2018年1月24日16:12:21

1. 安装playframework

从官网<https://www.playframework.com/下载play的安装包play-2.2.6.zip>

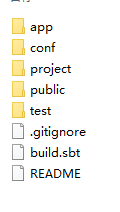
官网上从2.3版本开始就不提供这种开发包了。它的解释是这种传统的zip包，提供play命令来新建项目和运行项目，从2.3版本后就不再使用了。可以使用官方提供的项目模板来开发。

但是利用项目模板来开发效果并不好。下载好模板后，直接运行里面的sbt命令，系统会自动下载相关依赖，但是我每次都是错误，连接超时。项目根本不能用，还是用回旧版本吧。并且相关资料也比较少。

1. 将下载的zip包解压，并添加路径到系统环境变量里面。可以在命令行里面输入play命令，会显示一个play的图片，表示安装好了。
2. 创建play应用

进入以空的文件夹，使用play new 应用名 命令创建一个play应用。然后使用play run命令来运行程序。

生成的应用目录结构如下：



1. 将play项目转化为eclipse项目

进入项目目录后，输入play命令，然后再输入eclipse命令即可将paly项目转化为eclipse项目。

1. Play项目不需要手动编译，修改过java代码后，只要刷新页面play就可以自动更新项目了。
2. 第一次简单的修改play项目

Paly使用的是mvc框架，通过controller来接收http请求。修改新建的项目下app/controllers/application.java的内容。

**package** controllers;

**import** play.\*;

**import** play.mvc.\*;

**import** views.html.\*;

**public** **class** Application **extends** Controller {

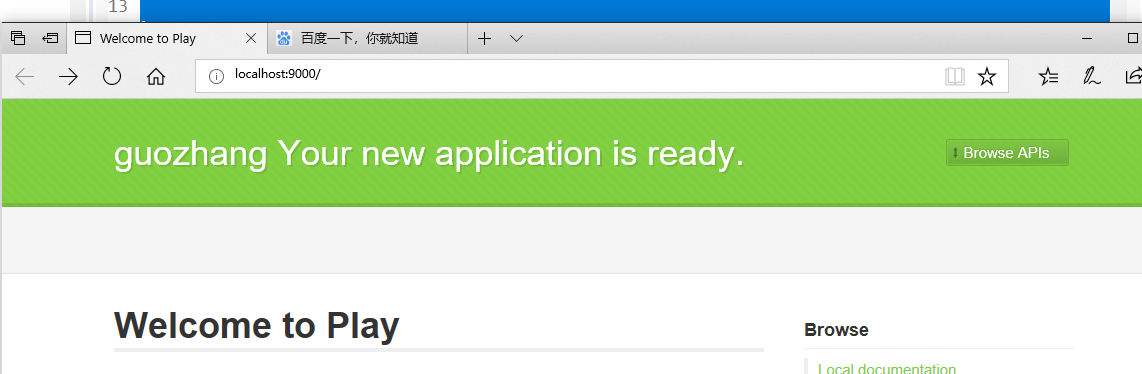
**public** **static** Result index() {

**return** *ok*(index.*render*("guozhang Your new application is ready."));

}

}

显示结果如下：



1. 修改main.scala.html下body的内容

<body>

<!-- @content -->

<div>

<p>hello world</p>

</div>

</body>

这下显示结果就只有hello world了

2018年1月25日11:04:47

1. 新的修改，将页面显示为自定义的html页面

要想设置新的页面，就得添加新的URL路劲或者不同的URL参数。这时需要在route里面修改路由表，这个表定义了http请求调用哪个controller下的方法，以及设置URL路劲的缩写。

在routes文件里面添加如下语句：

GET /test controllers.Application.bye()

表示当浏览器访问主页后面的test的时候，调用的是 controllers.Application.bye()方法。

然后就需要在application.java里面添加bye方法了。

方法写好后，在浏览器里面输入<http://localhost:9000/test> 就能直接调用这个方法了。

**public** **static** Result bye() {

**return** *ok*("<h1 style='color:red'>Hello Wrold!</h1>").as("text/html");

}

这里的方法比较特殊，可以参考自带的index()方法的写法。方法的返回值是Result，也就是标准的http响应码，比如200表示正常响应。

这里的方法官方定义为action，也就是执行http响应，并将结果返回给客户端。这里的方法都是写在一个继承自controller类的子类里面的。

响应结果可以有多种情况：

Result ok = ok("Hello world!");

Result notFound = notFound();

Result pageNotFound = notFound("<h1>Page not found</h1>").as("text/html");

Result badRequest = badRequest(views.html.form.render(formWithErrors));

Result oops = internalServerError("Oops");

Result anyStatus = status(488, "Strange response type");

返回结果通过ok方法来返回，参数是一个字符串，可以是html代码。可以设置响应的类型ContentType为text/html，如果没有设置为这个类型，就是普通的plain文本。

public static Result index() {

response().setContentType("text/html");

return ok("<h1>Hello World!</h1>");

}

1. 设置响应头

public static Result index() {

response().setContentType("text/html");

response().setHeader(CACHE\_CONTROL, "max-age=3600");

response().setHeader(ETAG, "xxx");

return ok("<h1>Hello World!</h1>");

}

2018年1月25日20:09:26

1. 使用session来保存用户状态

Session是用来保存多个http请求之间的状态的工具。比如用户登录后，保存用户的用户名。它使用了cookie技术，但数据大小很有限，最大4kb，并且只能保存字符串。

Session的使用就像使用键值对一样简单。

public static Result index() {

session("connected", "user@gmail.com");

return ok("Welcome!");

}

上面代码保存了一个key为connected的session。如果要读取这个session，只需要将这个key作为参数赋给session函数即可，返回值为value。

public static Result index() {

**String user = session("connected");**

if(user != null) {

return ok("Hello " + user);

} else {

return unauthorized("Oops, you are not connected");

}

}

移除session

session().remove("connected");

放弃整个session

session().clear();

1. 使用flash scope来保存状态数据

Flash scope和session很类似，但有区别。flash scope只能保存一个请求。用户能修改cookie。

flash scope只能用来传输success/error消息。

public static Result index() {

String message = flash("success");

if(message == null) {

message = "Welcome!";

}

return ok(message);

}

public static Result save() {

flash("success", "The item has been created");

return redirect("/home");

}

1. 请求体的解析

Htpp请求中的post和put请求都包含了请求体。get请求不包含。请求体的格式通过请求头的content-type定义。比如可以是文本，json，xml，多媒体文件等。

下面是一些常见的Content-Type字段的值。

* text/plain
* text/html
* text/css
* image/jpeg
* image/png
* image/svg+xml
* audio/mp4
* video/mp4
* application/javascript
* application/pdf
* application/zip
* application/atom+xml

不能直接用java来写请求体解析。Play提供了默认的bodyparse满足大多数需求。可以用java在这些的基础上拓展自己的解析器。

public static Result index() {

RequestBody body = request().body();

return ok("Got body: " + body);

}

public static Result save() {

RequestBody body = request().body();

String textBody = body.asText();

if(textBody != null) {

ok("Got: " + text);

} else {

badRequest("Expecting text/plain request body");

}

}

1. 使用模板引擎template engine

实际上就是建立一个html文件，系统自动生成对应的class文件，controller可以调用里面的render方法，然后就可以像客户端返回内容了。

If you create a views/Application/index.scala.html template file, it will generate a views.html.Application.index class that has a render() method.

For example, here is a simple template:

@(customer: Customer, orders: List[Order])

<h1>Welcome @customer.name!</h1>

<ul>

@for(order <- orders) {

<li>@order.getTitle()</li>

}

</ul>

You can then call this from any Java code as you would normally call a method on a class:

Content html = views.html.Application.index.render(customer, orders);

2018年1月26日10:44:26

1. 解决添加html后eclipse报无法解析xx类型的错误。

在views下面添加一个xx.scala.html后，play会自动生成对应的xx.class,并且包含render方法，供java代码调用。需要导入import views.html.\*;

之前只是添加了，可是不知道play是怎么添加class的。并且在浏览器里面刷新，系统也能正常运行，就是eclipse里面会报错。（eclipse在这里只是个编辑器，完全不用它的其他功能）。

多次尝试后终于找到正确的解决办法。

首先在命令行里面关闭程序ctrl+d

确保已经输入过play命令，然后输入compile命令，编译完成后，eclipse立马就不报错了。

四、控制台命令

play控制台常用命令  
　　第一：play help  
　　第二：play 启动play项目  
　　第三：run 使服务器在开发模式下运行，为你的项目访问浏览器做准备  
　　第四：exit 或Ctrl+D 离开控制台  
　　第五：compile 编译应用  
　　第六：console 启动交互式控制台：进入的是交互式的scala控制台可以测试你的代码  
　　第七：play debug 在开启JPDA调试端口(默认是9999)后可以链接使用Java调试器。  
　　      通过set JPDA\_PORT=数值(如1234)命令，可以设置JPDA端口  
　　第八：使用sbt特性如进入到项目文件夹运行play、run、compile等命令  
　　第九：直接在控制台使用play命令如进入CMD后直接输入play run、play help  
　　第十：play clean-all  清除所有的生成目录

1. Play项目的启动

进入项目目录后，输入play run就可以运行。也可以先输入play，然后再输入run命令。

系统项目默认的端口号是9000。 再浏览器里面输入localhost:9000即可访问项目

1. 简单的form提交程序的实现
2. 创建两个html页面， 一个用来提交页面内容，另一个用来显示后台处理结果

jump.scala.html 包含两个文本输入框和一个提交按钮

<!DOCTYPE html>

<html>

<head>

<title>测试Form</title>

</head>

<body>

<h1>form提交</h1>

<form action=*"say"* method=*"get"*>

<input type=*"text"* name=*"email"*/> <br/>

<input type=*"text"* name=*"password"*/><br/>

<input type=*"submit"*/>

</form>

</body>

</html>

say.scala.html 包含两个h标签，用来显示用户名和密码

@(user:String,password:String)

<html>

<head>

<title>ddfdf</title>

</head>

<body>

<h1>@user</h1>

<h2>@password</h2>

</body>

</html>

其中开头部分的@(user:String,password:String)是参数，是scala格式。通过java调用render方法的时候传递。在body里面也用@来调用这个参数。

1. 在controller类里面添加jump()和sayhello()方法，并在routes文件添加对应的指向说明。

**public** **static** Result jump()

{

**return** *ok*(views.html.jump.*render*());

}

**public** **static** Result sayhello() {

Form<User> userForm=Form.*form*(User.**class**);

User user= userForm.bindFromRequest().get();

**return** *ok*(views.html.say.*render*(user.email,user.password));

}

GET /say controllers.Application.sayhello()

GET /jump controllers.Application.jump()

这样在浏览器里面输入localhost:9000/jump就能访问jump方法了。然后能在浏览器里面打开jump.scala.html页面。然后点击里面的提交按钮，会通过get方式访问say方法，并传递参数。然后say方法就能接收到参数，最后通过say.scala.html页面显示结果。

注意Uers类里面的变量定义要和form表单提交的参数名一致。

**public** **class** User {

**public** String email;

**public** String password;

}

<form action=*"say"* method=*"get"*>

<input type=*"text"* name=*"email"*/> <br/>

<input type=*"text"* name=*"password"*/><br/>

<input type=*"submit"*/>

</form>

User里面是email和password，form里面也应该是email和password。否则form无法解析到客户端发送的请求。

另外要注意的是form的action，是直接在主地址的后面添加action的内容。即若action为say，则提交form的地址为localhost:9000/say 。 也就是说不要把from的action设置为localhost:9000/say 会出现重复的内容。

获取表单内容：User user= userForm.bindFromRequest().get();

也可以只获取一部分内容：

DynamicForm in = Form.form().bindFromRequest();  
 String result = in.get("content");

官网上说的使用DynamicForm检索没有model的数据

public static Result hello() {

DynamicForm requestData = Form.form().bindFromRequest();

String firstname = requestData.get("firstname");

String lastname = requestData.get("lastname");

return ok("Hello " + firstname + " " + lastname);

}

1. 额外的约束

public class User {

@Required

public String email;

public String password;

}

2018年1月29日16:35:21

1. 在Eclipse里面给java项目导入jar包。

右键工程项目，选择build path，选择add libraries。

或者右击 Reference Libries，选择build path。点击config build path。

点击右边的add Exteranl jars 按钮，选择jar文件即可。

1. 曹萌和黄江之前用play framework做的项目并没有用paly官方提供的数据库访问方式，而是使用了mybatis。官方的方式要在routes里面配置数据库参数，然后建一个model的子类来访问数据库。

Mybatis是一个轻量级的java持久层框架。也就是用来访问数据库的。与之对应的常用的是hibernate，但这个太庞大了。传统的用JDBC方式访问数据库的方式需要自己写sql语句，这样真实很不方便，容易出错。还可能遇到sql注入。于是出现了java持久层框架。

Mybatis现在的官网是GitHub。下载地址是： <https://github.com/mybatis/mybatis-3/releases>

直接下载zip文件，解压后，将jar包导入项目里。不仅要导入mybatis-3.4.5.jar，还要导入lib下面的jar包。

导入玩mybatis的jar包后，还要导入数据库厂商提供的jdbc驱动。比如mysql的驱动jar包是：mysql-connector-java-5.1.44-bin.jar

2018年1月30日16:26:25

1. 第一个简单的mybatis程序，实现对mysql数据库的查询
2. 在mysql里面创建以person表，包含id，name，age字段
3. 按照前面写的导入mybatis和mysql的jar包
4. Mybatis的使用需要一个核心配置文件sqlMapConfig.xml 和映射文件UserMapper.xml

sqlMapConfig.xml里面配置了数据库的相关属性，包括数据库类型，这里是mysql，数据库名称和用户名密码，以及对应的映射mapper文件。

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!-- 配置开发环境，可以配置多个，在具体用时再做切换 -->

<environments default=*"test"*>

<environment id=*"test"*>

<transactionManager type=*"JDBC"*></transactionManager> <!-- 事务管理类型：JDBC、MANAGED -->

<dataSource type=*"POOLED"*> <!-- 数据源类型：POOLED、UNPOOLED、JNDI -->

<property name=*"driver"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/travel?characterEncoding=utf-8"* />

<property name=*"username"* value=*"root"* />

<property name=*"password"* value=*""* />

</dataSource>

</environment>

</environments>

<!-- 加载映射文件 mapper -->

<mappers>

<!-- 路径用 斜线（/） 分割，而不是用 点(.) -->

<mapper resource=*"mybatisDemo/UserMapper.xml"*></mapper>

</mappers>

</configuration>

UserMapper.xml里面定义了sql语句等信息

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace=*"UserMapper"*> <!-- 命名空间，名字可以随意起，只要不冲突即可 -->

<!-- 对象映射，可以不写 -->

<!-- 查询功能，resultType 设置返回值类型 -->

<select id=*"findAll"* resultType=*"mybatisDemo.Person"*> <!-- 书写 SQL 语句 -->

SELECT \* FROM Person

</select>

</mapper>

写一个测试类

package mybatisDemo;

import java.io.IOException;

import java.io.InputStream;

import java.util.List;

import org.apache.ibatis.io.Resources;

import org.apache.ibatis.session.SqlSession;

import org.apache.ibatis.session.SqlSessionFactory;

import org.apache.ibatis.session.SqlSessionFactoryBuilder;

public class CeshiMyBatis {

public void ceshi() throws IOException {

/\*\*

\* 1、获得 SqlSessionFactory

\* 2、获得 SqlSession

\* 3、调用在 mapper 文件中配置的 SQL 语句

\*/

String resource = "mybatisDemo/sqlMapConfig.xml"; // 定位核心配置文件

InputStream inputStream = Resources.getResourceAsStream(resource);

SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream); // 创建 SqlSessionFactory

SqlSession sqlSession = sqlSessionFactory.openSession(); // 获取到 SqlSession

// 调用 mapper 中的方法：命名空间 + id

List<Person> personList = sqlSession.selectList("UserMapper.findAll");

for (Person p : personList){

System.out.println(p);

}

}

}

这里主要是创建 SqlSessionFactory和SqlSession对象

2018年2月6日15:47:18

1. Mysql的sql语法和sql server的sql语法有区别，mysql查询表名不能加方括号

SELECT uid, pwd FROM USER WHERE ( uid = 'admin' AND pwd = 'admin' )

2018年3月12日23:29:52

1. 关于多媒体文件路径的问题

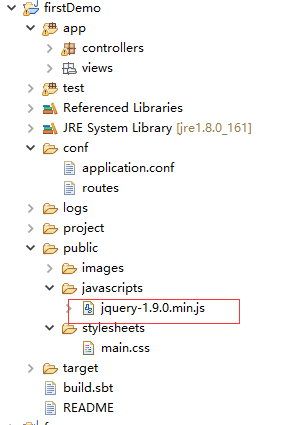
在application.conf文件里面有个host的属性

//host=<http://192.168.1.107/temp/>

表明了是文件的路径，这里设置了ip地址，之前一直不显示图片，是因为ip地址设置出错了。

2018年3月13日22:25:41

1. 原来paly framework新建的项目自带了jQuery



2018年3月26日16:27:33

1. 用提交form表单的方式演示一个简单的带参数的get请求

比如在浏览器里面输入：<http://192.168.43.232:9000/login?sex=girl>

浏览器地址栏输入的连接只能通过get请求访问

在playframework 的项目下的routes里面添加路由映射关系

GET /login @controllers.Application.login()

然后再Application类里面添加login方法，在login方法里面处理请求参数

**public** Result login()

{

DynamicForm requestData= Form.*form*().bindFromRequest();

String getsex= requestData.get("sex");

**return** *ok*("hello:"+getsex);

}

DynamicForm requestData= Form.*form*().bindFromRequest();

String getsex= requestData.get("sex");

这两句话就是得到请求参数的语句。

做这个的目的主要是学习Android使用httpurlconnection方法来发起get请求。

2018年3月26日22:55:48

1. 服务器端返回json数据的实例，（其实也可以发送json请求）

写了个hello的方法

**import** play.libs.Json;

**import** com.fasterxml.jackson.databind.JsonNode;

**import** com.fasterxml.jackson.databind.node.ObjectNode;

@BodyParser.Of(BodyParser.Json.**class**)

**public** Result hello()

{

JsonNode json = *request*().body().asJson();

ObjectNode result = Json.*newObject*();

// String name = json.findPath("name").textValue();

// if(name == null) {

// result.put("status", "KO");

// result.put("message", "Missing parameter [name]");

// return badRequest(result);

// } else {

result.put("status", "OK");

result.put("message", "Hello " + "jjj");

**return** *ok*(result);

// }

}

注意导入的包要正确。

当服务器请求这个方法的时候，会受到如下的数据：

{"status":"OK","message":"Hello jjj"}

**如果是Android客户端，也会受到同样的数据。这是json的标准格式。**

2018年3月28日16:07:31

1. 让服务器返回400的请求无效

**public** Result readF()

{

**return**(*badRequest*());

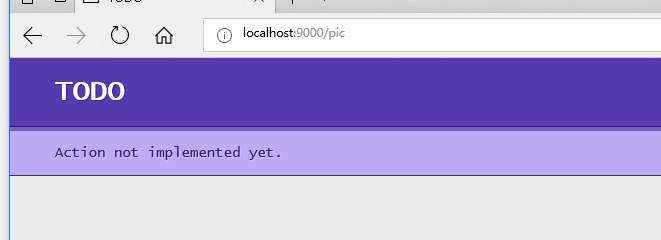
}

1. 重定向

return Results.redirect("login");

1. 返回 A 501 NOT\_IMPLEMENTED simple result.

**return** Results.*TODO*;



2018年4月2日10:18:38

1. 使用playframework自带的日志打印logger。

本来有非常强大的log4j的，但是由于不知道怎么使用，网上也没有关于在play中使用它的教程。然后发现play提供了commons-logging.jar包，这个也需要和log4j合用才行，我也没有找到相关教程。然后我查看了paly的api。发现play本身还自带了log打印功能。

在这个包里面：

D:\play-2.2.6\repository\local\com.typesafe.play\play\_2.10\2.2.6\jars\play\_2.10.jar

里面有个logger.class

要使用这个日志打印功能非常简单，什么初始化都不需要。直接在想要输出日志的地方使用logger即可，根据输出日志级别的不同，调用不同的方法。

Logger.*info*("hello");

Logger.*error*("hello world");

Logger.*of*("my.logger").info("Hello!");

Logger.*info*("A {} request was received at {}", *request*().method(), *request*().uri());

日志的查看：在项目的根目录下有个logs文件夹，里面有个application.log文件，用记事本打开就能看了。

1. playframework添加第三方jar包

本次需要添加jodd-3.4.10.jar这个包，使用里面的文件管理工具FieUtil，主要用它来保存上传的文件。

playframework的文件上传，并不是直接在eclipse里面添加buildpath就可以了。

首先在play项目的根目录下创建一个lib文件夹，注意这个文件夹不能随便取，不能是libs。

然后将需要的jar包放到这个目录下，这时play就可以正常使用这个jar包了。项目页面正常运行了。但是eclipse并不知道添加了这个jar包，因此代码里面会包红色错误，虽然不影响正常运行。这时，只需要在eclipse里面把它添加到buildpath就可以了。

2018年4月2日15:43:03

1. playframework使用*multipart/form-data实现文件上传，通过form表单的方式*

主要分为两大部分，前台文件上传界面和后台数据接收部分

1. 前台部分，使用在浏览器端提交form表单的方式，因此新建一个html文件：filettest.scala.html

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

<h1>hello world</h1>

<form action=*"/upload.do"* method=*"POST"* enctype=*"multipart/form-data"*>

<input type=*"file"* name=*"myfile"*>

<input type=*"submit"*/>

</form>

</body>

</html>

为了在浏览器里面能访问到这个页面，需要在routes里面添加路由

GET /filetest @controllers.Application.filetest()

然后再Application.java里面添加filetest方法，用于从服务器向浏览器返回这个html页面

**public** Result filetest()

{

**return** *ok*(views.html.filetest.*render*());

}

**说明：**在form表单中，提交方法必须是POST，enctype必须为*multipart/form-data*，表明是文件上传。

**<input type=*"file"* name=*"myfile"*>** 中的name为myfile是可以随便定义的，后台通过MultipartFormData对象的getFile（string key）方法来获取这个文件，通过参数来找到对应的提交文件，也就是说后台调用这个方法的时候，参数为string类型的myfile。

类似于okhttp中

builder.addFormDataPart(**"img"**, f.getName(), RequestBody.*create*(***MEDIA\_TYPE\_PNG***, f));

第一个参数为name，必须和后台一致。

在直接使用httpurlconnection上传文件时，使用：

fileSb.append(**"Content-Disposition: form-data; name=\""** + **"file"** + i + **"\"; filename=\""** + file.getName() + **"\""** + **"\r\n"**);  
fileSb.append(**"Content-Type: "** + **"image/jpeg"** + **"\r\n"**);

其中name属性就是要和后台保持一致的属性

（2）接下来就是后台数据接收的代码了，主要参考play官网教程。使用MultipartFormData 来接收。

官网例子如下：

public static Result upload() {

MultipartFormData body = request().body().asMultipartFormData();

FilePart picture = body.getFile("picture");

if (picture != null) {

String fileName = picture.getFilename();

String contentType = picture.getContentType();

File file = picture.getFile();

return ok("File uploaded");

} else {

flash("error", "Missing file");

return redirect(routes.Application.index());

}

}

它的例子中没有保存数据的部分，主要是先得到请求体，然后转换为MultipartFormData，通过该对象的getFile方法来获取FilePart对象，参数不同得到不同的文件。注意FilePart对象也有个getFileName方法，和MultipartFormData对象的getFileName方法结果不一样，后者得到的才是真实文件名。要获取到真实的File对象，还需要使用FilePart对象的getFile（）方法获取到File对象。获取到File对象后，就可以保存了。这里使用现成的工具即可，否则也可以使用java io的文件流来保存，但比较复杂。

这里使用jodd提供 的fileutil工具。jodd.io.FileUtil

前面已经讲过怎么在play中导入jar包了。

修改过的文件上传代码如下：

**public** Result fileupload2() **throws** IOException

{

// File file = request().body().asRaw().asFile();

// return ok("file uploaded:"+file.getName());

String destFilepath="D:\\temp3";

File destFile=**null**;

MultipartFormData body = *request*().body().asMultipartFormData();

FilePart picture = body.getFile("myfile");

**if**(picture!=**null**)

{

String fileName= picture.getFilename();

String contentType= picture.getContentType();

File file = picture.getFile();

destFile= **new** File(destFilepath+"\\"+fileName);

//makeDirs(destFilepath);

FileUtil.*mkdir*(destFilepath);

Logger.*error*(fileName);

FileUtil.*copy*(file, destFile);

**return** *ok*("file uploaded");

}

**else** {

*flash*("error", "Missing file");

**return** *redirect*(routes.***Application***.index());

}

}

1. 实现带参数的文件上传，其实这部分曹萌早就实现了

带参数的文件上传其实就是在上传文件的同时上传一个普通form表单，普通form表单本质上就是一些键值对。

前台增加一输入框

<form action=*"/upload.do"* method=*"POST"* enctype=*"multipart/form-data"*>

<tr>

<td>姓名:</td>

<td><input class=*"textbox"* type=*"text"* name=*"name"* data-options=*"required:true"*></input></td>

</tr>

<input type=*"file"* name=*"myfile"*>

<input type=*"submit"*/>

</form>

后台获取表单的代码如下：

DynamicForm requestData= Form.*form*().bindFromRequest();

String getName= requestData.get("name");

也可以在Form.*form*()的参数里指定接收的数据类型。

Form<model.Dataaccept> form = Form.*form*(model.Dataaccept.**class**);

model.Dataaccept uploadfile = form.bindFromRequest().get();

最后修改如下：

**public** Result fileupload4() **throws** IOException

{

String destFilepath="D:\\temp3";

File destFile=**null**;

MultipartFormData body = *request*().body().asMultipartFormData();

FilePart picture = body.getFile("myfile");

DynamicForm requestData= Form.*form*().bindFromRequest();

String getName= requestData.get("name");

Logger.*info*(getName);

**if**(picture!=**null**)

{

String fileName= picture.getFilename();

String contentType= picture.getContentType();

File file = picture.getFile();

destFile= **new** File(destFilepath+"\\"+fileName);

FileUtil.*mkdir*(destFilepath);

Logger.*error*(fileName);

FileUtil.*copy*(file, destFile);

**return** *ok*("file uploaded:"+getName);

}

**else** {

*flash*("error", "Missing file");

**return** *redirect*(routes.***Application***.index());

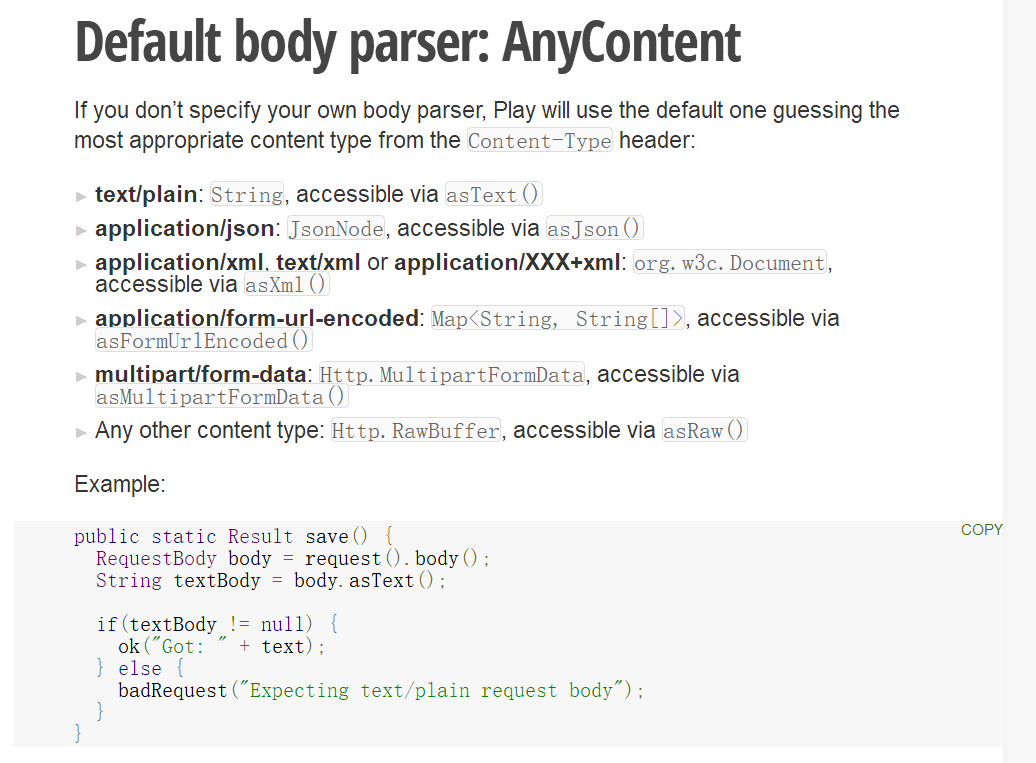
}

}

其实在接收部分，主要分为MultipartFormData和DynamicForm两部分来分别接收文件和表单数据，两者相互独立。

上传的时候不管是在Android还是浏览器里面，都不需要管上传顺序。

2018-4-4 10:47:17



1. playframework使用动态uri来访问本地资源

有时候URI是动态的，比如不同的用户id，不同的本地资源，不可能事先在routes文件里面定义好路由规则，因此需要使用动态的方式。

动态的方式可以使用正则表达式，也可以使用带参数的URI

比如：GET /files/\*name controllers.Application.download(name)

当访问的URI的files后面任意字符串都将被当做参数传递给name。然后再download里面处理这个参数

我的实例：GET /files/\*name @controllers.Application.getfile(name:String)

**public** Result getfile(String name)

{

String path="D:\\temp4\\"+name;

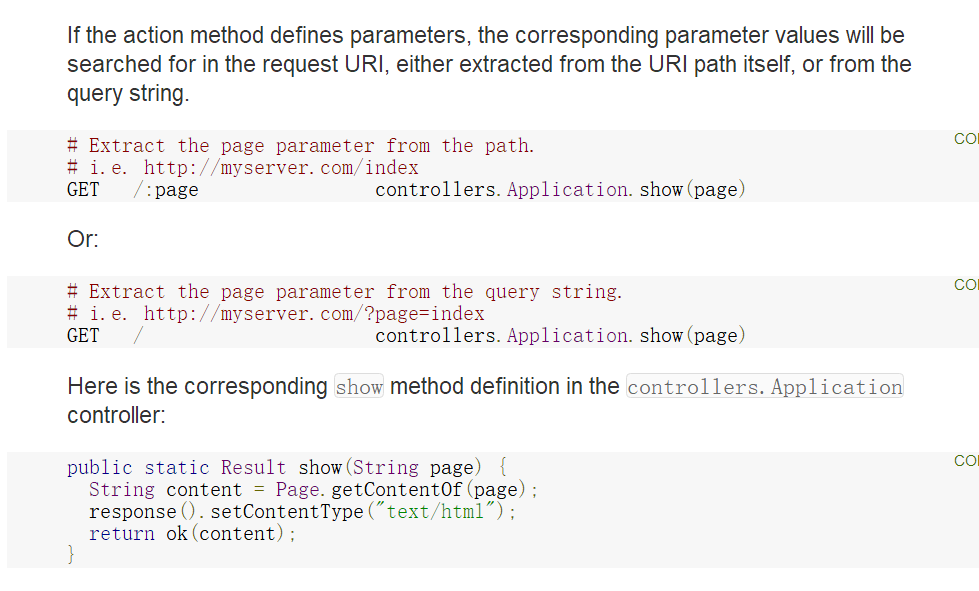
File file = **new** File(path);

**return** *ok*(file);

}

在浏览器里面调用方法<http://localhost:9000/files/2.jpg>

1. jpg将作为参数传递给name参数，然后拼接到path路径，返回一个文件给客户端。这样就形成了一个文件的URI。



动态uri传递参数，使用/:page的形式，

GET /:page controllers.Application.show(page)

调用方法为：# i.e. <http://myserver.com/index>

这种是直接在连接后面用斜线添加参数的方式，

如果uri的pattern没有设置内容，但是调用的方法有参数，这种情况也是可以的，

# Extract the page parameter from the query string.

# i.e. http://myserver.com/?page=index

GET / controllers.Application.show(page)

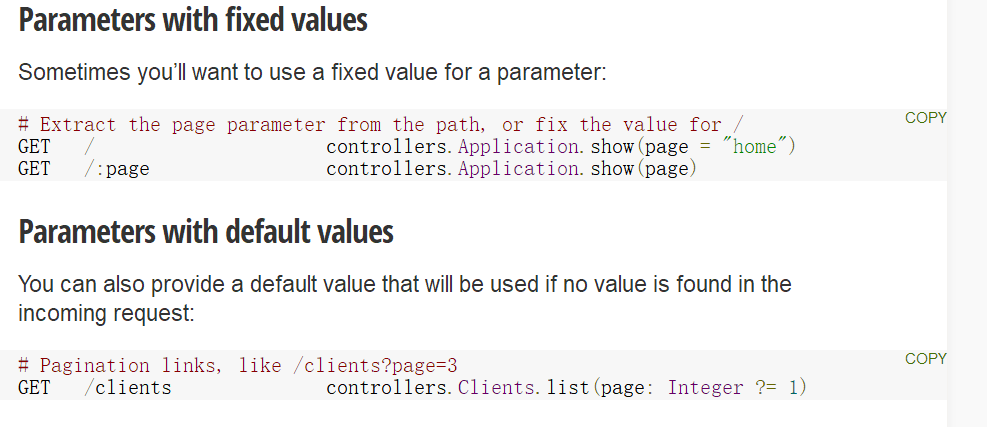
在原来的uri后面用？链接参数，就像用get方式提交form表单一样。

注意理由中参数可以设置数据类型，和默认值

@controllers.Application.getfile(name:String)

GET /unitmeasure/list @controllers.Unitmeasure.list(page:Integer ?=1,pageSize:Integer ?=20,sort:String ?="",order:String ?="")

1. 动态uri参数可以设置固定值或者默认值



用= 表示固定值，用？=表示默认值

比如工程自带的一个路由转换：

GET /assets/\*file controllers.Assets.at(path="/public", file)

表明at方法的path参数固定为“/public”,第二个参数为任意字符串。

# Pagination links, like /clients?page=3

GET /clients controllers.Clients.list(page: Integer ?= 1)

表示page参数默认为1，

路由冲突的话，会以第一个路由为标准。

1. 在return ok（）的参数里里面，可以设置一个file对象，用于向客户端返回一个文件，这个方法有很多重载方法。
2. 如果第一个参数数是File类型，第二个参数是string类型的文件名，如果文件时是图片，则在浏览器里面会直接显示这幅图片。
3. 如果只有一个参数，是File类型的，则在浏览器里面就只能是下载这个文件了。

2018年4月5日11:23:12

1. playframework的全局设置

类似于android的application类，是项目一启动就运行的，或者设置一些全局变量方法等。

play的全局类的定义要定义在root包里面，也就是没有包名，在eclipse里面是default package。

定义方法是在项目的app上右键，添加一个类Global，然后就会出现一个default package下的Global.java文件，但也会在app下出现一个多了的Global.java文件，把它删掉就可以了。

全局类需要继承于GlobalSettings 这个类，并且可以在里面写onStart方法和onStop方法。

import play.\*;

public class Global extends GlobalSettings {

public void onStart(Application app) {

Logger.info("Application has started");

}

public void onStop(Application app) {

Logger.info("Application shutdown...");

}

public Promise<SimpleResult> onError(RequestHeader request, Throwable t) {

return Promise.<SimpleResult>pure(internalServerError(

views.html.errorPage.render(t)

));

}

}

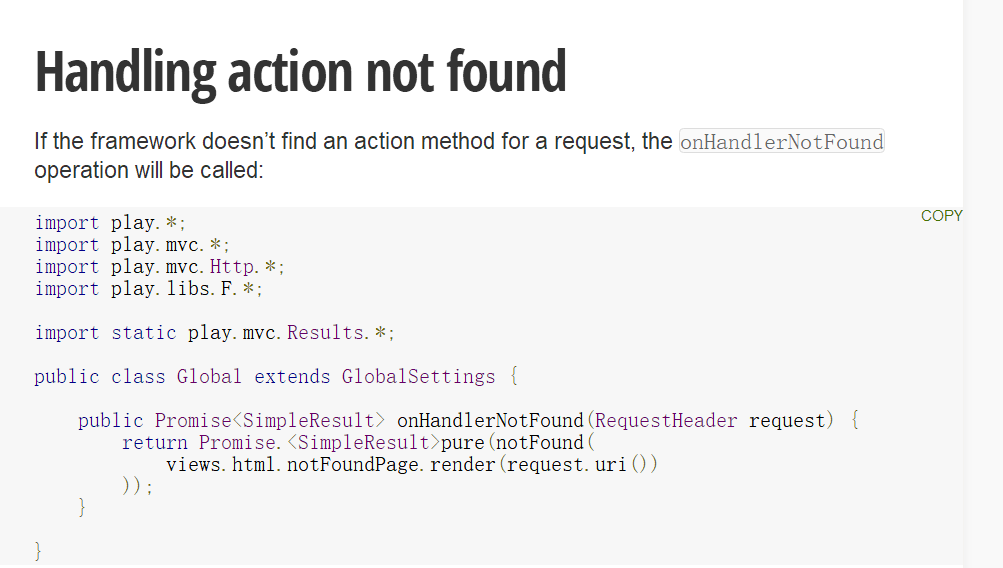
注意需要导入三个包：

**import** play.Application;

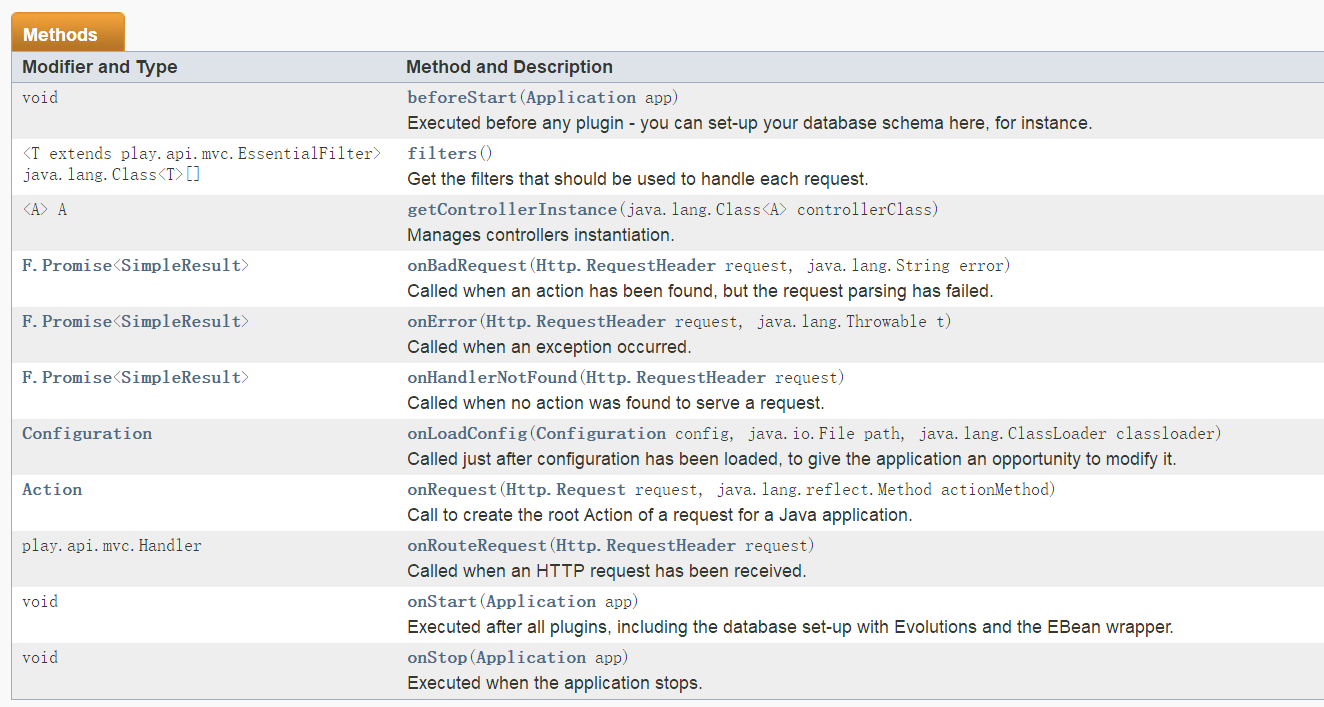
**import** play.GlobalSettings;

**import** play.Logger;

实测确实在项目启动和关闭的时候能打印日志。但是并不是在命令行里面输入run，就能运行onStart方法的，而是需要在浏览器里发起请求才可以。然后在命令行里面输入ctrl+d才会运行onStop方法。



**GlobalSettings**类中的方法定义：

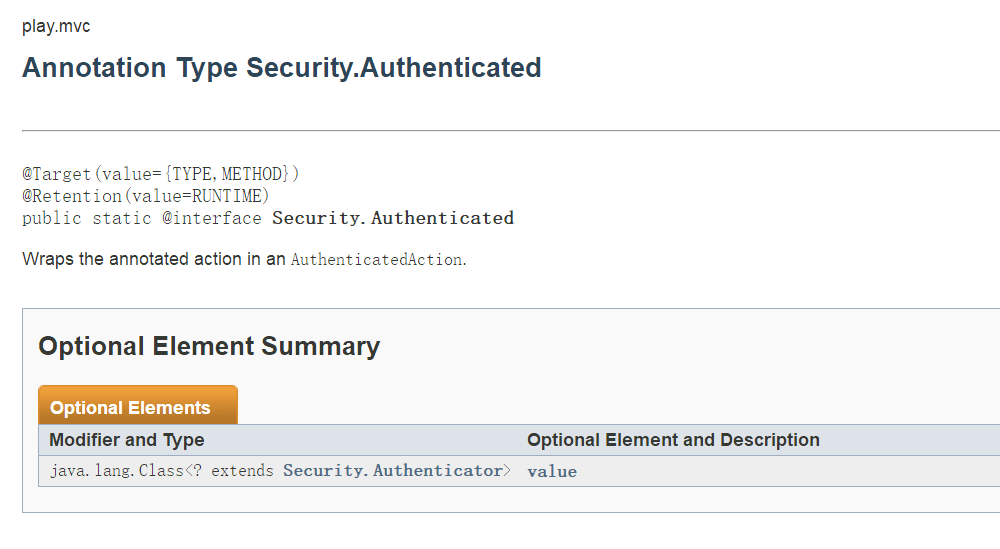


2018年4月6日19:50:00

1. play framework的安全认证

也就是是实现在登录之前访问服务器的除了登录页面之外的其他请求都跳转到登录页面。

play使用了注解annotation的方式来实现安全认证。



可以看出**Security.Authenticated**这个接口是一个自定义的注解。

## 自定义Annotation

Annotation定义格式：

public @interface Annotation名称

{

数据类型 变量名称():

}

定义一个简单的Annotation

**MyDefaultAnnotationNoneParam.java**

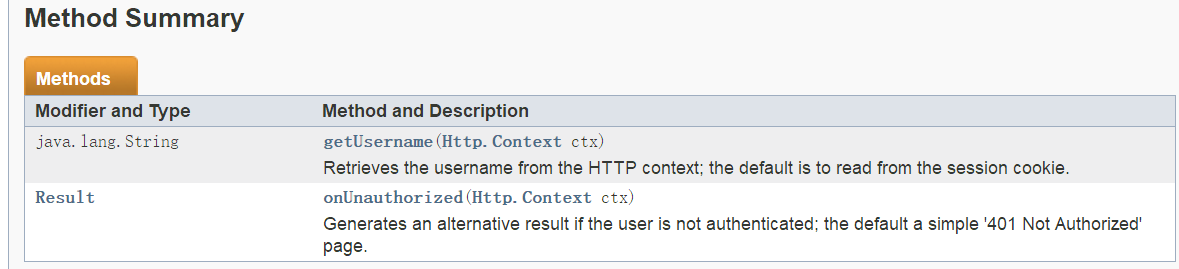
public @interface MyDefaultAnnotationNoneParam{

}

自定义annotation可以用在方法前，也可以用在类的定义之前，可以设置参数。

**Security.Authenticated**这个注解里面有个数据value是java的class，也就是说传递一个类给这个数据，使用的是java的反射机制。并且这个类继承自**Security.Authenticator**这个类。

Security.Authenticator类中的方法定义如下：



[Security.Authenticated这个注解上面使用了@Retention](mailto:Security.Authenticated这个注解上面使用了@Retention)这个注解，并且value值是runtime，

**Retention**这个注解是设置该注解的作用范围的，有三个取值（用枚举的方式来固定几个取值）：source，class，runtime

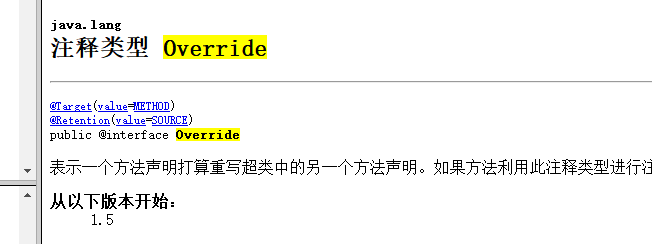
枚举类型为**RetentionPolicy**



**runtime**表明在程序执行的时候也会使用这个注解。

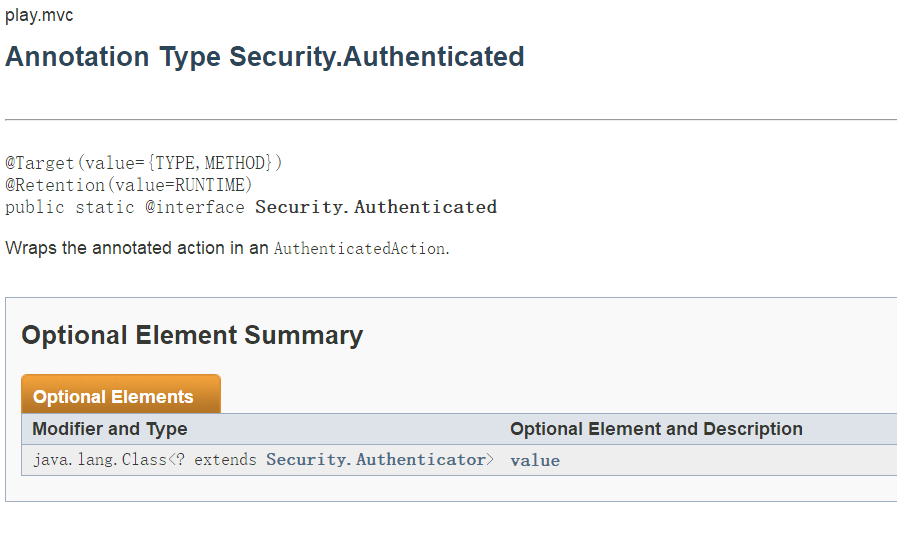
**soruce** 表明该注解只会用在java源代码中.java文件，

比如override这个注解的retention就是source



2018年4月7日10:25:28

1. play的安全认证功能的实现过程，即实现未登录的情况下，访问服务器先进入登录界面

play的安全认证使用了注解的方式来实现，自定义注解的接口为Security.Authenticated,定义如下：

从定义来看，有两点需要注意：（1）该注解可以用在类和方法前，也就是说如果类或者方法前有这个注解，调用前先调用注解的内容。

（2）注解中的内容value()必须是一个继承自[**Security.Authenticator**](https://www.playframework.com/documentation/2.2.x/api/java/play/mvc/Security.Authenticator.html)的子类，该类有两个方法：



因此，需要复写这个两个方法，getUsername方法，从session中获取指定的session，比如当前登录的用户名，如果找不到，表明未登录，因此需要转到未登录的方法onUnauthorized，在这里再决定是抓到登录页面还是返回给客户端别的信息。

**package** controllers;

**import** play.mvc.Http.Context;

**import** play.mvc.Security;

**import** play.mvc.Result;

**public** **class** Secured **extends** Security.Authenticator {

@Override

**public** String getUsername(Context ctx) {

**return** ctx.session().get("userName");

}

@Override

**public** Result onUnauthorized(Context ctx) {

//return redirect(routes.Application.login());

**return** *badRequest*("未授权");

}

}

最后就是使用该注解了，使用方法非常简单，但是有一点要非常注意的是，只能用在Controller的子类或者里面的方法上面。因为Controller类才是处理http请求和具体方法之间联系的类。

在类上的使用：

@Security.Authenticated(Secured.**class**)

**public** **class** Main **extends** Controller{

**public** Result index() {

**return** ok(views.html.main.render());

}

}

在方法上的使用：

@Security.Authenticated(Secured.**class**)

**public** Result hello2()

{

**return** *ok*("已经认证");

}

使用就是非常简单，先定义一个继承自Security.Authenticator的类，复写里面的getUsername和onUnauthorized方法。

[然后再类或者方法上调用注解@Security.Authenticated(Secured.class)](mailto:然后再类或者方法上调用注解@Security.Authenticated(Secured.class))，参数是刚刚复写的子类。

疑问:

（1）关于这个注解具体是在哪里通过反射来运行的，没有找到具体的实现过程，应该是在Controller类的某个方法里面吧

（2）在这个方法中：**public** String getUsername(Context ctx) {

**return** ctx.session().get("userName");

}

只是通过session获取用户名，如果别的人登录了，即session里面有个userName，那么别的未登录的人就不需要登录了页也能通过认证了，是不是这样呢？

我的理解是：session是在浏览器登录后创建的键值对，浏览器的一次访问就会创建一session，如果浏览器关闭了session就会注销，并且不同的用户间的session是相互隔离的。这样，不同的用户就不会相互影响。

2018年4月7日09:57:16

1. play的授权认证机制，使用deadbolt 2

<https://github.com/schaloner/deadbolt-2>

<http://deadbolt.ws/>

2018年4月8日21:51:48

1. **play中的form**

form是Form类中的一个方法，而Form是play.data包中的一个类。主要是用于处理表单的数据的。

在研究黄江写的离线编辑系统的登录功能时，发现他给登录页面的html传递了一个Form对象，

**public** Result login() {

**return** *ok*(

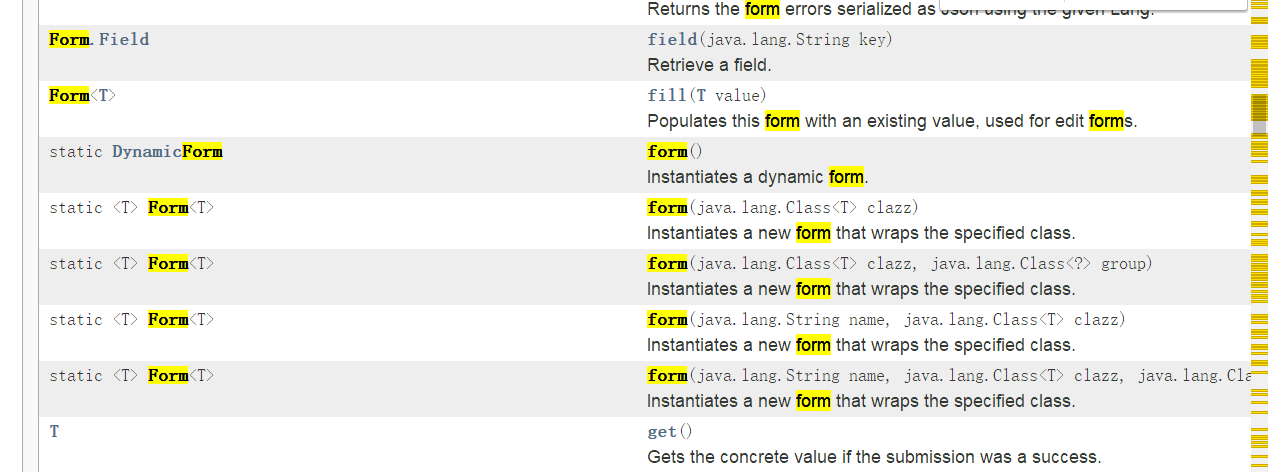
login.render(*form*(Login.**class**))

);

}

直接跳转到form的定义,（注意form是小写），发现进入到views.html.helper包下的form.class文件里面了，而paly官方的api里面根本找不到这个包的解释。

实际上应该是进入play.data包的Form类的form方法里面。



这个form方法可以传递多种参数，然后返回Form类的对象。

比如：static <T> Form<T> form(java.lang.Class<T> clazz)

可以传递一个类的定义，然后返回Form对象。

比如曹萌在baidu轨迹项目上的登录的实现过程：

**import** play.data.Form;

**public** Result login()

{

Form<model.User> puserForm=Form.*form*(model.User.**class**);

model.User pUser=puserForm.bindFromRequest().get();

*session*("uid", pUser.getUid());

*session*("pwd",pUser.getPwd());

model.UserExample pUserExample=**new** model.UserExample();

model.UserExample.Criteria pCriteria=pUserExample.or();

pCriteria.andUidEqualTo(pUser.getUid());

pCriteria.andPwdEqualTo(pUser.getPwd());

List<model.User> puserlists=**new** ArrayList<User>();

puserlists=pUserMapper.selectByExample(pUserExample);

**if**(puserlists.size()>0){

**return** *redirect*(routes.Application.main());

}

**else**{

**return** *redirect*(routes.Application.index());

}

}

我写的一个简单的登录过程，如果没有在参数里面指定数据类型，则返回一个DynamicForm

**public** Result login()

{

DynamicForm requestData= Form.*form*().bindFromRequest();

String getsex= requestData.get("sex");

//return ok("hello:"+getsex);

**return** *ok*("hello:"+getsex).as("text/html");

}

回到黄江写的登录过程，

**public** **static** **class** Login {

**public** String email;

**public** String password;

**public** String validate() {

//models.AuthorisedUser user= findUserByNamePass(email, password);

String usernameStr=play.Play.*application*().configuration().getString("admin.user");

String passwordStr=play.Play.*application*().configuration().getString("admin.password");

**if**(email.equals(usernameStr) && password.equals(passwordStr) ) {

**return** **null**;

}

**return** "用户名或密码错误";

}

}

**public** Result login() {

**return** *ok*(

login.render(*form*(Login.**class**))

);

}

然后再login.scala.html里面的第一个行写入导入的参数，用scala实现

@(form: Form[Application.Login])

2018年4月9日08:54:32

1. play可以利用scala给html传递一个对象，比如黄江写的登录过程，给login.scala.html传递了一个Form对象，Form里面又绑定了一个Login.class类。

官方例子：

@(customer: Customer, orders: List[Order])

<h1>Welcome @customer.name!</h1>

<ul>

@for(order <- orders) {

<li>@order.getTitle()</li>

}

</ul>

其中Customer和Order都是自定义的类。name是Customer里面的变量。在html里面调用方法和java的调用对象的属性是一样的，只是在前面加了个@符号，表示是scala代码。也可以调用方法。

然后是在html里面第一行引入的格式：

Parameter types are specified using a suffix syntax. Generic types are specified using the [] symbols instead of the usual <> Java syntax. For example, you write List[String], which is the same as List<String> in Java.

上面是官网的解释：数据类型使用后缀语法，也就是先写对象名，然后冒号和类名。如果有多个对象，用逗号隔开，写下一个对象。然后是对泛型的引用要使用[]，而不是<>,在java中泛型是在<>中指定数据类型。而这里要用[]

比如：@(form: Form[Application.Login])

@(customer: Customer, orders: List[Order])

第一个是Form类的泛型，第二个是List的泛型。

突然想起曹萌写的主页面传递用户名的例子了，用了个allNames方法将查询到的所有用户名组合成一个字符串传递给html，然用JavaScript解析这个字符串。其实也可以传递一个List集合，这样更明显。

可以这样循环遍历：

<ul>

@for(order <- orders) {

<li>@order.getTitle()</li>

}

</ul>

1. jQuery的表单验证

发现黄江写的登录界面的表单提交部分有如下这句话：

**var** isValid = $("#loginForm").form('validate');

关键是不知道后面部分form('validate')是什么意思，这个form是在html首行传递进来的对象吗？

不是的，这里是精确到表单验证，括号里面的validate是jquery的form方法的参数关键字。比如还有清空表单为 $('#loginForm').form('clear');

表单验证是什么意思呢？  
表单验证就是判断表单里面的输入项的格式是否正确的一种方法。需要引入jquery的validate插件。

比如判断输入框是否输入了内容，输入的是否是正确的email格式，是否是数字能，如果没有输入正确还能给出提示。

网上详细介绍：<http://www.runoob.com/jquery/jquery-plugin-validate.html>

比如： required:true 必须输入的字段。

黄江的用法是在input标签里面添加：data-options=*"required:true"*

网上的例子：

<fieldset>

<legend>输入您的名字，邮箱，URL，备注。</legend>

<p>

<label for="cname">Name (必需, 最小两个字母)</label>

<input id="cname" name="name" minlength="2" type="text" required>

</p>

<p>

<label for="cemail">E-Mail (必需)</label>

<input id="cemail" type="email" name="email" required>

</p>

<p>

<label for="curl">URL (可选)</label>

<input id="curl" type="url" name="url">

</p>

<p>

<label for="ccomment">备注 (必需)</label>

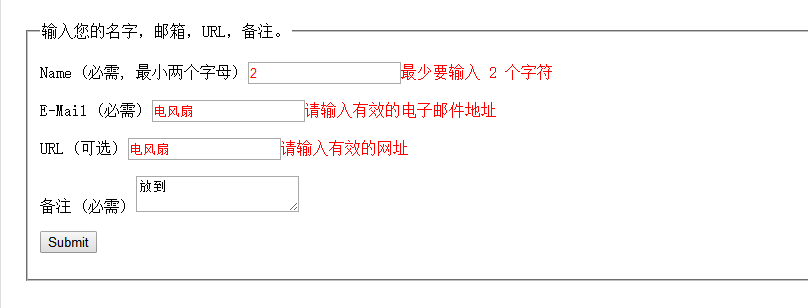
<textarea id="ccomment" name="comment" required></textarea>

</p>

<p>

<input class="submit" type="submit" value="Submit">

</p>



还可以指定提示内容：

$().ready(function() {

// 在键盘按下并释放及提交后验证提交表单

$("#signupForm").validate({

rules: {

firstname: "required",

lastname: "required",

username: {

required: true,

minlength: 2

},

password: {

required: true,

minlength: 5

},

confirm\_password: {

required: true,

minlength: 5,

equalTo: "#password"

},

email: {

required: true,

email: true

},

topic: {

required: "#newsletter:checked",

minlength: 2

},

agree: "required"

},

messages: {

firstname: "请输入您的名字",

lastname: "请输入您的姓氏",

username: {

required: "请输入用户名",

minlength: "用户名必需由两个字母组成"

},

password: {

required: "请输入密码",

minlength: "密码长度不能小于 5 个字母"

},

confirm\_password: {

required: "请输入密码",

minlength: "密码长度不能小于 5 个字母",

equalTo: "两次密码输入不一致"

},

email: "请输入一个正确的邮箱",

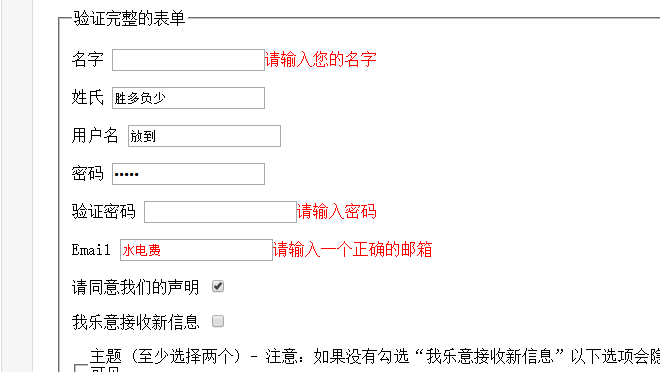
agree: "请接受我们的声明",

topic: "请选择两个主题"

}

})

});



接着就是判断是否验证成功，如果验证成功就提交表单，否则提示修改

**function** submitForm(){

**var** isValid = $("#loginForm").form('validate');

//alert(isValid);

**if**(isValid)

$('#formSubmit').click();

}

其实发现根本没有必要给登录界面传递一个Form对象，真不知道有什么用，还有用户名的value部分是什么意思id=*"email"* placeholder=*"用户名*[*"* value=*"@form("*email").value](mailto:%22%20value=%22@form(%22email%22).value)"

form的action定义如下：

<form action=*"@routes.Application.authenticate"* method=*"POST"* id=*"loginForm"* novalidate>

其实action可以写成login的，因为routes里面定义为：

POST /login @controllers.Application.authenticate()

实际上是一个意思

1. 发现登陆后认证部分一直无效

**public** Result authenticate() {

// return redirect(

// //routes.Main.index()

//

// );

**return** *ok*("hello");

}

于是写成返回ok的代码，发现能正常在浏览器显示hello。说明action路由等都没有问题，有问题的就是redirect部分。

其实代码没有错，routes.Main.index()确实是转到主页，这是在java中的写法，在html中要加@符号。

关键是main的index方法有安全验证，也就是没有设置session，就会一直跳转到登录页面

@Security.Authenticated(Secured.**class**)

**public** Result index() {

**return** *ok*("hello world");

// return ok(views.html.main.render());

}

因此需要在authenticate 方法里面添加session

改为： **public** Result authenticate() {

*session*("userName","guozhagn");

**return** *redirect*(

routes.Main.index()

);

// return ok("hxcfd");

}

后就能正常跳转了。

回到黄江自己写的登录验证代码：

**public** Result authenticate() {

Form<Login> loginForm = *form*(Login.**class**).bindFromRequest();

**if**(!loginForm.hasErrors()){

// session("userName", loginForm.get().email);

*session*("userName","guozhagn");

**return** *redirect*(

routes.Main.index()

);

}**else**{

**return** *badRequest*(login.render(loginForm));

}

}

注释掉的是原来他写的，下面是我写的。他的代码一直报错，找不到原因，改为我的就可以了。

其中有个关键的问题出在Login类了，看下他的定义：

**public** **static** **class** Login {

**public** String email;

**public** String password;

**public** String validate() {

//models.AuthorisedUser user= findUserByNamePass(email, password);

String usernameStr=play.Play.*application*().configuration().getString("admin.user");

String passwordStr=play.Play.*application*().configuration().getString("admin.password");

**if**(email.equals(usernameStr) && password.equals(passwordStr) ) {

**return** **null**;

}

**return** "用户名或密码错误";

}

}

Login类本来是用来接收form表单提交的内容的，里面的数据格式应该和form表单中的各个input的name值一致，否则就不用自定义类来接收，用DynamicForm来接收，就不用指定数据类型了。

黄江写的这个Login类里面有个validate方法，感觉是画蛇添足。

在接收表单数据时，使用

Form<Login> loginForm = *form*(Login.**class**).bindFromRequest();

无法正常接收，因为数据格式不匹配。导致loignForm为null。

将Login类中的validate方法注释掉就可以了。

通过调试发现原来是提交的表单莫名其妙地被添加了内容，本来密码的name是password，接收到的就是data[password],email也变成了data[email]了，这样Login自然就不能接收了，也许Login类中的validate方法是可以存在的。

我是改成DynamicForm然后调试才发现的。

**public** Result authenticate() {

// Form<Login> loginForm = Form.form(Login.class);

// Login user = loginForm.bindFromRequest().get();

//

DynamicForm requestData= Form.*form*().bindFromRequest();

String getsex= requestData.get("name");

**return** *ok*(getsex);

// if(!loginForm.hasErrors()){

// // session("userName", loginForm.get().email);

// session("userName",user.email);

// return redirect(

// routes.Main.index()

// );

// }else{

// return badRequest(login.render(loginForm));

// }

}

可是曹萌也是类似的写法，为什么他的没有被添加data[]呢

也许黄江给login的html传递Form对象是为了判断如果输入账号密码错误，能在页面上显示。

html中部分代码如下：

@if(form.hasGlobalErrors) {

<p style="text-align:*center*;color:*red*" class=*"error"*>

@form.globalError.message

</p>

}

@if(flash.contains("success")) {

<p style="text-align:*center*;color:*blue*" class=*"success"*>

@flash.get("success")

</p>

}

2018年4月9日11:54:19

把在html中设计form的部分换成曹萌写的代码，提交发现还是增加了data[],而曹萌做的轨迹调查项目怎么没有出现这个问题呢。

2018年4月9日15:27:36

1. 真是遇到很奇葩的现象了，上午一直想不通的问题，到了下午去没有了，一切正常。

上午不管是用了定义了类型的Form还是DynamicForm都不能正常接收道提交的表单数据，因为调试发现收到的数据的name属性增加了data[]。

在上午我将代码改为：

DynamicForm requestData= Form.*form*().bindFromRequest();

String name= requestData.get("data[email]");

String password = requestData.get("data[password]");

就能正常接收到数据，一度怀疑是jquer的问题，换成曹萌的前端代码无效。到了下午直接在我自己写的firstdemo里面测试接收表单数据，调试结果很惊奇，发现接收到数据的name都被play用data[]包裹了，然后要想获取数据不用管data[],直接在get方法的参数里写前端form里面的name属性的值就可以了。

也就是说下面的代码可以用了：

DynamicForm requestData= Form.*form*().bindFromRequest();

String name= requestData.get("email");

String password = requestData.get("password");

真是奇葩。

2018年4月9日15:42:44

1. 给form传递的自定义类型里面可以有方法，官网就是

public class User {

@Constraints.Required

public String email;

public String password;

public String validate() {

if (authenticate(email, password) == null) {

return "Invalid email or password";

}

return null;

}

}

validate method is called after checking annotation-based constraints and only if they pass. If validation passes you must return null . Returning any not-null value (empty string or empty list) is treated as failed validation.

官网解释说validate方法如果验证通过必须返回null，还可以返回其他类型，但都表示验证失败

The validate-method can return the following types: String, List<ValidationError> or Map<String,List<ValidationError>>

可实际问题是我只要加了validdate方法就不能用Form<User>的形式接收表单数据了。

之前黄江是在Application类里面定义了一个静态类Login来作为登录用户的数据类型。一直无法用Form方式接收数据。然后我在外面增加了一个User.java文件，写了个几乎一样的类User。能够正常用Form接收数据了，但是只要添加了validate方法就不能用了。

现在不管了，不想用它自己的validate方法了。直接在接收到数据后写验证逻辑更简洁明了，曹萌的就是这样的方法。官网说的validate方法调用是在检查注解约束后执行。官网的注解如下：

@Constraints.Required

public String email;

public String password;

可问题是我根部不知道这个注解是来自哪个包的，官网根本没有说明。导致无法使用。

真的可以彻底放弃这个validate方法了。

原来这个validate方法还是有好处的，如果验证失败，可以返回String或者包含ValidationError的List或者Map，然后这个form就用拥有了error。

然后 **l**oginForm.hasErrors() 这句代码就能返回true，再通过

return badRequest(login.render(loginForm));

传递给登录html的form，在html里面用scala判断是否有error，

有的话显示错误信息。

其实也可以给html增加一个新的接收数据，比如String类型的error，然后在html判断是否有error，有就显示。

由于无法在validate方法里面给form添加error。查阅paly的api中Form类的说明，发现除它除了有[hasErrors](https://www.playframework.com/documentation/2.2.x/api/java/play/data/Form.html#hasErrors())()方法外还有globalError()方法，还有 [**reject**](https://www.playframework.com/documentation/2.2.x/api/java/play/data/Form.html#reject(java.lang.String))(java.lang.String error)

这个方法可以给form添加一个errro。

[**discardErrors**](https://www.playframework.com/documentation/2.2.x/api/java/play/data/Form.html#discardErrors())() 删除form的所有error

因此在验证失败的部分添加如下的代码：

loginForm.reject("用户名或者密码错误");

**return** *badRequest*(login.render(loginForm));

就能在登录界面提示密码或账号错误了。

于是完整的登录验证代码如下：

**public** Result authenticate() {

Form<User> loginForm = Form.*form*(User.**class**).bindFromRequest();

String usernameStr=play.Play.*application*().configuration().getString("admin.user");

String passwordStr=play.Play.*application*().configuration().getString("admin.password");

loginForm.discardErrors();

User user= loginForm.get();

**if**(user.email.equals(usernameStr)&&user.password.equals(passwordStr))

{

*session*("userName",user.email);

**return** *redirect*(routes.Main.index());

}

**else**

{

loginForm.reject("用户名或者密码错误");

**return** *badRequest*(login.render(loginForm));

}

}

特色：(1)具有表单验证功能，能提示是否输入正确

（2）具有安全验证功能，如果没有登录就访问主页，先进入登录页面

（3）利用给html传递一个form对象，并且form对象可以携带error信息，使得前端能显示用户名或者密码错误等信息。也可以不用传递form对象给html，直接传递一字符串或者别的数据给html就可以了。

**总结登录功能的设计流程：**

1. 首先是设计系统的默认路由，即在URL中没有任何后缀的路由

**GET / @controllers.Main.index()**

1. 给Main.index()设计安全验证，当验证通过，直接跳转到主页面，否则进入登录页面。

在登录验证通过后，先增加session，再次进入Main.index() 这时因为已有session了，所以注解的安全认证就能通过，就能直接进入主页面了。

（3）安全验证的设计，在想要的方法前增加注解@Security.Authenticated(Secured.**class**)

（4）登录验证，本来应该是读取数据库中的用户名和密码，再匹配的，这里简化为访问play的配置文件了，方法为：

String usernameStr=play.Play.*application*().configuration().getString("admin.user");

1. 表单验证使用了jquery的validate方法

**function** submitForm(){

**var** isValid = $("#loginForm").form('validate');

//alert(isValid);

**if**(isValid)

$('#formSubmit').click();

}

1. 获取表单数据可以用DynamicForm，也可以用Form<T>在泛型里面指定数据类型为自定义的User类。

Form<User> loginForm = Form.*form*(User.**class**).bindFromRequest();

User user= loginForm.get();

1. 可以给form对象增加erro信息，增加使用reject（）方法，删除使用discardErrors方法。
2. 在html里面动态地显示错误信息，使用传递的form对象和scala方法。

@if(form.hasGlobalErrors) {

<p style="text-align:*center*;color:*red*" class=*"error"*>

@form.globalError.message

</p>

}

2018年4月9日21:05:31

1. 曹萌和黄江的主界面的用法有很大的不同，虽然都是使用的esay-ui。黄江用的是和bui类似的传递一个json字符串来表示菜单，而曹萌的是直接使用ul标签来添加菜单。

用前面的方法可以增加动态菜单的功能，进一步能区分不同的权限的用户。

**var** \_menus = {

"menus": [

{

"menuid": "1",

"icon": "icon-sys",

"menuname": "数据列表",

"menus": [{

"menuid": "12",

"menuname": "上传文件列表",

"icon": "icon-nav",

"child": [

{

"menuid": "120",

"menuname": "上传文件",

"url": "@routes.Uploadfile.index"

}

]

}]

}]

};