必须要添加，否则是无法启动的。如果使用maven 的spring-boot:run的话是不需要此配置的。（我在测试的时候，如果不配置下面的plugin也是直接在Main中运行的。）

```
<build>
```

```
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin </artifactId>
        </plugin>
    </plugins>
</build>
```

1.6 Hello之coding

第四步，真正的程序开始啦，我们需要一个启动类，然后在启动类申明让spring boot自动给我们配置spring需要的配置，比如：@SpringBootApplication,为了可以尽快让程序跑起来，我们简单写一个通过浏览器访问hello world字样的例子：

```
@RestController
@SpringBootApplication
publicclassApp {

    @RequestMapping("/")
    public String hello(){
        return"Hello world!";
    }
```

```
    publicstaticvoid main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}
```

其中@SpringBootApplication申明让spring boot自动给程序进行必要的配置，等价于以默认属性使用@Configuration，@EnableAutoConfiguration和@ComponentScan

@RestController返回json字符串的数据，直接可以编写RESTFul的接口；

1.7 Hello之Run

第五步，就是运行我们的Application了，我们先介绍第一种运行方式。第一种方式特别简单：右键Run As -> Java Application。之后打开浏览器输入地址：http://127.0.0.1:8080/ 就可以看到Hello world!了。第二种方式右键project – Run as – Maven build – 在Goals里输入spring-boot:run ,然后Apply,最后点击Run。

1.8 Hello之Error

顺利的情况下当然是皆大欢喜了，但是程序吧往往会给你开个小玩笑。那么我们要注意什么呢？主要是jdk的版本之类的，请看官方说明：

Name	Servlet Version	Java Version
Tomcat 8	3.1	Java 7+
Tomcat 7	3.0	Java 6+
Jetty 9	3.1	Java 7+
Jetty 8	3.0	Java 6+
Undertow 1.1	3.1	Java 7+

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
...

（15）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2292362

（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388
.....
（35）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/blog

附件下载:

- spring-boot-hello1.zip (12.7 KB)
- dl.iteye.com/topics/download/0a9d932b-f4cc-3002-8e59-e334aab431f3

1.4 （2）Spring Boot返回json数据【从零开始学Spring Boot】

发表时间: 2016-04-15 关键字: 从零开始学Spring Boot, Spring Boot, Spring Boot返回json数据

在做如下操作之前，我们对之前的Hello进行简单的修改，我们新建一个包com.kfit.test.web 然后新建一个类HelloControooler, 然后修改App.java类，主要是的这个类就是一个单纯的启动类。

主要代码如下：

App.java

```
package com.kfit;


import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

/**

 * Hello world!

 *

 */

//其中@SpringBootApplication申明让spring boot自动给程序进行必要的配置，等价于以默认属性使用@Configuration，@EnableAutoConfiguration和@ComponentScan

@SpringBootApplication

public class App {

        public static void main(String[] args) {

          SpringApplication.run(App.class, args);

    }

}
```

com.kfit.test.web.HelloController ：

```
package com.kfit.test.web;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController// 标记为：restful
publicclass HelloController {

  @RequestMapping("/")
  public String hello(){
    return"Hello world!";
  }
}
```

运行代码和之前是一样的效果的。

我们在编写接口的时候，时常会有需求返回json数据，那么在spring boot应该怎么操作呢？主要是在class中加入注解@RestController,。

返回JSON之步骤：

(1)编写一个实体类Demo

(2)编写DemoController；

(3)在DemoController加上@RestController和@RequestMapping注解；

(4)测试

具体代码如下：

com.kfit.test.bean.Demo：

```
package com.kfit.test.bean;

/**
* 测试实体类.
* @author Administrator
*
*/
publicclass Demo {
  privatelongid;//主键.
  private String name;//测试名称.
  publiclong getId() {
    returnid;
  }
  publicvoid setId(longid) {
    this.id = id;
  }
  public String getName() {
    returnname;
  }
  publicvoid setName(String name) {
    this.name = name;
  }
}
```

com.kfit.test.web.DemoController：

```
package com.kfit.test.web;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.kfit.test.bean.Demo;

/**
 * 测试.
 * @author Administrator
 *
 */
@RestController
@RequestMapping("/demo")
publicclass DemoController {

    /**
     * 返回demo数据:
     * 请求地址：http://127.0.0.1:8080/demo/getDemo
     * @return
     */
    @RequestMapping("/getDemo")
    public Demo getDemo(){
```

```
    Demo demo = new Demo();
    demo.setId(1);
    demo.setName("Angel");
    returndemo;
  }

}
```

那么在浏览器访问地址：http://127.0.0.1:8080/demo/getDemo 返回如下数据：

```
{
    id: 1,
    name: "Angel"
}
```

是不是很神奇呢，其实Spring Boot也是引用了JSON解析包Jackson，那么自然我们就可以在Demo对象上使用Jackson提供的json属性的注解，对时间进行格式化，对一些字段进行忽略等等。

【Spring Boot 系列博客】

0）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
（15）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2292362

16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

.....

（35）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/

附件下载:

- spring-boot-hello1.zip (17.1 KB)
- dl.iteye.com/topics/download/876342ca-ba8d-3303-85b9-d512b0e61595

1.5 （3）Spring Boot热部署【从零开始学Spring Boot】

发表时间: 2016-04-15 关键字: Spring Boot热部署, 从零开始学Spring Boot, Spring Boot

在编写代码的时候，你会发现我们只是简单把打印信息改变了下，就需要重新部署，如果是这样的编码方式，那么我们估计一天下来之后就真的是打几个Hello World之后就下班了。那么如何解决热部署的问题呢？那就是springloaded，加入如下配置：

```
<plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin </artifactId>
                <dependencies>
                  <!--springloaded  hot deploy -->
                  <dependency>
                     <groupId>org.springframework</groupId>
                     <artifactId>springloaded</artifactId>
                     <version>1.2.4.RELEASE</version>
                  </dependency>
                </dependencies>
                <executions>
                  <execution>
                     <goals>
                        <goal>repackage</goal>
                     </goals>
                     <configuration>
                        <classifier>exec</classifier>
                     </configuration>
                  </execution>
                 </executions>
</plugin>
```

如果是使用spring-boot:run的话，那么到此配置结束，现在你就可以体验coding...coding的爽了。

如果使用的run as – java application的话，那么还需要做一些处理哦：

把spring-loader-1.2.4.RELEASE.jar下载下来，放到项目的lib目录中，然后把IDEA的run参数里VM参数设置为：

-javaagent:.\lib\springloaded-1.2.4.RELEASE.jar -noverify

然后启动就可以了，这样在run as的时候，也能进行热部署了。

当然并不是所有的代码都支持热部署了，这个我自己也不是很明确，那些代码修改了可以直接不用重启查看。

Spring Boot 系列博客】

0）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

（15）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2292362

16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

.....

（35）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/

1.6 （4）Spring Boot使用别的json解析框架【从零开始学Spring Boot】

发表时间: 2016-04-16 关键字: 使用别的json解析框架, 从零开始学Spring Boot, Spring Boot

个人使用比较习惯的json框架是fastjson,所以spring boot默认的json使用起来就很陌生了，所以很自然我就想我能不能使用fastjson进行json解析呢。这里有一种方案：

就是在返回数据的时候，不直接返回对象，而是返回String，具体操作如下：

引入fastjson依赖库：

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.7</version>
</dependency>
```

代码修改为：

```
//地址：http://127.0.0.1:8080/demo/getFastJson

@RequestMapping("/getFastJson")

public String getFastJson(){

    Demo demo = new Demo();

    demo.setId(2);

    demo.setName("Angel2");
```

```
    return JSONObject.toJSONString(demo);

}
```

虽然看似多了一行代码，但是使用fastjson对数据的返回的可控性就很强了。当然可能会有更好的解决方案了，在这里这是抛装引玉。

1.7 （5）全局异常捕捉【从零开始学Spring Boot】

发表时间: 2016-04-16 关键字: 从零开始学Spring Boot, 全局异常捕捉, Spring Boot

在一个项目中的异常我们我们都会统一进行处理的，那么如何进行统一进行处理呢？

新建一个类GlobalDefaultExceptionHandler，

在class注解上@ControllerAdvice,

在方法上注解上@ExceptionHandler(value = Exception.class)，具体代码如下：

com.kfit.base.exception.GlobalDefaultExceptionHandler

```
package com.kfit.base.exception;

import javax.servlet.http.HttpServletRequest;

import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice
public class GlobalDefaultExceptionHandler {

    @ExceptionHandler(value = Exception.class)
    public void defaultErrorHandler(HttpServletRequest req, Exception e)  {
//      // If the exception is annotated with @ResponseStatus rethrow it and let
//      // the framework handle it - like the OrderNotFoundException example
//      // at the start of this post.
//      // AnnotationUtils is a Spring Framework utility class.
//      if (AnnotationUtils.findAnnotation(e.getClass(), ResponseStatus.class) != null)
//          throw e;
//
//      // Otherwise setup and send the user to a default error-view.
//      ModelAndView mav = new ModelAndView();
//      mav.addObject("exception", e);
```

```
//      mav.addObject("url", req.getRequestURL());
//      mav.setViewName(DEFAULT_ERROR_VIEW);
//      return mav;

     //打印异常信息：
      e.printStackTrace();
      System.out.println("GlobalDefaultExceptionHandler.defaultErrorHandler()");

      /*
       * 返回json数据或者String数据：
       * 那么需要在方法上加上注解：@ResponseBody
       * 添加return即可。
       */

      /*
       * 返回视图：
       * 定义一个ModelAndView即可，
       * 然后return;
       * 定义视图文件(比如：error.html,error.ftl,error.jsp);
       *
       */
  }

}
```

com.kfit.test.web.DemoController 加入方法：

```
@RequestMapping("/zeroException")
   public int zeroException(){
     return 100/0;

   }
```

访问：http://127.0.0.1:8080/zeroException 这个方法肯定是抛出异常的,那么在控制台就可以看到我们全局捕捉的异常信息了

Spring Boot 系列博客】

0）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

（15）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2292362

16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

.....

（35）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/

==========

您的打赏是我最大的动力，打开微信或者支付宝扫描二维码向我打赏吧：

（支付宝支付）

（微信支付）

附件下载:

- spring-boot-hello1.zip (410.8 KB)
- dl.iteye.com/topics/download/a6081ee4-89a4-34f3-a57d-10e0c4149b78

1.8 （6）Spring Boot datasource - mysql【从零开始学Spring Boot】

发表时间: 2016-04-16 关键字: 从零开始学Spring Boot, datasource - mysql, Spring Boot

在任何一个平台都逃离不了数据库的操作，那么在spring boot中怎么接入数据库呢？

很简单，我们需要在application.properties进行配置一下，application.properties路径是src/main/resources下，对于application.properties更多的介绍请自行百度进行查找相关资料进行查看，在此不进行过多的介绍，以下只是mysql的配置文件。

大体步骤：

(1)在application.properties中加入datasouce的配置

(2)在pom.xml加入mysql的依赖。

(3)获取DataSouce的Connection进行测试。

src/main/resouces/application.properties：

```
#########################################################
###datasource
#########################################################
spring.datasource.url = jdbc:mysql://localhost:3306/test
spring.datasource.username = root
spring.datasource.password = root
spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.datasource.max-active=20
spring.datasource.max-idle=8
spring.datasource.min-idle=8
spring.datasource.initial-size=10
```

pom.xml配置：

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

到此相关配置就ok了，那么就可以在项目中进行测试了，我们可以新建一个class Demo进行测试，实体类创建完毕之后，我们可能需要手动进行编写建表语句，这时候我们可能就会想起Hibernate的好处了。那么怎么在spring boot使用Hibernate好的特性呢？So easy,具体怎么操作，请看下篇文章之JPA – Hibernate。

1.9（7）JPA - Hibernate【从零开始学Spring Boot】

发表时间: 2016-04-16 关键字: 从零开始学Spring Boot, JPA - Hibernate, Spring Boot

在说具体如何在spring boot 使用Hibernate前，先抛装引玉些知识点？什么是JPA呢？

JPA全称Java Persistence API.JPA通过JDK 5.0注解或XML描述对象－关系表的映射关系，并将运行期的实体对象持久化到数据库中。

http://baike.baidu.com/link?url=LdqIXvzTr0RDjY2yoRdpogDdzaZ_L-DrIOpLLzK1z38quk6nf2ACoXEf3pWKTElHACS7vTawPTmoFv_QftgT_q

接下里就说本文章重点了，那么怎么操作呢？只需要如下配置就可以了？

pom.xml配置：

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

application.properties配置：

```
#################################################################
### Java Persistence Api
#################################################################
# Specify the DBMS
spring.jpa.database = MYSQL
# Show or not log for each sql query
spring.jpa.show-sql = true
# Hibernate ddl auto (create, create-drop, update)
spring.jpa.hibernate.ddl-auto = update
# Naming strategy
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy
# stripped before adding them to the entity manager)
```

```
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

那么就可以使用Hibernate带来的好处了，在实体类注解@Entity就会自动进行表的DDL操作了。

我们在com.kfit.test.bean.Demo 中加入注解：@Entity

```
@Entity//加入这个注解，Demo就会进行持久化了，在这里没有对@Table进行配置，请自行配置。
publicclass Demo {
   @Id @GeneratedValue
   private long id;//主键.

   private String name;//测试名称.

//其它代码省略.
```

这时候运行就会在数据库看到demo表了。

Spring Boot 系列博客】

0）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

（15）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2292362

16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

.....

（35）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/

附件下载:

- spring-boot-hello1.zip (412.6 KB)
- dl.iteye.com/topics/download/8c939bcf-048e-33bf-b99a-01b8c48e5fad

1.10 （8）使用JPA保存数据【从零开始学Spring Boot】

发表时间: 2016-04-16 关键字: 从零开始学Spring Boot, 使用JPA保存数据, Spring Boot

在看这一篇文档的话，需要先配置好JPA – Hibernate。

总体步骤：

(1) 创建实体类Demo,如果已经存在，可以忽略。

(2) 创建jpa repository类操作持久化。

(3) 创建service类。

(4) 创建restful请求类。

(5) 测试

代码如下：

com.kfit.test.bean.Demo ：

```
package com.kfit.test.bean;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

/**
 * 测试实体类.
 * @author Administrator
 *
 */
```

```
@Entity //加入这个注解，Demo就会进行持久化了，在这里没有对@Table进行配置，请自行配置。
public class Demo {
    @Id @GeneratedValue
    private long id;//主键.
    private String name;//测试名称.
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

com.kfit.test.dao.DemoRepository（这是一个接口，没有具体的实现，这就是JPA）：

```
package com.kfit.test.dao;
import org.springframework.data.repository.CrudRepository;
import com.kfit.test.bean.Demo;
```

```
/*
* 在CrudRepository自带常用的crud方法.
* 这样一个基本dao就写完了.
*/
public interface DemoRepository extends CrudRepository<Demo, Long>{

}
```

到这里保存数据的方法就写完了。CrudRepository类把一些常用的方法都已经进行定义和实现了。那么你现在就可以在别的类引入调用了。

另外就是在Spring Data的核心接口里面Repository是最基本的接口了，spring提供了很多实现了该接口的基本接口，如:CrudRepository，PagingAndSortingRepository，SimpleJpaRepository，QueryDslJpaRepository等大量查询接口

com.kfit.test.service.DemoService :

```
package com.kfit.test.service;

import javax.annotation.Resource;
import org.springframework.stereotype.Service;
import com.kfit.test.bean.Demo;
import com.kfit.test.dao.DemoRepository;

/**
* 提供Demo服务类.
* @author Administrator
*
*/
@Service
publicclass DemoService {
```

```
    @Resource
    private DemoRepository demoRepository;

    @Transactional
    public void save(Demo demo){
        demoRepository.save(demo);
    }
}
```

com.kfit.test.web.Demo2Controller(这里为了代码干净，新建了一个测试类)：

```
package com.kfit.test.web;

import javax.annotation.Resource;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.kfit.test.bean.Demo;
import com.kfit.test.service.DemoService;

@RestController
@RequestMapping("/demo2")
public class Demo2Controller {
    @Resource
    private DemoService demoService;

    /**
     * 测试保存数据方法.
     * @return
     */
    @RequestMapping("/save")
    public String save(){
        Demo d = new Demo();
        d.setName("Angel");
        demoService.save(d);//保存数据.
        return "ok.Demo2Controller.save";
    }
```

```
}
```

之后就可以进行测试了，访问地址：http://127.0.0.1:8080/demo2/save

查看数据库，就可以看到已经添加的数据了。

==========

您的打赏是我最大的动力，打开微信或者支付宝扫描二维码向我打赏吧：

（支付宝支付）

（微信支付）

附件下载:

- spring-boot-hello1.zip (416.8 KB)
- dl.iteye.com/topics/download/981be179-eb97-38f8-a049-21dcae9c26e7

1.11 （9）使用JdbcTemplate【从零开始学Spring Boot】

发表时间: 2016-04-17 关键字: 从零开始学Spring Boot, 使用JdbcTemplate, Spring Boot

整体步骤：

(1) 在pom.xml加入jdbcTemplate的依赖；

(2) 编写DemoDao类，声明为：@Repository，引入JdbcTemplate

(3) 编写DemoService类，引入DemoDao进行使用

(4) 编写Demo2Controller进行简单测试。

具体操作流程如下：

使用JdbcTemplate类需要加入（如果在JPA已经加入的话，这个步骤就可以忽略了）

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

那么只需要在需要使用的类中加入：

```
@Resource
private JdbcTemplate jdbcTemplate;
```

这样就可以使用jdbcTemplate进行数据库的操作了。

比如：

```
String sql = "insert into Demo(name,age) values(?,?)";
jdbcTemplate.update(sql, new Object[]{demo.getName(),demo.getAge()});
```

实战代码：

编写com.kfit.test.dao.DemoDao 数据库操作类：

```
package com.kfit.test.dao;

import javax.annotation.Resource;

import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Repository;

import com.kfit.test.bean.Demo;

/**
 * 使用JdbcTemplate操作数据库.
 * @author Administrator
 *
 */
@Repository
publicclass DemoDao {

    @Resource
    private JdbcTemplate jdbcTemplate;

    /**
     * 通过id获取demo对象.
     * @param id
     * @return
     */
    public Demo getById(long id){
        String sql = "select *from Demo where id=?";
        RowMapper<Demo> rowMapper = new BeanPropertyRowMapper<Demo>(Demo.class);
        returnjdbcTemplate.queryForObject(sql, rowMapper,id);
    }

}
```

com.kfit.test.service.DemoService :

```
package com.kfit.test.service;

import javax.annotation.Resource;
import org.springframework.stereotype.Service;
import com.kfit.test.bean.Demo;
import com.kfit.test.dao.DemoDao;
import com.kfit.test.dao.DemoRepository;

/**
 * 提供Demo服务类.
 * @author Administrator
 *
 */
@Service
public class DemoService {

    @Resource
    private DemoRepository demoRepository;

    @Resource
    private DemoDao demoDao;

    public void save(Demo demo){
        demoRepository.save(demo);
    }

    public Demo getById(long id){
        //demoRepository.findOne(id);//在demoRepository可以直接使用findOne进行获取.
        return demoDao.getById(id);
    }
}
```

com.kfit.test.web.Demo2Controller :

```
package com.kfit.test.web;
```

```
import javax.annotation.Resource;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.kfit.test.bean.Demo;
import com.kfit.test.service.DemoService;

@RestController
@RequestMapping("/demo2")
publicclass Demo2Controller {
    @Resource
    private DemoService demoService;



    /**
     * 测试保存数据方法.
     * @return
     */
    @RequestMapping("/save")
    public String save(){
        Demo d = new Demo();
        d.setName("Angel");
        demoService.save(d);//保存数据.
        return"ok.Demo2Controller.save";
    }

    //地址：http://127.0.0.1:8080/demo2/getById?id=1
    @RequestMapping("/getById")
    public Demo getById(longid){
        returndemoService.getById(id);
    }

}
```

剩下的就是启动进行测试了，访问地址：http://127.0.0.1:8080/demo2/getById?id=1

那么在浏览器中就可以看到：

{

```
    id: 1,
    name: "Angel"
}
```

当前前提是你的数据库中有id=1的数据了，不然会报错的：

org.springframework.dao.EmptyResultDataAccessException

==========

（微信支付）

附件下载:

- spring-boot-hello1.zip (418.8 KB)
- dl.iteye.com/topics/download/09093fd1-a6fe-3a16-91a3-47b21d1b2dff

1.12 （10）Spring Boot修改端口号【从零开始学Spring Boot】

发表时间: 2016-04-17 关键字: 从零开始学Spring Boot, Spring Boot, Spring Boot修改端口号

Spring boot 默认端口是8080，如果想要进行更改的话，只需要修改applicatoin.properties文件，在配置文件中加入：

```
server.port=9090
```

常用配置：

```
#--------------------------------------------------------
###EMBEDDED SERVER CONFIGURATION (ServerProperties)
#--------------------------------------------------------
#server.port=8080
#server.address= # bind to a specific NIC
#server.session-timeout= # session timeout in seconds
#the context path, defaults to '/'
#server.context-path=/spring-boot
#server.servlet-path= # the servlet path, defaults to '/'
#server.tomcat.access-log-pattern= # log pattern of the access log
#server.tomcat.access-log-enabled=false # is access logging enabled
#server.tomcat.protocol-header=x-forwarded-proto # ssl forward headers
#server.tomcat.remote-ip-header=x-forwarded-for
#server.tomcat.basedir=/tmp # base dir (usually not needed, defaults to tmp)
#server.tomcat.background-processor-delay=30; # in seconds
#server.tomcat.max-threads = 0 # number of threads in protocol handler
#server.tomcat.uri-encoding = UTF-8 # character encoding to use for URL decoding
```

1.13 （11）Spring Boot配置ContextPath【从零开始学Spring Boot】

发表时间: 2016-04-17 关键字: 从零开始学Spring Boot, 配置ContextPath, Spring Boot配置ContextPath

Spring boot默认是/ ，这样直接通过http://ip:port/就可以访问到index页面，如果要修改为http://ip:port/path/ 访问的话，那么需要在Application.properties文件中加入server.context-path = /你的path,比如：spring-boot,那么访问地址就是http://ip:port/spring-boot 路径。

```
server.context-path=/spring-boot
```

常用配置：

```
#--------------------------------------------------------
###EMBEDDED SERVER CONFIGURATION (ServerProperties)
#--------------------------------------------------------
#server.port=8080
#server.address= # bind to a specific NIC
#server.session-timeout= # session timeout in seconds
#the context path, defaults to '/'
#server.context-path=/spring-boot
#server.servlet-path= # the servlet path, defaults to '/'
#server.tomcat.access-log-pattern= # log pattern of the access log
#server.tomcat.access-log-enabled=false # is access logging enabled
#server.tomcat.protocol-header=x-forwarded-proto # ssl forward headers
#server.tomcat.remote-ip-header=x-forwarded-for
#server.tomcat.basedir=/tmp # base dir (usually not needed, defaults to tmp)
#server.tomcat.background-processor-delay=30; # in seconds
#server.tomcat.max-threads = 0 # number of threads in protocol handler
#server.tomcat.uri-encoding = UTF-8 # character encoding to use for URL decoding
```

1.14 （12）Spring Boot改变JDK编译版本【从零开始学Spring Boot】

发表时间: 2016-04-17 关键字: 从零开始学Spring Boot, 改变JDK编译版本, spring boot, Spring Boot改变JDK编译版本

Spring Boot在编译的时候，是有默认JDK版本的，如果我们期望使用我们要的JDK版本的话，那么要怎么配置呢？

这个只需要修改pom.xml文件的<build> -- <plugins>加入一个plugin即可。

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
  </configuration>
</plugin>
```

添加了plugin之后，需要右键Maven à Update Projects,这时候你可以看到工程根目录下的JRE System Library 版本更改了。

1.15 （13）处理静态资源(默认资源映射)【从零开始学Spring Boot】

发表时间: 2016-04-18 关键字: 从零开始学Spring Boot, 处理静态资源, Spring Boot, 默认资源映射

Spring Boot 默认为我们提供了静态资源处理，使用 WebMvcAutoConfiguration 中的配置各种属性。

建议大家使用Spring Boot的默认配置方式，如果需要特殊处理的再通过配置进行修改。

如果想要自己完全控制WebMVC，就需要在@Configuration注解的配置类上增加@EnableWebMvc（@SpringBootApplication 注解的程序入口类已经包含@Configuration），增加该注解以后WebMvcAutoConfiguration中配置就不会生效，你需要自己来配置需要的每一项。这种情况下的配置还是要多看一下WebMvcAutoConfiguration类。

我们既然是快速使用Spring Boot，并不想过多的自己再重新配置。本文还是主要针对Spring Boot的默认处理方式，部分配置在application 配置文件中（.properties 或 .yml）

默认资源映射

我们在启动应用的时候，可以在控制台中看到如下信息：

```
2016-01-08 09:29:30.362  INFO 24932 ---[           main]o.s.w.s.handler.Simp
2016-01-08 09:29:30.362  INFO 24932 ---[           main]o.s.w.s.handler.Simp
2016-01-08 09:29:30.437  INFO 24932 ---[           main]o.s.w.s.handler.Simp
```

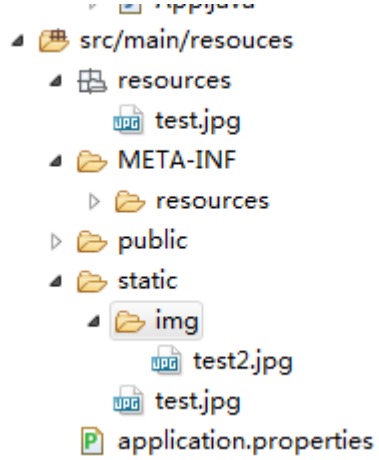
其中默认配置的 /** 映射到 /static（或/public、/resources、/META-INF/resources）

其中默认配置的 /webjars/** 映射到 classpath:/META-INF/resources/webjars/

PS：上面的 static、public、resources 等目录都在 classpath: 下面（如 src/main/resources/static）。

如果我按如下结构存放相同名称的图片，那么Spring Boot 读取图片的优先级是怎样的呢？

如下图：



当我们访问地址 http://localhost:8080/test.jpg 的时候，显示哪张图片？这里可以直接告诉大家，优先级顺序为：META/resources > resources > static > public（已进行测试）

如果我们想访问test2.jpg，请求地址 http://localhost:8080/img/test2.jpg

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496
（1）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500
（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
...
（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376
（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

1.16 （13）处理静态资源（自定义资源映射）【从零开始学Spring Boot】

发表时间: 2016-04-18 关键字: 从零开始学Spring Boot, spring boot, 自定义资源映射, 处理静态资源

上面我们介绍了Spring Boot 的默认资源映射，一般够用了，那我们如何自定义目录？

这些资源都是打包在jar包中的，然后实际应用中，我们还有很多资源是在管理系统中动态维护的，并不可能在程序包中，对于这种随意指定目录的资源，如何访问？

自定义目录

以增加 /myres/* 映射到 classpath:/myres/* 为例的代码处理为：

实现类继承 WebMvcConfigurerAdapter 并重写方法 addResourceHandlers （对于

```
package org.springboot.sample.config;

import org.springboot.sample.interceptor.MyInterceptor1;
import org.springboot.sample.interceptor.MyInterceptor2;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@Configuration
public class MyWebAppConfigurer
    extends WebMvcConfigurerAdapter {

  @Override
  public void addResourceHandlers(ResourceHandlerRegistry registry) {
    registry.addResourceHandler("/myres/**").addResourceLocations("classpath:/myres/");
    super.addResourceHandlers(registry);
  }

}
```

访问myres 文件夹中的test.jpg 图片的地址为 http://localhost:8080/myres/test.jpg

这样使用代码的方式自定义目录映射，并不影响Spring Boot的默认映射，可以同时使用。

如果我们将/myres/* 修改为 /* 与默认的相同时，则会覆盖系统的配置，可以多次使用 addResourceLocations 添加目录，优先级先添加的高于后添加的。

其中 addResourceLocations 的参数是动参，可以这样写 addResourceLocations(“classpath:/img1/”，“classpath:/img2/”，“classpath:/img3/”);

使用外部目录

如果我们要指定一个绝对路径的文件夹（如 D:/data/api_files ），则只需要使用 addResourceLocations 指定即可。

// 可以直接使用addResourceLocations 指定磁盘绝对路径，同样可以配置多个位置，注意路径写法需要加上 file:

```
registry.addResourceHandler("/api_files/**").addResourceLocations("file:D:/data/api_files");
```

附件下载:

- spring-boot-hello1.zip (1.4 MB)
- dl.iteye.com/topics/download/c929226e-bb99-3eac-8655-bc0abb0c5936

1.17 （14）Spring Boot定时任务的使用【从零开始学Spring Boot】

发表时间: 2016-04-18 关键字: 从零开始学Spring Boot, Spring Boot定时任务的使用

本文介绍在 Spring Boot 中如何使用定时任务，使用非常简单，就不做过多说明了。

com.kfit.base.scheduling.SchedulingConfig:

```
package com.kfit.base.scheduling;

import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.scheduling.annotation.Scheduled;

/**
 * 定时任务
 * @author Administrator
 *
 */
@Configuration
@EnableScheduling
publicclass SchedulingConfig {

    @Scheduled(cron = "0/20 * * * * ?") // 每20秒执行一次
    publicvoid scheduler() {
        System.out.println(">>>>>>>>>> SchedulingConfig.scheduler()");
    }

}
```

1.18 （ 15 ）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】

发表时间: 2016-04-19 关键字: 从零开始学Spring Boot, Spring Boot使用Druid和监控配置

Spring Boot 系列博客】

（ 0 ）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（ 1 ）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（ 2 ）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

...

（ 16 ）Spring Boot使用Druid（ 编程注入 ）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（ 17 ）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

Spring Boot默认的数据源是：org.apache.tomcat.jdbc.pool.DataSource

整体步骤：

(1) —— Druid简单介绍，具体看官网；

(2) —— 在pom.xml配置druid依赖包；

(3) —— 配置application.properties加入数据库源类型等参数；

(4) —— 编写druid servlet和filter提供监控页面访问；

(5) —— 输入地址进行测试；

Druid是Java语言中最好的数据库连接池，并且能够提供强大的监控和扩展功能。

业界把 Druid 和 HikariCP 做对比后，虽说 HikariCP 的性能比 Druid 高，但是因为 Druid 包括很多维度的统计和分析功能，所以这也是大家都选择使用它的原因。

下面来说明如何在 Spring Boot 中配置使用Druid

(1)添加Maven依赖 (或jar包)

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.0.18</version>
 </dependency>
```

(2)、配置数据源相关信息

```
# 数据库访问配置
# 主数据源，默认的
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/test
spring.datasource.username=root
spring.datasource.password=123456

# 下面为连接池的补充设置，应用到上面所有数据源中
# 初始化大小，最小，最大
spring.datasource.initialSize=5
spring.datasource.minIdle=5
spring.datasource.maxActive=20
# 配置获取连接等待超时的时间
```

```
spring.datasource.maxWait=60000
# 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒
spring.datasource.timeBetweenEvictionRunsMillis=60000
# 配置一个连接在池中最小生存的时间，单位是毫秒
spring.datasource.minEvictableIdleTimeMillis=300000
spring.datasource.validationQuery=SELECT 1 FROM DUAL
spring.datasource.testWhileIdle=true
spring.datasource.testOnBorrow=false
spring.datasource.testOnReturn=false
# 打开PSCache，并且指定每个连接上PSCache的大小
spring.datasource.poolPreparedStatements=true
spring.datasource.maxPoolPreparedStatementPerConnectionSize=20
# 配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙
spring.datasource.filters=stat,wall,log4j
# 通过connectProperties属性来打开mergeSql功能；慢SQL记录
spring.datasource.connectionProperties=druid.stat.mergeSql=true;druid.stat.slowSqlMillis=500
# 合并多个DruidDataSource的监控数据
```

```
#spring.datasource.useGlobalDataSourceStat=true
```

需要注意的是：spring.datasource.type旧的spring boot版本是不能识别的。

这时候启动应用就可以看到看到打印信息就是使用我们配置的数据源了：

[main] com.alibaba.druid.pool.DruidDataSource : {dataSource-1} inited

(3) 配置监控统计功能

配置Servlet

如下是在SpringBoot项目中基于注解的配置，如果是web.xml配置，按规则配置即可。

com.kfit.base.servlet.DruidStatViewServlet :

```
package com.kfit.base.servlet;
```

```
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;

import com.alibaba.druid.support.http.StatViewServlet;

/**
 * druid数据源状态监控.
 * @author Administrator
 *
 */

@WebServlet(urlPatterns="/druid/*",
        initParams={
                @WebInitParam(name="allow",value="192.168.1.72,127.0.0.1"),// IP白名单 (没有配置或者为空，则允许所有访问)
                @WebInitParam(name="deny",value="192.168.1.73"),// IP黑名单 (存在共同时，deny优先于allow)
                @WebInitParam(name="loginUsername",value="admin"),// 用户名
                @WebInitParam(name="loginPassword",value="123456"),// 密码
                @WebInitParam(name="resetEnable",value="false")// 禁用HTML页面上的“Reset All”功能
        }
)
public class DruidStatViewServlet extends StatViewServlet{
    private static final long serialVersionUID = 1L;

}
```

配置Filter

com.kfit.base.servlet.DruidStatFilter :

```
package com.kfit.base.servlet;

import javax.servlet.annotation.WebFilter;
import javax.servlet.annotation.WebInitParam;

import com.alibaba.druid.support.http.WebStatFilter;
```

```
/**
 * druid过滤器.
 * @author Administrator
 *
 */
@WebFilter(filterName="druidWebStatFilter",urlPatterns="/*",
    initParams={
            @WebInitParam(name="exclusions",value="*.js,*.gif,*.jpg,*.bmp,*.png,*.css,*.ico,/druid/*")// 忽略资源
     }
)
publicclass DruidStatFilter extends WebStatFilter{

}
```

最后在App.java类上加上注解：@ServletComponentScan是的spring能够扫描到我们自己编写的servlet和filter。

注意不要忘记在 SpringBootSampleApplication.java 上添加 @ServletComponentScan 注解，不然就是404了。

然后启动项目后访问 http://127.0.0.1:8080/druid/index.html 即可查看数据源及SQL统计等。

(4)配置监控系统方式二：

以上配置的监控方式是使用了原生的servlet，filter方式，然后通过@ServletComponentScan进行启动扫描包的方式进行处理的，你会发现我们的servlet，filter根本没有任何的编码。

在这里我们将使用另外一种方式进行处理：使用代码注册Servlet：

编写类：com.kfit.base.servlet.DruidConfiguration ：

```
package com.kfit.base.servlet;

import org.springframework.boot.context.embedded.FilterRegistrationBean;
import org.springframework.boot.context.embedded.ServletRegistrationBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import com.alibaba.druid.support.http.StatViewServlet;
import com.alibaba.druid.support.http.WebStatFilter;

/**
 * druid 配置.
 *
 * 这样的方式不需要添加注解：@ServletComponentScan
 * @author Administrator
 *
 */
@Configuration
publicclass DruidConfiguration {

    /**
     * 注册一个StatViewServlet
     * @return
     */
    @Bean
    public ServletRegistrationBean DruidStatViewServle2(){
        //org.springframework.boot.context.embedded.ServletRegistrationBean提供类的进行注册.
        ServletRegistrationBean servletRegistrationBean = new ServletRegistrationBean(new
StatViewServlet(),"/druid2/*");

        //添加初始化参数：initParams

        //白名单：
        servletRegistrationBean.addInitParameter("allow","127.0.0.1");
        //IP黑名单 (存在共同时，deny优先于allow) : 如果满足deny的话提示:Sorry, you are not permitted to view
this page.
        servletRegistrationBean.addInitParameter("deny","192.168.1.73");
        //登录查看信息的账号密码.
        servletRegistrationBean.addInitParameter("loginUsername","admin2");
        servletRegistrationBean.addInitParameter("loginPassword","123456");
        //是否能够重置数据.
        servletRegistrationBean.addInitParameter("resetEnable","false");
        returnservletRegistrationBean;
```

```
    }

    /**
     * 注册一个：filterRegistrationBean
     * @return
     */
    @Bean
    public FilterRegistrationBean druidStatFilter2(){

        FilterRegistrationBean filterRegistrationBean = new FilterRegistrationBean(new WebStatFilter());

        //添加过滤规则.
        filterRegistrationBean.addUrlPatterns("/*");

        //添加不需要忽略的格式信息.
    filterRegistrationBean.addInitParameter("exclusions","*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid2/*");
        returnfilterRegistrationBean;
    }

}
```

启动应用就可以访问：http://127.0.0.1:8080/druid2/index.html输入账号和密码：admin2/123456 就可以访问了。

附件下载:

- spring-boot-hello1.zip (1.4 MB)
- dl.iteye.com/topics/download/3c0307ef-d9cb-3fa2-9c68-0b47335e82f6

1.19（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】

发表时间: 2016-04-19 关键字: 从零开始学Spring Boot, Spring Boot使用Druid（编程注入)

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】:
http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508

...

（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

在上一节使用是配置文件的方式进行使用druid，这里在扩散下使用编程式进行使用Druid,在上一节我们新建了一个类：DruidConfiguration我在这个类进行编码：

```
package com.kfit.base.servlet;

import java.sql.SQLException;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Value;
```

```
import org.springframework.boot.context.embedded.FilterRegistrationBean;

import org.springframework.boot.context.embedded.ServletRegistrationBean;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import com.alibaba.druid.pool.DruidDataSource;

import com.alibaba.druid.support.http.StatViewServlet;

import com.alibaba.druid.support.http.WebStatFilter;



/**

 * druid 配置.

 *

 * 这样的方式不需要添加注解：@ServletComponentScan

 * @author Administrator

 *

 */

@Configuration

public class DruidConfiguration {



	/**
```

```
     * 注册一个StatViewServlet

     * @return

     */

    @Bean

    public ServletRegistrationBean DruidStatViewServle2(){

            //org.springframework.boot.context.embedded.ServletRegistrationBean提供类的进行注册

            ServletRegistrationBean servletRegistrationBean = new ServletRegistrationBean(new



            //添加初始化参数：initParams



            //白名单：

            servletRegistrationBean.addInitParameter("allow","127.0.0.1");

            //IP黑名单 (存在共同时，deny优先于allow) : 如果满足deny的话提示:Sorry, you are not p

            servletRegistrationBean.addInitParameter("deny","192.168.1.73");

            //登录查看信息的账号密码.

            servletRegistrationBean.addInitParameter("loginUsername","admin2");

            servletRegistrationBean.addInitParameter("loginPassword","123456");

            //是否能够重置数据.
```

```
        servletRegistrationBean.addInitParameter("resetEnable","false");

        return servletRegistrationBean;

    }



    /**

     * 注册一个：filterRegistrationBean

     * @return

     */

    @Bean

    public FilterRegistrationBean druidStatFilter2(){



        FilterRegistrationBean filterRegistrationBean = new FilterRegistrationBean(new We



        //添加过滤规则.

        filterRegistrationBean.addUrlPatterns("/*");



        //添加不需要忽略的格式信息.

        filterRegistrationBean.addInitParameter("exclusions","*.js,*.gif,*.jpg,*.png,*.cs

        return filterRegistrationBean;
```

```
    }



    /**
     * 注册dataSouce，这里作为一个例子，只注入了部分参数信息，其它的参数自行扩散思维。
     * @param driver
     * @param url
     * @param username
     * @param password
     * @param maxActive
     * @return
     */
    @Bean
public DataSource druidDataSource(@Value("${spring.datasource.driverClassName}") String dri
                                  @Value("${spring.datasource.url}") String url,
                                  @Value("${spring.datasource.username}") String username,
                                  @Value("${spring.datasource.password}") String password,
                                  @Value("${spring.datasource.maxActive}") int maxActive
                                                                        ) {
```

```
        DruidDataSource druidDataSource = new DruidDataSource();

        druidDataSource.setDriverClassName(driver);

        druidDataSource.setUrl(url);

        druidDataSource.setUsername(username);

        druidDataSource.setPassword(password);

        druidDataSource.setMaxActive(maxActive);



        System.out.println("DruidConfiguration.druidDataSource(),url="+url+",username="+usernam

        try {

            druidDataSource.setFilters("stat, wall");

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return druidDataSource;

    }



}
```

这里的区别在于加入一个方法：druidDataSource进行数据源的注入(当然这么一选择上一章节在application.properties配置的方式是比较好，如果有特殊需求的话，也可以在这里进行注入)。

如果同时进行了编程式的注入和配置的注入，配置的就无效了。

实际中推荐使用配置文件的方式，参考：

（15）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】：http://412887952-qq-com.iteye.com/blog/2292362

附件下载:

- spring-boot-hello1.zip (1.4 MB)
- dl.iteye.com/topics/download/50ac4718-9c07-340d-9a46-22e79b6e0f42

1.20 （17）Spring Boot普通类调用bean【从零开始学Spring Boot】

发表时间: 2016-04-19 关键字: 从零开始学Spring Boot, Spring Boot普通类调用bean

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496
（1）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500
（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
...
（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376
（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

我们知道如果我们要在一个类使用spring提供的bean对象，我们需要把这个类注入到spring容器中，交给spring容器进行管理，但是在实际当中，我们往往会碰到在一个普通的Java类中，想直接使用spring提供的其他对象或者说有一些不需要交给spring管理，但是需要用到spring里的一些对象。如果这是spring框架的独立应用程序，我们通过

```
ApplicationContext ac = new FileSystemXmlApplicationContext("applicationContext.xml");
ac.getBean("beanId");
```

这样的方式就可以很轻易的获取我们所需要的对象。
但是往往我们所做的都是Web Application，这时我们启动spring容器是通过在web.xml文件中配置，这样就不适合使用上面的方式在普通类去获取对象了，因为这样做就相当于加载了两次spring容器，而我们想是否可以通过在启动web服务器的时候，就把Application放在某一个类中，我们通过这个类在获取，这样就可以在普通类获取spring bean对象了，让我们接着往下看。

普通类调用Spring bean对象：

可以参考：http://412887952-qq-com.iteye.com/blog/1479445

这里有更多这方面的介绍，比较详细，在这里只是抛装引玉说明在Spring Boot是如何进行调用的。

17.1 在Spring Boot可以扫描的包下

假设我们编写的工具类为SpringUtil。

如果我们编写的SpringUtil在Spring Boot可以扫描的包下或者使用@ComponentScan引入自定义的包了，那么原理很简单，只需要使得SpringUtil实现接口：ApplicationContextAware，然后加上@Component 注解即可，具体编码如下：

com.kfit.base.util.SpringUtil:

```
package com.kfit.base.util;

import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.stereotype.Component;

/**
 * 普通类调用Spring bean对象：
 * 说明：
 * 1、此类需要放到App.java同包或者子包下才能被扫描，否则失效。
 * @author Administrator
 */
@Component
public class SpringUtil implements ApplicationContextAware{
    private static ApplicationContext applicationContext = null;

    @Override
    public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {
        if(SpringUtil.applicationContext == null){
            SpringUtil.applicationContext = applicationContext;
```

```
        }
        System.out.println("------------------------------------------------------------");
        System.out.println("------------------------------------------------------------");
    System.out.println("--------------com.kfit.base.util.SpringUtil-----------------------------------------------");
        System.out.println("========ApplicationContext配置成功,在普通类可以通过调用SpringUtils.getAppContext()获取applicationContext对象,applicationContext="+SpringUtil.applicationContext+"========");
        System.out.println("------------------------------------------------------------");
    }

    //获取applicationContext
    public static ApplicationContext getApplicationContext() {
        return applicationContext;
    }

    //通过name获取 Bean.
    public static Object getBean(String name){
        return getApplicationContext().getBean(name);
    }

    //通过class获取Bean.
    public static <T> T getBean(Class<T> clazz){
        return getApplicationContext().getBean(clazz);
    }

    //通过name,以及Clazz返回指定的Bean
    public static <T> T getBean(String name,Class<T> clazz){
        return getApplicationContext().getBean(name, clazz);
    }
}
```

启动应用，查看控制台的打印信息是否有：

======ApplicationContext配置成功,在普通类可以通过调用SpringUtils.getAppContext()获取applicationContext对象,

17.2 不在Spring Boot的扫描包下方式一

这种情况处理起来也很简单，先编写SpringUtil类，同样需要实现接口：ApplicationContextAware，具体编码如下：

simple.plugin.spring.SpringUtil

```
package simple.plugin.spring;

import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;

publicclass SpringUtil implements ApplicationContextAware{
   privatestatic ApplicationContext applicationContext = null;

   @Override
   publicvoid setApplicationContext(ApplicationContext applicationContext) throws BeansException {
     if(SpringUtil.applicationContext == null){
         SpringUtil.applicationContext = applicationContext;
     }
     System.out.println("----------------------------------------------------------------------");
     System.out.println("----------------------------------------------------------------------");
   System.out.println("---------------simple.plugin.spring.SpringUtil----------------------------------------------------");
      System.out.println("========ApplicationContext配置成功,在普通类可以通过调用SpringUtils.getAppContext()获取applicationContext对象,applicationContext="+SpringUtil.applicationContext+"========");
     System.out.println("----------------------------------------------------------------------");
   }

   //获取applicationContext
   publicstatic ApplicationContext getApplicationContext() {
     returnapplicationContext;
   }

   //通过name获取 Bean.
   publicstatic Object getBean(String name){
     return getApplicationContext().getBean(name);
   }
```

```
    //通过class获取Bean.
    public static <T> T getBean(Class<T> clazz){
        return getApplicationContext().getBean(clazz);
    }

    //通过name,以及Clazz返回指定的Bean
    public static <T> T getBean(String name,Class<T> clazz){
        return getApplicationContext().getBean(name, clazz);
    }
}
```

之后这一步才是关键，使用@Bean注解，在App.java类中将SpringUtil注解进来，代码如下：

```
package com.kfit;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.servlet.ServletComponentScan;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Import;

import simple.plugin.spring.SpringUtil;

/**
 * Hello world!
 *
 */
```

```
//其中@SpringBootApplication申明让spring boot自动给程序进行必要的配置，等价于以默认属性使用@Configuration， @EnableAutoConfiguration和@ComponentScan
@SpringBootApplication
@ServletComponentScan

public class App {

    /**注册Spring Util
     * 这里为了和上一个冲突，所以方面名为：springUtil2
     * 实际中使用springUtil
     */
    @Bean
    public SpringUtil springUtil2(){return new SpringUtil();}

    /**
     *
    参数里VM参数设置为：
     -javaagent:.\lib\springloaded-1.2.4.RELEASE.jar -noverify
     * @param args
     */
    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
```

```
}
```

17.3 不在Spring Boot的扫描包下方式二

代码基本和上面都是相同的，主要是在App.java中使用@Import进行导入。

而且在SpringUtil是不需要添加@Component注解

```
@SpringBootApplication
@ServletComponentScan
@Import(value={SpringUtil.class})
public class App {
    //省略其它代码.
}
```

说明以上3中方式都生效了，这3中方式根据实际情况选择一种方式就可以了。

那么这样子在普通类既可以使用:

SpringUtil.getBean() 获取到Spring IOC容器中的bean。

当然也可以在Spring管理的类中使用：

@Resouce或者@Autowired 进行注入使用，当然我们这个类的核心是普通类可以调用spring的bean进行使用了，是不是很神奇呢。

附件下载:

- spring-boot-hello1.zip (1.4 MB)

- dl.iteye.com/topics/download/26562524-043d-381b-a0ff-a5edb34ac6da

1.21 （18）使用模板（thymeleaf-freemarker）【从零开始学Spring Boot】

发表时间: 2016-04-19 关键字: 从零开始学Spring Boot, 使用模板

整体步骤：

（1） 在pom.xml中引入thymeleaf;

（2） 如何关闭thymeleaf缓存

（3） 编写模板文件.html

Spring Boot默认就是使用thymeleaf模板引擎的，所以只需要在pom.xml加入依赖即可：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Thymeleaf缓存在开发过程中，肯定是不行的，那么就要在开发的时候把缓存关闭，只需要在application.properties进行配置即可：

```
##########################################################
###THYMELEAF (ThymeleafAutoConfiguration)
##########################################################
#spring.thymeleaf.prefix=classpath:/templates/
#spring.thymeleaf.suffix=.html
#spring.thymeleaf.mode=HTML5
#spring.thymeleaf.encoding=UTF-8
# ;charset=<encoding> is added
#spring.thymeleaf.content-type=text/html
# set to false for hot refresh
```

```
spring.thymeleaf.cache=false
```

编写模板文件src/main/resouces/templates/helloHtml.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
    xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <h1 th:inline="text">Hello.v.2</h1>
    <p th:text="${hello}"></p>
  </body>
</html>
```

编写访问路径(com.kfit.test.web.TemplateController)：

```
package com.kfit.test.web;

import java.util.Map;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

/**
 * 模板测试.
 * @author Administrator
 *
 */
@Controller
publicclass TemplateController {

  /**
   * 返回html模板.
   */
  @RequestMapping("/helloHtml")
  public String helloHtml(Map<String,Object> map){
    map.put("hello","from TemplateController.helloHtml");
    return"/helloHtml";
```

```
    }

}
```

启动应用，输入地址：http://127.0.0.1:8080/helloHtml 会输出：

```
Hello.v.2
```

from TemplateController.helloHtml

18.2 使用freemarker

使用freemarker也很简单，

在pom.xml加入freemarker的依赖：

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-freemarker</artifactId>
</dependency>
```

剩下的编码部分都是一样的，说下application.properties文件：

```
#############################################################
###FREEMARKER (FreeMarkerAutoConfiguration)
#############################################################
spring.freemarker.allow-request-override=false
spring.freemarker.cache=true
spring.freemarker.check-template-location=true
spring.freemarker.charset=UTF-8
spring.freemarker.content-type=text/html
spring.freemarker.expose-request-attributes=false
spring.freemarker.expose-session-attributes=false
spring.freemarker.expose-spring-macro-helpers=false
#spring.freemarker.prefix=
```

```
#spring.freemarker.request-context-attribute=
#spring.freemarker.settings.*=
#spring.freemarker.suffix=.ftl
#spring.freemarker.template-loader-path=classpath:/templates/#comma-separatedlist
```

```
#spring.freemarker.view-names= #whitelistofviewnamesthatcanberesolved
```

com.kfit.test.web.TemplateController ：

```
/**
    * 返回html模板.
    */
   @RequestMapping("/helloFtl")
   public String helloFtl(Map<String,Object> map){
     map.put("hello","from TemplateController.helloFtl");
     return"/helloFtl";
```

```
   }
```

src/main/resouces/templates/helloFtl.ftl

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
     xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
   <head>
      <title>Hello World!</title>
   </head>
   <body>
      <h1>Hello.v.2</h1>
      <p>${hello}</p>
   </body>
</html>
```

访问地址：http://127.0.0.1:8080/helloFtl

```
Hello.v.2
```

from TemplateController.helloFtl

thymeleaf和freemarker是可以共存的。

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
...

（15）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2292362

（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388
.....
（35）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/blog

附件下载:

- spring-boot-hello1.zip (1.4 MB)
- dl.iteye.com/topics/download/96bfc16a-48f2-32f4-83a6-662a756e29e5

1.22 （19）Spring Boot 添加JSP支持【从零开始学Spring Boot】

发表时间: 2016-04-20 关键字: 从零开始学Spring Boot, Spring Boot 添加JSP支持

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496
（1）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500
（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
...
（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376
（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

【来也匆匆，去也匆匆，在此留下您的脚印吧,转发点赞评论；

您的认可是我最大的动力，感谢您的支持】

这个部分比较复杂，所以单独创建一个工程来进行讲解；

大体步骤：

（1） 创建Maven web project；

（2） 在pom.xml文件添加依赖；

（3） 配置application.properties支持jsp

（4） 编写测试Controller

（5） 编写JSP页面

（6） 编写启动类App.java

1，FreeMarker

2，Groovy

3，Thymeleaf（Spring 官网使用这个）

4，Velocity

5，JSP（貌似Spring Boot官方不推荐，STS创建的项目会在src/main/resources 下有个templates 目录，这里就是让我们放模版文件的，然后并没有生成诸如 SpringMVC 中的webapp目录）

不过本文还是选择大家都熟悉的JSP来举例，因为使用JSP与默认支持的模版需要特殊处理，所以拿来举例更好。

（1）创建Maven web project

使用Eclipse新建一个Maven Web Project ，项目取名为：

spring-boot-jsp

（2）在pom.xml文件添加依赖

```
<!-- spring boot parent节点，引入这个之后，在下面和spring boot相关的就不需要引入版本了; -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.3.RELEASE</version>

  </parent>
```

依赖包：

```
<!-- web支持: 1、web mvc; 2、restful; 3、jackjson支持; 4、aop ..... -->
```

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- servlet 依赖. -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <scope>provided</scope>
    </dependency>

    <!--
        JSTL ( JSP Standard Tag Library，JSP标准标签库)是一个不断完善的开放源代码的JSP标签库，是由apache的
jakarta小组来维护的。JSTL只能运行在支持JSP1.2和Servlet2.3规范的容器上，如tomcat 4.x。在JSP 2.0中也是作为标准支
持的。

        不然报异常信息：
        javax.servlet.ServletException: Circular view path [/helloJsp]: would dispatch back to the current
handler URL [/helloJsp] again. Check your ViewResolver setup! (Hint: This may be the result of an unspecified
view, due to default view name generation.)

     -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
    </dependency>



    <!-- tomcat 的支持.-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
    <dependency>
```

```
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
        <scope>provided</scope>
    </dependency>
```

Jdk编译版本：

```
<build>
    <finalName>spring-boot-jsp</finalName>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

（3）application.properties配置

上面说了spring-boot 不推荐JSP，想使用JSP需要配置application.properties。

添加src/main/resources/application.properties内容：

```
# 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 Angel From application
```

（4）编写测试Controller

编写类：com.kfit.jsp.controller.HelloController：

```
package com.kfit.jsp.controller;

import java.util.Map;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

/**
 * 测试
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Controller
public class HelloController {

    // 从 application.properties 中读取配置，如取不到默认值为Hello Shanhy
  @Value("${application.hello:Hello Angel}")
  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";
    }
}
```

（5）编写JSP页面

在 src/main 下面创建 webapp/WEB-INF/jsp 目录用来存放我们的jsp页面：helloJsp.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
  helloJsp
   <hr>
  ${hello}

</body>
</html>
```

(6)编写启动类

编写App.java启动类：

```
package com.kfit.jsp;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.web.SpringBootServletInitializer;

@SpringBootApplication
public class App extends SpringBootServletInitializer {
//    @Override
//    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
//        return application.sources(App.class);
//    }

    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}
```

右键Run As Java Application访问：http://127.0.0.1:8080/helloJsp 可以访问到：

helloJsp

Hello Angel From application

特别说明：针对el表达式，类似${hello}这个对于servlet的版本是有限制的，2.4版本版本以下是不支持的，是无法进行识别的，请注意。

附件下载:

- spring-boot-jsp.zip (17.1 KB)
- dl.iteye.com/topics/download/f9a750fb-2406-388d-9816-720065e89fae

1.23 （20）Spring Boot Servlet【从零开始学Spring Boot】

发表时间: 2016-04-20 关键字: Spring Boot Servlet, 从零开始学Spring Boot

Web开发使用 Controller 基本上可以完成大部分需求，但是我们还可能会用到 Servlet、Filter、Listener、Interceptor 等等。

当使用Spring-Boot时，嵌入式Servlet容器通过扫描注解的方式注册Servlet、Filter和Servlet规范的所有监听器（如HttpSessionListener监听器）。

Spring boot 的主 Servlet 为 DispatcherServlet，其默认的url-pattern为“/”。也许我们在应用中还需要定义更多的Servlet，该如何使用SpringBoot来完成呢？

在spring boot中添加自己的Servlet有两种方法，代码注册Servlet和注解自动注册（Filter和Listener也是如此）。

一、代码注册通过ServletRegistrationBean、 FilterRegistrationBean 和 ServletListenerRegistrationBean 获得控制。

也可以通过实现 ServletContextInitializer 接口直接注册。

二、在 SpringBootApplication 上使用@ServletComponentScan注解后，Servlet、Filter、Listener 可以直接通过 @WebServlet、@WebFilter、@WebListener 注解自动注册，无需其他代码。

这里我们新建一个spring-boot-hello2 java工程进行测试；这里不过多进行说明，如果这个还不会的话，请回到上上上一章进行查看。

通过代码注册Servlet示例代码：

com.kfit.servlet.MyServlet1：

```
package com.kfit.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
```

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

/**
 *
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
//这个不需要添加.
//@WebServlet(urlPatterns="/myServlet1/*", description="Servlet的说明")
public class MyServlet1 extends HttpServlet{

    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        System.out.println(">>>>>>>>>>>doGet()<<<<<<<<<<<");
        doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        System.out.println(">>>>>>>>>>>doPost()<<<<<<<<<<<");
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>这是：MyServlet1</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

在com.kfit.App中注册：

```
package com.kfit;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.embedded.ServletRegistrationBean;
import org.springframework.boot.web.servlet.ServletComponentScan;
import org.springframework.context.annotation.Bean;

import com.kfit.servlet.MyServlet1;

/**
 *
 *
 * 大家也许会看到有些demo使用了3个注解： @Configuration；
 *
 * @EnableAutoConfiguration
 * @ComponentScan
 *
 *                  其实：@SpringBootApplication申明让spring boot自动给程序进行必要的配置，
 *
 *              等价于以默认属性使用@Configuration，
```

```
*		@EnableAutoConfiguration和@ComponentScan
*
* 所以大家不要被一些文档误导了，让自己很迷茫了，希望本文章对您有所启发；
*
* @author Angel(QQ:412887952)
* @version v.0.1
*/
@SpringBootApplication
public class App {

	/**
	 * 注册Servlet.不需要添加注解：@ServletComponentScan
	 * @return
	 */
	@Bean
	public ServletRegistrationBean MyServlet1(){
		return new ServletRegistrationBean(new MyServlet1(),"/myServlet/*");
	}

	public static void main(String[] args) {
		SpringApplication.run(App.class, args);
	}
}
```

右键Run As Java Application进行访问http://127.0.0.1:8080/myServlet1

使用注解注册Servlet示例代码

com.kfit.servlet.MyServlet2.java

```
package com.kfit.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@WebServlet(urlPatterns="/myServlet2/*", description="Servlet的说明")
```

```
public class MyServlet2 extends HttpServlet{

    private static final long serialVersionUID = 1L;

    @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
    System.out.println(">>>>>>>>>>doGet()<<<<<<<<<<");
    doPost(req, resp);
  }

  @Override
  protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
    System.out.println(">>>>>>>>>>doPost()<<<<<<<<<<");
    resp.setContentType("text/html");
    PrintWriter out = resp.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Hello World</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>这是：myServlet2</h1>");
    out.println("</body>");
```

```
        out.println("</html>");
    }
}
```

SpringBootSampleApplication.java

```
package com.kfit;

import org.springboot.sample.servlet.MyServlet;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.embedded.ServletRegistrationBean;
import org.springframework.boot.web.servlet.ServletComponentScan;
import org.springframework.context.annotation.Bean;
import org.springframework.web.servlet.DispatcherServlet;

@SpringBootApplication
@ServletComponentScan//这个就是扫描相应的Servlet包;
public class SpringBootSampleApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootSampleApplication.class, args);
    }
}
```

访问http://127.0.0.1:8080/myServlet2

1.24 （21）Spring Boot过滤器、监听器【从零开始学Spring Boot】

发表时间: 2016-04-20 关键字: Spring Boot过滤器、监听器, 从零开始学Spring Boot

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508

...

（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

上一篇文章已经对定义Servlet 的方法进行了说明，过滤器（Filter）和监听器（Listener）的注册方法和 Servlet 一样，不清楚的可以查看下上一篇文章（20）： 本文将直接使用@WebFilter和@WebListener的方式，完成一个 Filter 和一个 Listener；使用注解

```
@ServletComponentScan//这个就是扫描相应的Servlet包;
```

过滤器（Filter）文件

com.kfit.filter.MyFilter.java

```
package com.kfit.filter;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;

/**
 *
 * 使用注解标注过滤器
 * @WebFilter将一个实现了javax.servlet.Filter接口的类定义为过滤器
 * 属性filterName声明过滤器的名称,可选
 * 属性urlPatterns指定要过滤的URL模式,也可使用属性value来声明.(指定要过滤的URL模式是必选属性)
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@WebFilter(filterName="myFilter",urlPatterns="/*")
publicclass MyFilter implements Filter{

    @Override
    publicvoid init(FilterConfig config) throws ServletException {
        System.out.println("过滤器初始化");
    }


    @Override
    publicvoid doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        System.out.println("执行过滤操作");
        chain.doFilter(request, response);
```

```
    }

    @Override
    publicvoid destroy() {
        System.out.println("过滤器销毁");
    }
}
```

ServletContext监听器（Listener）文件

com.kfit.listener.MyServletContextListener:

```
package com.kfit.listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

/**
 * 使用@WebListener注解，实现ServletContextListener接口
 *
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@WebListener
public class MyServletContextListener implements ServletContextListener {

    @Override
    public void contextDestroyed(ServletContextEvent arg0) {
        System.out.println("ServletContex销毁");
    }

    @Override
    public void contextInitialized(ServletContextEvent arg0) {
```

```
            System.out.println("ServletContex初始化");
        }
}
```

ServletContext监听器（Listener）文件（HttpSessionListener）

MyHttpSessionListener.java

```
package com.kfit.listener;

import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

/**
 * 监听Session的创建与销毁
 *
 */
@WebListener
publicclassMyHttpSessionListenerimplementsHttpSessionListener {

    @Override
    publicvoid sessionCreated(HttpSessionEvent se) {
        System.out.println("Session 被创建");
    }

    @Override
    publicvoid sessionDestroyed(HttpSessionEvent se) {
        System.out.println("ServletContex初始化");
    }

}
```

注意不要忘记在 SpringBootSampleApplication.java 上添加 @ServletComponentScan 注解。

启动的过程中我们会看到输出：

```
ServletContex初始化
```

过滤器初始化

服务启动后，随便访问一个页面，会看到输出：

执行过滤操作

Session 被创建

为什么无法看到session的过程： http://zhidao.baidu.com/link?url=EP-wlLvKpo8zI5NaIZrESzCdivq3Xg8VgOWQOvfpSLl3opTgvESerpo4wsG6tOs_dm6cQQMF_kQ6THNjNzr2Nq

至于如何使用代码的方式注册Filter和Listener，请参考上一篇文章关键Servlet的介绍。不同的是需要使用 FilterRegistrationBean 和 ServletListenerRegistrationBean 这两个类。

附件下载:

- spring-boot-hello2.zip (20.1 KB)
- dl.iteye.com/topics/download/dc42ccaa-a0f5-362c-aae8-7fa45a4f751d

1.25 （22）Spring Boot 拦截器HandlerInterceptor【从零开始学Spring Boot】

发表时间: 2016-04-20 关键字: Spring Boot 拦截器HandlerInterceptor, 从零开始学Spring Boot

【Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508

...

（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

上一篇对过滤器的定义做了说明，也比较简单。过滤器属于Servlet范畴的API，与Spring 没什么关系。

Web开发中，我们除了使用 Filter 来过滤请web求外，还可以使用Spring提供的HandlerInterceptor（拦截器）。

HandlerInterceptor 的功能跟过滤器类似，但是提供更精细的的控制能力：在request被响应之前、request被响应之后、视图渲染之前以及request全部结束之后。我们不能通过拦截器修改request内容，但是可以通过抛出异常（或者返回false）来暂停request的执行。

实现 UserRoleAuthorizationInterceptor 的拦截器有：

ConversionServiceExposingInterceptor

CorsInterceptor

LocaleChangeInterceptor

PathExposingHandlerInterceptor

ResourceUrlProviderExposingInterceptor

ThemeChangeInterceptor

UriTemplateVariablesHandlerInterceptor

UserRoleAuthorizationInterceptor

其中 LocaleChangeInterceptor 和 ThemeChangeInterceptor 比较常用。

配置拦截器也很简单，Spring 为什么提供了基础类WebMvcConfigurerAdapter ，我们只需要重写 addInterceptors 方法添加注册拦截器。

实现自定义拦截器只需要3步：

1、创建我们自己的拦截器类并实现 HandlerInterceptor 接口。

2、创建一个Java类继承WebMvcConfigurerAdapter，并重写 addInterceptors 方法。

2、实例化我们自定义的拦截器，然后将对像手动添加到拦截器链中（在addInterceptors方法中添加）。

PS：本文重点在如何在Spring-Boot中使用拦截器，关于拦截器的原理请大家查阅资料了解。

代码示例：

com.kfit.interceptor.MyInterceptor1.java

```
package com.kfit.interceptor;

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

import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

/**
 * 自定义拦截器1
 *
 * @author    Angel
 */
publicclass MyInterceptor1 implements HandlerInterceptor{
```

```
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
            throws Exception {
        System.out.println(">>>MyInterceptor1>>>>>>>在请求处理之前进行调用（Controller方法调用之前）");

        return true;// 只有返回true才会继续向下执行，返回false取消当前请求
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
            ModelAndView modelAndView) throws Exception {
        System.out.println(">>>MyInterceptor1>>>>>>>请求处理之后进行调用，但是在视图被渲染之前（Controller方法调用之后）");
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,
Exception ex)
            throws Exception {
        System.out.println(">>>MyInterceptor1>>>>>>>在整个请求结束之后被调用，也就是在DispatcherServlet 渲染了对应的视图之后执行（主要是用于进行资源清理工作）");
    }

}
```

com.kfit.interceptor.MyInterceptor2.java

```
package com.kfit.interceptor;

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

import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

/**
 * 自定义拦截器2
 *
 * @author    Angel
 */
public class MyInterceptor2 implements HandlerInterceptor {
```

```
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
            throws Exception {
        System.out.println(">>>MyInterceptor2>>>>>>>在请求处理之前进行调用（Controller方法调用之前）");

        return true;// 只有返回true才会继续向下执行，返回false取消当前请求
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
            ModelAndView modelAndView) throws Exception {
        System.out.println(">>>MyInterceptor2>>>>>>>请求处理之后进行调用，但是在视图被渲染之前（Controller方法调用之后）");
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,
Exception ex)
            throws Exception {
        System.out.println(">>>MyInterceptor2>>>>>>>在整个请求结束之后被调用，也就是在DispatcherServlet 渲染了对应的视图之后执行（主要是用于进行资源清理工作）");
    }

}
```

Com.kfit.config.MyWebAppConfigurer.java

```
package com.kfit.config;

import com.kfit.interceptor.MyInterceptor1;
import com.kfit.interceptor.MyInterceptor2;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@Configuration
public class MyWebAppConfigurer
    extends WebMvcConfigurerAdapter {

    @Override
```

```
    public void addInterceptors(InterceptorRegistry registry) {
        // 多个拦截器组成一个拦截器链
        // addPathPatterns 用于添加拦截规则
        // excludePathPatterns 用户排除拦截
        registry.addInterceptor(new MyInterceptor1()).addPathPatterns("/**");
        registry.addInterceptor(new MyInterceptor2()).addPathPatterns("/**");
        super.addInterceptors(registry);
    }

}
```

然后在浏览器输入地址： http://localhost:8080/index 后，控制台的输出为：

```
>>>MyInterceptor1>>>>>>>在请求处理之前进行调用（Controller方法调用之前）
>>>MyInterceptor2>>>>>>>在请求处理之前进行调用（Controller方法调用之前）
>>>MyInterceptor2>>>>>>>请求处理之后进行调用，但是在视图被渲染之前（Controller方法调用之后）
>>>MyInterceptor1>>>>>>>请求处理之后进行调用，但是在视图被渲染之前（Controller方法调用之后）
>>>MyInterceptor2>>>>>>>在整个请求结束之后被调用，也就是在DispatcherServlet 渲染了对应的视图之后执行（主要是用于进行资源清理工作）

>>>MyInterceptor1>>>>>>>在整个请求结束之后被调用，也就是在DispatcherServlet 渲染了对应的视图之后执行（主要是用于进行资源清理工作）
```

根据输出可以了解拦截器链的执行顺序（具体原理介绍，大家找度娘一问便知）

最后强调一点：只有经过DispatcherServlet 的请求，才会走拦截器链，我们自定义的Servlet 请求是不会被拦截的，比如我们自定义的Servlet地址http://localhost:8080/myServlet1 是不会被拦截器拦截的。并且不管是属于哪个Servlet 只要复合过滤器的过滤规则，过滤器都会拦截。

最后说明下，我们上面用到的 WebMvcConfigurerAdapter 并非只是注册添加拦截器使用，其顾名思义是做Web配置用的，它还可以有很多其他作用，通过下面截图便可以大概了解，具体每个方法都是干什么用的，留给大家自己研究（其实都大同小异也很简单）。

附件下载:

- spring-boot-hello2.zip (26.3 KB)
- dl.iteye.com/topics/download/d4555266-0693-376f-8d58-9e15bd969ef9

1.26 （23）Spring Boot启动加载数据CommandLineRunner【从零开始学Spring Boot】

发表时间: 2016-04-20 关键字: 从零开始学Spring Boot, 启动加载数据CommandLineRunner, spring boot

【Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
...

（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

实际应用中，我们会有在项目服务启动的时候就去加载一些数据或做一些事情这样的需求。

为了解决这样的问题，Spring Boot 为我们提供了一个方法，通过实现接口 CommandLineRunner 来实现。

很简单，只需要一个类就可以，无需其他配置。

创建实现接口 com.kfit.runner.CommandLineRunner 的类

```
package com.kfit.runner;

import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

/**
 * 服务启动执行
 *
 * @author    Angel(QQ:412887952)
 */
@Component
publicclassMyStartupRunner1implementsCommandLineRunner {

   @Override
   publicvoid run(String... args) throws Exception {
      System.out.println(">>>>>>>>>>>>>>>服务启动执行，执行加载数据等操作<<<<<<<<<<<<<");
   }

}
```

Spring Boot应用程序在启动后，会遍历CommandLineRunner接口的实例并运行它们的run方法。也可以利用

@Order注解（或者实现Order接口）来规定所有CommandLineRunner实例的运行顺序。

如下我们使用@Order 注解来定义执行顺序。

```
package com.kfit.runner;

import org.springframework.boot.CommandLineRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

/**
 * 服务启动执行
 *
 * @author    Angel(QQ:412887952)
 */
@Component
@Order(value=2)
publicclassMyStartupRunner1implementsCommandLineRunner {
```

```
    @Override
    public void run(String... args) throws Exception {
        System.out.println(">>>>>>>>>>>>>>>服务启动执行，执行加载数据等操作 11111111 <<<<<<<<<<<<");
    }

}
```

```
package com.kfit.runner;

import org.springframework.boot.CommandLineRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

/**
 * 服务启动执行
 *
 * @author    Angel(QQ:412887952)
 */
@Component
@Order(value=1)
public class MyStartupRunner2 implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
        System.out.println(">>>>>>>>>>>>>>>服务启动执行，执行加载数据等操作 22222222 <<<<<<<<<<<<");
    }

}
```

启动程序后，控制台输出结果为：

```
>>>>>>>>>>>>>>>服务启动执行，执行加载数据等操作 22222222 <<<<<<<<<<<<

>>>>>>>>>>>>>>>服务启动执行，执行加载数据等操作 11111111 <<<<<<<<<<<<
```

根据控制台结果可判断，@Order 注解的执行优先级是按value值从小到大顺序。

```
@Override
    public void run(String... args) throws Exception {
            System.out.println(Arrays.asList(args));
        System.out.println(">>>>>>>>>>>>>>>>>服务启动执行，执行加载数据等操作11111111<<<<<<<<<<<<<");
    }
```

这里的args就是程序启动的时候进行设置的:

```
SpringApplication.run(App.class, new String[]{"hello,","林峰"});
```

这里为了做演示，配置为固定值了，其实直接接收main中的args即可，那么在运行的时候，进行配置即可。

题外话：

eclipse中给java应用传args参数的方法如下：
1、先写好Java代码，比如文件名为IntArrqy.java；
2、在工具栏或菜单上点run as下边有个Run Configuration；
3、在弹出窗口点选第二个标签arguments；
4、把你想输入的参数写在program argumenst就可以了，多个参数使用空格隔开。
完成后点run即可通过运行结果看到参数使用情况了。

1.27 （24）Spring Boot环境变量读取和属性对象的绑定【从零开始学Spring Boot】

发表时间: 2016-04-21 关键字: 从零开始学Spring Boot, Spring Boot环境变量读取和属性对象的绑定, spring bot

凡是被Spring管理的类，实现接口 EnvironmentAware 重写方法 setEnvironment 可以在工程启动时，获取到系统环境变量和application配置文件中的变量。

com.kfit.environment.MyEnvironmentAware :

```
package com.kfit.environment;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.bind.RelaxedPropertyResolver;
import org.springframework.context.EnvironmentAware;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.env.Environment;

/**
 * 主要是@Configuration，实现接口：EnvironmentAware就能获取到系统环境信息;
 *
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Configuration
public class MyEnvironmentAware implements EnvironmentAware{
```

```
    //注入application.properties的属性到指定变量中.
    @Value("${spring.datasource.url}")
    private String myUrl;

    /**
     *注意重写的方法 setEnvironment 是在系统启动的时候被执行。
     */
    @Override
    public void setEnvironment(Environment environment) {

        //打印注入的属性信息.
        System.out.println("myUrl="+myUrl);

        //通过 environment 获取到系统属性.
        System.out.println(environment.getProperty("JAVA_HOME"));

        //通过 environment 同样能获取到application.properties配置的属性.
        System.out.println(environment.getProperty("spring.datasource.url"));

        //获取到前缀是"spring.datasource." 的属性列表值.
        RelaxedPropertyResolver relaxedPropertyResolver = new RelaxedPropertyResolver(environment,
"spring.datasource.");
        System.out.println("spring.datasource.url="+relaxedPropertyResolver.getProperty("url"));
```

```
System.out.println("spring.datasource.driverClassName="+relaxedPropertyResolver.getProperty("driverClassName
    }
}
```

其中application.properties文件信息是：

```
##############################################################
###datasource
##############################################################
spring.datasource.url = jdbc:mysql://localhost:3306/test
spring.datasource.username = root
spring.datasource.password = root
spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.datasource.max-active=20
spring.datasource.max-idle=8
spring.datasource.min-idle=8
spring.datasource.initial-size=10
```

@Controller @Service 等被Spring管理的类都支持，注意重写的方法 setEnvironment 是在系统启动的时候被执行。

或者如下Controller：

```
@Controller
publicclassPageControllerimplementsEnvironmentAware{

  @Override
  publicvoid setEnvironment(Environment environment) {
    String s = environment.getProperty("JAVA_HOME");
    System.out.println(s);
  }

}
```

我们还可以通过@ConfigurationProperties 读取application属性配置文件中的属性。

```
@Configuration
@ConditionalOnClass(Mongo.class)
@EnableConfigurationProperties(MongoProperties.class)
publicclassMongoAutoConfiguration {

  @Autowired
  private MongoProperties properties;

}
```

· @ConditionOnClass表明该@Configuration仅仅在一定条件下才会被加载，这里的条件是Mongo.class位于类路径上

· @EnableConfigurationProperties将Spring Boot的配置文件（application.properties）中的spring.data.mongodb.*属性映射为MongoProperties并注入到MongoAutoConfiguration中。

· @ConditionalOnMissingBean说明Spring Boot仅仅在当前上下文中不存在Mongo对象时，才会实例化一个Bean。这个逻辑也体现了Spring Boot的另外一个特性——自定义的Bean优先于框架的默认配置，我们如果显式的在业务代码中定义了一个Mongo对象，那么Spring Boot就不再创建。

```
@ConfigurationProperties(prefix = "spring.data.mongodb")
publicclass MongoProperties {

  private String host;
  privateint port = DBPort.PORT;
  private String uri = "mongodb://localhost/test";
  private String database;

  // ... getters/ setters omitted

}
```

它就是以spring.data.mongodb作为前缀的属性，然后通过名字直接映射为对象的属性，同时还包含了一些默认值。如果不配置，那么mongo.uri就是mongodb://localhost/test。

以上这个配置需要加入依赖：

```
<!--spring-boot-configuration:spring boot 配置处理器; -->
    <dependency>
```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-configuration-processor</artifactId>
        <optional>true</optional>
    </dependency>
```

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508

...

（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

1.28 （25）Spring Boot使用自定义的properties【从零开始学Spring Boot】

发表时间: 2016-04-21 关键字: Spring Boot使用自定义的properties, 从零开始学Spring Boot, spring boot

spring boot使用application.properties默认了很多配置。但需要自己添加一些配置的时候，我们应该怎么做呢。

若继续在application.properties中添加

如：

```
wisely2.name=wyf2
wisely2.gender=male2
```

定义配置类：

```
@ConfigurationProperties(prefix = "wisely2")
public class Wisely2Settings {
  private String name;
  private String gender;
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public String getGender() {
    return gender;
  }
  public void setGender(String gender) {
```

```
15.    this.gender = gender;
16.  }
17.
18.}
```

若新用新的配置文件

如我新建一个wisely.properties

```
1. wisely.name=wangyunfei
2. wisely.gender=male
```

需定义如下配置类

```
1. @ConfigurationProperties(prefix = "wisely",locations = "classpath:config/wisely.properties")
2. public class WiselySettings {
3.    private String name;
4.    private String gender;
5.    public String getName() {
6.      return name;
7.    }
8.    public void setName(String name) {
9.      this.name = name;
10.   }
11.   public String getGender() {
```

```
12.     return gender;
13.   }
14.   public void setGender(String gender) {
15.     this.gender = gender;
16.   }
17.
18.}
```

最后注意在spring Boot入口类加上@EnableConfigurationProperties

```
1. @SpringBootApplication
2. @EnableConfigurationProperties({WiselySettings.class,Wisely2Settings.class})
3. public class DemoApplication {
4.
5.   public static void main(String[] args) {
6.     SpringApplication.run(DemoApplication.class, args);
7.   }
8. }
```

使用定义的properties

```
1. @Controller
2. public class TestController {
3.   @Autowired
```

```
4.  WiselySettings wiselySettings;
5.  @Autowired
6.  Wisely2Settings wisely2Settings;
7.
8.  @RequestMapping("/test")
9.  public @ResponseBody String test(){
10.   System.out.println(wiselySettings.getGender()+"---"+wiselySettings.getName());
11.   System.out.println(wisely2Settings.getGender()+"==="+wisely2Settings.getGender());
12.   return "ok";
13. }
```

}

Spring Boot 系列博客】

（ 0 ）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（ 1 ）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（ 2 ）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

...

（ 16 ）Spring Boot使用Druid（ 编程注入 ）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（ 17 ）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

1.29 （26）改变自动扫描的包【从零开始学Spring Boot】

发表时间: 2016-04-21 关键字: 从零开始学Spring Boot, 改变自动扫描的包, spring boot

在开发中我们知道Spring Boot默认会扫描启动类同包以及子包下的注解，那么如何进行改变这种扫描包的方式呢，原理很简单就是：

@ComponentScan注解进行指定要扫描的包以及要扫描的类。

接下来我们简单写个例子进行测试下。

第一步：新建两个新包

我们在项目中新建两个包cn.kfit ; org.kfit ；

第二步：新建两个测试类；

在这里为了方便测试，我们让我们的类在启动的时候就进行执行，所以我们就编写两个类，实现接口CommandLineRunner，这样在启动的时候我们就可以看到打印信息了。

cn.kfit.MyCommandLineRunner1 ：

```
package cn.kfit;

import org.springframework.boot.CommandLineRunner;

@Configuration
publicclass MyCommandLineRunner1 implements CommandLineRunner {

   @Override
   publicvoid run(String... args) throws Exception {
      System.out.println("MyCommandLineRunner1.run()");
   }
}
```

org.kfit.MyCommandLineRunner2 ：

```
package org.kfit;

import org.springframework.boot.CommandLineRunner;

@Configuration
publicclass MyCommandLineRunner2 implements CommandLineRunner {

   @Override
   publicvoid run(String... args) throws Exception {
      System.out.println("MyCommandLineRunner2.run()");
   }
}
```

第三步：启动类进行注解指定

在App.java类中加入如下注解：

```
//可以使用：basePackageClasses={},basePackages={}
@ComponentScan(basePackages={"cn.kfit","org.kfit"})
```

启动如果看到打印信息：

```
MyCommandLineRunner1.run()
MyCommandLineRunner2.run()
```

说明我们配置成功了。

这时候你会发现，在App.java同包下的都没有被扫描了，所以如果也希望App.java包下的也同时被扫描的话，那么在进行指定包扫描的时候一定要进行指定配置：

```
@ComponentScan(basePackages={"cn.kfit","org.kfit","com.kfit"})
```

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496
（1）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500
（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
...
（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376
（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

附件下载:

- spring-boot-hello2.zip (40.7 KB)
- dl.iteye.com/topics/download/8c5fe95a-467f-321c-9e10-53ea57e60033

1.30 （27）Spring Boot Junit单元测试【从零开始学Spring Boot】

发表时间: 2016-04-21 关键字: Spring Boot Junit单元测试, 从零开始学Spring Boot, spring boot

Junit这种老技术，现在又拿出来说，不为别的，某种程度上来说，更是为了要说明它在项目中的重要性。

那么先简单说一下为什么要写测试用例

1. 可以避免测试点的遗漏，为了更好的进行测试，可以提高测试效率

2. 可以自动测试，可以在项目打包前进行测试校验

3. 可以及时发现因为修改代码导致新的问题的出现，并及时解决

那么本文从以下几点来说明怎么使用Junit，Junit4比3要方便很多，细节大家可以自己了解下，主要就是版本4中对方法命名格式不再有要求，不再需要继承TestCase，一切都基于注解实现。

那么Spring Boot如何使用Junit呢？

1). 加入Maven的依赖；

2). 编写测试service;

3). 编写测试类;

1). 加入Maven的依赖:

```
<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-test</artifactId>
		<scope>test</scope>
</dependency>
```

2). 编写测试service:

新建com.kfit.service.HelloService 提供测试方法：

```
package com.kfit.service;

import org.springframework.stereotype.Service;

@Service
publicclass HelloService {

   public String getName(){
     return"hello";
   }
}
```

3). 编写测试类：

在src/test/java下编写测试类： com.kfit.test.HelloServiceTest:

```
package com.kfit.test;

import javax.annotation.Resource;

import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.SpringApplicationConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;

import com.kfit.App;
import com.kfit.service.HelloService;

//// SpringJUnit支持，由此引入Spring-Test框架支持！
@RunWith(SpringJUnit4ClassRunner.class)

//// 指定我们SpringBoot工程的Application启动类
@SpringApplicationConfiguration(classes = App.class)
```

```
///由于是Web项目，Junit需要模拟ServletContext，因此我们需要给我们的测试类加上@WebAppConfiguration。
@WebAppConfiguration
publicclass HelloServiceTest {

   @Resource
   private HelloService helloService;

   @Test
   publicvoid testGetName(){
      Assert.assertEquals("hello",helloService.getName());
   }
}
```

这时候就可以进行测试了，右键—Run As – Junit Test。

Junit基本注解介绍

//在所有测试方法前执行一次，一般在其中写上整体初始化的代码
@BeforeClass

//在所有测试方法后执行一次，一般在其中写上销毁和释放资源的代码
@AfterClass

//在每个测试方法前执行，一般用来初始化方法（比如我们在测试别的方法时，类中与其他测试方法共享的值已经被改变，为了保证测试结果的有效性，我们会在@Before注解的方法中重置数据）
@Before

//在每个测试方法后执行，在方法执行完成后要做的事情
@After

// 测试方法执行超过1000毫秒后算超时，测试将失败
@Test(timeout = 1000)

// 测试方法期望得到的异常类，如果方法执行没有抛出指定的异常，则测试失败
@Test(expected = Exception.class)

// 执行测试时将忽略掉此方法，如果用于修饰类，则忽略整个类
@Ignore(“not ready yet”)

@Test

@RunWith

在JUnit中有很多个Runner，他们负责调用你的测试代码，每一个Runner都有各自的特殊功能，你要根据需要选择不同的Runner来运行你的测试代码。

如果我们只是简单的做普通Java测试，不涉及Spring Web项目，你可以省略@RunWith注解，这样系统会自动使用默认Runner来运行你的代码。

Spring Boot 系列博客】

（ 0 ）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（ 1 ）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（ 2 ）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

...

（ 16 ）Spring Boot使用Druid（ 编程注入 ）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（ 17 ）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

附件下载:

- spring-boot-hello2.zip (44 KB)
- dl.iteye.com/topics/download/e9551ed8-cb2d-32b0-97ef-06c1ffbd1b4b

1.31 （28）SpringBoot启动时的Banner设置【从零开始学Spring Boot】

发表时间: 2016-04-22 关键字: SpringBoot启动时的Banner设置, 从零开始学Spring Boot

对于使用过Spring Boot的开发者来说，程序启动的时候输出的由字符组成的Spring符号并不陌生。这个是Spring Boot为自己设计的Banner：

```
1.  .   ____          _            __ _ _
2. /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
3. ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
4.  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
5.   '  |____| .__|_| |_|_| |_\__, | / / / /
6.  =========|_|==============|___/=/_/_/_/
7.  :: Spring Boot ::        (v1.3.3.RELEASE)
```

如果有人不喜欢这个输出，本文说一下怎么修改。

第一种方式：修改的时候，进行设置,在Application的main方法中：

```
SpringApplication application = new SpringApplication(App.class);
        /*
         * Banner.Mode.OFF:关闭;
         * Banner.Mode.CONSOLE:控制台输出，默认方式;
         * Banner.Mode.LOG:日志输出方式;
         */
        application.setBannerMode(Banner.Mode.OFF);
```

```
        application.run(args);
```

第二种方式：修改banner.txt配置文件

在src/main/resouces下新建banner.txt，在文件中加入：

```
#这个是MANIFEST.MF文件中的版本号
${application.version}

#这个是上面的的版本号前面加v后上括号
${application.formatted-version}

#这个是springboot的版本号
${spring-boot.version}

#这个是springboot的版本号
${spring-boot.formatted-version}
```

第三种方式：重写接口Banner实现

SpringBoot提供了一个接口org.springframework.boot.Banner，他的实例可以被传给SpringApplication的setBanner(banner)方法。如果你闲得不行非要着重美化这个命令行输出的话，可以重写Banner接口的printBanner方法。

第四种方式：在application.properties进行配置

在application.proerpties进行banner的显示和关闭：

```
### 是否显示banner，可选值[true|false]
spring.main.show-banner=false
```

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

...

（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

1.32 （29）Spring boot 文件上传（多文件上传）【从零开始学Spring Boot】

发表时间: 2016-04-24 关键字: Spring boot 文件上传（多文件上传）, 从零开始学Spring Boot, spring boot

文件上传主要分以下几个步骤：

（1）新建maven java project；

（2）在pom.xml加入相应依赖；

（3）新建一个表单页面（这里使用thymleaf）；

（4）编写controller;

（5）测试；

（6）对上传的文件做一些限制；

（7）多文件上传实现

（1）新建maven java project

新建一个名称为spring-boot-fileupload maven java项目；

（2）在pom.xml加入相应依赖；

加入相应的maven依赖，具体看以下解释：

```
<!--
    spring boot 父节点依赖,
    引入这个之后相关的引入就不需要添加version配置，
    spring boot会自动选择最合适的版本进行添加。
   -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
```

```
    <version>1.3.3.RELEASE</version>
  </parent>
<dependencies>
     <!-- spring boot web支持：mvc,aop... -->
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- thmleaf模板依赖. -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
</dependencies>
  <build>
    <plugins>
       <!-- 编译版本; -->
       <plugin>
         <artifactId>maven-compiler-plugin</artifactId>
         <configuration>
            <source>1.8</source>
            <target>1.8</target>
         </configuration>
       </plugin>
    </plugins>

  </build>
```

（3）新建一个表单页面（这里使用thymleaf）

在src/main/resouces新建templates(如果看过博主之前的文章，应该知道，templates是spring boot存放模板文件的路径)，在templates下新建一个file.html:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
    xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
```

```
<head>
    <title>Hello World!</title>
</head>
<body>
    <form method="POST" enctype="multipart/form-data" action="/upload">
        <p>文件：<input type="file" name="file" /></p>
        <p><input type="submit" value="上传" /></p>
    </form>
</body>
</html>
```

（4）编写controller;

编写controller进行测试，这里主要实现两个方法：其一就是提供访问的/file路径；其二就是提供post上传的/upload方法，具体看代码实现：

```
package com.kfit;

import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;

@Controller
publicclass FileUploadController {

    //访问路径为：http://127.0.0.1:8080/file
    @RequestMapping("/file")
    public String file(){
        return"/file";
```

```
}

/**
 * 文件上传具体实现方法;
 * @param file
 * @return
 */
@RequestMapping("/upload")
@ResponseBody
public String handleFileUpload(@RequestParam("file")MultipartFile file){
  if(!file.isEmpty()){
    try {
      /*
       * 这段代码执行完毕之后，图片上传到了工程的跟路径；
       * 大家自己扩散下思维，如果我们想把图片上传到 d:/files大家是否能实现呢？
       * 等等;
       * 这里只是简单一个例子,请自行参考，融入到实际中可能需要大家自己做一些思考，比如：
       * 1、文件路径；
       * 2、文件名；
       * 3、文件格式;
       * 4、文件大小的限制;
       */
      BufferedOutputStream out = new BufferedOutputStream(new FileOutputStream(new
File(file.getOriginalFilename())));
      out.write(file.getBytes());
      out.flush();
      out.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
      return"上传失败,"+e.getMessage();
    } catch (IOException e) {
      e.printStackTrace();
      return"上传失败,"+e.getMessage();
    }
    return"上传成功";
  }else{
    return"上传失败，因为文件是空的.";
```

```
    }
  }
}
```

（5）编写**App.java**然后测试

App.java没什么代码，就是Spring Boot的启动配置，具体如下：

```
package com.kfit;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * Hello world!
 *
 */
//其中@SpringBootApplication申明让spring boot自动给程序进行必要的配置，等价于以默认属性使用
@Configuration，@EnableAutoConfiguration和@ComponentScan
@SpringBootApplication
publicclass App {


  publicstaticvoid main(String[] args) {
    SpringApplication.run(App.class, args);
  }
}
```

然后你就可以访问：http://127.0.0.1:8080/file 进行测试了，文件上传的路径是在工程的跟路径下，请刷新查看，其它的请查看代码中的注释进行自行思考。

（6）对上传的文件做一些限制；

对文件做一些限制是有必要的，在App.java进行编码配置：

```
@Bean
  public MultipartConfigElement multipartConfigElement() {
     MultipartConfigFactory factory = new MultipartConfigFactory();
     //// 设置文件大小限制 ,超了，页面会抛出异常信息，这时候就需要进行异常信息的处理了;
     factory.setMaxFileSize("128KB"); //KB,MB
     /// 设置总上传数据总大小
     factory.setMaxRequestSize("256KB");
     //Sets the directory location where files will be stored.
     //factory.setLocation("路径地址");
     returnfactory.createMultipartConfig();
  }
```

（7）多文件上传实现

多文件对于前段页面比较简单，具体代码实现：

在src/resouces/templates/mutifile.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
    xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
  <head>
     <title>Hello World!</title>
  </head>
  <body>
    <form method="POST" enctype="multipart/form-data" action="/batch/upload">
       <p>文件1：<input type="file" name="file" /></p>
       <p>文件2：<input type="file" name="file" /></p>
       <p>文件3：<input type="file" name="file" /></p>
       <p><input type="submit" value="上传" /></p>
    </form>
  </body>
</html>
```

com.kfit.FileUploadController中新增两个方法：

```
/**
   * 多文件具体上传时间，主要是使用了MultipartHttpServletRequest和MultipartFile
```

```
 * @param request
 * @return
 */
@RequestMapping(value="/batch/upload", method= RequestMethod.POST)
public@ResponseBody
String handleFileUpload(HttpServletRequest request){
    List<MultipartFile> files = ((MultipartHttpServletRequest)request).getFiles("file");
    MultipartFile file = null;
    BufferedOutputStream stream = null;
    for (inti =0; i< files.size(); ++i) {
        file = files.get(i);
        if (!file.isEmpty()) {
            try {
                byte[] bytes = file.getBytes();
                stream =
                        new BufferedOutputStream(new FileOutputStream(new File(file.getOriginalFilename())));
                stream.write(bytes);
                stream.close();
            } catch (Exception e) {
                stream =  null;
                return"You failed to upload " + i + " => " + e.getMessage();
            }
        } else {
            return"You failed to upload " + i + " because the file was empty.";
        }
    }
    return"upload successful";

}
```

访问路径：http://127.0.0.1:8080/mutifile 进行测试。

Spring Boot 系列博客】

（ 0 ）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（ 1 ）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（ 2 ）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

...

（ 16 ）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（ 17 ）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

附件下载:

- spring-boot-fileupload.zip (393.6 KB)
- dl.iteye.com/topics/download/41e423e3-0a3c-35ff-96e5-7fc77cc13d9d

1.33 （30）导入时如何定制spring-boot依赖项的版本【转载】【从零开始学Spring Boot】

发表时间: 2016-04-24 关键字: 导入时如何定制spring-boot依赖项的版本【转载】

此文章转载地址：http://www.tuicool.com/articles/RJJvMj3 请注重作者的版权。

spring-boot通过maven的依赖管理为我们写好了很多依赖项及其版本，我们可拿来使用。spring-boot文档介绍了两种使用方法，一是继承，二是导入。

通过<parent>继承：

```
<project>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.1.9.RELEASE</version>
  </parent>
</project>
```

或者在<dependencyManagement>中导入：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-dependencies</artifactId>
```

```
        <version>1.1.9.RELEASE</version>
        <type>pom</type>
        <scope>import</scope>
    </dependency>
    </dependencies>
</dependencyManagement>
```

此外，在其 文档 里说到，继承时可简单地通过属性定制依赖项版本。比如，改为使用较新的spring-4.1.6.RELEASE版本：

```
<properties>
    <spring.version>4.1.6.RELEASE<spring.version>
</properties>
```

不过，此法只对继承有效，导入无效。以下摘自其文档说明：

```
This only works if your Maven project inherits (directly or indirectly) from spring-boot-dependencies. If you have added spring-boot-dependencie
```

导入时有没有较简单的方法呢？我们可先继承后导入！

1、先建一个过渡性工程，继承后定制依赖项的版本。

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-dependencies</artifactId>
    <version>1.1.9.RELEASE</version>
  </parent>
  <groupId>mycomp</groupId>
  <artifactId>myproject-spring-boot-bom</artifactId>
  <version>1.1.9</version>

  <packaging>pom</packaging>

  <properties>
    <spring.version>4.1.6.RELEASE</spring.version>
  </properties>
</project>
```

2、然后导入到自己的工程里。

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>mycomp</groupId>
  <artifactId>myproject</artifactId>
```

```
    <version>0.0.1-SNAPSHOT</version>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>mycomp</groupId>
                <artifactId>myproject-spring-boot-bom</artifactId>
                <version>1.1.9</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
</project>
```

这样，虽然多建了一个过渡性工程，但定制依赖项版本同继承时一样简单。

Spring Boot 系列博客】

（ 0 ）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

...

（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

1.34 （31）Spring Boot导入XML配置【从零开始学Spring Boot】

发表时间: 2016-04-25 关键字: 从零开始学Spring Boot, Spring Boot导入XML配置, Spring Boot

【来也匆匆，去也匆匆，在此留下您的脚印吧,转发点赞评论；

您的认可是我最大的动力，感谢您的支持】

Spring Boot理念就是零配置编程，但是如果绝对需要使用XML的配置，我们建议您仍旧从一个@Configuration类开始，你可以使用@ImportResouce注解加载XML配置文件，我拿一个例子来进行讲解：

这个例子的大体步骤如下：

（1）新建一个工程;

（2）在App.java类编写HelloService2;

（3）在App.java类无法扫描的包下编写HelloService;

（4）编写application-bean.xml注入HelloService;

（5）编写ConfigClass注入配置文件application-bean.xml;

（6）编写App.java启动类进行测试；

（7）其它说明

（1）新建一个工程;

我们在前几节的例子已经写到hello2了，我们取一个新的名称为spring-boot-hello3,这里没有什么难点，不过多介绍，还有难处的可以查看之前的例子，当然这里加入spring-boot相应的web支持；

不懂的参考:

spring boot起步之Hello World【从零开始学Spring Boot】:

http://412887952-qq-com.iteye.com/blog/2291500

（2）在App.java类编写HelloService2;

首先我们这里有几个包：com.kfit,org.kfit,我们这里打算把App.java启动类放到com.kfit中，根据Spring Boot扫描（根包到子包的原则），我们把HelloService2写在Spring Boot可以扫描的位置，HelloService写在Spring Boot无法扫描到的位置，那么我们使用配置文件bean的方式进行引入，具体代码如下：

com.kfit.service.HelloService2：

```
package com.kfit.service;

import org.springframework.stereotype.Service;

@Service
publicclass HelloService2 {

    /**
     * 启动的时候观察控制台是否打印此信息;
     */
    public HelloService2() {
        System.out.println("HelloService2.HelloService2()");
        System.out.println("HelloService2.HelloService2()");
        System.out.println("HelloService2.HelloService2()");
    }
}
```

（3）在App.java类无法扫描的包下编写HelloService;

注意这个类是写在Spring Boot无法自动扫描的位置，正常启动之后，如果引入HelloService的话肯定会报异常的，因为它根本没有被注入成功，具体代码如下：

org.kfit.service.HelloService：

```
package org.kfit.service;

import org.springframework.stereotype.Service;
```

```
@Service
publicclass HelloService {

    /**
     * 启动的时候观察控制台是否打印此信息;
     */
    public HelloService() {
        System.out.println("HelloService.HelloService()");
        System.out.println("org.kfit.service.HelloService.HelloService()");
        System.out.println("HelloService.HelloService()");
    }

}
```

（4）编写application-bean.xml注入HelloService;

在src/main/resouces下编写配置文件application-bean.xml文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/
schema/beans/spring-beans.xsd">

    <!-- 注入spring boot无法扫描到的bean. -->
    <bean id="helloService" class="org.kfit.service.HelloService"></bean>

</beans>
```

（5）编写ConfigClass注入配置文件application-bean.xml;

在com.kfit.config包下编写类ConfigClass，这个确保能被Spring Boot可以扫描到，不然一切都付之东流了，具体代码如下：

com.kfit.config.ConfigClass：

```
package com.kfit.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.ImportResource;

/**
 * classpath路径：locations={"classpath:application-bean1.xml","classpath:application-bean2.xml"}
 * file路径： locations = {"file:d:/test/application-bean1.xml"};
 */
@Configuration
@ImportResource(locations={"classpath:application-bean.xml"})
//@ImportResource(locations={"file:d:/test/application-bean1.xml"})
publicclass ConfigClass {

}
```

（6）编写App.java启动类进行测试；

这个类Spring Boot正常的启动代码：

com.kfit.App：

```
package com.kfit;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
```

```
*
*
* 大家也许会看到有些demo使用了3个注解： @Configuration；
*
* @EnableAutoConfiguration
* @ComponentScan
*
*  其实：@SpringBootApplication申明让spring boot自动给程序进行必要的配置，
*
* 等价于以默认属性使用@Configuration，
*  @EnableAutoConfiguration和@ComponentScan
*
* 所以大家不要被一些文档误导了，让自己很迷茫了，希望本文章对您有所启发；
*
* @author Angel(QQ:412887952)
* @version v.0.1
*/
@SpringBootApplication
public class App {
    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}
```

在App.java 右键 Run As Java Application观察控制台输出可以看到：

```
HelloService2.HelloService2()
HelloService2.HelloService2()
HelloService2.HelloService2()
HelloService.HelloService()
org.kfit.service.HelloService.HelloService()
HelloService.HelloService()
```

说明我们引入编写的代码生效了，如果你不相信的话，可以把ConfigClass的注解去掉，测试下，是不是打印信息就少了HelloService的部分，是的话就对了。

（7）其它说明

ImportResouce有两种常用的引入方式：classpath和file，具体查看如下的例子：

classpath路径：locations={"classpath:application-bean1.xml",
"classpath:application-bean2.xml"
}

file路径：

locations = {"file:d:/test/application-bean1.xml"};

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

...

（ 15 ）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2292362

（ 16 ）Spring Boot使用Druid（ 编程注入 ）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（ 17 ）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

附件下载:

- spring-boot-hello3.zip (17.3 KB)
- dl.iteye.com/topics/download/e3ad4f4f-4866-3ec1-acef-24bb8c2ff948

1.35 （ 32 ）Spring Boot使用@SpringBootApplication注解，从零开始学Spring Boot

发表时间: 2016-04-25 关键字: Spring Boot使用@SpringBootApplication注解, 从零开始学Spring Boot

【来也匆匆，去也匆匆，在此留下您的脚印吧,转发点赞评论】

如果看了我之前的文章，这个节你就可以忽略了，这个是针对一些刚入门的选手存在的困惑进行写的一篇文章。

很多Spring Boot开发者总是使用 @Configuration ， @EnableAutoConfiguration 和 @ComponentScan 注解他们的main类。由于这些注解被如此频繁地一块使用（特别是你遵循以上最佳实践时），Spring Boot提供一个方便的 @SpringBootApplication 选择。该 @SpringBootApplication 注解等价于以默认属性使用 @Configuration ， @EnableAutoConfiguration 和 @ComponentScan 。

这是官方进行解析的，我个人自己第一次接触的时候也是有这个困惑的，希望此篇文章能解答在研究Spring Boot困惑的人。

Spring Boot 系列博客】

（ 0 ）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496
（ 1 ）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500
（ 2 ）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
...
（ 15 ）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2292362
（ 16 ）Spring Boot使用Druid（ 编程注入 ）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376
（ 17 ）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

1.36 （ 33 ）Spring Boot 监控和管理生产环境【从零开始学Spring Boot】

发表时间: 2016-04-26 关键字: Spring Boot 监控和管理生产环境, 从零开始学Spring Boot

【本文章是否对你有用以及是否有好的建议，请留言】

spring-boot-actuator模块提供了一个监控和管理生产环境的模块，可以使用http、jmx、ssh、telnet等拉管理和监控应用。审计（Auditing）、

健康（health）、数据采集（metrics gathering）会自动加入到应用里面。

首先，写一个最基本的spring boot项目。

基于Maven的项目添加 ‘starter’ 依赖：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

以下是所有监控描述：

HTTP方法	路径	描述	鉴权
GET	/autoconfig	查看自动配置的使用情况，该报告展示所有auto-configuration候选者及它们被应用或未被应用的原因	true
GET	/configprops	显示一个所有@ConfigurationProperties的整理列表	true

GET	/beans	显示一个应用中所有Spring Beans的完整列表	true
GET	/dump	打印线程栈	true
GET	/env	查看所有环境变量	true
GET	/env/{name}	查看具体变量值	true
GET	/health	查看应用健康指标	false
GET	/info	查看应用信息	false
GET	/mappings	查看所有url映射	true
GET	/metrics	查看应用基本指标	true
GET	/metrics/{name}	查看具体指标	true
POST	/shutdown	允许应用以优雅的方式关闭（默认情况下不启用）	true
GET	/trace	查看基本追踪信息	true

health

比如：http://localhost:8080/health

你可以得到结果

```
{
status: "UP",
diskSpace: {
status: "UP",
total: 107374174208,
free: 14877962240,
threshold: 10485760
```

```
}
}
```

可以检查的其他一些情况的健康信息。下面的HealthIndicators会被Spring Boot自动配置：

DiskSpaceHealthIndicator 低磁盘空间检测

DataSourceHealthIndicator 检查是否能从DataSource获取连接

MongoHealthIndicator 检查一个Mongo数据库是否可用（up）

RabbitHealthIndicator 检查一个Rabbit服务器是否可用（up）

RedisHealthIndicator 检查一个Redis服务器是否可用（up）

SolrHealthIndicator 检查一个Solr服务器是否可用（up）

自定义当然也可以，你可以注册实现了HealthIndicator接口的Spring beans，Health响应需要包含一个status和可选的用于展示的详情。

```
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.stereotype.Component;

@Component
public class MyHealth implements HealthIndicator {

```

```
    @Override
    public Health health() {
        int errorCode = check(); // perform some specific health check
        if (errorCode != 0) {
        return Health.down().withDetail("Error Code", errorCode).build();
        }
        return Health.up().build();
    }
}
```

trace

访问http://localhost:8080/trace 可以看到结果，默认为最新的一些HTTP请求

info

当执行 http://localhost:8080/info 的时候，结果什么没有

但是，在application.properties加入一些配置

```
info.app.name=ecs
info.app.version=1.0.0
```

```
info.build.artifactId=@project.artifactId@

info.build.version=@project.version@
```

执行/info就可以看到有些信息了。

/info 是用来在构建的时候，自动扩展属性的。对于Maven项目，可以通过 @..@ 占位符引用Maven的’ project properties’ 。

更多的细节和探索，需要自己看看源码和spring boot的官方文档。

Spring Boot 系列博客】

（ 0 ）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496

（ 1 ）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500

（ 2 ）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
…

（ 15 ）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2292362

（ 16 ）Spring Boot使用Druid（ 编程注入 ）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376

（ 17 ）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388

更多查看博客：http://412887952-qq-com.iteye.com/blog

1.37 （ 34 ）Spring Boot的启动器Starter详解【从零开始学Spring Boot】

发表时间: 2016-04-29 关键字: Spring Boot的启动器Starter详解, 从零开始学Spring Boot, spring boot

Spring Boot应用启动器基本的一共有N（现知道的是44）种：具体如下：

1）spring-boot-starter

这是Spring Boot的核心启动器，包含了自动配置、日志和YAML。

2）spring-boot-starter-actuator

帮助监控和管理应用。

3）spring-boot-starter-amqp

通过spring-rabbit来支持AMQP协议（Advanced Message Queuing Protocol）。

4）spring-boot-starter-aop

支持面向方面的编程即AOP，包括spring-aop和AspectJ。

5）spring-boot-starter-artemis

通过Apache Artemis支持JMS的API（Java Message Service API）。

6）spring-boot-starter-batch

支持Spring Batch，包括HSQLDB数据库。

7）spring-boot-starter-cache

支持Spring的Cache抽象。

8）spring-boot-starter-cloud-connectors

支持Spring Cloud Connectors，简化了在像Cloud Foundry或Heroku这样的云平台上连接服务。

9）spring-boot-starter-data-elasticsearch

支持ElasticSearch搜索和分析引擎，包括spring-data-elasticsearch。

10）spring-boot-starter-data-gemfire

支持GemFire分布式数据存储，包括spring-data-gemfire。

11）spring-boot-starter-data-jpa

支持JPA（Java Persistence API），包括spring-data-jpa、spring-orm、Hibernate。

12）spring-boot-starter-data-mongodb

支持MongoDB数据，包括spring-data-mongodb。

13）spring-boot-starter-data-rest

通过spring-data-rest-webmvc，支持通过REST暴露Spring Data数据仓库。

14）spring-boot-starter-data-solr

支持Apache Solr搜索平台，包括spring-data-solr。

15）spring-boot-starter-freemarker

支持FreeMarker模板引擎。

16）spring-boot-starter-groovy-templates

支持Groovy模板引擎。

17）spring-boot-starter-hateoas

通过spring-hateoas支持基于HATEOAS的RESTful Web服务。

18）spring-boot-starter-hornetq

通过HornetQ支持JMS。

19）spring-boot-starter-integration

支持通用的spring-integration模块。

20）spring-boot-starter-jdbc

支持JDBC数据库。

21）spring-boot-starter-jersey

支持Jersey RESTful Web服务框架。

22）spring-boot-starter-jta-atomikos

通过Atomikos支持JTA分布式事务处理。

23）spring-boot-starter-jta-bitronix

通过Bitronix支持JTA分布式事务处理。

24）spring-boot-starter-mail

支持javax.mail模块。

25）spring-boot-starter-mobile

支持spring-mobile。

26）spring-boot-starter-mustache

支持Mustache模板引擎。

27）spring-boot-starter-redis

支持Redis键值存储数据库，包括spring-redis。

28）spring-boot-starter-security

支持spring-security。

29）spring-boot-starter-social-facebook

支持spring-social-facebook

30）spring-boot-starter-social-linkedin

支持pring-social-linkedin

31）spring-boot-starter-social-twitter

支持pring-social-twitter

32）spring-boot-starter-test

支持常规的测试依赖，包括JUnit、Hamcrest、Mockito以及spring-test模块。

33）spring-boot-starter-thymeleaf

支持Thymeleaf模板引擎，包括与Spring的集成。

34）spring-boot-starter-velocity

支持Velocity模板引擎。

35）spring-boot-starter-web

S支持全栈式Web开发，包括Tomcat和spring-webmvc。

36）spring-boot-starter-websocket

支持WebSocket开发。

37）spring-boot-starter-ws

支持Spring Web Services。

Spring Boot应用启动器面向生产环境的还有2种，具体如下：

1）spring-boot-starter-actuator

增加了面向产品上线相关的功能，比如测量和监控。

2）spring-boot-starter-remote-shell

增加了远程ssh shell的支持。

最后，Spring Boot应用启动器还有一些替换技术的启动器，具体如下：

1）spring-boot-starter-jetty

引入了Jetty HTTP引擎（用于替换Tomcat）。

2）spring-boot-starter-log4j

支持Log4J日志框架。

3）spring-boot-starter-logging

引入了Spring Boot默认的日志框架Logback。

4）spring-boot-starter-tomcat

引入了Spring Boot默认的HTTP引擎Tomcat。

5）spring-boot-starter-undertow

引入了Undertow HTTP引擎（用于替换Tomcat）。

1.38 （ 35 ）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】

发表时间: 2016-04-29 关键字: Spring Boot集成Redis实现缓存机制, 从零开始学Spring Boot, spring boot

【本文章是否对你有用以及是否有好的建议，请留言】

本文章牵涉到的技术点比较多：Spring Data JPA、Redis、Spring MVC,Spirng Cache，所以在看这篇文章的时候，需要对以上这些技术点有一定的了解或者也可以先看看这篇文章，针对文章中实际的技术点在进一步了解（注意，您需要自己下载Redis Server到您的本地，所以确保您本地的Redis可用，这里还使用了MySql数据库，当然你也可以内存数据库进行测试）。这篇文章会提供对应的Eclipse代码示例，具体大体的分如下几个步骤：

（1）新建Java Maven Project;

（2）在pom.xml中添加相应的依赖包；

（3）编写Spring Boot启动类；

（4）配置application.properties;

（5）编写RedisCacheConfig配置类；

（6）编写DemoInfo测试实体类；

（7）编写DemoInfoRepository持久化类；

（8）编写DemoInfoService类；

（9）编写DemoInfoController类；

（10）测试代码是否正常运行了

（11）自定义缓存key;

（1）新建Java Maven Project;

这个步骤就不细说，新建一个spring-boot-redis Java maven project;

（2）在pom.xml中添加相应的依赖包；

在Maven中添加相应的依赖包,主要有：spring boot 父节点依赖；spring boot web支持；缓存服务spring-context-support；添加redis支持；JPA操作数据库；mysql 数据库驱动，具体pom.xml文件如下：

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.kfit</groupId>
 <artifactId>spring-boot-redis</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>

 <name>spring-boot-redis</name>
 <url>http://maven.apache.org</url>

 <properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <!-- 配置JDK编译版本. -->
  <java.version>1.8</java.version>
 </properties>

<!-- spring boot 父节点依赖,
    引入这个之后相关的引入就不需要添加version配置，
     spring boot会自动选择最合适的版本进行添加。
 -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.3.RELEASE</version>
  </parent>


<dependencies>

    <dependency>
      <groupId>junit</groupId>
```

```
    <artifactId>junit</artifactId>
    <scope>test</scope>
</dependency>

<!-- spring boot web支持：mvc,aop... -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<!--
   包含支持UI模版（Velocity，FreeMarker，JasperReports），
   邮件服务，
   脚本服务(JRuby)，
   缓存Cache（EHCache），
   任务计划Scheduling（uartz）。
-->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>
</dependency>

<!-- 添加redis支持-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-redis</artifactId>
</dependency>

<!-- JPA操作数据库. -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<!-- mysql 数据库驱动. -->
<dependency>
  <groupId>mysql</groupId>
```

```
        <artifactId>mysql-connector-java</artifactId>
    </dependency>

    <!-- 单元测试. -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>

 </dependencies>
</project>
```

上面是完整的pom.xml文件，每个里面都进行了简单的注释。

（ 3 ）编写Spring Boot启动类（ com.kfit.App ）；

```
package com.kfit;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
/**
 * Spring Boot启动类;
 *
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */

@SpringBootApplication
public class App {
    /**
     * -javaagent:.\lib\springloaded-1.2.4.RELEASE.jar -noverify
     * @param args
     */
    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
```

```
}
```

（4）配置application.properties;

这里主要是配置两个资源，第一就是数据库基本信息；第二就是redis配置； 第三就是JPA的配置；

Src/main/resouces/application.properties：

```
#########################################################
###datasource  配置MySQL数据源；
#########################################################
spring.datasource.url = jdbc:mysql://localhost:3306/test
spring.datasource.username = root
spring.datasource.password = root
spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.datasource.max-active=20
spring.datasource.max-idle=8
spring.datasource.min-idle=8
spring.datasource.initial-size=10



#########################################################
###REDIS (RedisProperties) redis基本配置；
#########################################################
# database name
spring.redis.database=0
# server host1
spring.redis.host=127.0.0.1
# server password
#spring.redis.password=
#connection port
spring.redis.port=6379
# pool settings ...
spring.redis.pool.max-idle=8
spring.redis.pool.min-idle=0
spring.redis.pool.max-active=8
```

```
spring.redis.pool.max-wait=-1
# name of Redis server
#spring.redis.sentinel.master=
# comma-separated list of host:port pairs
#spring.redis.sentinel.nodes=

##########################################################
### Java Persistence Api 自动进行建表
##########################################################
# Specify the DBMS
spring.jpa.database = MYSQL
# Show or not log for each sql query
spring.jpa.show-sql = true
# Hibernate ddl auto (create, create-drop, update)
spring.jpa.hibernate.ddl-auto = update
# Naming strategy
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy
# stripped before adding them to the entity manager)
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

（5）编写RedisCacheConfig配置类；

缓存主要有几个要实现的类：其一就是CacheManager缓存管理器；其二就是具体操作实现类；其三就是CacheManager工厂类（这个可以使用配置文件配置的进行注入，也可以通过编码的方式进行实现）；其四就是缓存key生产策略（当然Spring自带生成策略，但是在Redis客户端进行查看的话是系列化的key,对于我们肉眼来说就是感觉是乱码了，这里我们先使用自带的缓存策略）。

com.kfit.config/RedisCacheConfig：

```
package com.kfit.config;

import org.springframework.cache.CacheManager;
import org.springframework.cache.annotation.CachingConfigurerSupport;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.data.redis.cache.RedisCacheManager;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;

/**
 * redis 缓存配置;
 *
 * 注意：RedisCacheConfig这里也可以不用继承：CachingConfigurerSupport，也就是直接一个普通的Class就好了；
 *
 * 这里主要我们之后要重新实现 key的生成策略，只要这里修改KeyGenerator，其它位置不用修改就生效了。
 *
 * 普通使用普通类的方式的话，那么在使用@Cacheable的时候还需要指定KeyGenerator的名称;这样编码的时候比较麻烦。
 *
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Configuration
@EnableCaching//启用缓存，这个注解很重要；
publicclass RedisCacheConfig extends CachingConfigurerSupport {


    /**
     * 缓存管理器.
     * @param redisTemplate
     * @return
     */
    @Bean
    public CacheManager cacheManager(RedisTemplate<?,?> redisTemplate) {
        CacheManager cacheManager = new RedisCacheManager(redisTemplate);
        returncacheManager;
    }


    /**
     * redis模板操作类,类似于jdbcTemplate的一个类;
     *
```

```
     * 虽然CacheManager也能获取到Cache对象，但是操作起来没有那么灵活；
     *
     * 这里在扩展下：RedisTemplate这个类不见得很好操作，我们可以在进行扩展一个我们
     *
     * 自己的缓存类，比如：RedisStorage类;
     *
     * @param factory : 通过Spring进行注入，参数在application.properties进行配置；
     * @return
     */
    @Bean
    public RedisTemplate<String, String> redisTemplate(RedisConnectionFactory factory) {
        RedisTemplate<String, String> redisTemplate = new RedisTemplate<String, String>();
        redisTemplate.setConnectionFactory(factory);

        //key序列化方式;（不然会出现乱码;）,但是如果方法上有Long等非String类型的话，会报类型转换错误；
        //所以在没有自己定义key生成策略的时候，以下这个代码建议不要这么写，可以不配置或者自己实现
ObjectRedisSerializer
        //或者JdkSerializationRedisSerializer序列化方式;
//        RedisSerializer<String> redisSerializer = new StringRedisSerializer();//Long类型不可以会出现异常信息;
//        redisTemplate.setKeySerializer(redisSerializer);
//        redisTemplate.setHashKeySerializer(redisSerializer);

        return redisTemplate;
    }

}
```

在以上代码有很详细的注释，在这里还是在简单的提下：

RedisCacheConfig这里也可以不用继承：CachingConfigurerSupport，也就是直接一个普通的Class就好了；这里主要我们之后要重新实现 key的生成策略，只要这里修改KeyGenerator，其它位置不用修改就生效了。普通使用普通类的方式的话，那么在使用@Cacheable的时候还需要指定KeyGenerator的名称;这样编码的时候比较麻烦。

（6）编写DemoInfo测试实体类；

编写一个测试实体类：com.kfit.bean.DemoInfo：

```
package com.kfit.bean;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
/**
 * 测试实体类，这个随便;
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Entity
public class DemoInfo  implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id@GeneratedValue
    private long id;
    private String name;
    private String pwd;
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPwd() {
        return pwd;
    }
```

```
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }

    @Override
    public String toString() {
        return "DemoInfo [id=" + id + ", name=" + name + ", pwd=" + pwd + "]";
    }
}
```

（7）编写DemoInfoRepository持久化类；

DemoInfoRepository使用Spirng Data JPA实现：

com.kfit.repository.DemoInfoRepository：

```
package com.kfit.repository;

import org.springframework.data.repository.CrudRepository;

import com.kfit.bean.DemoInfo;

/**
 * DemoInfo持久化类
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
public interface DemoInfoRepository extends CrudRepository<DemoInfo,Long> {

}
```

（8）编写DemoInfoService类；

编写DemoInfoService，这里有两个技术方面，第一就是使用Spring @Cacheable注解方式和RedisTemplate对象进行操作，具体代码如下：

com.kfit.service.DemoInfoService:

```
package com.kfit.service;

import com.kfit.bean.DemoInfo;

/**
 * demoInfo 服务接口
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
publicinterface DemoInfoService {

    public DemoInfo findById(longid);

    publicvoid deleteFromCache(longid);

    void test();
}
```

com.kfit.service.impl.DemoInfoServiceImpl:

```
package com.kfit.service.impl;

import javax.annotation.Resource;

import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.ValueOperations;
import org.springframework.stereotype.Service;

import com.kfit.bean.DemoInfo;
import com.kfit.repository.DemoInfoRepository;
import com.kfit.service.DemoInfoService;

/**
```

```
 *
 *DemoInfo数据处理类
 *
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Service
publicclass DemoInfoServiceImpl implements DemoInfoService {

    @Resource
    private DemoInfoRepository demoInfoRepository;

    @Resource
    private RedisTemplate<String,String> redisTemplate;

    @Override
    publicvoid test(){
        ValueOperations<String,String> valueOperations = redisTemplate.opsForValue();
        valueOperations.set("mykey4", "random1="+Math.random());
        System.out.println(valueOperations.get("mykey4"));
    }

    //keyGenerator="myKeyGenerator"
    @Cacheable(value="demoInfo") //缓存,这里没有指定key.
    @Override
    public DemoInfo findById(longid) {
        System.err.println("DemoInfoServiceImpl.findById()==========从数据库中进行获取的....id="+id);
        returndemoInfoRepository.findOne(id);
    }

    @CacheEvict(value="demoInfo")
    @Override
    publicvoid deleteFromCache(longid) {
        System.out.println("DemoInfoServiceImpl.delete().从缓存中删除.");
    }

}
```

（9）编写DemoInfoController类；

```
package com.kfit.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import com.kfit.bean.DemoInfo;
import com.kfit.service.DemoInfoService;

/**
 * 测试类.
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Controller
publicclass DemoInfoController {

    @Autowired
     DemoInfoService demoInfoService;


    @RequestMapping("/test")
    public@ResponseBody String test(){
        DemoInfo loaded = demoInfoService.findById(1);
            System.out.println("loaded="+loaded);
            DemoInfo cached = demoInfoService.findById(1);
        System.out.println("cached="+cached);
        loaded = demoInfoService.findById(2);
        System.out.println("loaded2="+loaded);
        return"ok";
    }
```

```
    @RequestMapping("/delete")
    public@ResponseBody String delete(longid){
        demoInfoService.deleteFromCache(id);
        return"ok";
    }

    @RequestMapping("/test1")
    public@ResponseBody String test1(){
        demoInfoService.test();
        System.out.println("DemoInfoController.test1()");
        return"ok";
    }

}
```

（10）测试代码是否正常运行了

启动应用程序，访问地址：http://127.0.0.1:8080/test

查看控制台可以查看：

```
DemoInfoServiceImpl.findById()=========从数据库中进行获取的....id=1
loaded=DemoInfo [id=1, name=张三, pwd=123456]
cached=DemoInfo [id=1, name=张三, pwd=123456]
DemoInfoServiceImpl.findById()=========从数据库中进行获取的....id=2
loaded2=DemoInfo [id=2, name=张三, pwd=123456]
```

如果你看到以上的打印信息的话，那么说明缓存成功了。

访问地址：http://127.0.0.1:8080/test1

```
random1=0.9985031320746356
DemoInfoController.test1()
```

二次访问：http://127.0.0.1:8080/test

```
loaded=DemoInfo [id=1, name=张三, pwd=123456]
cached=DemoInfo [id=1, name=张三, pwd=123456]
loaded2=DemoInfo [id=2, name=张三, pwd=123456]
```

这时候所有的数据都是执行缓存的。

这时候执行删除动作：http://127.0.0.1:8080/delete?id=1

然后在访问：http://127.0.0.1:8080/test

```
DemoInfoServiceImpl.findById()=========从数据库中进行获取的....id=1
loaded=DemoInfo [id=1, name=张三, pwd=123456]
cached=DemoInfo [id=1, name=张三, pwd=123456]
loaded2=DemoInfo [id=2, name=张三, pwd=123456]
```

（11）自定义缓存key;

在com.kfit.config.RedisCacheConfig类中重写CachingConfigurerSupport中的keyGenerator ,具体实现代码如下：

```
/**
    * 自定义key.
    * 此方法将会根据类名+方法名+所有参数的值生成唯一的一个key,即使@Cacheable中的value属性一样，key也会不一样。
    */
   @Override
   public KeyGenerator keyGenerator() {
      System.out.println("RedisCacheConfig.keyGenerator()");
      return new KeyGenerator() {
         @Override
         public Object generate(Object o, Method method, Object... objects) {
            // This will generate a unique key of the class name, the method name
            //and all method parameters appended.
```

```
            StringBuilder sb = new StringBuilder();
            sb.append(o.getClass().getName());
            sb.append(method.getName());
            for (Object obj : objects) {
                sb.append(obj.toString());
            }
            System.out.println("keyGenerator=" + sb.toString());
            returnsb.toString();
        }
    };
}
```

这时候在redis的客户端查看key的话还是序列化的肉眼看到就是乱码了，那么我改变key的序列方式，这个很简单，redis底层已经有具体的实现类了，我们只需要配置下：

```
//key序列化方式;（不然会出现乱码;）,但是如果方法上有Long等非String类型的话，会报类型转换错误；
//所以在没有自己定义key生成策略的时候，以下这个代码建议不要这么写，可以不配置或者自己实现
ObjectRedisSerializer
//或者JdkSerializationRedisSerializer序列化方式;
    RedisSerializer<String> redisSerializer = new StringRedisSerializer();//Long类型不可以会出现异常信息;
    redisTemplate.setKeySerializer(redisSerializer);
    redisTemplate.setHashKeySerializer(redisSerializer);
```

综上以上分析:RedisCacheConfig类的方法调整为：

```
package com.kfit.config;

import java.lang.reflect.Method;

import org.springframework.cache.CacheManager;
import org.springframework.cache.annotation.CachingConfigurerSupport;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.cache.interceptor.KeyGenerator;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.cache.RedisCacheManager;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.RedisSerializer;
```

```
import org.springframework.data.redis.serializer.StringRedisSerializer;

/**
 * redis 缓存配置;
 *
 * 注意：RedisCacheConfig这里也可以不用继承：CachingConfigurerSupport，也就是直接一个普通的Class就好了；
 *
 * 这里主要我们之后要重新实现 key的生成策略，只要这里修改KeyGenerator，其它位置不用修改就生效了。
 *
 * 普通使用普通类的方式的话，那么在使用@Cacheable的时候还需要指定KeyGenerator的名称;这样编码的时候比较麻烦。
 *
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Configuration
@EnableCaching//启用缓存，这个注解很重要；
publicclass RedisCacheConfig extends CachingConfigurerSupport {

    /**
     * 缓存管理器.
     * @param redisTemplate
     * @return
     */
    @Bean
    public CacheManager cacheManager(RedisTemplate<?,?> redisTemplate) {
        CacheManager cacheManager = new RedisCacheManager(redisTemplate);
        returncacheManager;
    }


    /**
     * RedisTemplate缓存操作类,类似于jdbcTemplate的一个类;
     *
     * 虽然CacheManager也能获取到Cache对象，但是操作起来没有那么灵活；
     *
     * 这里在扩展下：RedisTemplate这个类不见得很好操作，我们可以在进行扩展一个我们
```

```
    *
    * 自己的缓存类，比如：RedisStorage类;
    *
    * @param factory : 通过Spring进行注入，参数在application.properties进行配置;
    * @return
    */
    @Bean
    public RedisTemplate<String, String> redisTemplate(RedisConnectionFactory factory) {
        RedisTemplate<String, String> redisTemplate = new RedisTemplate<String, String>();
        redisTemplate.setConnectionFactory(factory);

        //key序列化方式;（不然会出现乱码;）,但是如果方法上有Long等非String类型的话，会报类型转换错误；
        //所以在没有自己定义key生成策略的时候，以下这个代码建议不要这么写，可以不配置或者自己实现
ObjectRedisSerializer
        //或者JdkSerializationRedisSerializer序列化方式;
        RedisSerializer<String> redisSerializer = new StringRedisSerializer();//Long类型不可以会出现异常信息;
        redisTemplate.setKeySerializer(redisSerializer);
        redisTemplate.setHashKeySerializer(redisSerializer);

        return redisTemplate;
    }

    /**
    * 自定义key.
    * 此方法将会根据类名+方法名+所有参数的值生成唯一的一个key,即使@Cacheable中的value属性一样，key也会不
一样。
    */
    @Override
    public KeyGenerator keyGenerator() {
        System.out.println("RedisCacheConfig.keyGenerator()");
        return new KeyGenerator() {
            @Override
            public Object generate(Object o, Method method, Object... objects) {
                // This will generate a unique key of the class name, the method name
                //and all method parameters appended.
                StringBuilder sb = new StringBuilder();
                sb.append(o.getClass().getName());
```

```
            sb.append(method.getName());
            for (Object obj : objects) {
                sb.append(obj.toString());
            }
            System.out.println("keyGenerator=" + sb.toString());
            returnsb.toString();
        }
    };
}
```

```
}
```

这时候在访问地址：http://127.0.0.1:8080/test

这时候看到的Key就是：com.kfit.service.impl.DemoInfoServiceImplfindById1

在控制台打印信息是：

```
（1）keyGenerator=com.kfit.service.impl.DemoInfoServiceImplfindById1
（2）DemoInfoServiceImpl.findById()=========从数据库中进行获取的....id=1
（3）keyGenerator=com.kfit.service.impl.DemoInfoServiceImplfindById1
（4）loaded=DemoInfo [id=1, name=张三, pwd=123456]
（5）keyGenerator=com.kfit.service.impl.DemoInfoServiceImplfindById1
（6）cached=DemoInfo [id=1, name=张三, pwd=123456]
（7）keyGenerator=com.kfit.service.impl.DemoInfoServiceImplfindById2
（8）keyGenerator=com.kfit.service.impl.DemoInfoServiceImplfindById2
（10）DemoInfoServiceImpl.findById()=========从数据库中进行获取的....id=2
（11）loaded2=DemoInfo [id=2, name=张三, pwd=123456]
```

其中@Cacheable,@CacheEvict下节进行简单的介绍，这节的东西实在是太多了，到这里就打住吧，剩下的就需要靠你们自己进行扩展了。

Spring Boot 系列博客】

（0）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
...

（15）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2292362

（16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388
.....
（35）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/blog

附件下载:

- spring-boot-redis.zip (424.5 KB)
- dl.iteye.com/topics/download/2506c03c-cbcc-3f92-a4f1-8a45b6fb3c80

1.39 （ 36 ）Spring Boot Cache理论篇【从零开始学Spring Boot】

发表时间: 2016-04-30 关键字: Spring Boot Cache理论篇, 从零开始学Spring Boot, spring boot

Spring Boot Cache理论篇

在上一篇中我们介绍了Spring Boot集成Redis的实战例子，里面使用到了Spring Cache，那么什么是Spring Cache呢，本章将会做一个理论介绍，至于实战的话，可以在上一章节进行实战测试。

（ 35 ）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2294942

Spring 3.1 引入了激动人心的基于注释（annotation）的缓存（cache）技术，它本质上不是一个具体的缓存实现方案（例如EHCache 或者 OSCache、Redis等），而是一个对缓存使用的抽象，通过在既有代码中添加少量它定义的各种 annotation，即能够达到缓存方法的返回对象的效果。

Spring 的缓存技术还具备相当的灵活性，不仅能够使用 SpEL（Spring Expression Language）来定义缓存的 key 和各种 condition，还提供开箱即用的缓存临时存储方案，也支持和主流的专业缓存例如 EHCache 集成。

其特点总结如下：

· 通过少量的配置 annotation 注释即可使得既有代码支持缓存

· 支持开箱即用 Out-Of-The-Box，即不用安装和部署额外第三方组件即可使用缓存

· 支持 Spring Express Language，能使用对象的任何属性或者方法来定义缓存的 key 和 condition

· 支持 AspectJ，并通过其实现任何方法的缓存支持

· 支持自定义 key 和自定义缓存管理者，具有相当的灵活性和扩展性

我们以前如何自己实现缓存的呢？

以前我们是在service层通过先判断缓存中是否存在缓存数据，如果存在直接返回数据，如果不存在从数据库中进行查询数据的方式，那么我们会发现这种方式的代码耦合型太高了。如果你之前的代码就是这么设计的，那么你可以考虑使用Spring Cache优化你的代码，你的代码将会变得优雅很多。

实现方式主要在方法上加上注解，可以注解的注解类有：

Cacheable 支持如下几个参数：

value：缓存位置名称，不能为空，如果使用EHCache，就是ehcache.xml中声明的cache的name

key：缓存的key，默认为空，既表示使用方法的参数类型及参数值作为key，支持SpEL

condition：触发条件，只有满足条件的情况才会加入缓存，默认为空，既表示全部都加入缓存，支持SpEL

以下有一个例子：

```
1. //将缓存保存进andCache，并使用参数中的userId加上一个字符串(这里使用方法名称)作为缓存的key
2. @Cacheable(value="andCache",key="#userId + 'findById'")
3. public SystemUser findById(String userId) {
4.     SystemUser user = (SystemUser) dao.findById(SystemUser.class, userId);
5.     return user ;
6. }
7. //将缓存保存进andCache，并当参数userId的长度小于32时才保存进缓存，默认使用参数值及类型作为缓存的key
8. @Cacheable(value="andCache",condition="#userId.length < 32")
9. public boolean isReserved(String userId) {
10.     System.out.println("hello andCache"+userId);
11.     return false;
12.}
```

@CacheEvict 支持如下几个参数：

value：缓存位置名称，不能为空，同上

key：缓存的key，默认为空，同上

condition：触发条件，只有满足条件的情况才会清除缓存，默认为空，支持SpEL

allEntries：true表示清除value中的全部缓存，默认为false

以下是一个小例子：

```
1. //清除掉指定key的缓存
2. @CacheEvict(value="andCache",key="#user.userId + 'findById'")
3. public void modifyUserRole(SystemUser user) {
4.       System.out.println("hello andCache delete"+user.getUserId());
5. }
6.
7. //清除掉全部缓存
8. @CacheEvict(value="andCache",allEntries=true)
9. public void setReservedUsers() {
10.   System.out.println("hello andCache deleteall");
11.}
```

一般来说，我们的更新操作只需要刷新缓存中某一个值，所以定义缓存的key值的方式就很重要，最好是能够唯一，因为这样可以准确的清除掉特定的缓存，而不会影响到其它缓存值，

比如我这里针对用户的操作，使用(userId+方法名称)的方式设定key值，当然，你也可以找到更适合自己的方式去设定。

@CachePut 注释，这个注释可以确保方法被执行，同时方法的返回值也被记录到缓存中，实现缓存与数据库的同步更新，理解为update语句。

更具体的可以参考：http://hanqunfeng.iteye.com/blog/1158824 因为本系列教材并不是要介绍Spring 的一些特定的东西，只是在上一章节以及下一章节会用到，所以在这里进行提及下。

Spring Boot 系列博客】

（ 0 ）前言【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291496

（ 1 ）spring boot起步之Hello World【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2291500

（ 2 ）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
...

（ 15 ）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2292362

（ 16 ）Spring Boot使用Druid（ 编程注入 ）【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blogs/2292376

（ 17 ）Spring Boot普通类调用bean【从零开始学Spring Boot】：
http://412887952-qq-com.iteye.com/blog/2292388
.....
（ 35 ）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/blog

1.40 （37）Spring Boot集成EHCache实现缓存机制【从零开始学Spring Boot】

发表时间: 2016-04-30 关键字: Spring Boot集成EHCache实现缓存机制, 从零开始学Spring Boot, spring boot

【本文章是否对你有用以及是否有好的建议，请留言】

写后感：博主写这么一系列文章也不容易啊，请评论支持下。

如果看过我之前（35）的文章这一篇的文章就会很简单，没有什么挑战性了。

那么我们先说说这一篇文章我们都会学到的技术点：Spring Data JPA,Spring Boot 使用Mysql,Spring MVC,EHCache,Spring Cache等（其中@Cacheable请看上一节的理论知识），具体分如下几个步骤：

（1）新建Maven Java Project

（2）在pom.xml中加入依赖包

（3）编写Spring Boot启动类；

（4）配置application.properties;

（5）编写缓存配置类以及ehcache.xml配置文件；

（6）编写DemoInfo实体类进行测试；

（7）编写持久类DemoInfoRepository;

（8）编写处理类DemoInfoService;

（9）编写DemoInfoController测试类；

（10）运行测试；

以上就是具体的步骤了，那么接下来我们一起按照这个步骤来进行实现吧。

（1）新建Maven Java Project

新建一个工程名为spring-boot-ehcache的maven java project。

（2）在pom.xml中加入依赖包

在pom.xml文件中加入相应的依赖包，Spring Boot父节点依赖包；spring boot web支持；缓存依赖spring-context-support；集成ehcache需要的依赖；JPA操作数据库；mysql 数据库驱动，具体pom.xml文件：

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.kfit</groupId>
  <artifactId>spring-boot-ehcache</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>spring-boot-ehcache</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <!-- 配置JDK编译版本. -->
    <java.version>1.8</java.version>
  </properties>


  <!-- spring boot 父节点依赖,
    引入这个之后相关的引入就不需要添加version配置，
     spring boot会自动选择最合适的版本进行添加。
  -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
```

```
    <version>1.3.3.RELEASE</version>
</parent>

<dependencies>
    <!-- 单元测试. -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <scope>test</scope>
    </dependency>

    <!-- spring boot web支持：mvc,aop... -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!--
        包含支持UI模版（Velocity，FreeMarker，JasperReports），
        邮件服务，
        脚本服务(JRuby)，
        缓存Cache（EHCache），
        任务计划Scheduling（uartz）。
     -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
    </dependency>

    <!-- 集成ehcache需要的依赖-->
    <dependency>
        <groupId>net.sf.ehcache</groupId>
        <artifactId>ehcache</artifactId>
    </dependency>

    <!-- JPA操作数据库. -->
    <dependency>
```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
      </dependency>

      <!-- mysql 数据库驱动. -->
      <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
      </dependency>

      <!-- Spring boot单元测试. -->
      <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-test</artifactId>
          <scope>test</scope>
      </dependency>
    </dependencies>
</project>
```

（3）编写Spring Boot启动类（com.kfit.App.java）；

```
package com.kfit;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 *
 *
 * @SpringBootApplication申明让spring boot自动给程序进行必要的配置，
 *
@SpringBootApplication
等待于：
@Configuration
@EnableAutoConfiguration
```

```
@ComponentScan
 *
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@SpringBootApplication
publicclass App {
    publicstaticvoid main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}
```

（4）配置application.properties;

在application.properties中主要配置数据库连接和JPA的基本配置,具体如下：

Src/main/resouces/application.properties：

```
#########################################################
###datasource ,mysql数据库连接配置
#########################################################
spring.datasource.url = jdbc:mysql://localhost:3306/test
spring.datasource.username = root
spring.datasource.password = root
spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.datasource.max-active=20
spring.datasource.max-idle=8
spring.datasource.min-idle=8
spring.datasource.initial-size=10



#########################################################
### Java Persistence Api ，JPA自动建表操作配置
#########################################################
# Specify the DBMS
```

```
spring.jpa.database = MYSQL
# Show or not log for each sql query
spring.jpa.show-sql = true
# Hibernate ddl auto (create, create-drop, update)
spring.jpa.hibernate.ddl-auto = update
# Naming strategy
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy
# stripped before adding them to the entity manager)
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

（5）编写缓存配置类以及ehcache.xml配置文件：

这个类主要是注册缓存管理对象EhCacheCacheManager、缓存工厂对象EhCacheManagerFactoryBean，具体代码如下：

EhCacheManagerFactoryBean：

```
package com.kfit.config;

import org.springframework.cache.annotation.EnableCaching;
import org.springframework.cache.ehcache.EhCacheCacheManager;
import org.springframework.cache.ehcache.EhCacheManagerFactoryBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;

/**
 * 缓存配置.
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Configuration
@EnableCaching//标注启动缓存.
publicclass CacheConfiguration {

    /**
```

```
     * ehcache 主要的管理器
     * @param bean
     * @return
     */
    @Bean
    public EhCacheCacheManager ehCacheCacheManager(EhCacheManagerFactoryBean bean){
        System.out.println("CacheConfiguration.ehCacheCacheManager()");
        return new EhCacheCacheManager(bean.getObject());
    }

    /*
     * 据shared与否的设置,
     * Spring分别通过CacheManager.create()
     * 或new CacheManager()方式来创建一个ehcache基地.
     *
     * 也说是说通过这个来设置cache的基地是这里的Spring独用,还是跟别的(如hibernate的Ehcache共享)
     *
     */
    @Bean
    public EhCacheManagerFactoryBean ehCacheManagerFactoryBean(){
        System.out.println("CacheConfiguration.ehCacheManagerFactoryBean()");
        EhCacheManagerFactoryBean cacheManagerFactoryBean = new EhCacheManagerFactoryBean ();
        cacheManagerFactoryBean.setConfigLocation (new ClassPathResource("conf/ehcache.xml"));
        cacheManagerFactoryBean.setShared(true);
        return cacheManagerFactoryBean;
    }

}
```

在src/main/resouces/conf下编写ehcache.xml配置文件，当然这个文件你可以放在其它目录下：

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd"
    updateCheck="false">
```

```
<!--
diskStore：为缓存路径，ehcache分为内存和磁盘两级，此属性定义磁盘的缓存位置。参数解释如下：
  user.home – 用户主目录
  user.dir  – 用户当前工作目录
  java.io.tmpdir – 默认临时文件路径

 -->
<diskStore path="java.io.tmpdir/Tmp_EhCache" />

<!--
  defaultCache：默认缓存策略，当ehcache找不到定义的缓存时，则使用这个缓存策略。只能定义一个。
 -->

 <!--
  name:缓存名称。
  maxElementsInMemory:缓存最大数目
  maxElementsOnDisk：硬盘最大缓存个数。
  eternal:对象是否永久有效，一但设置了，timeout将不起作用。
  overflowToDisk:是否保存到磁盘，当系统当机时
  timeToIdleSeconds:设置对象在失效前的允许闲置时间（单位：秒）。仅当eternal=false对象不是永久有效时使用，可选属性，默认值是0，也就是可闲置时间无穷大。
  timeToLiveSeconds:设置对象在失效前允许存活时间（单位：秒）。最大时间介于创建时间和失效时间之间。仅当eternal=false对象不是永久有效时使用，默认是0.，也就是对象存活时间无穷大。
  diskPersistent：是否缓存虚拟机重启期数据 Whether the disk store persists between restarts of the Virtual Machine. The default value is false.
  diskSpoolBufferSizeMB：这个参数设置DiskStore（磁盘缓存）的缓存区大小。默认是30MB。每个Cache都应该有自己的一个缓冲区。
  diskExpiryThreadIntervalSeconds：磁盘失效线程运行时间间隔，默认是120秒。
  memoryStoreEvictionPolicy：当达到maxElementsInMemory限制时，Ehcache将会根据指定的策略去清理内存。默认策略是LRU（最近最少使用）。你可以设置为FIFO（先进先出）或是LFU（较少使用）。
   clearOnFlush：内存数量最大时是否清除。
    memoryStoreEvictionPolicy:可选策略有：LRU（最近最少使用，默认策略）、FIFO（先进先出）、LFU（最少访问次数）。
      FIFO，first in first out，这个是大家最熟的，先进先出。
      LFU， Less Frequently Used，就是上面例子中使用的策略，直白一点就是讲一直以来最少被使用的。如上面所讲，缓存的元素有一个hit属性，hit值最小的将会被清出缓存。
```

```
        LRU，Least Recently Used，最近最少使用的，缓存的元素有一个时间戳，当缓存容量满了，而又需要腾出地方来缓存新的元素的时候，那么现有缓存元素中时间戳离当前时间最远的元素将被清出缓存。
    -->
    <defaultCache
      eternal="false"
      maxElementsInMemory="1000"
      overflowToDisk="false"
      diskPersistent="false"
      timeToIdleSeconds="0"
      timeToLiveSeconds="600"
      memoryStoreEvictionPolicy="LRU" />

    <cache
      name="demo"
      eternal="false"
      maxElementsInMemory="100"
      overflowToDisk="false"
      diskPersistent="false"
      timeToIdleSeconds="0"
      timeToLiveSeconds="300"
      memoryStoreEvictionPolicy="LRU" />

</ehcache>
```

（6）编写DemoInfo实体类进行测试；

在com.kfit.bean下编写DemoInfo实体类进行缓存测试：

```
package com.kfit.bean;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

/**
 * 测试实体类.
 * @author Angel(QQ:412887952)
```

```
* @version v.0.1
*/
@Entity
publicclass DemoInfo {
   @Id@GeneratedValue
   privatelongid;//主键.
   private String name;//名称;
   private String pwd;//密码;
   privateintstate;
   publiclong getId() {
      returnid;
   }
   publicvoid setId(longid) {
      this.id = id;
   }
   public String getName() {
      returnname;
   }
   publicvoid setName(String name) {
      this.name = name;
   }
   public String getPwd() {
      returnpwd;
   }
   publicvoid setPwd(String pwd) {
      this.pwd = pwd;
   }
   publicint getState() {
      returnstate;
   }
   publicvoid setState(intstate) {
      this.state = state;
   }
   @Override
   public String toString() {
      return"DemoInfo [id=" + id + ", name=" + name + ", pwd=" + pwd + ", state=" + state + "]";
   }
```

```
}
```

（7）编写持久类DemoInfoRepository;

编写持久类DemoInfoRepository：

com.kfit.repository.DemoInfoRepository：

```
package com.kfit.repository;

import org.springframework.data.repository.CrudRepository;

import com.kfit.bean.DemoInfo;

/**
 * 操作数据库.
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
public interface DemoInfoRepository extends CrudRepository<DemoInfo,Long>{

}
```

（8）编写处理类DemoInfoService;

编写增删改查的方法，在这几个方法中都使用注解缓存，进行缓存的创建以及删除，修改等操作：

com.kfit.service.DemoInfoService：

```
package com.kfit.service;

import com.kfit.bean.DemoInfo;

import javassist.NotFoundException;

public interface DemoInfoService {

   void delete(Long id);
```

```
    DemoInfo update(DemoInfo updated) throws NotFoundException;

    DemoInfo findById(Long id);

    DemoInfo save(DemoInfo demoInfo);

}
```

com.kfit.service.impl.DemoInfoServiceImpl：

```
package com.kfit.service.impl;

import javax.annotation.Resource;

import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;

import com.kfit.bean.DemoInfo;
import com.kfit.repository.DemoInfoRepository;
import com.kfit.service.DemoInfoService;

import javassist.NotFoundException;

@Service
publicclass DemoInfoServiceImpl implements DemoInfoService {

    //这里的单引号不能少，否则会报错，被识别是一个对象;
    publicstaticfinal String CACHE_KEY = "'demoInfo'";

     @Resource
     private DemoInfoRepository demoInfoRepository;

    /**
     * value属性表示使用哪个缓存策略，缓存策略在ehcache.xml
```

```
 */
publicstaticfinal String DEMO_CACHE_NAME = "demo";

/**
 * 保存数据.
 * @param demoInfo
 */
@CacheEvict(value=DEMO_CACHE_NAME,key=CACHE_KEY)
@Override
public DemoInfo save(DemoInfo demoInfo){
   returndemoInfoRepository.save(demoInfo);
}

/**
 * 查询数据.
 * @param id
 * @return
 */
@Cacheable(value=DEMO_CACHE_NAME,key="'demoInfo_'+#id")
@Override
public DemoInfo findById(Long id){
   System.err.println("没有走缓存！"+id);
   returndemoInfoRepository.findOne(id);
}

/**
 * http://www.mincoder.com/article/2096.shtml:
 *
 * 修改数据.
 *
 * 在支持Spring Cache的环境下，对于使用@Cacheable标注的方法，Spring在每次执行前都会检查Cache中是否存在相同key的缓存元素，如果存在就不再执行该方法，而是直接从缓存中获取结果进行返回，否则才会执行并将返回结果存入指定的缓存中。@CachePut也可以声明一个方法支持缓存功能。与@Cacheable不同的是使用@CachePut标注的方法在执行前不会去检查缓存中是否存在之前执行过的结果，而是每次都会执行该方法，并将执行结果以键值对的形式存入指定的缓存中。

   @CachePut也可以标注在类上和方法上。使用@CachePut时我们可以指定的属性跟@Cacheable是一样的。
```

```
     *
     * @param updated
     * @return
     *
     * @throws NotFoundException
     */
    @CachePut(value = DEMO_CACHE_NAME,key = "'demoInfo_'+#updated.getId()")
    //@CacheEvict(value = DEMO_CACHE_NAME,key = "'demoInfo_'+#updated.getId()")//这是清除缓存.
    @Override
    public DemoInfo update(DemoInfo updated) throws NotFoundException{
        DemoInfo demoInfo = demoInfoRepository.findOne(updated.getId());
        if(demoInfo == null){
            thrownew NotFoundException("No find");
        }
        demoInfo.setName(updated.getName());
        demoInfo.setPwd(updated.getPwd());
        returndemoInfo;
    }


    /**
     * 删除数据.
     * @param id
     */
    @CacheEvict(value = DEMO_CACHE_NAME,key = "'demoInfo_'+#id")//这是清除缓存.
    @Override
    publicvoid delete(Long id){
        demoInfoRepository.delete(id);
    }


}
```

（9）编写DemoInfoController测试类；

编写一个rest进行测试：

com.kfit.controller.DemoInfoController：

```
package com.kfit.controller;

import javax.annotation.Resource;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.kfit.bean.DemoInfo;
import com.kfit.service.DemoInfoService;

import javassist.NotFoundException;

@RestController
publicclass DemoInfoController {

    @Resource
    private DemoInfoService demoInfoService;

    @RequestMapping("/test")
    public String test(){

      //存入两条数据.
       DemoInfo demoInfo = new DemoInfo();
       demoInfo.setName("张三");
       demoInfo.setPwd("123456");
       DemoInfo demoInfo2 = demoInfoService.save(demoInfo);

       //不走缓存.
       System.out.println(demoInfoService.findById(demoInfo2.getId()));
       //走缓存.
       System.out.println(demoInfoService.findById(demoInfo2.getId()));



       demoInfo = new DemoInfo();
       demoInfo.setName("李四");
```

```
        demoInfo.setPwd("123456");
        DemoInfo demoInfo3 = demoInfoService.save(demoInfo);

        //不走缓存.
        System.out.println(demoInfoService.findById(demoInfo3.getId()));
        //走缓存.
        System.out.println(demoInfoService.findById(demoInfo3.getId()));

        System.out.println("============修改数据====================");
        //修改数据.
        DemoInfo updated = new DemoInfo();
        updated.setName("李四-updated");
        updated.setPwd("123456");
        updated.setId(demoInfo3.getId());
        try {
            System.out.println(demoInfoService.update(updated));
        } catch (NotFoundException e) {
            e.printStackTrace();
        }

        //不走缓存.
        System.out.println(demoInfoService.findById(updated.getId()));

        return"ok";
    }


}
```

（10）运行测试；

运行App.java进行测试，访问：http://127.0.0.1:8080/test 进行测试，主要是观察控制台的打印信息。

```
Hibernate: insert into demo_info (name, pwd, state) values (?, ?, ?)
没有走缓存！52
DemoInfo [id=52, name=张三, pwd=123456, state=0]
DemoInfo [id=52, name=张三, pwd=123456, state=0]
```

```
Hibernate: insert into demo_info (name, pwd, state) values (?, ?, ?)
没有走缓存！53
DemoInfo [id=53, name=李四, pwd=123456, state=0]
DemoInfo [id=53, name=李四, pwd=123456, state=0]
==========修改数据==========
DemoInfo [id=53, name=李四-updated, pwd=123456, state=0]
DemoInfo [id=53, name=李四-updated, pwd=123456, state=0]
C:\Users\ADMINI~1.ANG\AppData\Local\Temp\
Hibernate: insert into demo_info (name, pwd, state) values (?, ?, ?)
没有走缓存！54
DemoInfo [id=54, name=张三, pwd=123456, state=0]
DemoInfo [id=54, name=张三, pwd=123456, state=0]
Hibernate: insert into demo_info (name, pwd, state) values (?, ?, ?)
没有走缓存！55
DemoInfo [id=55, name=李四, pwd=123456, state=0]
DemoInfo [id=55, name=李四, pwd=123456, state=0]
==========修改数据==========
DemoInfo [id=55, name=李四-updated, pwd=123456, state=0]
```

```
DemoInfo [id=55, name=李四-updated, pwd=123456, state=0]
```

好了本篇文章就写到这里吧，打烊休息了，实在是动不了！

Spring Boot 系列博客】

0 ）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（ 1 ）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

（15）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2292362

16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

.....

（35）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/

附件下载:

- spring-boot-ehcache.zip (31.1 KB)
- dl.iteye.com/topics/download/75623e2d-cc80-3463-8850-d170ecfac048

1.41 （38）Spring Boot分布式Session状态保存Redis【从零开始学Spring Boot】

发表时间: 2016-05-01 关键字: 从零开始学Spring Boot, Spring Boot分布式Session状态保存Redis

【本文章是否对你有用以及是否有好的建议，请留言】

在使用spring boot做负载均衡的时候，多个app之间的session要保持一致，这样负载到不同的app时候，在一个app登录之后，而访问到另外一台服务器的时候，session丢失。

常规的解决方案都是使用：如apache使用mod_jk.conf，使用Memcached进行共享。

在开发spring boot app的时候可以借助 spring session 和redis或者ehcache，用外置的redis或者ehcache来存储session的状态,这里使用redis进行介绍，ehcache实现是一样的。

增加相关依赖

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-redis</artifactId>
    </dependency>

  <dependency>
        <groupId>org.springframework.session</groupId>
        <artifactId>spring-session-data-redis</artifactId>
</dependency>
```

RedisSessionConfig.java

```
package com.wisely.base;

import org.springframework.context.annotation.Configuration;

import org.springframework.session.data.redis.config.annotation.web.http.EnableRedisHttpSession;

@Configuration

@EnableRedisHttpSession

public class RedisSessionConfig {

}
```

如果需要添加失效时间可以使用以下的写法：

@EnableRedisHttpSession(maxInactiveIntervalInSeconds = 60) //1分钟失效

相关配置修改

在application.properties修改redis配置信息（请自行安装redis），请根据实际修改。如：

```
spring.redis.host=127.0.0.1
```

所有实体类实现Serializable接口

```
public class UserInfo implements Serializable
```

查看效果

这时候登录系统在不同的app之间跳转的时候，session都是一致了，redis上可以看到：

总结

使用这些代码之后 ，无论你使用nginx或者apache，都无须在关心多个app之间的session一致的问题了。

注意事项

（1）redis版本号需要是2.8以上否则会抛异常：ERR Unsupported CONFIG parameter: notify-keyspace-events；

（2）RedisSessionConfig需要放在App.java启动类可以扫描的位置；

【Spring Boot 系列博客】

0）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

（15）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2292362

16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

.....

（35）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/

附件下载:

- spring-boot-session.zip (17.2 KB)
- dl.iteye.com/topics/download/38b13e2d-b422-3b48-bcbb-73c96c5a718a

1.42 （ 39.1 ） Spring Boot Shiro权限管理【从零开始学Spring Boot】

发表时间: 2016-05-21 关键字: Spring Boot Shiro权限管理, 从零开始学Spring Boot

距上一个章节过了二个星期了，最近时间也是比较紧，一直没有时间可以写博客，今天难得有点时间，就说说Spring Boot如何集成Shiro吧。这个章节会比较复杂，牵涉到的技术点也会比较，如果没有Spring Boot基础的还是建议先学习基础，不然此博客看起来会比较费劲。好了废话不都说了，还是开始我们的Spring Boot Shiro之旅吧。还是依照之前的风格，我们分解下我们的目标：

(1). Shiro简单介绍

(2). 集成Shiro核心分析

(3). 无Shiro的Spring Boot

(4). 集成Shiro 进行用户授权

(5). Shiro缓存

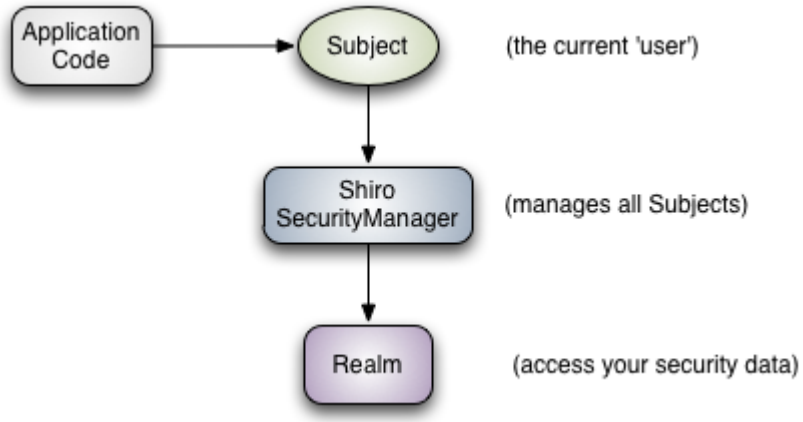
(6). Shiro记住密码

(7). Shiro验证码

(1). Shiro简单介绍

Shiro是Apache下的一个开源项目，我们称之为Apache Shiro。它是一个很易用与Java项目的的安全框架，提供了认证、授权、加密、会话管理，与 Spring Security 一样都是做一个权限的安全框架，但是与Spring Security 相比，在于 Shiro 使用了比较简单易懂易于使用的授权方式。

Apache Shiro 的三大核心组件



这里写图片描述 <!--[endif]-->

- Subject 当前用户操作

- SecurityManager 用于管理所有的Subject

- Realms 用于进行权限信息的验证，也是我们需要自己实现的。

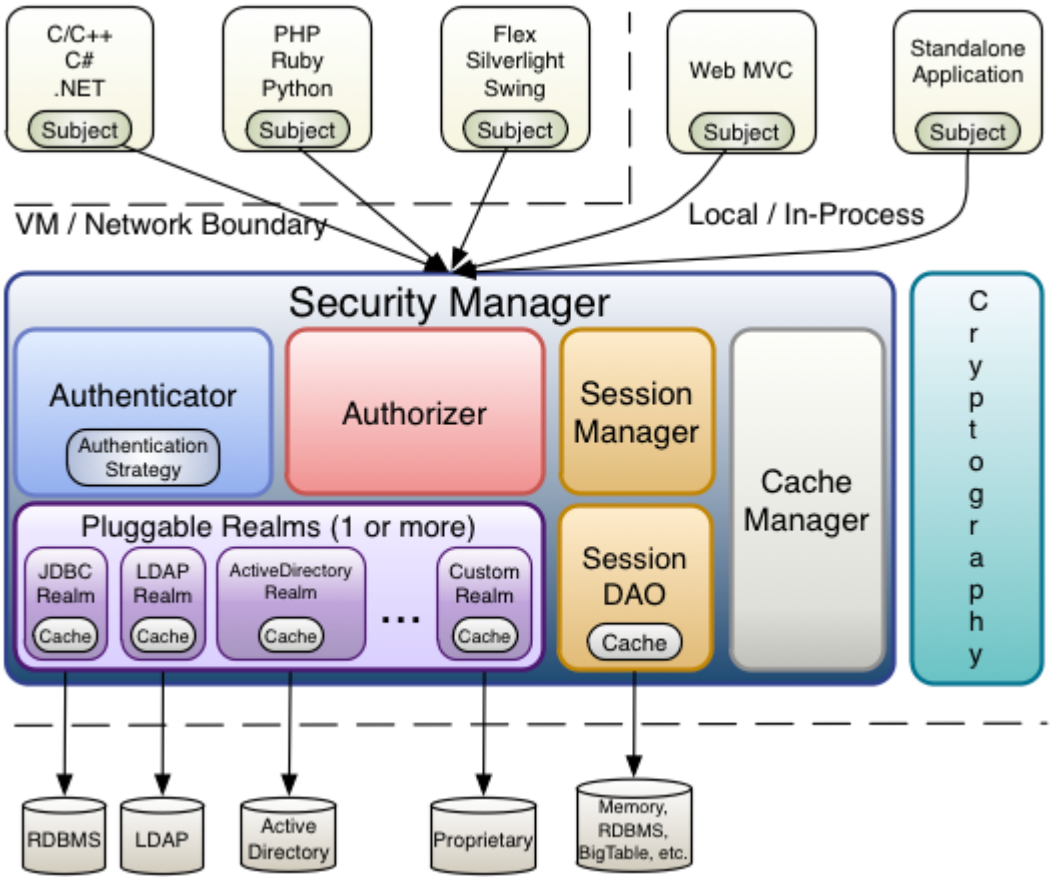
我们需要实现Realms的Authentication 和 Authorization。其中 Authentication 是用来验证用户身份，Authorization 是授权访问控制，用于对用户进行的操作授权，证明该用户是否允许进行当前操作，如访问某个链接，某个资源文件等。

Apache Shiro 核心通过 Filter 来实现，就好像SpringMvc 通过DispachServlet 来主控制一样。
既然是使用 Filter 一般也就能猜到，是通过URL规则来进行过滤和权限校验，所以我们需要定义一系列关于URL的规则和访问权限。

另外我们可以通过Shiro 提供的会话管理来获取Session中的信息。Shiro 也提供了缓存支持，使用 CacheManager 来管理。

官方网站：http://shiro.apache.org/

完整架构图：



Shiro是很强大的一个安全框架，这里只是抛装引玉下，还有很多的需要大家自己去学习Shiro。

(2). 集成Shiro核心分析

集成Shiro的话，我们需要知道Shiro框架大概的一些管理对象。

第一：ShiroFilterFactory，Shiro过滤器工程类，具体的实现类是：ShiroFilterFactoryBean，此实现类是依赖于SecurityManager安全管理器。

第二：SecurityManager,Shiro的安全管理，主要是身份认证的管理，缓存管理，cookie管理，所以在实际开发中我们主要是和SecurityManager进行打交道的，ShiroFilterFactory主要配置好了Filter就可以了。当然SecurityManager并进行身份认证缓存的实现，我们需要进行对应的编码然后进行注入到安全管理器中。

第三：Realm,用于身份信息权限信息的验证。

第四：其它的就是缓存管理，记住登录之类的，这些大部分都是需要自己进行简单的实现，然后注入到SecurityManager让Shiro的安全管理器进行管理就好了。

(3). 无Shiro的Spring Boot

我们先编写一个无Shiro的简单的框架，在这个框架中我们可以访问到index,login,userInfo,userInfoAdd。

这个步骤对于有Spring Boot基础的就应该很简单了，在这里简单的介绍下：

(a) 新建一个maven java project,取名为spring-boot-shiro1

(b) 在pom.xml中引入基本依赖，在这里还没有引入shiro等的依赖：

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>com.kfit</groupId>
<artifactId>spring-boot-shiro1</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>spring-boot-shiro1</name>
<url>http://maven.apache.org</url>

<properties>
```

```
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <java.version>1.8</java.version>
 </properties>

  <!--
    spring boot 父节点依赖,
    引入这个之后相关的引入就不需要添加version配置，
    spring boot会自动选择最合适的版本进行添加。
   -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.3.RELEASE</version>
  </parent>

 <dependencies>

     <!-- spring boot web支持：mvc,aop... -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```

```
    <!-- thmleaf模板依赖. -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>

 </dependencies>
</project>
```

在这里只引入了Spirng Boot的web依赖以及对thymleaf模板引擎的依赖。

(c) 编写网页文件:

index.html,login.html,userInfo.html,userInfoAdd.html

这个文件存在在src/main/resouces/templates, 这几个文件中都是简单的代码，只有登录界面中有账号和密码：

index.html ：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Insert title here</title>
</head>
<body>
  <h3>index</h3>
</body>
</html>
```

login.html :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Insert title here</title>
</head>
<body>
        错误信息：<h4 th:text="${msg}"></h4>
    <form action="" method="post">
        <p>账号：<input type="text" name="username" value="admin"/></p>
        <p>密码：<input type="text" name="password" value="123456"/></p>
        <p><input type="submit" value="登录"/></p>
    </form>
</body>
</html>
```

其它的页面都是简单的一个标签而已：

```
<h3>用户查询界面</h3>
```

```
<h3>用户添加界面</h3>
```

请自行编码。

(d)编写启动类

编写启动类com.kfit.App.java ：

```
package com.kfit;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
* 启动类.
* @author Angel(QQ:412887952)
* @version v.0.1
*/
@SpringBootApplication
publicclass App {

  /**
   * 参数里VM参数设置为：
-javaagent:.\lib\springloaded-1.2.4.RELEASE.jar -noverify
   * @param args
   */
  publicstaticvoid main(String[] args) {
    SpringApplication.run(App.class, args);
  }
```

```
}
```

这样类似的代码我们已经介绍很多了，没有什么可以过多的介绍了，

这时候我们右键run as 运行App.java类访问index,login页面，会报Error Page，因为我们还没编写Controller处理类呢。

(e)编写HomeController类

在com.kfit.root.controller新建HomeController类：

```
package com.kfit.root.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
publicclass HomeController {

  @RequestMapping({"/","/index"})
  public String index(){
    return"/index";
  }

  @RequestMapping(value="/login",method=RequestMethod.GET)
  public String login(){
    return"login";
```

```
    }

}
```

在这里我们并没有把UserInfo对应的处理也在页面进行编码了，因为之后我们创建了UserInfo之后，打算新建一个UserInfoController进行处理，所以这里就没有相应的userInfo的跳转处理。

这时候我们在运行我们的程序就应该可以访问index,login页面了。

到此这个小节就结束了，现在我们的程序还有问题，就是index页面在没有登录的时候，就可以进行访问了，我们希望是如果直接访问index页面，如果没有登录的话，直接跳转到login进行登录。

那么下一小节我们将会介绍如何在当前代码中集成shiro。

如果在一篇博客中介绍整个的集成过程的话，可能会比较乱，另外就是也没法介绍的那么细致，所以博主决定还是分开进行讲解，这样大家会比较好理解。那么下一篇博客大家再见吧，我要先去健健身，回来接着写。

附件下载:

- spring-boot-shiro1.zip (15.2 KB)
- dl.iteye.com/topics/download/b6169768-fe72-33e5-be5a-4701075f85e8

1.43 （ 39.2 ）. Spring Boot Shiro权限管理【从零开始学Spring Boot】

发表时间: 2016-05-21 关键字: 从零开始学Spring Boot, Spring Boot Shiro权限管理

(4). 集成Shiro 进行用户授权

在看此小节前，您可能需要先看：

http://412887952-qq-com.iteye.com/blog/2299732

紧着上一小节，在上一节我们编写了简单的一个小程序，但是我们会发现我们随便访问index,login 以及任何一个界面，无需登录也可以进行访问，但是这不是我们所想要的，我们想要的是希望在用户没有登录的情况下，跳转login页面进行登录。那么这个时候Shiro就闪亮登场了。

集成shiro大概分这么一个步骤：

(a) pom.xml中添加Shiro依赖；

(b) 注入Shiro Factory和SecurityManager。

(c) 身份认证

(d) 权限控制

(a) pom.xml中添加Shiro依赖；

要使用Shiro进行权限控制，那么很明显的就需要添加对Shiro的依赖包，在pom.xml中加入如下配置：

```
<!-- shiro spring. -->
    <dependency>
      <groupId>org.apache.shiro</groupId>
      <artifactId>shiro-spring</artifactId>
      <version>1.2.2</version>
    </dependency>
```

(b) 注入Shiro Factory和SecurityManager。

在Spring中注入类都是使用配置文件的方式，在Spring Boot中是使用注解的方式，那么应该如何进行实现呢？

我们在上一节说过，Shiro几个核心的类，第一就是ShiroFilterFactory,第二就是SecurityManager，那么最简单的配置就是注入这两个类就ok了，那么如何注入呢？看如下代码：

新建类 com.kfit.config.shiro.ShiroConfiguration ：

```
package com.kfit.config.shiro;

import java.util.LinkedHashMap;
import java.util.Map;

import org.apache.shiro.mgt.SecurityManager;
import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * Shiro 配置
 *
Apache Shiro 核心通过 Filter 来实现，就好像SpringMvc 通过DispachServlet 来主控制一样。
既然是使用 Filter 一般也就能猜到，是通过URL规则来进行过滤和权限校验，所以我们需要定义一系列关于URL的规则和访问权限。
 *
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Configuration
publicclass ShiroConfiguration {



   /**
    * ShiroFilterFactoryBean 处理拦截资源文件问题。
```

```
 * 注意：单独一个ShiroFilterFactoryBean配置是或报错的，以为在
 * 初始化ShiroFilterFactoryBean的时候需要注入：SecurityManager
 *
   Filter Chain定义说明
  1、一个URL可以配置多个Filter，使用逗号分隔
  2、当设置多个过滤器时，全部验证通过，才视为通过
  3、部分过滤器可指定参数，如perms，roles
 *
 */
@Bean
public ShiroFilterFactoryBean shirFilter(SecurityManager securityManager){
  System.out.println("ShiroConfiguration.shirFilter()");
  ShiroFilterFactoryBean shiroFilterFactoryBean  = new ShiroFilterFactoryBean();

   // 必须设置 SecurityManager
  shiroFilterFactoryBean.setSecurityManager(securityManager);



  //拦截器.
  Map<String,String> filterChainDefinitionMap = new LinkedHashMap<String,String>();

  //配置退出过滤器,其中的具体的退出代码Shiro已经替我们实现了
  filterChainDefinitionMap.put("/logout", "logout");

  //<!-- 过滤链定义，从上向下顺序执行，一般将 /**放在最为下边 -->:这是一个坑呢，一不小心代码就不好使了;
   //<!-- authc:所有url都必须认证通过才可以访问; anon:所有url都都可以匿名访问-->
  filterChainDefinitionMap.put("/**", "authc");

  // 如果不设置默认会自动寻找Web工程根目录下的"/login.jsp"页面
   shiroFilterFactoryBean.setLoginUrl("/login");
   // 登录成功后要跳转的链接
   shiroFilterFactoryBean.setSuccessUrl("/index");
   //未授权界面;
   shiroFilterFactoryBean.setUnauthorizedUrl("/403");

  shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);
```

```
        return shiroFilterFactoryBean;
    }



    @Bean
    public SecurityManager securityManager(){
        DefaultWebSecurityManager securityManager =  new DefaultWebSecurityManager();
        return securityManager;
    }



}
```

这里说下：ShiroFilterFactory中已经由Shiro官方实现的过滤器：

Shiro内置的FilterChain

Filter Name	Class
anon	org.apache.shiro.web.filter.authc.AnonymousFilter
authc	org.apache.shiro.web.filter.authc.FormAuthenticationFilter
authcBasic	org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter
perms	org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter
port	org.apache.shiro.web.filter.authz.PortFilter
rest	org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter
roles	org.apache.shiro.web.filter.authz.RolesAuthorizationFilter
ssl	org.apache.shiro.web.filter.authz.SslFilter
user	org.apache.shiro.web.filter.authc.UserFilter

anon:所有url都都可以匿名访问;

authc: 需要认证才能进行访问;

user:配置记住我或认证通过可以访问；

这几个是我们会用到的，在这里说明下，其它的请自行查询文档进行学习。

这时候我们运行程序，访问/index页面我们会发现自动跳转到了login页面，当然这个时候输入账号和密码是无法进行访问的。下面这才是重点：任何身份认证，如何权限控制。

(c) 身份认证

在认证、授权内部实现机制中都有提到，最终处理都将交给Real进行处理。因为在Shiro中，最终是通过Realm来获取应用程序中的用户、角色及权限信息的。通常情况下，在Realm中会直接从我们的数据源中获取Shiro需要的验证信息。可以说，Realm是专用于安全框架的DAO.

认证实现

Shiro的认证过程最终会交由Realm执行，这时会调用Realm的getAuthenticationInfo(token)方法。

该方法主要执行以下操作:

1、检查提交的进行认证的令牌信息

2、根据令牌信息从数据源(通常为数据库)中获取用户信息

3、对用户信息进行匹配验证。

4、验证通过将返回一个封装了用户信息的AuthenticationInfo实例。

5、验证失败则抛出AuthenticationException异常信息。

而在我们的应用程序中要做的就是自定义一个Realm类，继承AuthorizingRealm抽象类，重载doGetAuthenticationInfo ()，重写获取用户信息的方法。

既然需要进行身份权限控制，那么少不了创建用户实体类，权限实体类。

在权限管理系统中，有这么几个角色很重要，这个要是不清楚的话，那么就很难理解，我们为什么这么编码了。第一是用户表：在用户表中保存了用户的基本信息，账号、密码、姓名，性别等；第二是：权限表（资源+控制权限）：这个表中主要是保存了用户的URL地址，权限信息；第三就是角色表：在这个表重要保存了系统存在的角色；第四就是关联表：用户-角色管理表（用户在系统中都有什么角色，比如admin，vip等），角色-权

限关联表（每个角色都有什么权限可以进行操作）。依据这个理论，我们进行来进行编码，很明显的我们第一步就是要进行实体类的创建。在这里我们使用Mysql和JPA进行操作数据库。

那么我们先在pom.xml中引入mysql和JPA的依赖：

```
<!-- Spirng data JPA依赖; -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<!-- mysql驱动; -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>

</dependency>
```

配置src/main/resouces/application.properties配置数据库和jpa(application.properties新建一个即可):

```
#########################################################
###datasource
#########################################################
spring.datasource.url = jdbc:mysql://localhost:3306/test
spring.datasource.username = root
spring.datasource.password = root
spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.datasource.max-active=20
spring.datasource.max-idle=8
spring.datasource.min-idle=8
spring.datasource.initial-size=10



#########################################################
### Java Persistence Api
```

```
#########################################################
# Specify the DBMS
spring.jpa.database = MYSQL
# Show or not log for each sql query
spring.jpa.show-sql = true
# Hibernate ddl auto (create, create-drop, update)
spring.jpa.hibernate.ddl-auto = update
# Naming strategy
#[org.hibernate.cfg.ImprovedNamingStrategy | org.hibernate.cfg.DefaultNamingStrategy]
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.DefaultNamingStrategy
# stripped before adding them to the entity manager)
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

准备工作准备好之后，那么就可以编写实体类了：

UserInfo.java、 SysRole.java、 SysPermission.java至于之前的关联表我们使用JPA进行自动生成。

用户：com.kfit.core.bean.UserInfo ：

```
package com.kfit.core.bean;

import java.io.Serializable;
import java.util.List;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;

/**
 * 用户信息.
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Entity
```

```
public class UserInfo implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id @GeneratedValue
    private long uid;//用户id;

    @Column(unique=true)
    private String username;//账号.

    private String name;//名称（昵称或者真实姓名，不同系统不同定义）

    private String password; //密码;
    private String salt;//加密密码的盐

    private byte state;//用户状态,0:创建未认证（比如没有激活，没有输入验证码等等）--等待验证的用户，1:正常状态,2：用户被锁定.


    @ManyToMany(fetch=FetchType.EAGER)//立即从数据库中进行加载数据;
    @JoinTable(name = "SysUserRole", joinColumns = { @JoinColumn(name = "uid") }, inverseJoinColumns ={@JoinColumn(name = "roleId") })
    private List<SysRole> roleList;// 一个用户具有多个角色

    public List<SysRole> getRoleList() {
        return roleList;
    }

    public void setRoleList(List<SysRole> roleList) {
        this.roleList = roleList;
    }

    public long getUid() {
        return uid;
    }

    public void setUid(long uid) {
        this.uid = uid;
    }
```

```
public String getUsername() {
    returnusername;
}

publicvoid setUsername(String username) {
    this.username = username;
}

public String getName() {
    returnname;
}

publicvoid setName(String name) {
    this.name = name;
}

public String getPassword() {
    returnpassword;
}

publicvoid setPassword(String password) {
    this.password = password;
}

public String getSalt() {
    returnsalt;
}

publicvoid setSalt(String salt) {
    this.salt = salt;
}

publicbyte getState() {
    returnstate;
}
```

```
    public void setState(byte state) {
        this.state = state;
    }

    /**
     * 密码盐.
     * @return
     */
    public String getCredentialsSalt(){
        return this.username+this.salt;
    }

    @Override
    public String toString() {
        return "UserInfo [uid=" + uid + ", username=" + username + ", name=" + name + ", password=" +
password
                + ", salt=" + salt + ", state=" + state + "]";
    }


}
```

在这里salt主要是用来进行密码加密的，当然也可以使用明文进行编码测试，实际开发中还是建议密码进行加密。

getCredentialsSalt()

这个方法重新对盐重新进行了定义，用户名+salt，这样就更加不容易被破解了。

角色类 > com.kfit.core.bean.SysRole：

```
package com.kfit.core.bean;

import java.io.Serializable;
import java.util.List;
```

```
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;



/**
 * 系统角色实体类;
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Entity
publicclass SysRole implements Serializable{
    privatestaticfinallongserialVersionUID = 1L;
    @Id@GeneratedValue
    private Long id; // 编号
    private String role; // 角色标识程序中判断使用,如"admin",这个是唯一的:
    private String description; // 角色描述,UI界面显示使用
    private Boolean available = Boolean.FALSE; // 是否可用,如果不可用将不会添加给用户

    //角色 -- 权限关系：多对多关系;
    @ManyToMany(fetch=FetchType.EAGER)
@JoinTable(name="SysRolePermission",joinColumns={@JoinColumn(name="roleId")},inverseJoinColumns={@JoinColumn(name=
    private List<SysPermission> permissions;

    // 用户 - 角色关系定义;
    @ManyToMany
@JoinTable(name="SysUserRole",joinColumns={@JoinColumn(name="roleId")},inverseJoinColumns={@JoinColumn(name="uid"
    private List<UserInfo> userInfos;// 一个角色对应多个用户

    public List<UserInfo> getUserInfos() {
      returnuserInfos;
    }
    publicvoid setUserInfos(List<UserInfo> userInfos) {
```

```
    this.userInfos = userInfos;
}
public Long getId() {
    returnid;
}
publicvoid setId(Long id) {
    this.id = id;
}
public String getRole() {
    returnrole;
}
publicvoid setRole(String role) {
    this.role = role;
}
public String getDescription() {
    returndescription;
}
publicvoid setDescription(String description) {
    this.description = description;
}
public Boolean getAvailable() {
    returnavailable;
}
publicvoid setAvailable(Boolean available) {
    this.available = available;
}
public List<SysPermission> getPermissions() {
    returnpermissions;
}
publicvoid setPermissions(List<SysPermission> permissions) {
    this.permissions = permissions;
}
@Override
public String toString() {
    return"SysRole [id=" + id + ", role=" + role + ", description=" + description + ", available=" + available
            + ", permissions=" + permissions + "]";
}
```

```
}
```

权限 > com.kfit.core.bean.SysPermission ：

```
package com.kfit.core.bean;

import java.io.Serializable;
import java.util.List;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;

/**
 * 权限实体类;
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
@Entity
publicclass SysPermission implements Serializable{
    privatestaticfinallongserialVersionUID = 1L;

    @Id@GeneratedValue
    privatelongid;//主键.
    private String name;//名称.

    @Column(columnDefinition="enum('menu','button')")
    private String resourceType;//资源类型，[menu|button]
    private String url;//资源路径.
    private String permission; //权限字符串,menu例子：role:*，button例子：
role:create,role:update,role:delete,role:view
    private Long parentId; //父编号
    private String parentIds; //父编号列表
```

```
    private Boolean available = Boolean.FALSE;

    @ManyToMany
@JoinTable(name="SysRolePermission",joinColumns={@JoinColumn(name="permissionId")},inverseJoinColumns={@JoinColumn
    private List<SysRole> roles;

    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getResourceType() {
        return resourceType;
    }
    public void setResourceType(String resourceType) {
        this.resourceType = resourceType;
    }
    public String getUrl() {
        return url;
    }
    public void setUrl(String url) {
        this.url = url;
    }
    public String getPermission() {
        return permission;
    }
    public void setPermission(String permission) {
        this.permission = permission;
    }
    public Long getParentId() {
```

```
    return parentId;
  }
  public void setParentId(Long parentId) {
    this.parentId = parentId;
  }
  public String getParentIds() {
    return parentIds;
  }
  public void setParentIds(String parentIds) {
    this.parentIds = parentIds;
  }
  public Boolean getAvailable() {
    return available;
  }
  public void setAvailable(Boolean available) {
    this.available = available;
  }
  public List<SysRole> getRoles() {
    return roles;
  }
  public void setRoles(List<SysRole> roles) {
    this.roles = roles;
  }
  @Override
  public String toString() {
    return "SysPermission [id=" + id + ", name=" + name + ", resourceType=" + resourceType + ", url=" + url
        + ", permission=" + permission + ", parentId=" + parentId + ", parentIds=" + parentIds + ",
available="
        + available + ", roles=" + roles + "]";
  }

}
```

ok，到这里实体类就编码完毕了，在这里我们看到的是3个实体类，
UserInfo,SysRole,SysPermission,对应的是数据库的五张表：

1表UserInfo、 2表SysUserRole、 3表SysRole、 4表SysRolePermission、 5表SysPermission

这时候运行程序，就会自动建表，然后我们添加一些数据：

用户管理

用户添加

用户删除

管理员

会员

管理员

这时候数据都准备完毕了，那么接下来就应该编写Repository进行访问数据了（在下载源码中有shiro.sql文件，可自行导入使用即可）。

com.kfit.core.repository.UserInfoRepository ：

```
package com.kfit.core.repository;

import org.springframework.data.repository.CrudRepository;
```

```
import com.kfit.core.bean.UserInfo;

/**
 * UserInfo持久化类;
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
publicinterface UserInfoRepository extends CrudRepository<UserInfo,Long>{

    /**通过username查找用户信息;*/
    public UserInfo findByUsername(String username);

}
```

在这里你会发现我们只编写了UserInfo的数据库操作，那么我们怎么获取我们的权限信息了，通过userInfo.getRoleList()可以获取到对应的角色信息，然后在通过对应的角色可以获取到权限信息，当然这些都是JPA帮我们实现了，我们也可以进行直接获取到权限信息，只要写一个关联查询然后过滤掉重复的权限即可，这里不进行实现。

编写一个业务处理类UserInfoService>

com.kfit.core.service.UserInfoService :

```
package com.kfit.core.service;

import com.kfit.core.bean.UserInfo;

publicinterface UserInfoService {

    /**通过username查找用户信息;*/
    public UserInfo findByUsername(String username);

}
```

com.kfit.core.service.impl.UserInfoServiceImpl :

```
package com.kfit.core.service.impl;

import javax.annotation.Resource;

import org.springframework.stereotype.Service;

import com.kfit.core.bean.UserInfo;
import com.kfit.core.repository.UserInfoRepository;
import com.kfit.core.service.UserInfoService;

@Service
publicclass UserInfoServiceImpl implements UserInfoService{

    @Resource
    private UserInfoRepository userInfoRepository;

    @Override
    public UserInfo findByUsername(String username) {
        System.out.println("UserInfoServiceImpl.findByUsername()");
        returnuserInfoRepository.findByUsername(username);
    }

}
```

这里主要是为了满足MVC编程模式，在例子中直接访问DAO层也未尝不可，实际开发中建议还是遵循MVC开发模式，至于有什么好处，请自行百度MVC。

好了以上都是为了实现身份认证，权限控制的准备工作，想必大家看了也有点困惑了，坚持住，这个技术点不是每个人都能进行编码的，你会发现在一个公司里都是技术领导帮你实现了这一部分很复杂的编码，你只需要通过界面进行配置而已，所以嘛，要做一个技术领导这一部分不掌握好像还不行了。好了，说重点吧，基本工作准备好之后，剩下的才是重点，shiro的认证最终是交给了Realm进行执行了，所以我们需要自己重新实现一个Realm，此Realm继承AuthorizingRealm。

com.kfit.config.shiro.MyShiroRealm ：

```
package com.kfit.config.shiro;
```

```
import javax.annotation.Resource;

import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.SimpleAuthenticationInfo;
import org.apache.shiro.authc.UsernamePasswordToken;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.apache.shiro.util.ByteSource;

import com.kfit.core.bean.SysPermission;
import com.kfit.core.bean.SysRole;
import com.kfit.core.bean.UserInfo;
import com.kfit.core.service.UserInfoService;

/**
 *  身份校验核心类;
 * @author Angel(QQ:412887952)
 * @version v.0.1
 */
public class MyShiroRealm extends AuthorizingRealm{


    @Resource
    private UserInfoService userInfoService;

    /**
     * 认证信息.(身份验证)
     * :
     * Authentication 是用来验证用户身份
     * @param token
     * @return
     * @throws AuthenticationException
     */
    @Override
```

```
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
    System.out.println("MyShiroRealm.doGetAuthenticationInfo()");

    //获取用户的输入的账号.
    String username = (String)token.getPrincipal();
    System.out.println(token.getCredentials());

    //通过username从数据库中查找 User对象，如果找到，没找到.
    //实际项目中，这里可以根据实际情况做缓存，如果不做，Shiro自己也是有时间间隔机制，2分钟内不会重复执行该方法
    UserInfo userInfo = userInfoService.findByUsername(username);
    System.out.println("----->>userInfo="+userInfo);
    if(userInfo == null){
        return null;
    }

    /*
     * 获取权限信息:这里没有进行实现，
     * 请自行根据UserInfo,Role,Permission进行实现；
     * 获取之后可以在前端for循环显示所有链接;
     */
    //userInfo.setPermissions(userService.findPermissions(user));

    //账号判断;

    //加密方式;
    //交给AuthenticatingRealm使用CredentialsMatcher进行密码匹配，如果觉得人家的不好可以自定义实现
    SimpleAuthenticationInfo authenticationInfo = new SimpleAuthenticationInfo(
            userInfo, //用户名
            userInfo.getPassword(), //密码
             ByteSource.Util.bytes(userInfo.getCredentialsSalt()),//salt=username+salt
             getName()  //realm name
    );
```

```
        //明文: 若存在，将此用户存放到登录认证info中，无需自己做密码对比，Shiro会为我们进行密码对比校验
//      SimpleAuthenticationInfo authenticationInfo = new SimpleAuthenticationInfo(
//              userInfo, //用户名
//              userInfo.getPassword(), //密码
//               getName()  //realm name
//      );

        return authenticationInfo;
    }



    /**
     * 此方法调用  hasRole,hasPermission的时候才会进行回调.
     *
     * 权限信息.(授权):
     * 1、如果用户正常退出，缓存自动清空；
     * 2、如果用户非正常退出，缓存自动清空；
     * 3、如果我们修改了用户的权限，而用户不退出系统，修改的权限无法立即生效。
     * （需要手动编程进行实现；放在service进行调用）
     * 在权限修改后调用realm中的方法，realm已经由spring管理，所以从spring中获取realm实例，
     * 调用clearCached方法；
     * :Authorization 是授权访问控制，用于对用户进行的操作授权，证明该用户是否允许进行当前操作，如访问某个链接，某个资源文件等。
     * @param principals
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
        /*
         * 当没有使用缓存的时候，不断刷新页面的话，这个代码会不断执行，
         * 当其实没有必要每次都重新设置权限信息，所以我们需要放到缓存中进行管理；
         * 当放到缓存中时，这样的话，doGetAuthorizationInfo就只会执行一次了，
         * 缓存过期之后会再次执行。
         */
        System.out.println("权限配置-->MyShiroRealm.doGetAuthorizationInfo()");
```

```
    SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
    UserInfo userInfo  = (UserInfo)principals.getPrimaryPrincipal();

    //实际项目中，这里可以根据实际情况做缓存，如果不做，Shiro自己也是有时间间隔机制，2分钟内不会重复执行该方法
//    UserInfo userInfo = userInfoService.findByUsername(username)


    //权限单个添加;
    // 或者按下面这样添加
     //添加一个角色,不是配置意义上的添加,而是证明该用户拥有admin角色
//    authorizationInfo.addRole("admin");
     //添加权限
//    authorizationInfo.addStringPermission("userInfo:query");



    ///在认证成功之后返回.
    //设置角色信息.
    //支持 Set集合,
    //用户的角色对应的所有权限，如果只使用角色定义访问权限，下面的四行可以不要
//        List<Role> roleList=user.getRoleList();
//        for (Role role : roleList) {
//            info.addStringPermissions(role.getPermissionsName());
//        }
    for(SysRole role:userInfo.getRoleList()){
        authorizationInfo.addRole(role.getRole());
        for(SysPermission p:role.getPermissions()){
            authorizationInfo.addStringPermission(p.getPermission());
        }
    }

    //设置权限信息.
//    authorizationInfo.setStringPermissions(getStringPermissions(userInfo.getRoleList()));

    return authorizationInfo;
  }
```

```
    /**
     * 将权限对象中的权限code取出.
     * @param permissions
     * @return
     */
//  public Set<String> getStringPermissions(Set<SysPermission> permissions){
//     Set<String> stringPermissions = new HashSet<String>();
//     if(permissions != null){
//         for(SysPermission p : permissions) {
//             stringPermissions.add(p.getPermission());
//         }
//     }
//       return stringPermissions;
// }

}
```

继承AuthorizingRealm主要需要实现两个方法：

doGetAuthenticationInfo();

doGetAuthorizationInfo();

其中doGetAuthenticationInfo主要是用来进行身份认证的，也就是说验证用户输入的账号和密码是否正确。

```
SimpleAuthenticationInfoauthenticationInfo =
 new SimpleAuthenticationInfo(
            userInfo, //用户名
            userInfo.getPassword(), //密码
            ByteSource.Util.bytes(userInfo.getCredentialsSalt()),//salt=username+salt
            getName()  //realm name
```

```
    );
```

交给AuthenticatingRealm使用CredentialsMatcher进行密码匹配，如果觉得人家的不好可以自定义实现

如果你是进行明文进行编码的话，那么使用使用如下方式：

```
SimpleAuthenticationInfo authenticationInfo = new SimpleAuthenticationInfo(
        userInfo, //用户名
        userInfo.getPassword(), //密码
        getName()  //realm name
    );
```

至于doGetAuthorizationInfo()是权限控制，当访问到页面的时候，使用了相应的注解或者shiro标签才会执行此方法否则不会执行，所以如果只是简单的身份认证没有权限的控制的话，那么这个方法可以不进行实现，直接返回null即可。

在这个方法中主要是使用类：SimpleAuthorizationInfo

进行角色的添加和权限的添加。

```
authorizationInfo.addRole(role.getRole());

authorizationInfo.addStringPermission(p.getPermission());
```

当然也可以添加集合：

```
authorizationInfo.setRoles(roles);

authorizationInfo.setStringPermissions(stringPermissions);
```

到这里我们还需要有一个步骤很重要就是将我们自定义的Realm注入到SecurityManager中。

在com.kfit.config.shiro.ShiroConfiguration中添加方法：

```
/**
    * 身份认证realm;
    * (这个需要自己写，账号密码校验；权限等)
```

```
     * @return
     */
    @Bean
    public MyShiroRealm myShiroRealm(){
        MyShiroRealm myShiroRealm = new MyShiroRealm();
        returnmyShiroRealm;

    }
```

将myShiroRealm注入到securityManager中：

```
@Bean
    public SecurityManager securityManager(){
        DefaultWebSecurityManager securityManager =  new DefaultWebSecurityManager();
        //设置realm.
        securityManager.setRealm(myShiroRealm());
        returnsecurityManager;

    }
```

到这里的话身份认证权限控制基本是完成了，最后我们在编写一个登录的时候，登录的处理：

在com.kfit.root.controller.HomeController中添加login post处理：

```
// 登录提交地址和applicationontext-shiro.xml配置的loginurl一致。 (配置文件方式的说法)
    @RequestMapping(value="/login",method=RequestMethod.POST)
    public String login(HttpServletRequest request, Map<String, Object> map) throws Exception {
        System.out.println("HomeController.login()");
        // 登录失败从request中获取shiro处理的异常信息。
        // shiroLoginFailure:就是shiro异常类的全类名.
        String exception = (String) request.getAttribute("shiroLoginFailure");

        System.out.println("exception=" + exception);
        String msg = "";
        if (exception != null) {
            if (UnknownAccountException.class.getName().equals(exception)) {
                System.out.println("UnknownAccountException -- > 账号不存在：");
```

```
            msg = "UnknownAccountException -- > 账号不存在：";
        } elseif (IncorrectCredentialsException.class.getName().equals(exception)) {
            System.out.println("IncorrectCredentialsException -- > 密码不正确：");
            msg = "IncorrectCredentialsException -- > 密码不正确：";
        } elseif ("kaptchaValidateFailed".equals(exception)) {
            System.out.println("kaptchaValidateFailed -- > 验证码错误");
            msg = "kaptchaValidateFailed -- > 验证码错误";
        } else {
            msg = "else >> "+exception;
            System.out.println("else -- >" + exception);
        }
    }
    map.put("msg", msg);
    // 此方法不处理登录成功,由shiro进行处理.
    return"/login";
}
```

这时候我们启动应用程序，访问http://127.0.0.1:8080/index

会自动跳转到http://127.0.0.1:8080/login 界面，然后输入账号和密码：admin/123456,这时候会提示：IncorrectCredentialsException -- >密码不正确。

这主要是因为我们在上面进行了密文的方式，那么怎么加密方式，我们并没有告诉Shiro，所以认证失败了。

在这里我们需要编写一个加密算法类，当然Shiro也已经有了具体的实现 HashedCredentialsMatcher

我们只需要进行注入使用即可：

在com.kfit.config.shiro.ShiroConfiguration中加入方法：

```
/**
    * 凭证匹配器
    * （由于我们的密码校验交给Shiro的SimpleAuthenticationInfo进行处理了
    * 所以我们需要修改下doGetAuthenticationInfo中的代码;
    * ）
```

```
     * @return
     */
    @Bean
    public HashedCredentialsMatcher hashedCredentialsMatcher(){
        HashedCredentialsMatcher hashedCredentialsMatcher = new HashedCredentialsMatcher();

        hashedCredentialsMatcher.setHashAlgorithmName("md5");//散列算法:这里使用MD5算法;
        hashedCredentialsMatcher.setHashIterations(2);//散列的次数，比如散列两次，相当于 md5(md5(""));

        return hashedCredentialsMatcher;
    }
```

在myShiroRealm()方法中注入凭证匹配器：

```
/**
     * 身份认证realm;
     * (这个需要自己写，账号密码校验；权限等)
     * @return
     */
    @Bean
    public MyShiroRealm myShiroRealm(){
        MyShiroRealm myShiroRealm = new MyShiroRealm();
        myShiroRealm.setCredentialsMatcher(hashedCredentialsMatcher());;
        return myShiroRealm;
    }
```

这时候在访问/login进行登录就可以登陆到/index界面了。

这一节就先到这里吧，实在是有点头大了是吧。到时候奉上源代码。

(d) 权限控制

在上一小节我们已经可以登录了。

在我们新建一个UserInfoController

com.kfit.core.controller.UserInfoController ：

```
package com.kfit.core.controller;

import org.apache.shiro.authz.annotation.RequiresPermissions;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/userInfo")
publicclass UserInfoController {

    /**
     * 用户查询.
     * @return
     */
    @RequestMapping("/userList")
    public String userInfo(){
      return"userInfo";
    }

    /**
     * 用户添加;
     * @return
     */
    @RequestMapping("/userAdd")
    public String userInfoAdd(){
      return"userInfoAdd";
    }

}
```

然后运行登录进行访问：http://127.0.0.1:8080/userInfo/userAdd

并没有执行doGetAuthorizationInfo()打印信息，所以我们会发现我们的身份认证是好使了，但是权限控制好像没有什么作用哦。

我们少了几部分代码，

第一就是开启shiro aop注解支持，这个只需要在com.kfit.config.shiro.ShiroConfiguration加入如下方法进行开启即可：

```
/**
    * 开启shiro aop注解支持.
    * 使用代理方式;所以需要开启代码支持;
    * @param securityManager
    * @return
    */
   @Bean
   public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager securityManager){
      AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor = new AuthorizationAttributeSourceAdvisor();
      authorizationAttributeSourceAdvisor.setSecurityManager(securityManager);
      returnauthorizationAttributeSourceAdvisor;

   }
```

第二就是在controller方法中加入相应的注解：

```
/**
    * 用户添加;
    * @return
    */
   @RequestMapping("/userAdd")
   @RequiresPermissions("userInfo:add")//权限管理;
   public String userInfoAdd(){
     return"userInfoAdd";

   }
```

这时候在访问http://127.0.0.1:8080/userInfo/userAdd 会看到控制台打印信息：

```
权限配置-->MyShiroRealm.doGetAuthorizationInfo()
```

如果访问：http://127.0.0.1:8080/userInfo/userDel 会看到

```
Whitelabel Error Page
```

This application has no explicit mapping for /error, so you are seeing this as a fallback.

```
Sat May 21 22:17:55 CST 2016
There was an unexpected error (type=Internal Server Error, status=500).
Subject does not have permission [userInfo:del]
```

当然我们需要在UserInfoController方法中加入：

```
/**
    * 用户删除;
    * @return
    */
   @RequestMapping("/userDel")
   @RequiresPermissions("userInfo:del")//权限管理;
   public String userDel(){
     return"userInfoDel";

   }
```

在上面的错误信息中Subject does not have permission可以看出此用户没有这个权限。好了，至此Shiro的权限控制到此先告一段落。在这里我先抛出一个问题：我们不断的访问 http://127.0.0.1:8080/userInfo/userAdd 你会看到

```
权限配置-->MyShiroRealm.doGetAuthorizationInfo()
2016-05-21 22:20:26.263  INFO 11692 --- [nio-8080-exec-1] org.apache.shiro.realm.AuthorizingRealm  : No cache or cacheManager properties have been set.  Authorization cache cannot be obtained.
权限配置-->MyShiroRealm.doGetAuthorizationInfo()
2016-05-21 22:20:26.385  INFO 11692 --- [nio-8080-exec-2] org.apache.shiro.realm.AuthorizingRealm  : No cache or cacheManager properties have been set.  Authorization cache cannot be obtained.
权限配置-->MyShiroRealm.doGetAuthorizationInfo()
2016-05-21 22:20:26.538  INFO 11692 --- [nio-8080-exec-3] org.apache.shiro.realm.AuthorizingRealm  : No cache or cacheManager properties have been set.  Authorization cache cannot be obtained.
```

```
权限配置-->MyShiroRealm.doGetAuthorizationInfo()
```

这说明我们不断的访问权限信息，但是实际中我们的权限信息是不怎么会改变的，所以我们希望是第一次访问，然后进行缓存处理，那么Shiro是否支持呢，答案是肯定的，我们在下一小节进行讲解，如何在Shiro中加入缓存机制。

【Spring Boot 系列博客】

0 ）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（ 1 ）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（ 2 ）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
（ 15 ）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2292362

16 ）Spring Boot使用Druid（ 编程注入 ）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（ 17 ）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

.....

（ 35 ）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/

附件下载:

- spring-boot-shiro1.zip (44.4 KB)

- dl.iteye.com/topics/download/d3ff2380-538d-35e3-b882-72cf044953d1

- spring-boot-shiro1.zip (44.4 KB)
- dl.iteye.com/topics/download/7588e430-78cc-316f-9017-b9212a2d251f

1.44 （ 39.3 ） Spring Boot Shiro权限管理【从零开始学Spring Boot】

发表时间: 2016-05-21 关键字: Spring Boot Shiro权限管理, 从零开始学Spring Boot

在学习此小节之前您可能还需要学习：

（ 39.1 ） Spring Boot Shiro权限管理【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2299732

（ 39.2 ）. Spring Boot Shiro权限管理【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2299777

相对于上一小节这个就比较简单了。

主要分这么几个步骤：在pom.xml中加入缓存依赖；注入缓存；

(a) 在pom.xml文件中加入依赖：

```
<!-- shiro ehcache -->
    <dependency>
       <groupId>org.apache.shiro</groupId>
       <artifactId>shiro-ehcache</artifactId>
       <version>1.2.2</version>
    </dependency>

    <!--
       包含支持UI模版（Velocity，FreeMarker，JasperReports），
       邮件服务，
       脚本服务(JRuby)，
       缓存Cache（EHCache），
       任务计划Scheduling（uartz）。
     -->
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context-support</artifactId>

</dependency>
```

(b)注入缓存

在com.kfit.config.shiro.ShiroConfiguration中加入如下方法：

```
/**
 * shiro缓存管理器;
 * 需要注入对应的其它的实体类中：
 * 1、安全管理器：securityManager
 * 可见securityManager是整个shiro的核心；
 * @return
 */
@Bean
public EhCacheManager ehCacheManager(){
  System.out.println("ShiroConfiguration.getEhCacheManager()");
  EhCacheManager cacheManager = new EhCacheManager();
  cacheManager.setCacheManagerConfigFile("classpath:config/ehcache-shiro.xml");
  return cacheManager;

}
```

将缓存对象注入到SecurityManager中：

```
@Bean
public SecurityManager securityManager(){
  DefaultWebSecurityManager securityManager =  new DefaultWebSecurityManager();
  //设置realm.
  securityManager.setRealm(myShiroRealm());

  //注入缓存管理器;
  securityManager.setCacheManager(ehCacheManager());//这个如果执行多次，也是同样的一个对象;
```

```
    return securityManager;
}
```

(c)添加缓存配置文件：

在src/main/resouces/config添加ehcache-shiro.xml配置文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache name="es">

  <diskStore path="java.io.tmpdir"/>

  <!--
    name:缓存名称。
    maxElementsInMemory:缓存最大数目
    maxElementsOnDisk：硬盘最大缓存个数。
    eternal:对象是否永久有效，一但设置了，timeout将不起作用。
    overflowToDisk:是否保存到磁盘，当系统当机时
    timeToIdleSeconds:设置对象在失效前的允许闲置时间（单位：秒）。仅当eternal=false对象不是永久有效时使用，可选属性，默认值是0，也就是可闲置时间无穷大。
    timeToLiveSeconds:设置对象在失效前允许存活时间（单位：秒）。最大时间介于创建时间和失效时间之间。仅当eternal=false对象不是永久有效时使用，默认是0.，也就是对象存活时间无穷大。
    diskPersistent：是否缓存虚拟机重启期数据 Whether the disk store persists between restarts of the Virtual Machine. The default value is false.
    diskSpoolBufferSizeMB：这个参数设置DiskStore（磁盘缓存）的缓存区大小。默认是30MB。每个Cache都应该有自己的一个缓冲区。
    diskExpiryThreadIntervalSeconds：磁盘失效线程运行时间间隔，默认是120秒。
    memoryStoreEvictionPolicy：当达到maxElementsInMemory限制时，Ehcache将会根据指定的策略去清理内存。默认策略是LRU（最近最少使用）。你可以设置为FIFO（先进先出）或是LFU（较少使用）。
     clearOnFlush：内存数量最大时是否清除。
      memoryStoreEvictionPolicy:
        Ehcache的三种清空策略;
        FIFO，first in first out，这个是大家最熟的，先进先出。
        LFU， Less Frequently Used，就是上面例子中使用的策略，直白一点就是讲一直以来最少被使用的。如上面所讲，缓存的元素有一个hit属性，hit值最小的将会被清出缓存。
```

```
        LRU，Least Recently Used，最近最少使用的，缓存的元素有一个时间戳，当缓存容量满了，而又需要腾出地方来缓存新的元素的时候，那么现有缓存元素中时间戳离当前时间最远的元素将被清出缓存。
    -->
     <defaultCache
            maxElementsInMemory="10000"
            eternal="false"
            timeToIdleSeconds="120"
            timeToLiveSeconds="120"
            overflowToDisk="false"
            diskPersistent="false"
            diskExpiryThreadIntervalSeconds="120"
            />


    <!-- 登录记录缓存锁定10分钟 -->
    <cache name="passwordRetryCache"
           maxEntriesLocalHeap="2000"
           eternal="false"
           timeToIdleSeconds="3600"
           timeToLiveSeconds="0"
           overflowToDisk="false"
           statistics="true">
    </cache>

</ehcache>
```

在配置文件上已经有很详细的解释了，所以这里就过多介绍ehcache的配置了。

运行程序访问：http://127.0.0.1:8080/userInfo/userAdd

查看控制台的打印信息：

权限配置-->MyShiroRealm.doGetAuthorizationInfo()

这个信息就只打印一次了，说明我们的缓存生效了。

0 ）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（ 1 ）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（ 2 ）Spring Boot返回json数据【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2291508
（ 15 ）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】
http://412887952-qq-com.iteye.com/blog/2292362

16 ）Spring Boot使用Druid（ 编程注入 ）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（ 17 ）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

.....

（ 35 ）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/

附件下载:

- spring-boot-shiro1.zip (49 KB)
- dl.iteye.com/topics/download/c9462ac6-ec2b-3c5f-81df-1ed3ca48c65b

1.45 (39.4) Spring Boot Shiro权限管理【从零开始学Spring Boot】

发表时间: 2016-05-21 关键字: 从零开始学Spring Boot, Spring Boot Shiro权限管理

在读此文章之前您还可能需要先了解：

（39.1）Spring Boot Shiro权限管理【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2299732

（基本搭建）

（39.2）. Spring Boot Shiro权限管理【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2299777

（shiro权限引入）

（39.3）Spring Boot Shiro权限管理【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2299780

（shiro 缓存）

(6). Shiro记住密码

记住密码实现起来也是比较简单的，主要看下是如何实现的。

在com.kfit.config.shiro.ShiroConfiguration加入两个方法：

```
/**
    * cookie对象;
    * @return
```

```
     */
    @Bean
    public SimpleCookie rememberMeCookie(){
        System.out.println("ShiroConfiguration.rememberMeCookie()");
        //这个参数是cookie的名称，对应前端的checkbox的name = rememberMe
        SimpleCookie simpleCookie = new SimpleCookie("rememberMe");
        //<!-- 记住我cookie生效时间30天 ,单位秒;-->
        simpleCookie.setMaxAge(259200);
        return simpleCookie;
    }

    /**
     * cookie管理对象;
     * @return
     */
    @Bean
    public CookieRememberMeManager rememberMeManager(){
        System.out.println("ShiroConfiguration.rememberMeManager()");
        CookieRememberMeManager cookieRememberMeManager = new CookieRememberMeManager();
        cookieRememberMeManager.setCookie(rememberMeCookie());
        return cookieRememberMeManager;
    }
```

将rememberMeManager注入到SecurityManager中

```
@Bean
    public SecurityManager securityManager(){
        DefaultWebSecurityManager securityManager =  new DefaultWebSecurityManager();
        //设置realm.
        securityManager.setRealm(myShiroRealm());

        //注入缓存管理器;
        securityManager.setCacheManager(ehCacheManager());//这个如果执行多次，也是同样的一个对象;

        //注入记住我管理器;
        securityManager.setRememberMeManager(rememberMeManager());
```

```
    return securityManager;
}
```

在ShiroFilterFactoryBean添加记住我过滤器：

```
@Bean
public ShiroFilterFactoryBean shirFilter(SecurityManager securityManager){
    System.out.println("ShiroConfiguration.shirFilter()");
    ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();

     // 必须设置 SecurityManager
    shiroFilterFactoryBean.setSecurityManager(securityManager);



    //拦截器.
    Map<String,String> filterChainDefinitionMap = new LinkedHashMap<String,String>();

    //配置退出过滤器,其中的具体的退出代码Shiro已经替我们实现了
    filterChainDefinitionMap.put("/logout", "logout");


    //配置记住我或认证通过可以访问的地址
     filterChainDefinitionMap.put("/index", "user");
     filterChainDefinitionMap.put("/", "user");


    //<!-- 过滤链定义，从上向下顺序执行，一般将 /**放在最为下边 -->:这是一个坑呢，一不小心代码就不好使了;
     //<!-- authc:所有url都必须认证通过才可以访问; anon:所有url都都可以匿名访问-->
    filterChainDefinitionMap.put("/**", "authc");

    // 如果不设置默认会自动寻找Web工程根目录下的"/login.jsp"页面
     shiroFilterFactoryBean.setLoginUrl("/login");
     // 登录成功后要跳转的链接
     shiroFilterFactoryBean.setSuccessUrl("/index");
```

```
        //未授权界面;
        shiroFilterFactoryBean.setUnauthorizedUrl("/403");

        shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);
        return shiroFilterFactoryBean;
    }
```

主要是加入了：

```
//配置记住我或认证通过可以访问的地址
filterChainDefinitionMap.put("/index", "user");
filterChainDefinitionMap.put("/", "user");
```

修改登录界面加入rememberMe复选框：

在login.html中加入：

```
<P><input type="checkbox" name="rememberMe" />记住我</P>
```

这时候运行程序，登录之后跳转到/index页面，然后我们关闭浏览器，然后直接访问/index还是可以访问的，说明我们写的记住密码已经生效了，如果访问http://127.0.0.1:8080/userInfo/userAdd 的

话还是需要重新登录的。

至此spring boot集成shiro就真的告一段落了。

当然shiro还是博大精深的，还有很多需要大家去研究，在这里博主抛出几个问题（当然博主自己已经实现了）：

问题1：如何在登录的时候加入验证码校验；

问题2：如何在thymleaf使用shiro标签，比如：shiro:hasPermission；

问题3：如何限制密码输错3次就不能登录了；

问题4：如何编写无状态的过滤器；

问题5：在同一个工程中如何有状态和无状态的过滤器同时并存；（这个问题博主还没解决哦，有谁知道怎么解决的，欢迎留言或者加QQ告知，不胜感激）。

【Spring Boot 系列博客】

0）前言【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291496

（1）spring boot起步之Hello World【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2291500

（2）Spring Boot返回json数据【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2291508

（15）Spring Boot使用Druid和监控配置【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2292362

16）Spring Boot使用Druid（编程注入）【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blogs/2292376

（17）Spring Boot普通类调用bean【从零开始学Spring Boot】：

http://412887952-qq-com.iteye.com/blog/2292388

.....

（35）Spring Boot集成Redis实现缓存机制【从零开始学Spring Boot】

http://412887952-qq-com.iteye.com/blog/2294942

更多查看博客：http://412887952-qq-com.iteye.com/

附件下载:

- spring-boot-shiro1.zip (49.9 KB)
- dl.iteye.com/topics/download/060fba7e-e588-33b0-82a4-1adacdbafb01

从零开始学Spring Boot

作者: 林祥纤
http://412887952-qq-com.iteye.com