



Woodland
Academy

6.8 ウェブ開発補足

- ログ出力
- Lombok
- プロパティファイル



Shape Your Future

目次

- 1 ログ出力
- 2 Lombok
- 3 プロパティファイル

ログ

- これまでは、`System.out.println()` などのメソッドを使って、テストや Debug の情報を出力しています。実際の開発現場では、このようなテスト情報やシステムの動作記録を、**ログ**^[Log]と呼ばれる**外部ファイルに保存**しておきます。
- コンソールに出力するだけでなく、ログを使用することにはいくつかのメリットがあります：
 - **毎回の実行情報**が自動保存され、過去の記録はいつでも閲覧可能
 - エラーでプログラムが**異常終了**した場合でも記録が保存
 - 開発者は、**ユーザー**からもらったログ情報をもとに**デバッグ**できる

ログのレベル

- 実際のログの出力は、出力情報の重要度をいくつかの**レベル**に分けて出力します。よく使われるレベル構造は右の表に示します：
- 開発か、テストか、リリースされた製品がユーザーに使用される時など、様々な状況でプログラムが実行された時に、どのレベル**以上**の（どれくらい細かい）情報をコンソールや外部ログファイルに出力するかを制御することができます。

レベル	意味
TRACE	最も詳細な情報
DEBUG	デバッグに役立つくらいの詳細情報
INFO	一般情報
WARN	問題が発生する可能性がある警告情報
ERROR	エラーや例外に関する重要な情報

SLF4J

- 出力ログに関する機能を提供する Java のライブラリには、Log4J、Logback、SLF4J などがあります。ここでは、簡単に使える **SLF4J** を簡単に紹介します。
- Spring Boot の依存関係には既に SLF4J が含まれているため、別のパッケージを Maven とか追加する必要はありません。関連するメソッドは org.slf4j パッケージにあります。

Logger の取得

- ログを出力するために、まず、**Logger** クラスのオブジェクトを取得する必要があります：

```
Logger logger = LoggerFactory.getLogger("Logger Name");
```

- Logger Name はこのロガーの名前です。一般的に、現在の**クラス名**をロガーの名前として使用します：

```
Logger logger = LoggerFactory.getLogger(MainController.class);
```

ログの出力

- ロガーを取得したら、その**同名のメソッド**を使用して、特定なレベルのログが出力できます。例えば、以下のコードでは WARN レベルのログを出力します：

```
logger.warn( "Caution!" );
```

- System.out.println() と同様に、各レベルのログの出力を**コンソール**で確認できます：

```
2022-08-13 07:01:47.692 INFO 17308 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring Dispa
2022-08-13 07:01:47.692 INFO 17308 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dis
2022-08-13 07:01:47.693 INFO 17308 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization
2022-08-13 07:01:47.720 INFO 17308 --- [nio-8080-exec-1] n.l.s.s.controllers.MainController : Hello!
2022-08-13 07:01:47.720 WARN 17308 --- [nio-8080-exec-1] n.l.s.s.controllers.MainController : Caution!
2022-08-13 07:01:47.720 ERROR 17308 --- [nio-8080-exec-1] n.l.s.s.controllers.MainController : Something went wrong...
```

- ログの出力日時、スレッド名なども記録されます。

ログ出力の設定

- 現在、INFO レベル以上のメッセージのみが出力されることがわかりました。設定情報を application.properties に追加することで、出力ログの**レベル制限が変更**できます：

```
logging.level.[ローガーのクラス名かパッケージ名]=[出力レベル]
```

- 例えば、この Security プロジェクトで使ったローガーの出力レベルを DEBUG 以上に設定するには、次のような設定を書くだけです：

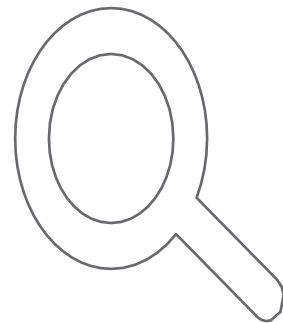
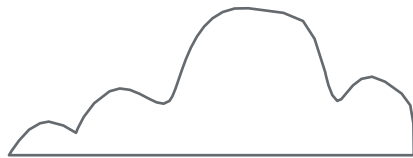
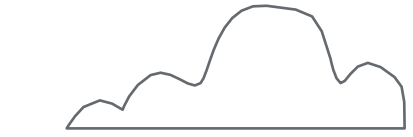
```
logging.level.net.lighthouseplan.spring.security=DEBUG
```

- また、以下の設定でログを**外部ファイル**にも出力できます：

```
logging.file.name=[ログファイルのパス]
```




Q&A




目次

- ① ログ出力
- ② Lombok
- ③ プロパティファイル

JavaBeans の仕様

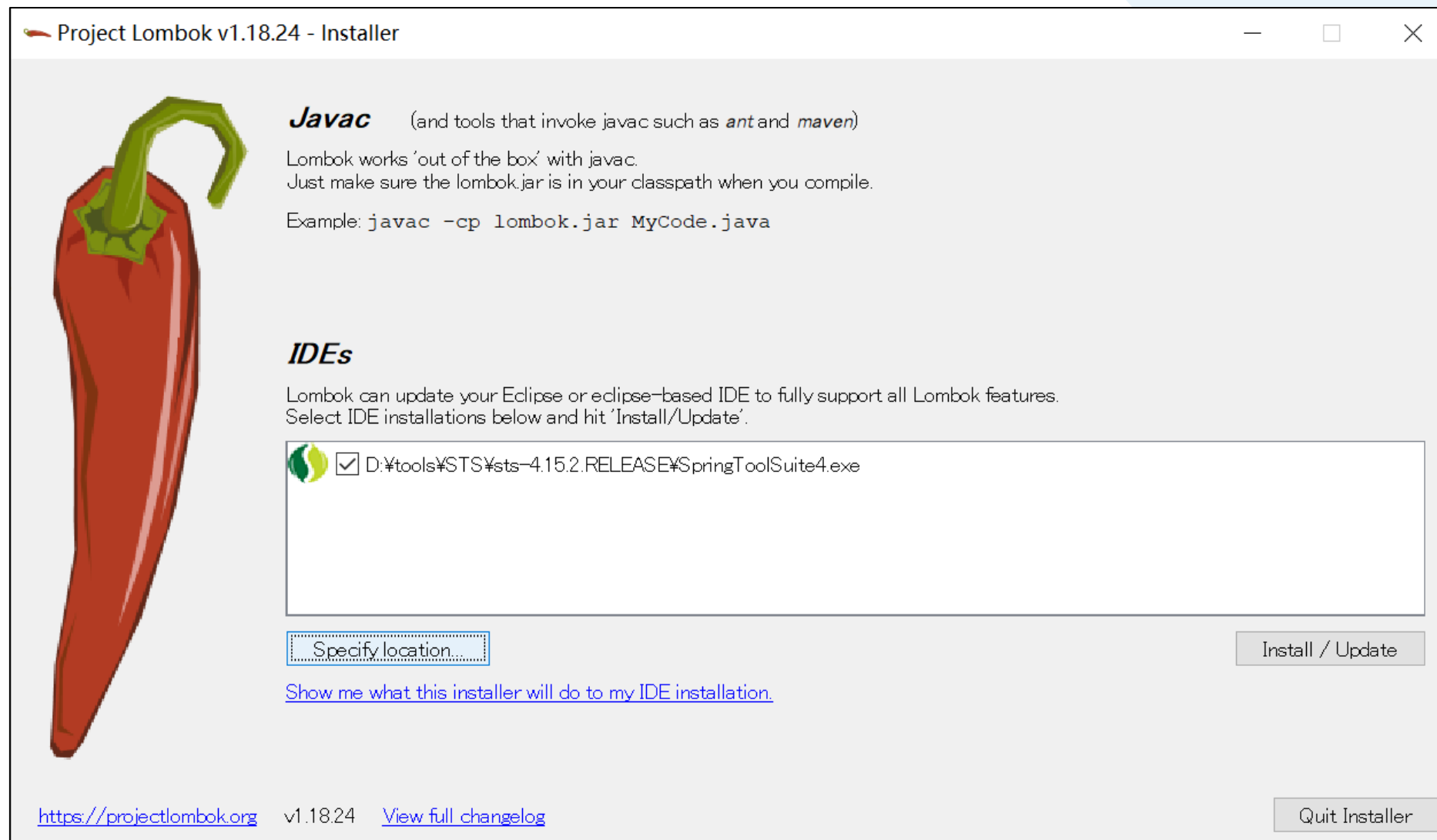
- クラスは以下の条件を満たす場合、JavaBean といいます：
 - すべての属性は **private** である
 - すべての属性は、それに対応する**ゲッター**と**セッター**を持つ
 - **public** の**パラメータなしコンストラクタ**がある
 - **Serializable** インタフェースを実装。
- 実際では、データを管理する DTO やパラメータクラスの標準として、JavaBean のような仕様がよく使われます。一部の仕様（Plain Old Java Object、略して POJO など）は、コンストラクタが必ずしもパラメータなしではなく、**Serializable** を実装しないなど、若干の違いがあります。

Lombok

- 異なる JavaBean クラスは、ほとんど同様の方法で定義されています。更に、全て変数をパラメータとして持つコンストラクタや、toString() メソッドなどの、**毎回似たものを書かなければならない**コードがたくさんあります。
- **Lombok** は、このようなコードの生成を自動化し、開発を効率化するためのツールキットです。
- Lombok は追加インストールが必要です。以下のリンクからインストールパッケージをダウンロードしてください：
 <https://projectlombok.org/download>

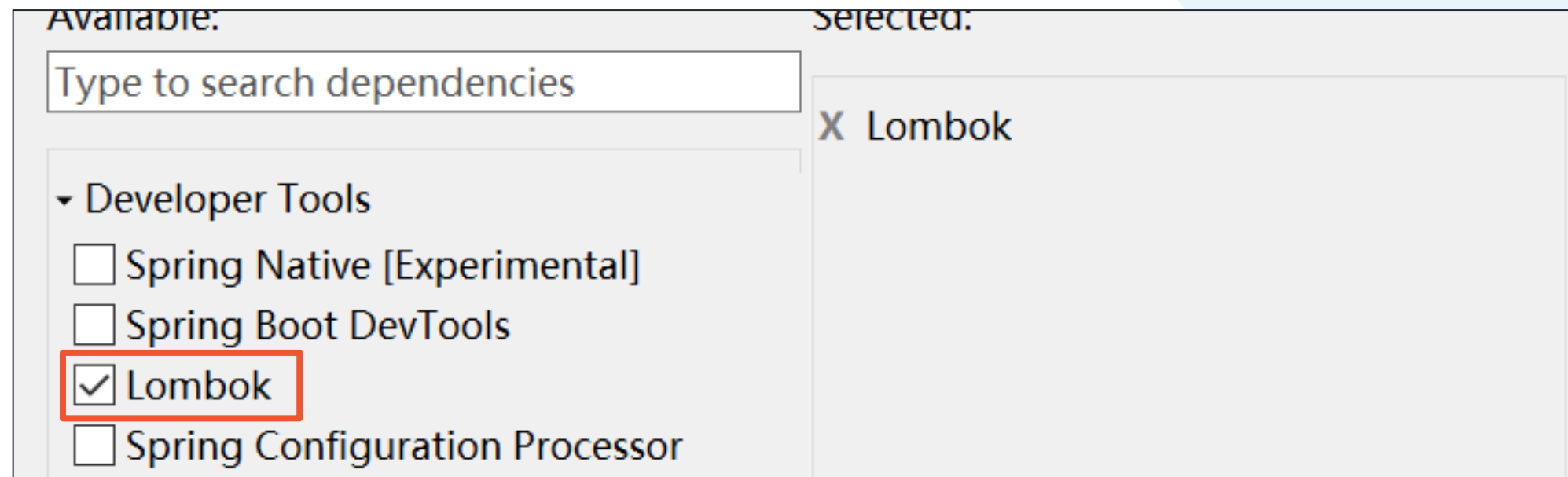
Lombok のインストール

- STS が表示されていない場合は、「Specify Location」をクリックして STS を探します。STS を選択し、「Install / Update」をクリックしてインストールします：



Lombok の依存関係の追加

- STS を再起動し、新規プロジェクトに Lombok の依存関係を追加します（もちろん、Maven 経由で後から追加することもできます）：



@Getter と @Setter

- Lombok の主な機能は、特定の宣言の前に付く**アノテーション**によって実現します。
- 例えば、**@Getter** または **@Setter** アノテーションを変数の宣言前に追加すると、その変数に対応するゲッターとセッターが自動的に生成されます：

```
1 public class Student {
2     @Getter
3     @Setter
4     private String name;
5 }
```

```
1 Student student = new Student();
2 student.setName("Alice");
3 System.out.println(student.getName()); // => Alice
```

よく使われるアノテーション

- その他によく使われるアノテーションを以下の表に：

アノテーション	機能
@NoArgsConstructor	パラメーターなしのコンストラクターの生成
@RequiredArgsConstructor	必須変数がパラメータなコンストラクタの生成
@AllArgsConstructor	全変数がパラメータなコンストラクタの生成
@ToString	toString() メソッドの生成
@EqualsAndHashCode	equals() および hashCode() の生成
@Data	クラスに @ToString、@EqualsAndHashCode、@RequiredArgsConstructor を追加し、全ての変数に @Getter と @Setter を追加
@Log (@Slf4j)	log という (SLF4J) ロガーの生成

@Builder

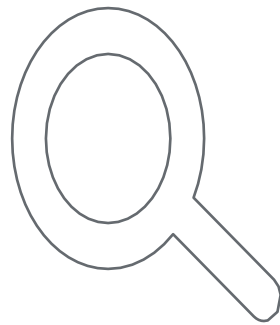
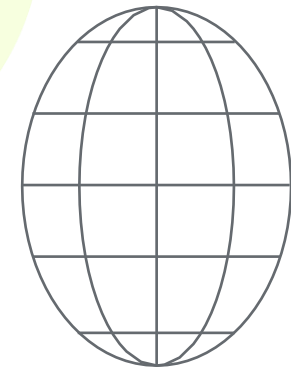
- **@Builder** アノテーションは、オブジェクトを作成するための **builder()** メソッドを自動生成します。インスタンス化のコードをより読みやすく、拡張しやすくなります：

```
1 @Data
2 @Builder
3 public class Student {
4     private Long id;
5     private String name;
6     private int score;
7 }
```

```
1 Student student = Student.builder()
2     .id(1L)
3     .name("Alice")
4     .score(90)
5     .build();
6 System.out.println(student); // => Student(id=1, name=Alice, score=90)
```



Q&A



目次

- ① ログ出力
- ② Lombok
- ③ プロパティファイル

application.properties

- Spring Boot は、プロジェクト全体の**設定情報**を格納するために、**application.properties** を使用します。
- src/main/resources ディレクトリ、またはプロジェクトパスの /config ディレクトリに配置する必要があります。
- この Properties ファイルを使って、ポート番号などのデフォルトの設定値が変更できます。ポート番号はデフォルトで 8080 に設定され、server.port で設定を変更できます：


```
server.port=8090
```

設定可能なパラメータ

- これまで使用したもの以外にも、application.propertiesで設定できる項目は多数あります：

パラメータ	機能
spring.application.name	アプリケーションの名前
server.address	サーバーのアドレス
spring.mail.host	サーバーのメールホストアドレス
logging.file.path	ログファイルのパス
spring.config.name	プロパティのアドレス(デフォルトは application)

- 設定可能な全パラメータは、ここに記載されています：

 <https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

Properties ファイルの問題

- 実際に使用されるプロパティのパラメータは、通常、一定の**階層的な構造**を満たします。しかし、従来の .properties ファイルはこの性質を利用していないため、パラメータが多くて名前が長い場合、非常に読みづらくなることがあります：

```

1 spring.datasource.url=jdbc:postgresql://localhost:5432/security
2 spring.datasource.username=postgres
3 spring.datasource.password=123456
4
5 logging.file.name=application.log
6 logging.level.root=INFO
7 logging.level.net.lighthouseplan.spring.security=DEBUG

```

YAML フォーマット

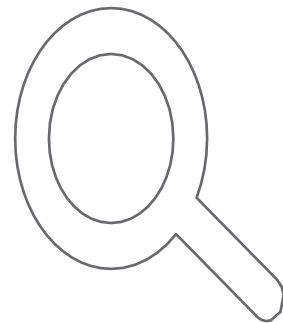
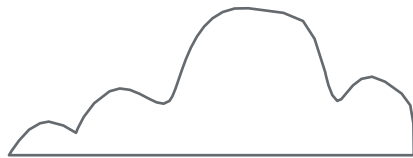
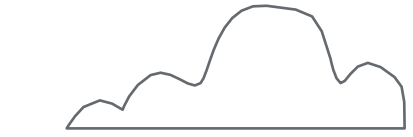
- Spring Boot は **YAML** 形式でのプロパティ設定もできます。情報を階層的に表現するため、プロパティの記述は簡単：

```
1 spring:
2   datasource:
3     url: jdbc:postgresql://localhost:5432/security
4     username: postgres
5     password: 123456
6
7   logging:
8     file:
9       name: application.log
10    level:
11      root: INFO
12    net.lighthouseplan.spring.security: DEBUG
```

- YAML で記述されたプロパティは元の .properties ファイルを置き換えた **application.yml** ファイルに保存されます。



Q&A



まとめ

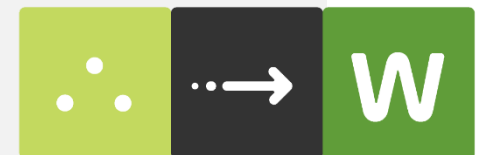
Sum Up



1. ログの概念とロギングライブラリの使い方。
2. Lombok の使用方法。
3. プロパティファイルの書き方。

Thank you!

From Seeds to Woodland — Shape Your Future.



Shape Your Future