链滴

springboot 关闭的正确姿势

作者：howepeng

原文链接：https://ld246.com/article/1488335661664

来源网站：链滴

  使用springboot框架生成的项目，不需要部署在服务器上，只需用maven的install命令项目达成jar包，然后使用java -jar 命令就可以将项目启动起来了（不会用java -jar命令请自行百度）
  那么问题来了……
  这东西起来以后怎么关？…
  如果使用杀进程的方法，那么你需要人为的去查询pid（会获取指定进程pid然后杀掉的算）。
  下面的方法可以通过给主函数传参的方式来进行springboot项目的开或关。

  上代码

```
package cn.crm;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ConnectException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.security.AccessControlException;
import java.util.Random;

import javax.sql.DataSource;

import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.log4j.Logger;
import org.mybatis.spring.SqlSessionFactoryBean;
import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.ExitCodeGenerator;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.core.io.support.PathMatchingResourcePatternResolver;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.transaction.PlatformTransactionManager;

@EnableAutoConfiguration
@SpringBootApplication
@ComponentScan
@MapperScan("cn.crm.mapper")
public class Application {
    private static Logger logger = Logger.getLogger(Application.class);
    private static Application app = null;
    // 是否阻塞
    protected boolean await = true;
    // 关闭端口
    private int port = 9876;
    // 主机地址
```

```
    private String address = "localhost";
    // 系统阻塞socket
    private volatile ServerSocket awaitSocket = null;
    // 是否停止等待
    private volatile boolean stopAwait = false;
    // 关闭系统指令
    private String startup= "STARTUP";
    // 启动系统指令
    private String shutdown = "SHUTDOWN";
    private Random random = null;

    @Bean
    @ConfigurationProperties(prefix="spring.datasource")
    public DataSource dataSource() {
        return new org.apache.tomcat.jdbc.pool.DataSource();
    }

    @Bean
    public SqlSessionFactory sqlSessionFactoryBean() throws Exception {

        SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();
        sqlSessionFactoryBean.setDataSource(dataSource());

        PathMatchingResourcePatternResolver resolver = new PathMatchingResourcePatternRe
olver();

        sqlSessionFactoryBean.setMapperLocations(resolver.getResources("classpath:/mybatis/*.
ml"));

        return sqlSessionFactoryBean.getObject();
    }

    @Bean
    public PlatformTransactionManager transactionManager() {
        return new DataSourceTransactionManager(dataSource());
    }


    /**
     * Start
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {
        if (app == null) {
            app = new Application();
        }
        if (args.length &lt; 3) {
          logger.info("SpringBoot args error");
          return;
        }
        String command  = args[0];
        app.port = Integer.valueOf(args[1]) ;
        app.address = args[2];
        if (app.startup.equals(command)) {
```

```
            app.start();
        }
        if (app.shutdown.equals(command)) {
            app.stop();
        }
    }

    private void start() throws Exception {
        logger.info("-----------------------Server startup-----------------------");
        long t1 = System.nanoTime();
        String[] args = new String[]{};
        ApplicationContext ctx = SpringApplication.run(Application.class, args);
        logger.info("SpringBoot Start Success");
        long t2 = System.nanoTime();
        logger.info("Server startup in " + ((t2 - t1) / 1000000) + " ms");
        if (await) {
            await();
        }
        ExitCodeGenerator exitCodeGenerator = new ExitCodeGenerator() {
            @Override
            public int getExitCode() {
              return 200;
            }
        };
        SpringApplication.exit(ctx, exitCodeGenerator);
        logger.info("-----------------------Server shutdown-----------------------");
    }

    private void await() {
        try {
            awaitSocket = new ServerSocket(port, 1, InetAddress.getByName(address));
        } catch (IOException e) {
            logger.error("await: create[" + address + ":" + port + "]: ", e);
            return;
        }
        try {
            while (!stopAwait) {
                ServerSocket serverSocket = awaitSocket;
                if (serverSocket == null) {
                    break;
                }
                Socket socket = null;
                StringBuilder command = new StringBuilder();
                try {
                    InputStream stream;
                    long acceptStartTime = System.currentTimeMillis();
                    try {
                        socket = serverSocket.accept();
                        socket.setSoTimeout(10 * 1000);
                        stream = socket.getInputStream();
                    } catch (SocketTimeoutException ste) {
                        logger.error(
                                "Provider.accept.timeout:" + Long.valueOf(System.currentTimeMillis() - ac
eptStartTime),
```

```
                    ste);
                continue;
            } catch (AccessControlException ace) {
                logger.error("Provider.accept security exception: " + ace.getMessage(), ace);
                continue;
            } catch (IOException e) {
                if (stopAwait) {
                    break;
                }
                logger.error("Provider.await: accept: ", e);
                break;
            }

            // 读取
            int expected = 1024;
            while (expected &lt; shutdown.length()) {
                if (random == null)
                    random = new Random();
                expected += (random.nextInt() % 1024);
            }
            while (expected &gt; 0) {
                int ch = -1;
                try {
                    ch = stream.read();
                } catch (IOException e) {
                    ch = -1;
                }
                if (ch &lt; 32)
                    break;
                command.append((char) ch);
                expected--;
            }
        } finally {
            try {
                if (socket != null) {
                    socket.close();
                }
            } catch (IOException e) {
                // Ignore

            }
        }
        boolean match = command.toString().equals(shutdown);
        if (match) {
            break;
        } else {

        }
    }
} finally {
    ServerSocket serverSocket = awaitSocket;
    awaitSocket = null;

    if (serverSocket != null) {
```

```
                try {
                    serverSocket.close();
                } catch (IOException e) {
                    // Ignore

                }
            }
        }
    }

    private void stop() {
        try (Socket socket = new Socket(address, port); OutputStream stream = socket.getOutpu
Stream()) {
            String shutdown = this.shutdown;
            for (int i = 0; i &lt; shutdown.length(); i++) {
                stream.write(shutdown.charAt(i));
            }
            stream.flush();
        } catch (ConnectException ce) {
            logger.error("stopServer.connectException [" + address + ":" + port + "]", ce);
            ce.printStackTrace();
            System.exit(1);
        } catch (IOException e) {
            logger.error("stopServer.IOException", e);
            e.printStackTrace();
            System.exit(1);
        }
    }

}
```

  这样，只要在启动的时候加入启动或停止的参数可以实现启动或停止服务的功能了(注：wait方法一直在监听ip和端口是否收到shutdown的命令，接收到shutdown命令后则调用SpringApplcation的exit方法，停止服务）。