一、实验名称

自动化测试（12. Selenium 实战项目）

二、实验目的

使用 Selenium WebDriver 与 pytest 对“登录 / 注册 / 搜索”三个流程进行自动化测试。

三、实验内容

本实验选择直接完成“12. Selenium 实战项目”。参考资料中登录/注册示例使用 `https://example.com/*` 作为占位地址，实际不可用，无法执行稳定断言。为保证用例可运行，实现了两张本地静态页面（登录/注册）作为被测对象，并在其上编写 Selenium 自动化测试脚本；搜索流程使用 Bing。

为保证用例可执行性，做了两点处理：

1. 登录/注册使用离线页面作为被测对象，页面与断言由本实验控制。
2. 搜索流程使用 Bing。

四、实验仪器

实验运行环境如下：

1. 操作系统：`Linux-6.18.2-2-cachyos-x86_64-with-glibc2.42`
2. Python：`3.13.11`
3. pytest：`9.0.2`
4. selenium：`4.39.0`
5. 浏览器：Chrome `143.0.7499.169`

五、实验原理

Selenium WebDriver 通过浏览器驱动控制真实浏览器实例，并通过元素定位、用户交互与等待机制实现 Web 自动化测试。本实验主要涉及以下原理与机制：

1. 元素定位：使用 `By.ID` 定位输入框、按钮与提示文本等元素，依赖页面中稳定的 `id` 属性保证定位可靠性。
2. 交互模拟：通过 `send_keys()` 完成输入，通过 `click()` 触发前端逻辑，模拟用户在页面上的主要操作路径。

3. 显式等待：使用 `WebDriverWait(...).until(...)` 等待关键状态出现（如提示文本包含预期字符串、页面标题包含关键字），降低因页面渲染与脚本执行时序导致的偶发失败。
4. 离线页面：登录/注册页面以本地 HTML 文件作为被测对象，通过 `file://` URL 打开，使页面内容与断言点可控，避免外部站点变更对测试结果的影响。
5. pytest 断言：pytest 负责用例组织与结果判定，通过 `assert` 给出可自动化评估的测试结论；环境不可用（如 WebDriver 启动失败）通过 `pytest.skip(...)` 与业务失败区分。

六、实验过程及实验结果

1. 用例设计与断言点

为每个流程选取了明确的断言点，并把输入数据与操作步骤固定下来，便于复现。三个用例如下：

用例	输入/数据	操作步骤（概述）	断言点
登录（离线）	用户名 `testuser`；密码 `password123`	打开离线登录页 → 输入用户名/密码 → 点击登录	页面提示文本包含 `Welcome`
注册（离线）	用户名 `newuser`；邮箱 `newuser@example.com`；密码两次一致	打开离线注册页 → 填写表单 → 点击注册	页面提示文本包含 `Registration successful`
搜索（联网）	关键词 `runoob`	打开 Bing 搜索结果页	页面标题包含 `runoob`

下面依次给出关键实现与代码。

2. 离线被测页面

离线页面仅保留测试需要的交互，并为关键元素提供固定的 `id`（例如 `username`、`password`、`message`），用于 Selenium 定位与断言。

离线登录页面完整代码如下：

```
<!doctype html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Login</title>
    <style>
      :root {
        color-scheme: light dark;
        --bg: #0b1220;
        --card: #111a2e;
        --text: #e8edf7;
        --muted: #aab6d1;
        --accent: #4f8cff;
        --danger: #ff5a7a;
        --ok: #35c07a;
      }

      body {
        margin: 0;
        min-height: 100vh;
        display: grid;
        place-items: center;
        background: radial-gradient(1200px 800px at 20% 10%, #1b2a55 0%, var(--bg)
55%);
        font-family: ui-sans-serif, system-ui, -apple-system, BlinkMacSystemFont,
"Segoe UI",
          "PingFang SC", "Hiragino Sans GB", "Microsoft YaHei", Arial, sans-serif;
        color: var(--text);
      }

      .card {
        width: min(420px, calc(100vw - 40px));
        background: color-mix(in srgb, var(--card) 90%, #000 10%);
        border: 1px solid color-mix(in srgb, #fff 16%, transparent);
        border-radius: 16px;
        padding: 22px 22px 18px;
        box-shadow: 0 20px 60px rgba(0, 0, 0, 0.55);
      }

      h1 {
        font-size: 18px;
        margin: 0 0 14px;
      }

      label {
        display: block;
        margin: 10px 0;
```

```
  color: var(--muted);
  font-size: 13px;
}

input {
  margin-top: 6px;
  width: 100%;
  box-sizing: border-box;
  padding: 10px 12px;
  border-radius: 10px;
  border: 1px solid color-mix(in srgb, #fff 14%, transparent);
  background: color-mix(in srgb, #000 14%, transparent);
  color: var(--text);
  outline: none;
}

input:focus {
  border-color: color-mix(in srgb, var(--accent) 70%, #fff 10%);
  box-shadow: 0 0 0 4px color-mix(in srgb, var(--accent) 18%, transparent);
}

button {
  width: 100%;
  margin-top: 10px;
  padding: 10px 12px;
  border-radius: 10px;
  border: 0;
  background: var(--accent);
  color: #fff;
  font-weight: 600;
  cursor: pointer;
}

#message {
  margin: 12px 0 0;
  min-height: 18px;
  font-size: 13px;
  color: var(--muted);
}

#message[data-status="success"] {
  color: var(--ok);
}

#message[data-status="error"] {
  color: var(--danger);
}

.hint {
```

```
        margin-top: 10px;
        font-size: 12px;
        color: color-mix(in srgb, var(--muted) 85%, #fff 15%);
      }
    </style>
  </head>
  <body>
    <main class="card">
      <h1>登录</h1>
      <label>
        用户名
        <input id="username" type="text" autocomplete="username" />
      </label>
      <label>
        密码
        <input id="password" type="password" autocomplete="current-password" />
      </label>
      <button id="login-button" type="button">登录</button>
      <p id="message" aria-live="polite"></p>
      <p class="hint">测试账号：testuser / password123</p>
    </main>
    <script>
      (function () {
        const usernameInput = document.getElementById("username");
        const passwordInput = document.getElementById("password");
        const loginButton = document.getElementById("login-button");
        const message = document.getElementById("message");

        function setMessage(text, ok) {
          message.textContent = text;
          message.dataset.status = ok ? "success" : "error";
        }

        loginButton.addEventListener("click", function () {
          const username = usernameInput.value.trim();
          const password = passwordInput.value;

          if (username === "testuser" && password === "password123") {
            setMessage("Welcome, " + username, true);
            return;
          }

          setMessage("Invalid credentials", false);
        });
      })();
    </script>
  </body>
</html>
```

```
<!doctype html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Register</title>
    <style>
      :root {
        color-scheme: light dark;
        --bg: #0b1220;
        --card: #111a2e;
        --text: #e8edf7;
        --muted: #aab6d1;
        --accent: #4f8cff;
        --danger: #ff5a7a;
        --ok: #35c07a;
      }

      body {
        margin: 0;
        min-height: 100vh;
        display: grid;
        place-items: center;
        background: radial-gradient(1200px 800px at 20% 10%, #1b2a55 0%, var(--bg)
55%);
        font-family: ui-sans-serif, system-ui, -apple-system, BlinkMacSystemFont,
"Segoe UI",
          "PingFang SC", "Hiragino Sans GB", "Microsoft YaHei", Arial, sans-serif;
        color: var(--text);
      }

      .card {
        width: min(460px, calc(100vw - 40px));
        background: color-mix(in srgb, var(--card) 90%, #000 10%);
        border: 1px solid color-mix(in srgb, #fff 16%, transparent);
        border-radius: 16px;
        padding: 22px 22px 18px;
        box-shadow: 0 20px 60px rgba(0, 0, 0, 0.55);
      }

      h1 {
        font-size: 18px;
        margin: 0 0 14px;
      }

      label {
        display: block;
        margin: 10px 0;
```

```
      color: var(--muted);
      font-size: 13px;
    }

    input {
      margin-top: 6px;
      width: 100%;
      box-sizing: border-box;
      padding: 10px 12px;
      border-radius: 10px;
      border: 1px solid color-mix(in srgb, #fff 14%, transparent);
      background: color-mix(in srgb, #000 14%, transparent);
      color: var(--text);
      outline: none;
    }

    input:focus {
      border-color: color-mix(in srgb, var(--accent) 70%, #fff 10%);
      box-shadow: 0 0 0 4px color-mix(in srgb, var(--accent) 18%, transparent);
    }

    button {
      width: 100%;
      margin-top: 10px;
      padding: 10px 12px;
      border-radius: 10px;
      border: 0;
      background: var(--accent);
      color: #fff;
      font-weight: 600;
      cursor: pointer;
    }

    #message {
      margin: 12px 0 0;
      min-height: 18px;
      font-size: 13px;
      color: var(--muted);
    }

    #message[data-status="success"] {
      color: var(--ok);
    }

    #message[data-status="error"] {
      color: var(--danger);
    }
  </style>
</head>
```

```
<body>
  <main class="card">
    <h1>注册</h1>
    <label>
      用户名
      <input id="username" type="text" autocomplete="username" />
    </label>
    <label>
      邮箱
      <input id="email" type="email" autocomplete="email" />
    </label>
    <label>
      密码
      <input id="password" type="password" autocomplete="new-password" />
    </label>
    <label>
      确认密码
      <input id="confirm-password" type="password" autocomplete="new-password" />
    </label>
    <button id="register-button" type="button">注册</button>
    <p id="message" aria-live="polite"></p>
  </main>
  <script>
    (function () {
      const usernameInput = document.getElementById("username");
      const emailInput = document.getElementById("email");
      const passwordInput = document.getElementById("password");
      const confirmPasswordInput = document.getElementById("confirm-password");
      const registerButton = document.getElementById("register-button");
      const message = document.getElementById("message");

      function setMessage(text, ok) {
        message.textContent = text;
        message.dataset.status = ok ? "success" : "error";
      }

      registerButton.addEventListener("click", function () {
        const username = usernameInput.value.trim();
        const email = emailInput.value.trim();
        const password = passwordInput.value;
        const confirmPassword = confirmPasswordInput.value;

        if (!username || !email || !password) {
          setMessage("Registration failed", false);
          return;
        }

        if (password !== confirmPassword) {
          setMessage("Registration failed", false);
```

```
            return;
          }

          setMessage("Registration successful", true);
        });
      })();
    </script>
  </body>
</html>
```

离线页面通过 `file://` 方式打开；Python 侧使用 `Path.as_uri()` 生成 file URL（见 `to_file_url`）。

3. WebDriver 管理与可移植性

为避免在每个用例里重复初始化浏览器，将 WebDriver 的创建、无头配置与超时设置封装成 `create_webdriver()`。运行时 WebDriver 使用 Chrome `143.0.7499.169`；无头模式由 `SELENIUM_HEADLESS` 控制，代码默认值为 `true`。

`create_webdriver()` 与 `to_file_url()` 代码如下：

```python
from __future__ import annotations

import os
from pathlib import Path
from typing import Literal, cast

from selenium import webdriver
from selenium.common.exceptions import WebDriverException
from selenium.webdriver.chrome.options import Options as ChromeOptions
from selenium.webdriver.chrome.service import Service as ChromeService
from selenium.webdriver.firefox.options import Options as FirefoxOptions
from selenium.webdriver.firefox.service import Service as FirefoxService
from selenium.webdriver.remote.webdriver import WebDriver

BrowserName = Literal["chrome", "firefox"]


class WebDriverUnavailableError(RuntimeError):
    pass


def _env_bool(name: str, default: bool) -> bool:
    raw = os.getenv(name)
    if raw is None:
        return default
    value = raw.strip().lower()
    if value in {"1", "true", "yes", "y", "on"}:
        return True
    if value in {"0", "false", "no", "n", "off"}:
        return False
    return default


def _resolve_driver_path(env_name: str) -> str | None:
    raw = os.getenv(env_name)
    if not raw:
        return None
    path = Path(raw).expanduser()
    return str(path)


def create_webdriver(
    *,
    browser: BrowserName | None = None,
    headless: bool | None = None,
) -> WebDriver:
    raw_browser = (browser or os.getenv("SELENIUM_BROWSER", "chrome")).strip().lower()
    if raw_browser not in {"chrome", "firefox"}:
```

```
        raise ValueError(f"不支持的浏览器类型：{raw_browser!r}（仅支持
chrome/firefox）")

    selected_browser = cast(BrowserName, raw_browser)

    use_headless = _env_bool("SELENIUM_HEADLESS", True) if headless is None else
headless

    try:
        if selected_browser == "chrome":
            options = ChromeOptions()
            if use_headless:
                options.add_argument("--headless")
            options.add_argument("--window-size=1280,720")
            options.add_argument("--disable-gpu")

            driver_path = _resolve_driver_path("CHROMEDRIVER_PATH")
            if driver_path:
                service = ChromeService(executable_path=driver_path)
                driver = webdriver.Chrome(service=service, options=options)
            else:
                driver = webdriver.Chrome(options=options)

        else:
            options = FirefoxOptions()
            options.headless = use_headless

            driver_path = _resolve_driver_path("GECKODRIVER_PATH")
            if driver_path:
                service = FirefoxService(executable_path=driver_path)
                driver = webdriver.Firefox(service=service, options=options)
            else:
                driver = webdriver.Firefox(options=options)

        driver.set_page_load_timeout(15)
        return driver
    except WebDriverException as exc:
        raise WebDriverUnavailableError(
            "WebDriver 启动失败：请确认已安装浏览器与对应驱动；"
            "或设置环境变量 CHROMEDRIVER_PATH/GECKODRIVER_PATH 指向驱动可执行文件。"
        ) from exc
    except RuntimeError as exc:
        message = str(exc)
        if "Can't find free port" in message or "Unable to bind" in message:
            raise WebDriverUnavailableError(
                "WebDriver 启动失败：当前环境不允许创建本地监听端口（socket bind 被拒
绝）。"
            ) from exc
        raise
```

```
def to_file_url(path: Path) -> str:
    return path.resolve().as_uri()
```

4. Selenium 示例脚本与断言

本实验实现了三个脚本函数 `run()`：离线登录、离线注册、Bing 搜索。登录/注册脚本打开本地页面，填写表单并点击按钮，再通过显式等待等待提示文本出现并返回；搜索脚本打开 Bing 搜索结果页，等待标题包含关键字并返回标题。

离线登录脚本代码如下：

```
from __future__ import annotations

from pathlib import Path

from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as ec
from selenium.webdriver.support.ui import WebDriverWait

from src.driver_factory import create_webdriver, to_file_url


def run() -> str:
    driver = create_webdriver()
    try:
        src_dir = Path(__file__).resolve().parents[1]
        page_url = to_file_url(src_dir / "webapp" / "login.html")

        driver.get(page_url)

        driver.find_element(By.ID, "username").send_keys("testuser")
        driver.find_element(By.ID, "password").send_keys("password123")
        driver.find_element(By.ID, "login-button").click()

        message_locator = (By.ID, "message")
        WebDriverWait(driver,
5).until(ec.text_to_be_present_in_element(message_locator, "Welcome"))
        return driver.find_element(*message_locator).text
    finally:
        driver.quit()


def main() -> None:
    print(run())


if __name__ == "__main__":
    main()
```

离线注册脚本代码如下：

```python
from __future__ import annotations

from pathlib import Path

from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as ec
from selenium.webdriver.support.ui import WebDriverWait

from src.driver_factory import create_webdriver, to_file_url


def run() -> str:
    driver = create_webdriver()
    try:
        src_dir = Path(__file__).resolve().parents[1]
        page_url = to_file_url(src_dir / "webapp" / "register.html")

        driver.get(page_url)

        driver.find_element(By.ID, "username").send_keys("newuser")
        driver.find_element(By.ID, "email").send_keys("newuser@example.com")
        driver.find_element(By.ID, "password").send_keys("newpassword123")
        driver.find_element(By.ID, "confirm-password").send_keys("newpassword123")
        driver.find_element(By.ID, "register-button").click()

        message_locator = (By.ID, "message")
        WebDriverWait(driver, 5).until(
            ec.text_to_be_present_in_element(message_locator, "Registration
successful")
        )
        return driver.find_element(*message_locator).text
    finally:
        driver.quit()


def main() -> None:
    print(run())


if __name__ == "__main__":
    main()
```

Bing 搜索脚本代码如下：

```python
from __future__ import annotations

from urllib.parse import quote_plus

from selenium.webdriver.support.ui import WebDriverWait

from src.driver_factory import create_webdriver


def run(*, keyword: str = "runoob") -> str:
    driver = create_webdriver()
    try:
        # 直接打开搜索结果页，降低首页弹窗/策略页对自动化操作的影响。
        driver.get(f"https://www.bing.com/search?q={quote_plus(keyword)}")

        WebDriverWait(driver, 15).until(lambda d: keyword.lower() in d.title.lower())
        return driver.title
    finally:
        driver.quit()


def main() -> None:
    print(run())


if __name__ == "__main__":
    main()
```

三个示例脚本均把关键结果（提示文本或页面标题）作为返回值返回给 pytest 用例，使测试层只关注断言本身，从而保持职责边界清晰。

5. pytest 用例

本实验使用 pytest 作为统一执行入口，pytest 用例直接调用上一节脚本的 `run()` 并做断言。当 WebDriver 启动失败或页面等待超时时，使用 `pytest.skip(...)` 结束并记录原因。

登录用例代码如下：

```python
import pytest

from src.driver_factory import WebDriverUnavailableError
from src.examples import login_example


def test_login_example():
    try:
        message = login_example.run()
    except WebDriverUnavailableError as exc:
        pytest.skip(str(exc))

    assert "Welcome" in message
```

注册用例代码如下：

```python
import pytest

from src.driver_factory import WebDriverUnavailableError
from src.examples import register_example


def test_register_example():
    try:
        message = register_example.run()
    except WebDriverUnavailableError as exc:
        pytest.skip(str(exc))

    assert "Registration successful" in message
```

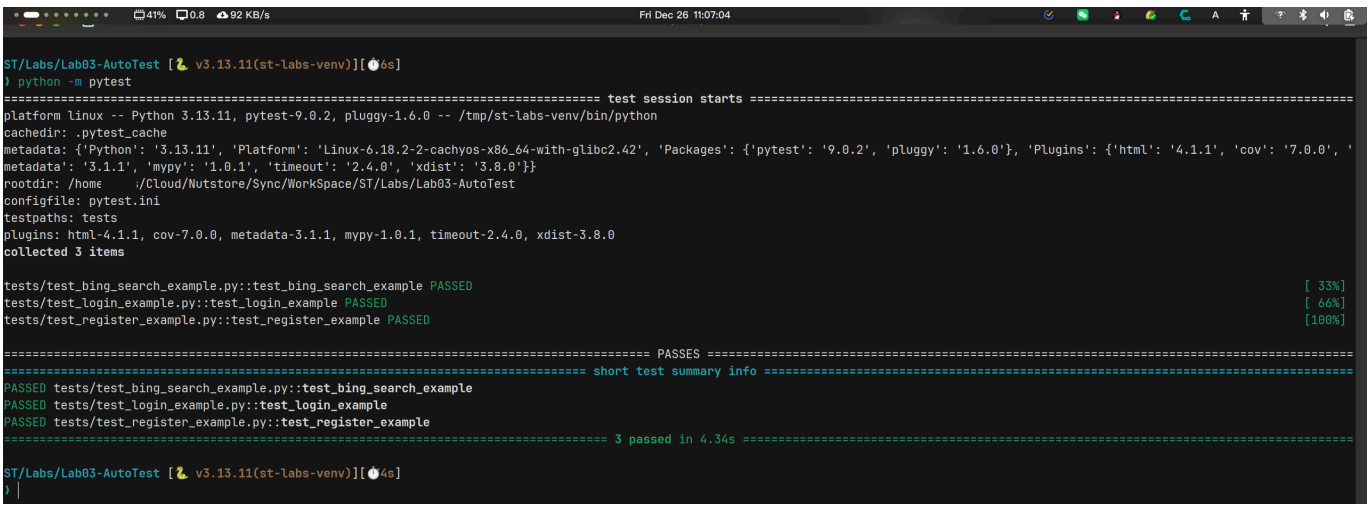
联网搜索用例代码如下：

```python
import pytest
from selenium.common.exceptions import TimeoutException, WebDriverException

from src.driver_factory import WebDriverUnavailableError
from src.examples import bing_search_example


def test_bing_search_example():
    try:
        title = bing_search_example.run(keyword="runoob")
    except WebDriverUnavailableError as exc:
        pytest.skip(str(exc))
    except TimeoutException as exc:
        pytest.skip(f"Bing 页面加载/元素等待超时：{exc}")
    except WebDriverException as exc:
        pytest.skip(f"Bing 用例执行异常：{exc}")

    assert "runoob" in title.lower()
```

6. 实验步骤与运行结果

执行命令：`python -m pytest`

终端输出：



七、讨论与结论

1. 占位站点不可用：参考资料中的 `https://example.com/*` 为占位地址，无法用于稳定断言。实验实现离线页面（登录/注册），并用固定 `id` 作为定位入口，使被测对象与断言点稳定可控。

2. 百度搜索用例失败：实现阶段曾按“以百度为例”编写搜索用例，但执行 `python -m pytest` 时出现如下报错：

```
selenium.common.exceptions.ElementNotInteractableException: Message: element not
interactable
selenium.common.exceptions.TimeoutException: Message:
```

为保证全量用例可运行通过，最终将搜索用例切换为 Bing，并移除百度相关代码，仅保留 Bing 搜索用例。

结论如下：本实验将三个脚本示例落地为三个独立的可执行用例，并通过离线页面解决了“示例站点为占位地址导致无法断言”的问题。实现过程中，显式等待用于降低 UI 自动化的偶发失败率；统一的 WebDriver 工厂用于隔离环境差异、减少重复代码。整体结构按“driver 工厂 → 示例脚本 → pytest 用例”分层，既能直接运行示例脚本，也能以 pytest 方式批量执行，更符合工程化自动化测试的组织方式。