平衡k臂赌博机问题

10臂赌博机实验

k = 10, 动作的期望价值$q_*(a)$通过均值为0，方差为1的正态分布随机得到，a = 1,2,3 . . . ,10

In [20]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#设置中文字体与美化样式
plt.rcParams["font.sans-serif"] = ["SimHei"]
plt.rcParams["axes.unicode_minus"] = False
sns.set(style="whitegrid")
```

In [21]:
```
#定义赌博机环境
class KArmedBandit:
    def __init__(self, k = 10, seed = None):
        """
        初始化k臂赌博机
        K: 动作数量
        seed: 随机种子，使实验可复现
        """
        self.k = k
        if seed is not None:
            np.random.seed(seed)
        self.q_star = np.random.randn(k)
        self.best_action = np.argmax(self.q_star)

    def get_q_star(self):
        return self.q_star

    def step(self,action):
        """
        执行动作，返回奖励
        奖励 ~ N(q_star[action],1)
        """
        return np.random.randn()+self.q_star[action]


# 定义 ε-贪婪智能体
class EpsilonGreedyAgent:
    def __init__(self,k, epsilon, seed=None):
        """
        k: 动作数量
        epsilon: 探索概率
        """
        self.k = k
        self.epsilon = epsilon
        if seed is not None:
            np.random.seed(seed)

        self.Q = np.zeros(k)     # 定义奖励函数
```

```python
        self.N = np.zeros(k)    # 保存每个动作被选择的次数，方便计算状态奖励

    def select_action(self):
        """
        动作选择
        """
        if np.random.rand()<self.epsilon:
            # exploration 探索，选择随机动作
            return np.random.randint(self.k)
        else:
            # exploitation
            max_Q = np.max(self.Q)
            best_actions = np.where(self.Q==max_Q)[0]  #where函数返回满足条件的位
            return np.random.choice(best_actions)   #当有多个最大奖励时，随机返回

    def update(self,action,reward):
        """
        更新
        """
        self.N[action] += 1
        self.Q[action] += (reward-self.Q[action])/self.N[action]
```

In [27]:
```python
# 运行单次实验
def run_expriment(bandit, agent, steps=1000):
    """
    运行一次实验
    返回：每一步的奖励，是否选择了最优动作
    """
    rewards = np.zeros(steps)
    is_best_actions = np.zeros(steps,dtype=bool)

    for t in range(steps):
        action = agent.select_action()
        reward = bandit.step(action)
        agent.update(action,reward)

        rewards[t] = reward
        is_best_actions[t] = (action == bandit.best_action)

    return rewards, is_best_actions
```

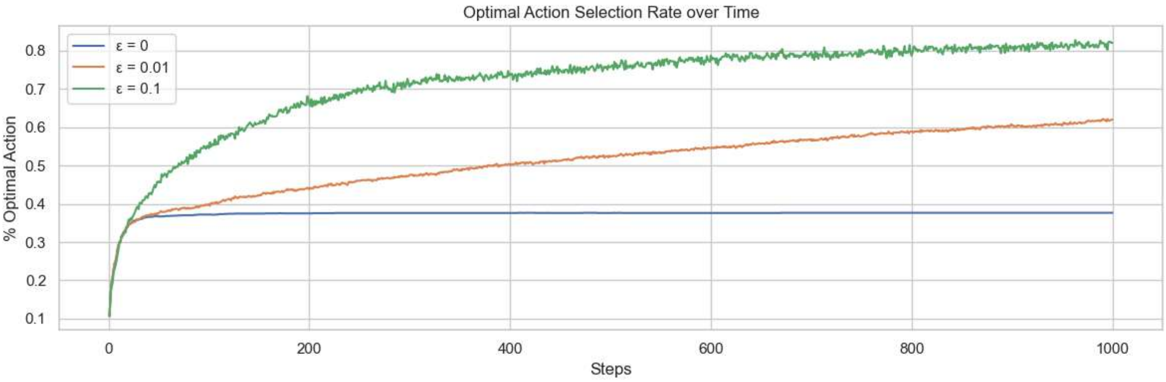
In [28]:
```python
# 主实验
def main_expriment(epsilons = [0,0.01,0.1], runs = 2000, steps = 1000, k = 10, s
    """
    对每一个epsilon，运行runs次实验，每次steps步。
    return: 平均奖励矩阵，最优动作选择矩阵
    """
    average_rewards = np.zeros((len(epsilons),steps))
    average_best_actions = np.zeros((len(epsilons),steps))

    for i,epsilon in enumerate(epsilons):
        print(f"running expriments for E = {epsilon}...")
        total_rewards = np.zeros(steps)
        total_best_actions = np.zeros(steps)

        for run in range(runs):
            """
            为每次运行创建新的赌博机
            """
```

```python
            bandit = KArmedBandit(k = k, seed=seed +run)
            agent = EpsilonGreedyAgent(k = k, epsilon=epsilon, seed = seed+run)
            # print(bandit.get_q_star())
            rewards, is_best = run_expriment(bandit,agent,steps)
            total_rewards += rewards
            total_best_actions += is_best

        average_best_actions[i] = total_best_actions/runs
        average_rewards[i] = total_rewards/runs

    return average_rewards, average_best_actions
```

In [29]:
```python
def plot_results(avg_rewards, avg_best_action_rates, epsilons, steps=1000):
    fig, axes = plt.subplots(2, 1, figsize=(12, 8))

    # 平均奖励
    for i, epsilon in enumerate(epsilons):
        axes[0].plot(range(1, steps + 1), avg_rewards[i], label=f'ε = {epsilon}'
    axes[0].set_xlabel('Steps')
    axes[0].set_ylabel('Average Reward')
    axes[0].legend()
    axes[0].set_title('Average Reward over Time')

    # 最优动作选择率
    for i, epsilon in enumerate(epsilons):
        axes[1].plot(range(1, steps + 1), avg_best_action_rates[i], label=f'ε =
    axes[1].set_xlabel('Steps')
    axes[1].set_ylabel('% Optimal Action')
    axes[1].legend()
    axes[1].set_title('Optimal Action Selection Rate over Time')

    plt.tight_layout()
    plt.show()
```

In [30]:
```python
if __name__ == "__main__":
    # 设置参数
    EPSILONS = [0, 0.01, 0.1]
    RUNS = 2000
    STEPS = 1000

    # 运行实验
    avg_rewards, avg_best_action_rates = main_expriment(
        epsilons=EPSILONS,
        runs=RUNS,
        steps=STEPS,
        k=10,
        seed=42  # 为了复现性，你可以改变或移除
    )
    print(avg_rewards.shape,avg_best_action_rates.shape)
    print(avg_best_action_rates,avg_rewards)
    # 绘图
    plot_results(avg_rewards, avg_best_action_rates, EPSILONS, STEPS)

    # 打印长期（最后100步）平均表现
    print("\n=== 长期表现（最后100步平均） ===")
    for i, eps in enumerate(EPSILONS):
        last_100_reward = np.mean(avg_rewards[i, -100:])
        last_100_optimal = np.mean(avg_best_action_rates[i, -100:])
        print(f"ε = {eps:.2f}: 平均奖励 = {last_100_reward:.3f}, 最优动作率 = {l
```

```
running expriments for E = 0...
running expriments for E = 0.01...
running expriments for E = 0.1...
(3, 1000) (3, 1000)
[[0.1065 0.1635 0.196  ... 0.3765 0.3765 0.3765]
 [0.1065 0.1625 0.1935 ... 0.6185 0.6195 0.6205]
 [0.1065 0.159  0.1845 ... 0.8235 0.8205 0.8205]] [[0.07387915 0.31048855 0.49677
893 ... 1.03448335 1.01456309 0.99924449]
 [0.07387915 0.30822287 0.49773834 ... 1.34087822 1.35666832 1.33263078]
 [0.07387915 0.28742308 0.46430849 ... 1.37327314 1.37787789 1.36039334]]
```




```
=== 长期表现（最后100步平均） ===
ε = 0.00: 平均奖励 = 1.022, 最优动作率 = 0.377
ε = 0.01: 平均奖励 = 1.318, 最优动作率 = 0.610
ε = 0.10: 平均奖励 = 1.362, 最优动作率 = 0.813
```

In [ ]: