



链滴

Latke Code View - Latke.java

作者: [ZephyrJung](#)

原文链接: <https://ld246.com/article/1478834122842>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

大致的代码结构：

- java
- b3log
- cache,cron,event,image,intercept,ioc
- logging,mail,model,plugin,remote,repository
- service,servlet,taskqueue,thread,urlfetch,user,util
- **Latkes.java**
- Keys.java,Runtimetabase.java,RuntimeEnv.java,RuntimeMode.java
- json
- weborganic
- resources
- beans.xml

Latkes.java 下面那一行涉及的代码基本没有逻辑，全是枚举或者常量值的定义，想来无可研究。

开始的字段定义继续忽略，首先是静态代码块（删去了 LOGGER）：

```
static {  
    try {  
        final InputStream resourceAsStream = Latkes.class.getResourceAsStream("/latke.properties");  
        if (null != resourceAsStream) {  
            LATKEPROPS.load(resourceAsStream);  
        }  
    } catch (final Exception e) {  
        throw new RuntimeException("Not found latke.properties");  
    }  
    try {  
        final InputStream resourceAsStream = Latkes.class.getResourceAsStream("/local.properties");  
        if (null != resourceAsStream) {  
            LOCALPROPS.load(resourceAsStream);  
        }  
    } catch (final Exception e) {  
    }  
    try {  
        final InputStream resourceAsStream = Latkes.class.getResourceAsStream("/remote.properties");  
        if (null != resourceAsStream) {  
            REMOTEPROPS.load(resourceAsStream);  
        }  
    } catch (final Exception e) {  
    }  
}
```

需要注意的点：

- 静态代码块，意味着这段代码将在类实例化的时候首先得到执行
- Latkes.class.getResourceAsStream，这个用法不知是什么
- LATKE_PROPS 是 Properties 类型

—
接下来大部分是从 properties 文件里获取配置的方法，不一一列举，如：

```
public static String getStaticResourceVersion() {  
    //如果没有值，则从配置文件中读取，否则直接返回  
    if (null == staticResourceVersion) {  
        staticResourceVersion = LATKEPROPS.getProperty("staticResourceVersion");  
        //如果没有读取到，则赋予默认值  
        if (null == staticResourceVersion) {  
            staticResourceVersion = startupTimeMillis;  
        }  
    }  
    return staticResourceVersion;  
}
```

—
初始化运行环境，Starter 类会调用该方法：

```
public static void initRuntimeEnv() {  
    if (null != runtimeEnv) {  
        return;  
    }  
    final String runtimeEnvValue = LATKEPROPS.getProperty("runtimeEnv"); //latke.propertie  
  
    if (null != runtimeEnvValue) {  
        //如果不为空,说明配置里有设置,读取设置  
        runtimeEnv = RuntimeEnv.valueOf(runtimeEnvValue);  
    }  
    //配置为空,设置为-本地  
    runtimeEnv = RuntimeEnv.LOCAL;  
    //获取设置运行模式  
    if (null == runtimeMode) {  
        final String runtimeModeValue = LATKEPROPS.getProperty("runtimeMode");  
        if (null != runtimeModeValue) {  
            runtimeMode = RuntimeMode.valueOf(runtimeModeValue);  
        } else {  
            runtimeMode = RuntimeMode.PRODUCTION;  
        }  
    }  
    if (RuntimeEnv.LOCAL == runtimeEnv) {  
        // Read local database configurations  
        final RuntimeDatabase runtimeDatabase = getRuntimeDatabase();  
        //local.properties  
        if (RuntimeDatabase.H2 == runtimeDatabase) {  
            final String newTCPServer = Latkes.getLocalProperty("newTCPServer");  
            if ("true".equals(newTCPServer)) {
```

```

final String jdbcURL = Latkes.getLocalProperty("jdbc.URL");
if (Strings.isEmptyOrNull(jdbcURL)) {
    throw new IllegalStateException("The jdbc.URL in local.properties is required");
}
final String[] parts = jdbcURL.split(":");
if (parts.length != Integer.valueOf("5")/* CheckStyle.... */ ) {
    throw new IllegalStateException("jdbc.URL should like [jdbc:h2:tcp://localhost:250/~/] (the port part is required)");
}
String port = parts[parts.length - 1];
port = StringUtils.substringBefore(port, "/");
try {
    h2 = org.h2.tools.Server.createTcpServer(new String[]{"-tcpPort", port, "-tcpAllwOthers"}).start();
} catch (final SQLException e) {
    final String msg = "H2 TCP server create failed";
    throw new IllegalStateException(msg);
}
}
}
locale = new Locale("enUS");
}

```

- 这段代码中包含了当数据库是 H2 时的处理，大概时使用该数据库时要自己做的事情吧，毕竟是纯 Java 库，想来不必深究，其文档应该有说明。
- Locale 类的作用意义不明。

关闭 Latke

```

public static void shutdown() {
    try {
        if (RuntimeEnv.LOCAL != getRuntimeEnv()) {
            return;
        }
        //关闭数据库连接池
        Connections.shutdownConnectionPool();
        final RuntimeDatabase runtimeDatabase = getRuntimeDatabase();
        switch (runtimeDatabase) {
            case H2:
                final String newTCPServer = Latkes.getLocalProperty("newTCPServer");
                if ("true".equals(newTCPServer)) {
                    h2.stop();
                    h2.shutdown();
                }
                break;
            default:
        }
        //关闭定时任务
        CronService.shutdown();
        //关闭执行的线程
    }
}

```

```
    LocalThreadService.EXECUTORSERVICE.shutdown();
} catch (final Exception e) {
}
//停止应用生命周期
Lifecycle.endApplication();
// This manually deregisters JDBC driver, which prevents Tomcat from complaining about
memory leaks
//注销数据库连接驱动
final Enumeration<Driver> drivers = DriverManager.getDrivers();
while (drivers.hasMoreElements()) {
    final Driver driver = drivers.nextElement();
    try {
        DriverManager.deregisterDriver(driver);
    } catch (final SQLException e) {
    }
}
}
```

- Connections, CronService, LocalThreadService, Lifecycle 均为 b3log 包下的类，此时暂不做进一步研究。
- Driver 为 java.sql 包下的类。

加载皮肤，这个看似跟 MVC 无关的内容，放在这里，有点琢磨不透，可能是因为预先想到了由不同皮肤，而在公共的框架上提供了支持吧：

```
public static void loadSkin(final String skinDirName) {
    final ServletContext servletContext = AbstractServletListener.getServletContext();
    Templates.MAINCFG.setServletContextForTemplateLoading(servletContext, "skins/" + sk
nDirName);
    Latkes.setTimeZone("Asia/Shanghai");
}
```

- AbstractServletListener, Templates 为 b3log 的类，暂不详查
- ServletContext 有什么特殊的地方？

获取皮肤名称：

```
public static String getSkinName(final String skinDirName) {
    try {
        final Properties ret = new Properties();
        final File file = getWebFile("/skins/" + skinDirName + "/skin.properties");
        ret.load(new FileInputStream(file));
        return ret.getProperty("name");
    } catch (final Exception e) {
        return null;
    }
}
```

```
public static File getWebFile(final String path) {
    final ServletContext servletContext = AbstractServletListener.getServletContext();
    File ret;
    try {
        final URL resource = servletContext.getResource(path);
        if (null == resource) {
            return null;
        }
        ret = FileUtils.toFile(resource);
        if (null == ret) {
            final File tempdir = (File) servletContext.getAttribute("javax.servlet.context.tempdir")
                ret = new File(tempdir.getPath() + path);
                FileUtils.copyURLToFile(resource, ret);
                ret.deleteOnExit();
        }
        return ret;
    } catch (final Exception e) {
        return null;
    }
}
```
