Оглавление

[Функциональные интерфейсы 2](#_Toc89900750)

[java.lang 7](#_Toc89900751)

[class Object 7](#_Toc89900752)

[final class String 8](#_Toc89900753)

[Spring Security 11](#_Toc89900754)

# Функциональные интерфейсы

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

Predicate<T>

|  |  |
| --- | --- |
| 1. | boolean test(T t); |
| 1.  2.  3.  4.  5. | default Predicate<T> and(  Predicate<? super T> other) {  Objects.requireNonNull(other);  return (t) -> test(t) && other.test(t);  } |
| 1.  2.  3. | default Predicate<T> negate() {  return (t) -> !test(t);  } |
| 1.  2.  3.  4.  5. | default Predicate<T> or(  Predicate<? super T> other) {  Objects.requireNonNull(other);  return (t) -> test(t) || other.test(t);  } |
| 1.  2.  3.  4.  5.  6. | static <T> Predicate<T> isEqual(  Object targetRef) {  return (null == targetRef)  ? Objects::isNull  : object -> targetRef.equals(object);  } |

DoublePredicate

|  |  |
| --- | --- |
| 1. | boolean test(double value); |
| 1.  2.  3.  4.  5.  6. | default DoublePredicate and(  DoublePredicate other) {  Objects.requireNonNull(other);  return (value) ->  test(value) && other.test(value);  } |
| 1.  2.  3. | default DoublePredicate negate() {  return (value) -> !test(value);  } |
| 1.  2.  3.  4.  5.  6. | default DoublePredicate or(  DoublePredicate other) {  Objects.requireNonNull(other);  return (value) ->  test(value) || other.test(value);  } |

BiPredicate<T, U>

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

Adf

Consumer<T>

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

DoubleConsumer

|  |  |
| --- | --- |
| 1. | void accept(double value); |
| 1.  2.  3.  4.  5.  6.  7.  8. | default DoubleConsumer andThen(  DoubleConsumer after) {  Objects.requireNonNull(after);  return (double t) -> {  accept(t);  after.accept(t);  };  } |

BiConsumer<T, U>

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

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.  2.  3.  4.  5. | default <V> Function<V, R> compose(  Function<? super V, ? extends T> before) {  Objects.requireNonNull(before);  return (V v) -> apply(before.apply(v));  } |
| 1.  2.  3.  4.  5. | default <V> Function<T, V> andThen(  Function<? super R, ? extends V> after) {  Objects.requireNonNull(after);  return (T t) -> after.apply(apply(t));  } |
| 1.  2.  3. | static <T> Function<T, T> identity() {  return t -> t;  } |

DoubleFunction<R>

|  |  |
| --- | --- |
| 1. | R apply(double value); |

DoubleToIntFunction

|  |  |
| --- | --- |
| 1. | int applyAsInt(double value); |

DoubleToLongFunction

|  |  |
| --- | --- |
| 1. | long applyAsLong(double value); |

DoubleUnaryOperator

|  |  |
| --- | --- |
| 1. | double applyAsDouble(double operand); |
| 1.  2.  3.  4.  5.  6. | default DoubleUnaryOperator compose(  DoubleUnaryOperator before) {  Objects.requireNonNull(before);  return (double v) ->  applyAsDouble(before.applyAsDouble(v));  } |
| 1.  2.  3.  4.  5.  6. | default DoubleUnaryOperator andThen(  DoubleUnaryOperator after) {  Objects.requireNonNull(after);  return (double t) ->  after.applyAsDouble(applyAsDouble(t));  } |
| 1.  2.  3. | static DoubleUnaryOperator identity() {  return t -> t;  } |

BiFunction<T, U, R>

|  |  |
| --- | --- |
| 1. | R apply(T t, U u); |
| 1.  2.  3.  4.  5. | 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));  } |

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

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

DoubleBinaryOperator

|  |  |
| --- | --- |
| 1. | double applyAsDouble(double left, double right); |

Другие

ыв

# java.lang

## class Object

Конструктор

|  |  |
| --- | --- |
| 1. | public Object() |

Методы

|  |  |
| --- | --- |
| 1.  2. | protected native Object clone()  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.  2. | final native void wait(long timeoutMillis)  throws InterruptedException |
| 1.  2. | final void wait(long timeoutMillis, int nanos)  throws InterruptedException |

## final class String

extends [Object](#_class_Object)

implements

|  |  |
| --- | --- |
| 1. | CharSequence |
| 1. | Comparable<String> |
| 1. | java.io.Serializable |

Поле

|  |  |
| --- | --- |
| 1.  2.  3. | static final  Comparator<String> CASE\_INSENSITIVE\_ORDER =  new CaseInsensitiveComparator(); |

Конструкторы

|  |  |
| --- | --- |
| 1. | String() |
| 1. | String(byte[] bytes) |
| 1. | String(byte bytes[], int offset, int length) |
| 1.  2. | String(byte bytes[], int offset, int length,  String charsetName) |
| 1.  2. | String(byte bytes[], int offset, int length,  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.  2. | static String copyValueOf(char data[], int offset,  int count) |
| 1. | boolean endsWith(String suffix) |
| 1. | boolean equals(Object anObject) |
| 1. | boolean equalsIgnoreCase(String anotherString) |
| 1.  2. | static String format(String format,  Object... args) |
| 1.  2. | static String format(Locale l, String format,  Object... args) |
| 1. | byte[] getBytes() |
| 1. | byte[] getBytes(String charsetName) |
| 1. | byte[] getBytes(Charset charset) |
| 1.  2. | void getChars(int srcBegin, int srcEnd,  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.  2. | static String join(CharSequence delimiter,  CharSequence... elements) |
| 1.  2. | static String join(CharSequence delimiter,  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.  2. | int offsetByCodePoints(int index,  int codePointOffset) |
| 1.  2.  3. | boolean regionMatches(boolean ignoreCase,  int toffset, String other, int ooffset,  int len) |
| 1.  2. | boolean regionMatches(int toffset, String other,  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:

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41. | @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:

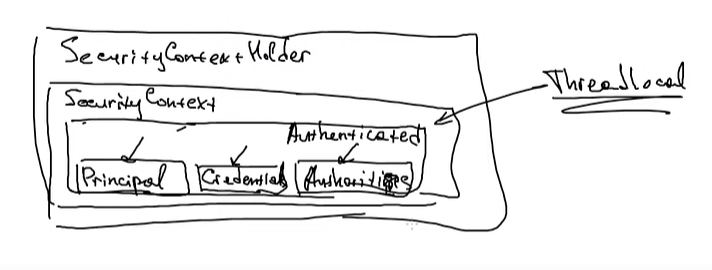
|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20.  21.  22.  23.  24. | <form class="form-signin" method="post"  action="/security/login">  <h2 class="form-signin-heading">  Please sign in</h2>  <p>  <label for="username" class="sr-only">  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:

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5. | @GetMapping("/authenticated")  public String pageForAuthenticatedUser(  Principal principal) {  return principal.getName();  } |



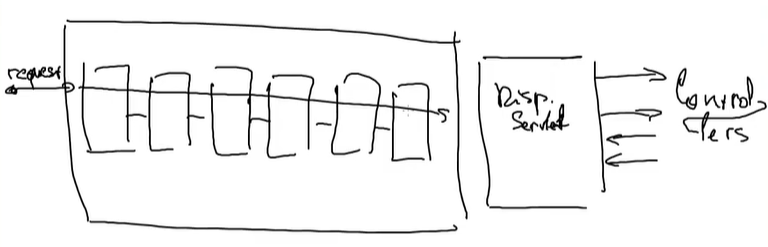
SecurityContextHolder — основное хранилище.

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

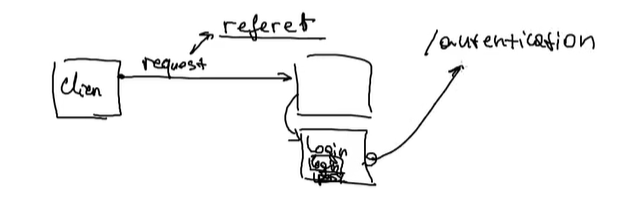
Authenticated: Principal (информация о пользователе), Credentials (пароль, который нужно проверить), Authorities (права доступа).

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

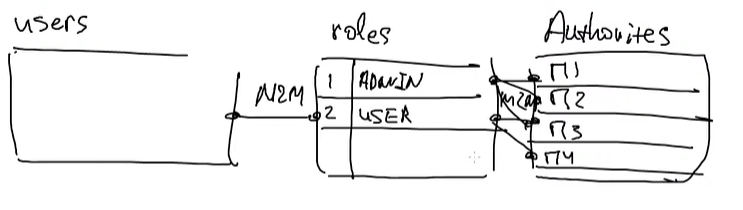
Данные храниться во ThreadLocal переменной тоже в целях безопасности. Пользователь в своем потоке работает, и только о себе информацию знает.



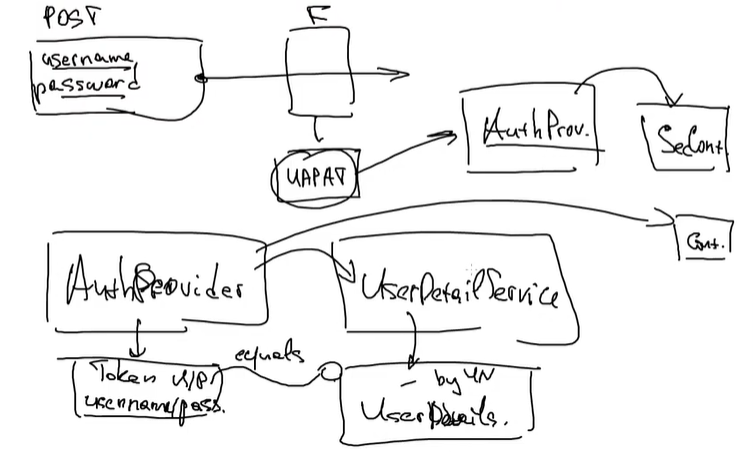
Процесс аутентификация происходит до диспетчера сервлета и обрабатывается множеством фильтров.



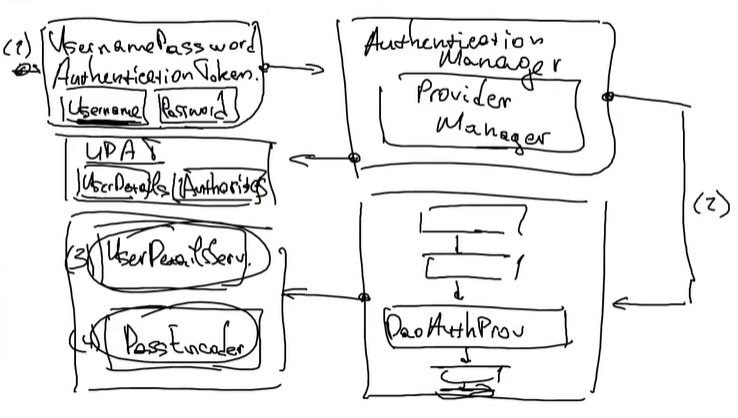
Asd



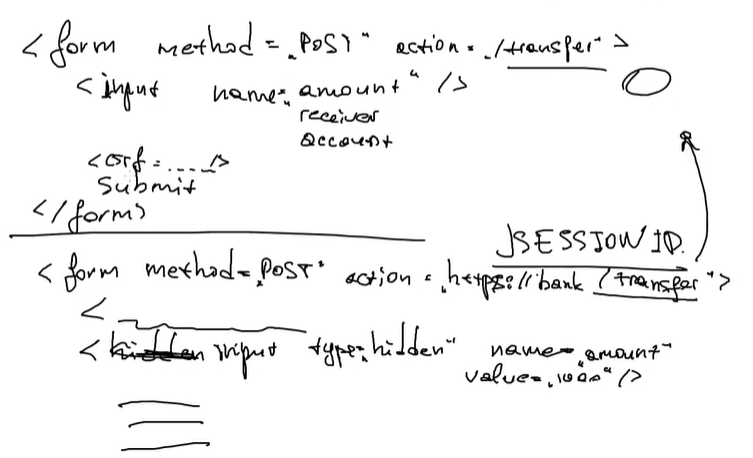
asd



Asd



Asdf



увфы