LightHouse Academy

6.6 Spring Boot 補足

- ファイルのアップロード
- URL パラメーター
- リダイレクトとフォーワード
- パスワードの暗号化

www.lighthouseit.academy

目次

LightHouse Academy

ファイルのアップロード（フロントエンド）

- 実際のアプリケーションでは、元々サーバーにあるファイル以外、ユーザが画像、音声とかのファイルをアプロードすることがあります。それらのファイルは、サーバーのファイルシステムに直接に保存する必要があります。

- ファイルをアップロードために、フロントエンドでは他のデータと同じように、フォームの <input> を使用します。ただし、type 属性は「**file**」に設定して、form に「**enctype=“multipart/form-data”**」を追加します：

```
<form action="/image/add" method="POST" enctype="multipart/form-data">
  <input type="file" name="image">
  <input type="submit">
</form>
```

ファイルのアップロード（コントローラー）

- コントローラーでは、このようなファイルを他のデータと同じように、「@RequestParam」のつけた引数でファイルを受け取ります。ただし、引数のタイプは「**MultipartFile**」に指定します：

```
1 @PostMapping("/image/add")
2 public String addImage(@RequestParam MultipartFile image) {
3     String fileName = image.getOriginalFilename();
4     //...
5 }
```

受け取ったファイルを保存

- 受け取ったファイルオブジェクトは、名前を取得したり、サーバー保存したりできます。ファイルを保存するには、例えば **Files.copy** メソッドを利用する方法があります：

```
1 @PostMapping("/image/add")
2 public String addImage(@RequestParam MultipartFile image) throws IOException {
3     String fileName = image.getOriginalFilename();
4     Files.copy(image.getInputStream(), Path.of("/images/" + fileName));
5     //...
6 }
```

ファイルの利用

- サーバーに保存したら、後はすでにサーバーに入れたファイルのように、相対パスや絶対パスで利用すればいいです。例えば、画像ファイルのパスを利用し、html ファイルに表示することも可能です：

```
1 <div id="image-list">
2   <div th:each="image:${images}">
3     <span th:text="${image.id}"></span>
4     <img th:src="'/images/' + ${image.fileName}">
5   </div>
6 </div>
```

Try

ImageController

目次

LightHouse Academy

URL パラメーター

- ID、ユーザー名などの簡単のパラメーターを伝えるためには、これらの変数を直接に URL パスに書くことも可能です。コントローラーでは、「**@PathVariable**」を付けた引数でそれらの変数を受け取れます：

```
1 @GetMapping("/path/content/{num}")
2 public String showContent(@PathVariable int num) {
3     //...
4 }
```

複数の URL パラメーター

- **一つの URL に複数のパラメーター**を入れることも可能です：

```
1 @GetMapping("/path/content/{num1}/{num2}")
2 public String showContentMult(@PathVariable int num1,
3                                @PathVariable int num2) {
4     //...
5 }
```

Try

PathVariableController

目次

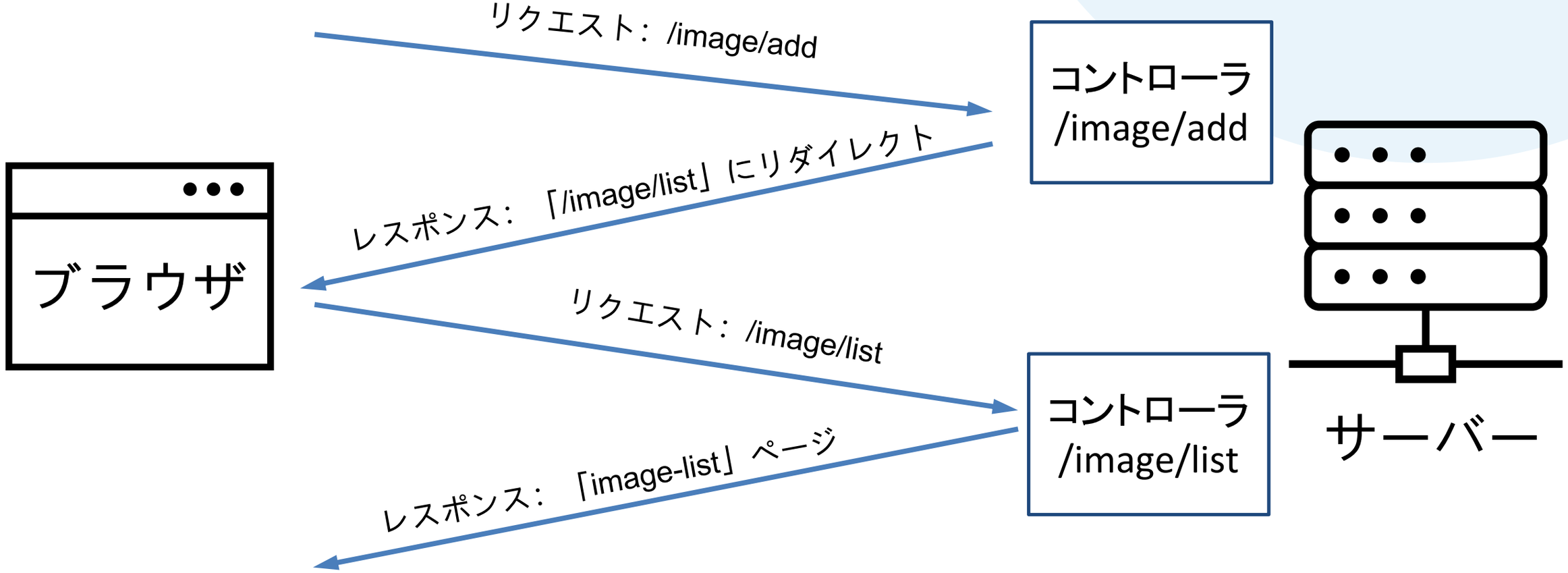
リダイレクト

- 前も紹介したように、**リダイレクト**で、コントローラーのレスポンスとしてユーザーを別の画面に遷移させることができます：

```
1 @PostMapping("/image/add")
2 public String addImage(@RequestParam MultipartFile image) {
3     // ...
4     return "redirect:/image/list";
5 }
```

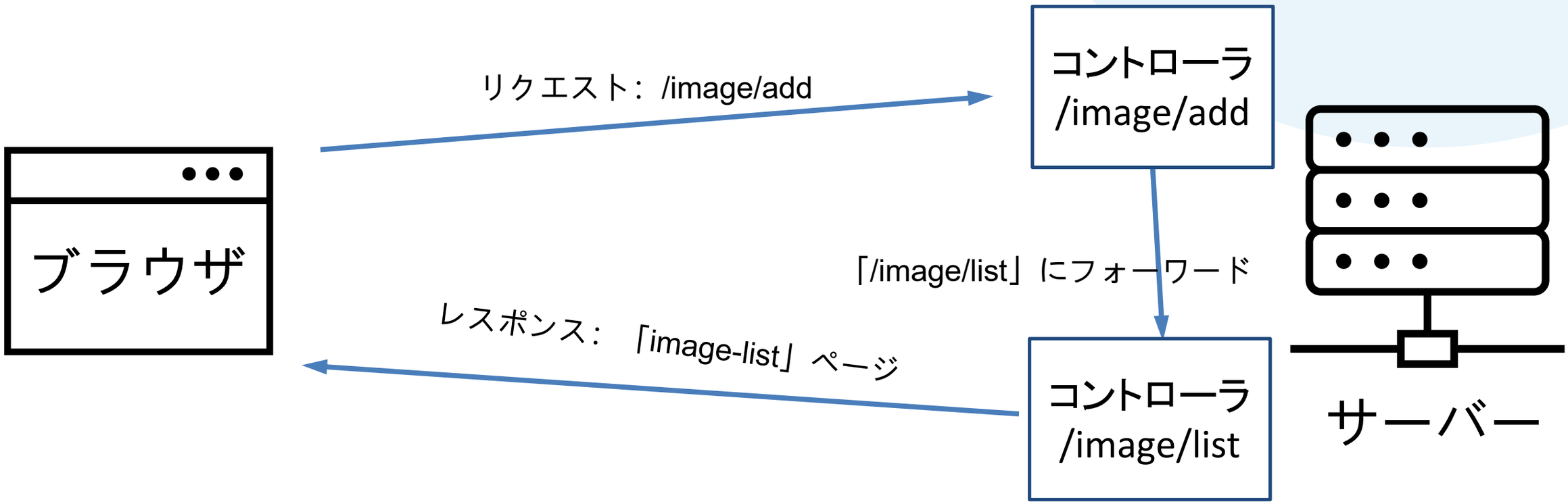
リダイレクトの仕組み

- リダイレクトの原理として、サーバーはユーザー側のブラウザに「他のページに行ってください」と要求したら、ブラウザが他の URL に、もう一度リクエストします：



フォーワードの仕組み

- リダイレクト以外、もう一つの画面遷移の方法があります。それは、**フォーワード**です。フォワードはリダイレクトと違って、ブラウザを経由せず、直接に URL に対応するコントローラーメソッドを呼び出すことが可能です：



フォーワードの書き方

- Spring Boot でフォーワードの書き方はリダイレクトと似てて、ただ「**redirect**」を「**forward**」に置き換えるだけです：

```
1 @PostMapping("/image/add")
2 public String addImage(@RequestParam MultipartFile image) {
3     // ...
4     return "forward:/image/list";
5 }
```

リダイレクトとフォーワードの選択

- リダイレクトは、サーバー内外問わずに URL に転移できますが、**複雑なデータをコントローラ間に渡すのは難しい**です。

- ただし、簡単なデータを渡したいのなら、パスにパラメーターを直接に入れるのが有効です：

```
public String redirectToTarget() {
    String param1 = "apple";
    String param2 = "banana";
    return "redirect:/rdr/target" + "?param1=" + param1 + "&param2=" + param2;
}
```

- その代わりに、リダイレクトは**サーバー内外問わず、どんな URL にも転移できる**という利点があります。

次へ

- それに対して、フォーワードは**外部サーバーに遷移できないが**、複雑なデータでも「Model」「ModelAndView」とかの Java オブジェクトによって**コントローラー間でやり取りできます**：

```
1 public ModelAndView forwardToTarget(ModelAndView mav) {
2     mav.addObject("param1", "Alice");
3     mav.addObject("param2", "Bob");
4     mav.setViewName("forward:/fwd/target");
5     return mav;
6 }
```

Try

RedirectController

目次

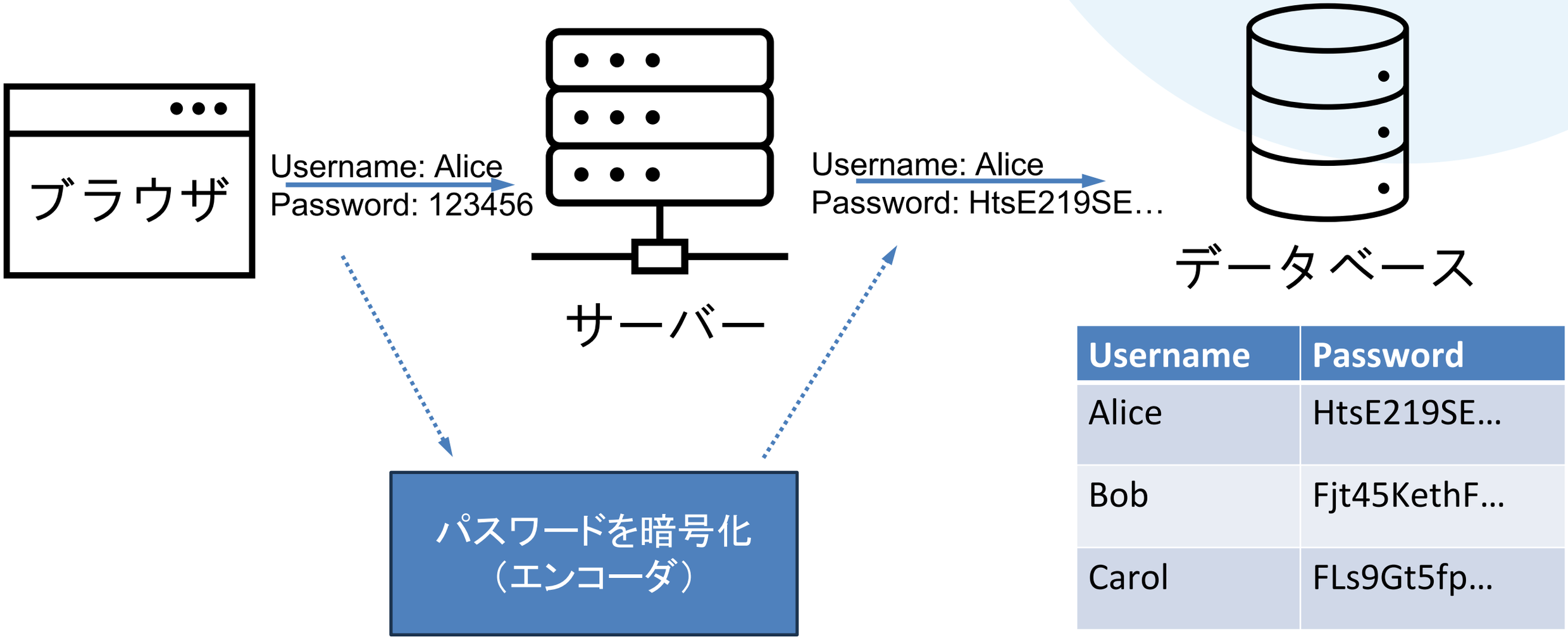
パスワード保存の問題

- 以前紹介した例では、ユーザーのパスワードをそのままデータベースに保存します。しかし、このままでは、もし万が一データベースの情報が漏洩したら非常に危険です。このサイトのパスワードだけではなく、他のサイトで使われるパスワードも知られる恐れがあるから。

- この故に、実際のアプリケーションでは、パスワードを暗号化してから保存します。

暗号化の仕組み（登録）

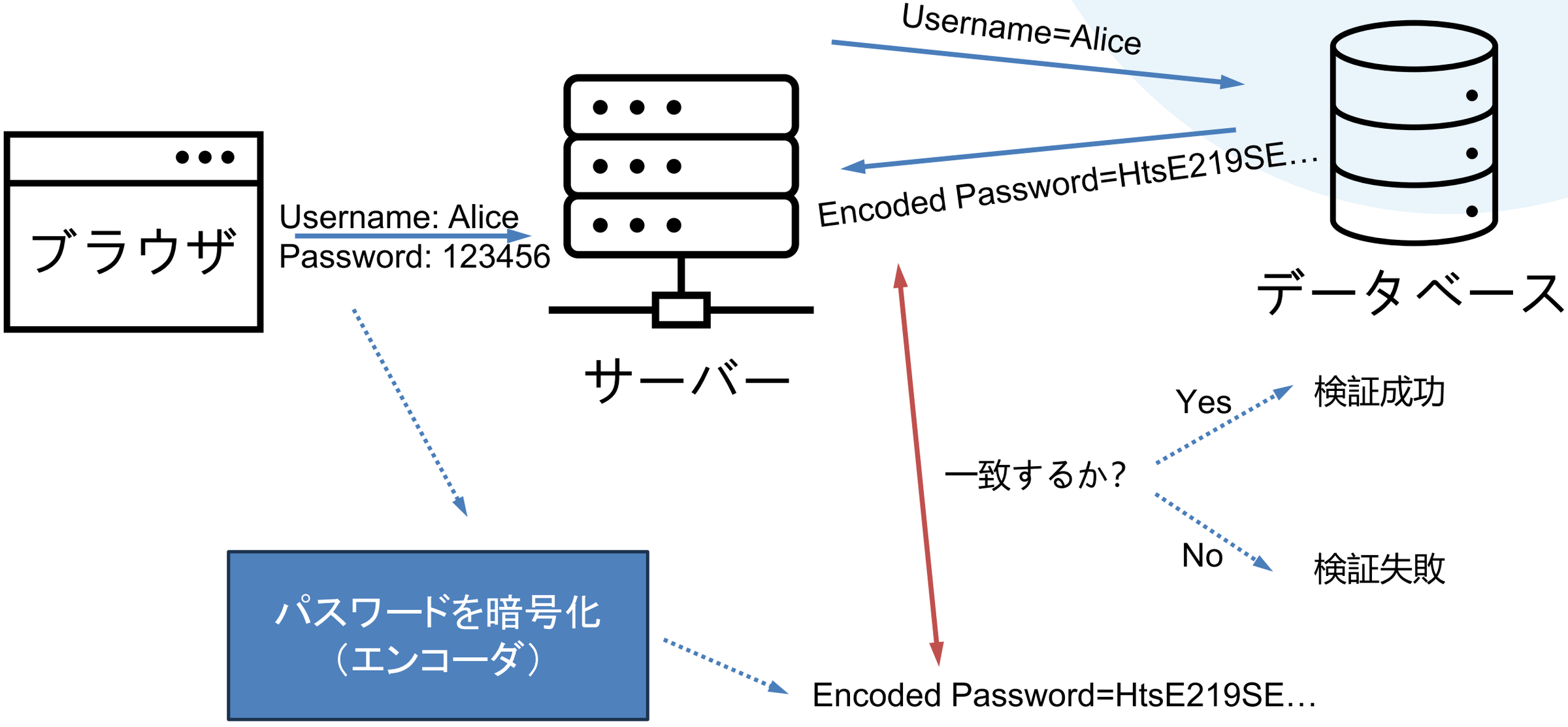
- パスワードの暗号化では、元のパスワードに**絶対復号できない**暗号文にする必要があります。そして、ユーザー情報を登録する際はサーバー（データベース）に元のパスワードを保存せず、暗号文だけを保存する：



Username	Password
Alice	HtsE219SE…
Bob	Fjt45KethF…
Carol	FLs9Gt5fp…

暗号化の仕組み（検証）

- 暗号化したパスワードを検証するには、ユーザーが入力したパスワードを同じ手順で暗号化し、得られた暗号文がデータベースに記されたものと一致するかどうかを判断します：



Spring Boot での暗号化（登録）

- Spring Boot では、PasswordEncoder インタフェースの **encode** メソッドによって簡単にパスワードを暗号化できます。サンプルコードでは、一番良く使われる「**BCryptPasswordEncoder**」を使用し、ユーザー情報を登録する際に、暗号化したパスワードを保存します：

```
BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
String encodedPassword = passwordEncoder.encode(password);
userDao.save(new UserEntity(username, encodedPassword));
```

Spring Boot での暗号化（ログイン検証）

- Spring Security では自動的にユーザログインを検証するために、自分で暗号化したパスワードを検証する必要がないが、暗号文の前に「**{bcrypt}**」を加えることによって暗号化のアルゴリズムを Spring に知らせる必要があります：

```
manager.createUser(User.builder()
        .username(username)
        .password("{bcrypt}" + password)
        .roles("USER")
        .build());
```

Spring Boot での暗号化（その他検証）

- ログイン以外のところで（パスワードの編集とか）パスワードを自分で検証したい場合は、**maches** メソッドを利用できます：

```
public boolean validate(String username, String rawPassword) {
    String encodedPassword = userDao.findByUsername(username).getPassword();

    BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
    return passwordEncoder.matches(rawPassword, encodedPassword);
}
```

Try

PasswordEncodeSample

まとめ

Sum Up

1.ファイルのアップロード。

2.URL パラメータ。

3.リダイレクトとフォーワード：
① Redirect：外部のサーバに遷移可能。
② Forward：データのやり取りが効率。

4.パスワード暗号化の概念：
① パスワード暗号化の理由：情報漏洩対策。
② パスワード暗号化のやり方。
③ Spring Security での実現。

Light in Your Career.

THANK YOU!