```
            state, reward, done, info = env.step(action)
            episode_reward += reward
            if done:
                break
        print(
            'Testing | Episode: {}/{} | Episode Reward: {:.4f} | Running Time:
                {:.4f}'.format(
                episode + 1, TEST_EPISODES, episode_reward,
                time.time() - t0))
```

5.10.6　PPO: Pendulum-V0

PPO 是一种一阶方法，与 TRPO 这样的二阶算法不同。

在 PPO-Penalty 中，是通过给目标函数增加一个 KL 散度惩罚项的，以解决像 TRPO 这样带 KL 约束的更新问题。PPO 类的结构如下所示：

```
class PPO(object):
    def __init__(self, state_dim, action_dim, action_bound, method='clip'): # 初始化
        ...
    def train_actor(self, state, action, adv, old_pi): # 行动者训练函数
        ...
    def train_critic(self, reward, state): # 批判者训练函数
        ...
    def update(self): # 主更新函数
        ...
    def get_action(self, s, greedy=False): # 选择动作
        ...
    def save(self): # 存储网络
        ...
    def load(self): # 载入网络
        ...
    def store_transition(self, state, action, reward): # 存储每步的状态、动作、奖励
        ...
    def finish_path(self, next_state): # 计算累积奖励
        ...
```

在 PPO 算法中，我们在初始化函数中建立行动者网络和批判者网络。PPO 有两种方法：PPO-Penalty 和 PPO-Clip。我们在选用不同的方法时，要设置其相对应的参数。由于环境是一个连续运