链滴

mini-spring 第四期：类扫描器，控制器初始化

作者：ChenforCode
原文链接：https://ld246.com/article/1573035552395
来源网站：链滴

1.在core包中定义ClassScanner,并定义scanClass方法，目的是给定一个包路径，扫描出该包下面的有类

主要的流程是：

@1.拿到改包路径下的所有资源，遍历

@2.如果是jar包，就继续遍历这个jar包，把jar包的需要的class扫描进来，也就是以我们包名的开头的class文件

@3.jar包中的jarentryname的格式是cn/chenforcode/xxx.class，所以必须转换成包名，然后才可用类加载器进行扫描。

```
package cn.chenforcode.core;

import java.io.IOException;
import java.net.JarURLConnection;
import java.net.URL;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;
import java.util.jar.JarEntry;
import java.util.jar.JarFile;

public class ClassSacnner {
    public static List<Class<?>> scanClass(String packageName) throws IOException, ClassNot
oundException {
        List<Class<?>> classList = new ArrayList<>();
        //根据包名获取路径
        String path = packageName.replace(".", "/");
        ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
        //获取到这个路径下的所有资源
        Enumeration<URL> resources = classLoader.getResources(path);
        //遍历这些资源
        while (resources.hasMoreElements()) {
            URL resource = resources.nextElement();
            //如果这个文件是一个jar包，需要单独处理
            if (resource.getProtocol().contains("jar")) {
                //拿到jar包连接
                JarURLConnection jarURLConnection = (JarURLConnection) resource.openConnecti
n();
                //拿到jar包路径
                String jarFilePath = jarURLConnection.getJarFile().getName();
                classList.addAll(getClassesFromJar(jarFilePath, path));
            } else {
                //todo 处理不是jar包类型的资源
            }
        }
        return classList;
    }

    public static List<Class<?>> getClassesFromJar(String jarFilePath, String path) throws IOEx
eption, ClassNotFoundException {
        //todo 从jar包中获取所需要的类
        List<Class<?>> classes = new ArrayList<>();
        //获取这个jar文件
```

```
        JarFile jarFile = new JarFile(jarFilePath);
        //拿到jar文件的entry集合
        Enumeration<JarEntry> jarEntries = jarFile.entries();
        //遍历jar文件
        while (jarEntries.hasMoreElements()) {
            JarEntry jarEntry = jarEntries.nextElement();
            //拿到某个类的entryname，集体格式如cn/chenforcode/xxx.class
            String entryName = jarEntry.getName();
            //判断这个文件是否是我们需要的
            if (entryName.startsWith(path) && entryName.endsWith(".class")) {
                String classFullName = entryName.replace("/", ".")
                        .substring(0, entryName.length() - 6);
                //如果是把它加载进去，并加入class的list返回
                classes.add(Class.forName(classFullName));
            }
        }
        return classes;
    }
}
```

2.开始编写handler

```
package cn.chenforcode.web.handler;

import java.lang.reflect.Method;

public class MappingHandler {
    private String uri;
    private Method method;
    private Class<?> controller;
    private String[] args;

    MappingHandler(String uri, Method method, Class<?> cls, String[] args) {
        this.uri = uri;
        this.method = method;
        this.cls = cls;
        this.args = args;
    }
}
```

3.创建handlerManager来管理mappingHandler

4.创建resolveMappingHandler方法，对给出的类列表进行遍历

```
/**
     * @Author <a href="http://www.chenforcode.cn">PKUCoder</a>
     * @Date 2019/11/6 4:50 下午
     * @Param [classList]
     * @Return void
     * @Description 对给出的一个类列表遍历，解析其中的方法
     **/
    public static void resolveMappingHandler(List<Class<?>> classList) {
        for (Class<?> cls : classList) {
```

```
            if (cls.isAnnotationPresent(Controller.class)) {
                parseHandlerFromController(cls);
            }
        }
    }
```

5.创建parseHandlerFromController方法，对某个类中的方法进行解析

```
/**
     * @Author <a href="http://www.chenforcode.cn">PKUCoder</a>
     * @Date 2019/11/6 5:17 下午
     * @Param [cls]
     * @Return void
     * @Description 对某个类中的方法进行解析
     **/
    private static void parseHandlerFromController(Class<?> cls) {
        Method[] methods = cls.getDeclaredMethods();
        for (Method method : methods) {
            //如果没有被requestMapping注解修饰就不用解析了
            if (!method.isAnnotationPresent(RequestMapping.class)) {
                continue;
            }
            //得到mapping需要的参数
            //获取uri
            String uri = method.getDeclaredAnnotation(RequestMapping.class).value();
            //获取参数
            List<String> paramNameList = new ArrayList<>();
            //遍历参数，如果带有了requestParam注解的话，就进行解析
            for (Parameter parameter: method.getParameters()) {
                if (parameter.isAnnotationPresent(RequsetParam.class)) {
                    //得到所有的参数名称，加入一个list
                    paramNameList.add(parameter.getDeclaredAnnotation(RequsetParam.class).valu
());
                }
            }
            //将参数list转化成参数数组
            String[] args = paramNameList.toArray(new String[paramNameList.size()]);
            //构造一个mappingHandler，注意这个handler是方法级别的，每一个方法都对应着一个ha
dler
            MappingHandler mappingHandler = new MappingHandler(uri, method, cls, args);
            //加入到handler的集合中
            HandlerManager.mappingHandlerList.add(mappingHandler);
        }
    }
```

6.接下来是要在dispatcherservlet中使用handler

```
@Override
    public void service(ServletRequest req, ServletResponse res) throws ServletException, IOExc
ption {
        for (MappingHandler handler: HandlerManager.mappingHandlerList) {
            try {
                if (handler.handle(req, res)) {
                    return;
```

```
            }
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            e.printStackTrace();
        }
    }
}
```

7.编写handler的handle方法，即开始处理请求

```
public boolean handle(ServletRequest req, ServletResponse res) throws IllegalAccessException
 InstantiationException, InvocationTargetException, IOException {
    String requestURI = ((HttpServletRequest)req).getRequestURI();
    //如果uri不相等，说明这个handler是不能处理的
    if (!uri.equals(requestURI)) {
        return false;
    }
    //开始处理请求
    Object[] parameters = new Object[args.length];
    //初始化参数
    for (int i = 0; i < args.length; i++) {
        parameters[i] = args[i];
    }
    Object ctl = controller.newInstance();
    //利用反射调用controller，这个response就相当于controller执行完返回的结果
    Object response = method.invoke(ctl, parameters);
    res.getWriter().println(response.toString());
    return true;
}
```

8.这个时候可以捋一捋。。首先dispatcherServlet已经封装在了服务器中，并以一个“/”路径进行截，也就是说，每一个请求都会经过这个servlet，并通过他的service方法。然后在这个方法里，会行这次请求的uri比对，根据这次请求来的uri，在所有的handlermapping中遍历，如果能够找到一个anlermapping与之相对应，那么就让这个handlermapping进行处理，即调用handle方法。同时呢在这个方法里，给controller实例化一个对象，然后利用method的invoke反射机制调用controller，到处理结果，放入response并返回。

9.这个时候重新打包项目，运行，已经能够响应controller的请求了。

10.写到这里我仍让有个疑问，mappinghandler中保存的参数，我觉得应该仅仅是参数的名称的一个组吧，如果仅仅把名称传入invoke，是如何调用的呢？？？那个真正的参数值是什么时候传递过来的。。