What are Selenium exceptions?

Definition

An exception is known as an unusual or unprecedented event that occurs during the execution of a software program or application. It is a runtime error of an unexpected result or event which influence and disrupt usual program flow. An exception is also considered as a fault.

Classification of Selenium exceptions

Selenium exceptions are divided into two types including Checked Exceptions and Unchecked Exceptions.

1. Checked Exceptions

Checked Exceptions are handled during the process of writing codes. These exceptions are handled before compiling the code, therefore, such exceptions are examined at the compile time.

2. Unchecked Exceptions

These exceptions are thrown at runtime. Unchecked exceptions are more catastrophic than the compile-time exception as it causes problems while running Automation pack in headless.

The complete list of exceptions in Selenium

1. ConnectionClosedException: This exception takes place when there is a disconnection in the driver.

2. ElementClickInterceptedException: The command could not be completed as the element receiving the events is concealing the element which was requested clicked.

3. ElementNotInteractableException: This Selenium exception is thrown when an element is presented in the DOM but it is impossible to interact with such element.

4. ElementNotSelectableException: This Selenium exception is thrown when an element is presented in the DOM but is unavailable for selection. Hence, it is impossible to interact with.

5. ElementNotVisibleException: This type of Selenium exception takes place when an existing element in DOM has a feature set as hidden. In this situation, elements are there, but you can not see and interact with the WebDriver.

6. ErrorHandler.UnknownServerException: Exception is used as a placeholder if the server returns an error without a stack trace.

7. ErrorInResponseException: This exception is thrown when a fault has occurred on the server-side. You can see it happens when interacting with the Firefox extension or the remote driver server.

8. ImeActivationFailedException: This exception occurs when IME engine activation has failed.

9. ImeNotAvailableException: This exception takes place when IME support is unavailable.

10. InsecureCertificateException: Navigation made the user agent to hit a certificate warning, which is caused by an invalid or expired TLS certificate.

11. InvalidArgumentException: This Selenium exception is thrown if an argument does not belong to the expected type.

12. InvalidCookieDomainException: This happens when you try to add a cookie under a different domain rather than the current URL.

13. InvalidCoordinatesException: This happens if the coordinates offered to an interacting operation are not valid.

14. InvalidElementStateException: This Selenium exception occurs if a command cannot be finished as the element is invalid.

15. InvalidSessionIdException: Takes place when the given session ID is not included in the list of active sessions, which means the session does not exist or is inactive either.

16. InvalidSwitchToTargetException: Happens if the frame or window target to be switched does not exist.

17. JavascriptException: This problem happens when executing JavaScript supplied by the user.

18. JsonException: Happens when you afford to get the session capabilities where the session is not created.

19. MoveTargetOutOfBoundsException: Takes place if the target provided to the ActionChains move() methodology is not valid. For example: out of the document.

20. NoAlertPresentException: Happens when you switch to no presented alert.

21. NoSuchAttributeException: Occurs when the attribute of the element could not be found.

22. NoSuchContextException: Happens in mobile device testing and is thrown by ContextAware.

23. NoSuchCookieException: This exception is thrown if there is no cookie matching with the given path name found amongst the associated cookies of the current browsing context's active document.

24. NoSuchElementException: Happens if an element could not be found.

25. NoSuchFrameException: Takes place if frame target to be switch does not exist.

26. NoSuchWindowException: Occurs if window target to be switched does not exist.

27. NotFoundException: This exception is a subclass of WebDriverException. It happens when an element on the DOM does not exist.

28. RemoteDriverServerException: This Selenium exception is thrown when the server does not respond due to the problem that the capabilities described are not proper.

29. ScreenshotException: It is impossible to capture a screen.

30. ScriptTimeoutException: Thrown when executeAsyncScript takes more time than the given time limit to return the value.

31. SessionNotCreatedException: A new session could not be successfully created.

32. SessionNotFoundException: The WebDriver is performing the action right after you quit the browser.

33. StaleElementReferenceException: This Selenium exception happens if the web element is detached from the current DOM.

34. TimeoutException: Thrown when there is not enough time for a command to be completed.

35. UnableToCreateProfileException: You can open a browser with certain options using profiles, but sometimes a new version of the Selenium driver server or browser may not support the profiles.

36. UnableToSetCookieException: Occurs if a driver is unable to set a cookie.

37. UnexpectedAlertPresentException: This Selenium exception happens when there is the appearance of an unexpected alert.

38. UnexpectedTagNameException: Happens if a support class did not get a web element as expected.

39. UnhandledAlertException: It happens when there is an alert, but WebDriver is unable to perform Alert operation.

40. UnknownMethodException: Thrown when the requested command matches with a known URL but not matching with a methodology for that URL.

41. UnreachableBrowserException: This Selenium exception happens if the browser is unable to be opened or has crashed because of some reasons.

42. UnsupportedCommandException: Occurs when remote WebDriver does not send valid commands as expected.

43. WebDriverException: This takes place when the WebDriver is performing the action right after you close the browser.

Selenium exceptions handling

What is Exception handling?

In a software program, an atypical event (e.g. unrecognized messages) can interfere with the regular execution flow. These events, if not handled properly, can result in termination of the program by immediate throwing exceptions. Handling exceptions is the process of managing these atypical events in order to prevent such problems from arising.

Selenium exception solutions

In order to handle these above types of Selenium exceptions, this article will discuss some compiler directives to support exception handling.

- **Throw:** Throw keyword is used to throw exception to the runtime to handle it.

```
// Method Signature\ public static void anyFunction() throws Exception{ try{ // write your code here }catch (Exception e){ // Do whatever you wish to do here // Now throw the exception back to the system throw(e); } }
```

```
1  // Method Signature\
2
3  public static void anyFunction ( ) throws Exception {
4
5  try {
6
7  // write your code here
8
9  } catch ( Exception e ) {
10
11 // Do whatever you wish to do here
12
13 // Now throw the exception back to the system
14
15 throw ( e ) ;
16
17 }
18
19 }
```

- **Multiple Catch blocks**: You can have multiple @catch() blocks to catch different types of exception. The syntax for multiple catch blocks looks like the following:

```
try { //Some code }catch(ExceptionType1 e1){ //Code for Handling the Exception 1 }catch(ExceptionType2 e2){
//Code for Handling the Exception 2 }
```

```
try
{
//Some code
} catch ( ExceptionType1 e1 ) {
//Code for Handling the Exception 1
} catch ( ExceptionType2 e2 ) {
//Code for Handling the Exception 2
}
```

- **Try/Catch**: A @try block encloses code that can potentially throw an exception. A @catch() block contains exception-handling logic for exceptions thrown in a @try block. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try { // Some code }catch(Exception e){ // Code for Handling the exception }
```

```
try
{
// Some code
} catch ( Exception e ) {
// Code for Handling the exception
}
```

- **Finally**: A @finally block contains code that must be executed whether an exception is thrown or not:

try { //Protected code }catch(ExceptionType1 e1) { //Catch block }catch(ExceptionType2 e2) { //Catch block }catch(ExceptionType3 e3) { //Catch block }finally { //The finally block always executes. }

```
1  try
2
3  {
4
5  //Protected code
6
7  } catch ( ExceptionType1 e1 )
8
9  {
10
11 //Catch block
12
13 } catch ( ExceptionType2 e2 )
14
15 {
16
17 //Catch block
18
19 } catch ( ExceptionType3 e3 )
20
21 {
22
23 //Catch block
24
25 } finally
26
27 {
28
29 //The finally block always executes.
30
31 }
```