Оглавление

Функциональные интерфейсы

```
1.  package java.util.function;
2.
3.  @FunctionalInterface
4.  public interface ... {
5.    ...
6.  }
```

Predicate<T>

```
1.  boolean test(T t);
```

```
1.  default Predicate<T> and(
2.      Predicate<? super T> other) {
3.    Objects.requireNonNull(other);
4.    return (t) -> test(t) && other.test(t);
5.  }
```

```
1.  default Predicate<T> negate() {
2.    return (t) -> !test(t);
3.  }
```

```
1.  default Predicate<T> or(
2.      Predicate<? super T> other) {
3.    Objects.requireNonNull(other);
4.    return (t) -> test(t) || other.test(t);
5.  }
```

```
1.  static <T> Predicate<T> isEqual(
2.      Object targetRef) {
3.    return (null == targetRef)
4.        ? Objects::isNull
5.        : object -> targetRef.equals(object);
6.  }
```

DoublePredicate

```
1.  boolean test(double value);
```

```
1.  default DoublePredicate and(
2.      DoublePredicate other) {
3.    Objects.requireNonNull(other);
4.    return (value) ->
5.        test(value) && other.test(value);
6.  }
```

```
1.  default DoublePredicate negate() {
2.    return (value) -> !test(value);
3.  }
```

```
1.  default DoublePredicate or(
2.      DoublePredicate other) {
```

```
3.    Objects.requireNonNull(other);
4.    return (value) ->
5.        test(value) || other.test(value);
6.  }
```

BiPredicate<T, U>

```
1.  boolean test(T t, U u);
```

```
1.  default BiPredicate<T, U> and(
2.      BiPredicate<? super T, ? super U> other) {
3.    Objects.requireNonNull(other);
4.    return (T t, U u) ->
5.        test(t, u) && other.test(t, u);
6.  }
```

```
1.  default BiPredicate<T, U> negate() {
2.    return (T t, U u) -> !test(t, u);
3.  }
```

```
1.  default BiPredicate<T, U> or(
2.      BiPredicate<? super T, ? super U> other) {
3.    Objects.requireNonNull(other);
4.    return (T t, U u) ->
5.        test(t, u) || other.test(t, u);
6.  }
```

Adf

Consumer<T>

```
1.  void accept(T t);
```

```
1.  default Consumer<T> andThen(
2.      Consumer<? super T> after) {
3.    Objects.requireNonNull(after);
4.    return (T t) -> {
5.      accept(t);
6.      after.accept(t);
7.    };
8.  }
```

DoubleConsumer

```
1.  void accept(double value);
```

```
1.  default DoubleConsumer andThen(
2.      DoubleConsumer after) {
3.    Objects.requireNonNull(after);
```

```
4.    return (double t) -> {
5.        accept(t);
6.        after.accept(t);
7.    };
8.  }
```

BiConsumer<T, U>

```
1.  void accept(T t, U u);
```

```
1.  default BiConsumer<T, U> andThen(
2.      BiConsumer<? super T, ? super U> after) {
3.    Objects.requireNonNull(after);
4.    return (l, r) -> {
5.      accept(l, r);
6.      after.accept(l, r);
7.    };
8.  }
```

Supplier<T>

```
1.  T get();
```

DoubleSupplier

```
1.  double getAsDouble();
```

BooleanSupplier

```
1.  boolean getAsBoolean();
```

(нулевая функция до R).

Function<T, R>

```
1.  R apply(T t);
```

```
1.  default <V> Function<V, R> compose(
2.      Function<? super V, ? extends T> before) {
3.    Objects.requireNonNull(before);
4.    return (V v) -> apply(before.apply(v));
```

```
}
```

```
default <V> Function<T, V> andThen(
    Function<? super R, ? extends V> after) {
  Objects.requireNonNull(after);
  return (T t) -> after.apply(apply(t));
}
```

```
static <T> Function<T, T> identity() {
  return t -> t;
}
```

DoubleFunction<R>

```
R apply(double value);
```

DoubleToIntFunction

```
int applyAsInt(double value);
```

DoubleToLongFunction

```
long applyAsLong(double value);
```

DoubleUnaryOperator

```
double applyAsDouble(double operand);
```

```
default DoubleUnaryOperator compose(
    DoubleUnaryOperator before) {
  Objects.requireNonNull(before);
  return (double v) ->
      applyAsDouble(before.applyAsDouble(v));
}
```

```
default DoubleUnaryOperator andThen(
    DoubleUnaryOperator after) {
  Objects.requireNonNull(after);
  return (double t) ->
      after.applyAsDouble(applyAsDouble(t));
}
```

```
static DoubleUnaryOperator identity() {
  return t -> t;
}
```

BiFunction<T, U, R>

```
R apply(T t, U u);
```

```
default <V> BiFunction<T, U, V> andThen(
    Function<? super R, ? extends V> after) {
  Objects.requireNonNull(after);
  return (T t, U u) -> after.apply(apply(t, u));
```

```
5.  }
```

BinaryOperator<T>
extends BiFunction<T,T,T>

```
1.  public static <T> BinaryOperator<T> minBy(
2.      Comparator<? super T> comparator) {
3.    Objects.requireNonNull(comparator);
4.    return (a, b) ->
5.        comparator.compare(a, b) <= 0 ? a : b;
6.  }
```

```
1.  public static <T> BinaryOperator<T> maxBy(
2.      Comparator<? super T> comparator) {
3.    Objects.requireNonNull(comparator);
4.    return (a, b) ->
5.        comparator.compare(a, b) >= 0 ? a : b;
6.  }
```

DoubleBinaryOperator

```
1.  double applyAsDouble(double left, double right);
```

Другие

ыв

java.lang

class Object

Конструктор

```
1.  public Object()
```

Методы

```
1.  protected native Object clone()
2.      throws CloneNotSupportedException
1.  boolean equals(Object obj)
1.  final native Class<?> getClass()
1.  native int hashCode()
1.  final native void notify()
1.  final native void notifyAll()
1.  String toString()
1.  final void wait() throws InterruptedException
1.  final native void wait(long timeoutMillis)
2.      throws InterruptedException
1.  final void wait(long timeoutMillis, int nanos)
2.      throws InterruptedException
```

final class String

extends Object

implements

```
1. CharSequence
1. Comparable<String>
1. java.io.Serializable
```

Поле

```
1. static final
2.     Comparator<String> CASE_INSENSITIVE_ORDER =
3.     new CaseInsensitiveComparator();
```

Конструкторы

```
1. String()
1. String(byte[] bytes)
1. String(byte bytes[], int offset, int length)
1. String(byte bytes[], int offset, int length,
2.     String charsetName)
1. String(byte bytes[], int offset, int length,
2.     Charset charset)
1. String(byte bytes[], String charsetName)
1. String(byte bytes[], Charset charset)
1. String(char value[])
1. String(char value[], int offset, int count)
1. String(int[] codePoints, int offset, int count)
1. String(String original)
1. String(StringBuffer buffer)
1. String(StringBuilder builder)
```

Методы

```
1. char charAt(int index)
1. IntStream chars()
1. int codePointAt(int index)
1. int codePointBefore(int index)
1. int codePointCount(int beginIndex, int endIndex)
1. IntStream codePoints()
1. int compareTo(String anotherString)
1. int compareToIgnoreCase(String str)
1. String concat(String str)
```

```
1.  boolean contains(CharSequence s)
1.  boolean contentEquals(CharSequence cs)
1.  boolean contentEquals(StringBuffer sb)
1.  static String copyValueOf(char data[])
1.  static String copyValueOf(char data[], int offset,
2.      int count)
1.  boolean endsWith(String suffix)
1.  boolean equals(Object anObject)
1.  boolean equalsIgnoreCase(String anotherString)
1.  static String format(String format,
2.      Object... args)
1.  static String format(Locale l, String format,
2.      Object... args)
1.  byte[] getBytes()
1.  byte[] getBytes(String charsetName)
1.  byte[] getBytes(Charset charset)
1.  void getChars(int srcBegin, int srcEnd,
2.      char dst[], int dstBegin)
1.  int hashCode()
1.  int indexOf(int ch)
1.  int indexOf(int ch, int fromIndex)
1.  int indexOf(String str)
1.  int indexOf(String str, int fromIndex)
1.  native String intern()
1.  boolean isBlank()
1.  boolean isEmpty()
1.  static String join(CharSequence delimiter,
2.      CharSequence... elements)
1.  static String join(CharSequence delimiter,
2.      Iterable<? extends CharSequence> elements)
1.  int lastIndexOf(int ch)
1.  int lastIndexOf(int ch, int fromIndex)
1.  int lastIndexOf(String str)
1.  int lastIndexOf(String str, int fromIndex)
1.  int length()
1.  Stream<String> lines()
1.  boolean matches(String regex)
1.  int offsetByCodePoints(int index,
2.      int codePointOffset)
1.  boolean regionMatches(boolean ignoreCase,
2.      int toffset, String other, int ooffset,
3.      int len)
1.  boolean regionMatches(int toffset, String other,
```

2.	int ooffset, int len)
1.	String repeat(int count)
1.	String replace(char oldChar, char newChar)
1. 2.	String replace(CharSequence target, CharSequence replacement)
1. 2.	String replaceAll(String regex, String replacement)
1. 2.	String replaceFirst(String regex, String replacement)
1.	String[] split(String regex)
1.	String[] split(String regex, int limit)
1.	boolean startsWith(String prefix)
1.	boolean startsWith(String prefix, int toffset)
1.	String strip()
1.	String stripLeading()
1.	String stripTrailing()
1. 2.	CharSequence subSequence(int beginIndex, int endIndex)
1.	String substring(int beginIndex)
1.	String substring(int beginIndex, int endIndex)
1.	char[] toCharArray()
1.	String toLowerCase()
1.	String toLowerCase(Locale locale)
1.	String toString()
1.	String toUpperCase()
1.	String toUpperCase(Locale locale)
1.	String trim()
1.	static String valueOf(boolean b)
1.	static String valueOf(char c)
1.	static String valueOf(char data[])
1. 2.	static String valueOf(char data[], int offset, int count)
1.	static String valueOf(double d)
1.	static String valueOf(float f)
1.	static String valueOf(int i)
1.	static String valueOf(long l)
1.	static String valueOf(Object obj)

Spring Security

Конфигурационный файл:

```
@EnableWebSecurity
public class SecurityConfig
    extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(HttpSecurity http)
      throws Exception {

    http.authorizeRequests()

          // авторизированные пользователи
          .antMatchers("/authenticated/**")
              .authenticated()

          // доступ по ролям
          .antMatchers("/admin/**")
              .hasAnyRole("ADMIN", "SUPERADMIN")

          // доступ по правам authority
          .antMatchers("/profile/**")
              .hasAuthority()

        .and()
          // всплывающее окно
          .httpBasic()
          // своя форма логина
          .formLogin()
          // url страницы для входа
          .loginProcessingUrl("/hellologin")
          // ...
          .successForwardUrl("/authenticated")
          // страница успешного входа
          .defaultSuccessUrl("/authenticated")
          // обработчик успешной аутентификации
          .successHandler()

        .and()
          // страница после выхода
          .logout().logoutSuccessUrl("/");
    }
}
```

Стандартная форма входа:

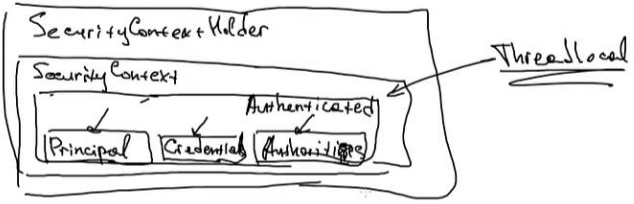
```
<form class="form-signin" method="post"
    action="/security/login">
  <h2 class="form-signin-heading">
      Please sign in</h2>
  <p>
    <label for="username" class="sr-only">
        Username</label>
    <input type="text" id="username"
        name="username" class="form-control"
        placeholder="Username" required autofocus>
  </p>
  <p>
    <label for="password" class="sr-only">
        Password</label>
    <input type="password" id="password"
        name="password" class="form-control"
        placeholder="Password" required>
  </p>
  <input name="_csrf" type="hidden"
      value="b65c8057-6296-44b9-af77-23158cccb80d"
      />
  <button class="btn btn-lg btn-primary btn-block"
      type="submit">Sign in</button>
</form>
```

Выделенные атрибуты name не стоит изменять. Их считывает Spring Security.

Для входа без БД можно воспользоваться логином user и паролем, сгенерированным в консоли.

Объект Principal можно заключить в параметры метода контроллера и получить информацию о пользователе:

```
@GetMapping("/authenticated")
public String pageForAuthenticatedUser(
    Principal principal) {
  return principal.getName();
}
```



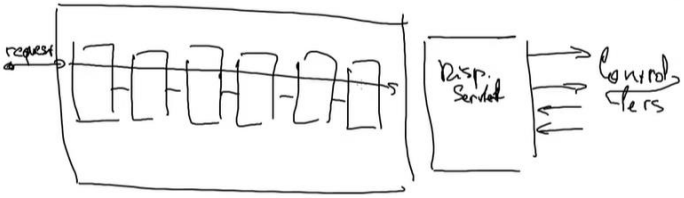
SecurityContextHolder — основное хранилище.

SecurityContext — хранилище данных, хранит данные в ThreadLocal переменной (для каждого потока свои данные)

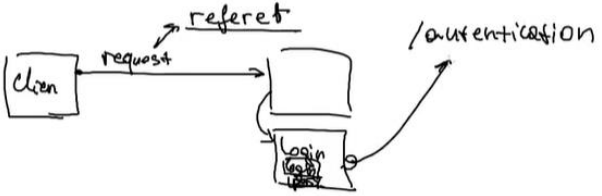
Authenticated: Principal (информация о пользователе), Credentials (пароль, который нужно проверить), Authorities (права доступа).

Credentials чистится сразу после проверки пароля. Principal не хранит в себе пароля. Сделано в целях безопасности.

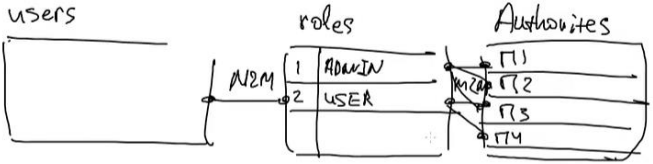
Данные храниться во ThreadLocal переменной тоже в целях безопасности. Пользователь в своем потоке работает, и только о себе информацию знает.



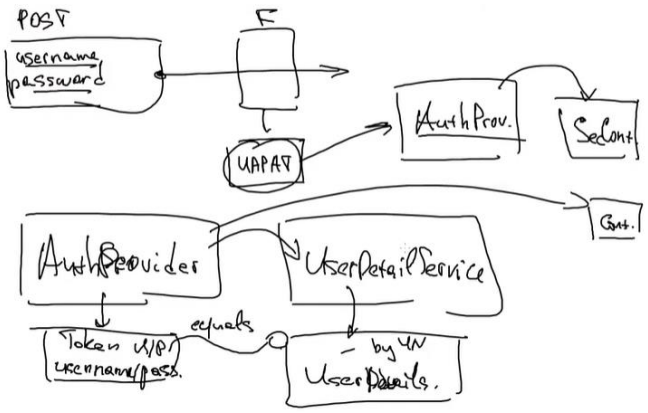
Процесс аутентификация происходит до диспетчера сервлета и обрабатывается множеством фильтров.



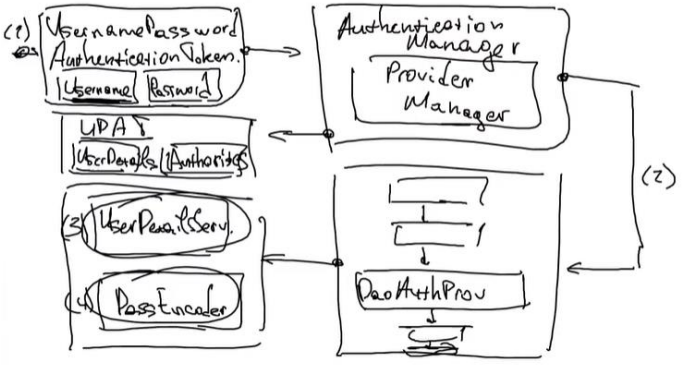
Asd



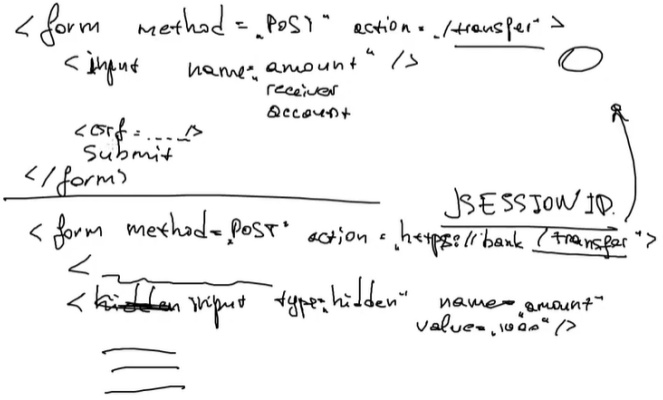
asd


POST
F
username
password
UPAFT
AuthProv.
SecCont.
Cont.
AuthProvider
UserDetailService
equals
by UN

Asd


(1)
UsernamePassword
AuthenticationToken.
Username
Password
UPAT
UserDetails
Authorities
AuthenticationManager
ProviderManager
(2)
(3)
UserDetailServ.
(4)
PassEncoder
DaoAuthProv

Asdf


<form method="POST" action="/transfer">
<input name="amount" />
receiver account
<csrf=...>
Submit
</form>
JSESSIONID
<form method="POST" action="https://bank/transfer">
<input type="hidden" name="amount" value="1000" />

увфы