linear regression-Gradient descent

In [1]:
```
import numpy as np
import pandas as pd
```

波士顿房价数据集字段说明

CRIM 房屋所在镇的犯罪率
ZN 面积大于25000平方英尺住宅所占的比例
INDUS 房屋所在镇非零售区域所占比例
CHAS 房屋是否位于河边，如果位于河边，则值为1，否则值为0
NOX 一氧化氮的浓度
RM 平均房间数量
AGE 1940年前建成房屋所占的比例
DIS 房屋距离波士顿五大就业中心的加权距离
RAD 距离房屋最近的公路
TAX 财产税额度
PIRATIO 房屋所在镇师生比例
B 计算公式：1000（房屋所在镇非美籍人口所占比例-0.63）*2
LSTAT 弱势群体人口所占比例
MEDV 房屋的平均价格 （需预测值）

In [2]:
```
data = pd.read_csv(r"boston.csv")
# data.head()
# # 查看数据基本信息，是否存在缺失值（如有缺失值，需删除）
data.info()
# # 查看是否有重复值，返回False则无重复值
# data.duplicated().any()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 452 entries, 0 to 451
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     452 non-null    float64
 1   ZN       452 non-null    float64
 2   INDUS    452 non-null    float64
 3   CHAS     452 non-null    int64  
 4   NOX      452 non-null    float64
 5   RM       452 non-null    float64
 6   AGE      452 non-null    float64
 7   DIS      452 non-null    float64
 8   RAD      452 non-null    int64  
 9   TAX      452 non-null    int64  
 10  PIRATIO  452 non-null    float64
 11  B        452 non-null    float64
 12  LSTAT    452 non-null    float64
 13  MEDV     452 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 49.6 KB
```

In [16]:
```
class LinearRegression:
    """使用python实现的线性回归（梯度下降）"""
```

```
    def __init__(self,alpha,times):
        """初始化方法
        Parameters
        ------
        alpha:float
             学习率，用来控制步长（权重调整的幅度）
        times:int
             循环迭代的次数
        """
        self.alpha = alpha
        self.times = times

    def fit(self,X,y):
        """根据提供的训练数据，对模型进行训练
        Parameters
        ------
        X: 类数组类型。形状：[样本数量，特征数量]
           待训练的样本特征属性（特征矩阵）
        y:类数组类型，形状：[样本数量]
          目标值（标签信息）
        ------
        """
        X = np.asarray(X)
        y = np.asarray(y)
        # 创建权重的向量，初始值为0（或任何其他的值），长度比特征数量多1，多出的值
        self.w_ = np.zeros(1 + X.shape[1])
        # 创建损失 列表，用来保存每次迭代后的损失值，损失值计算：（预测值-真实值）
        self.loss_ = []

        # 进行循环，多次迭代，不断调整权重值，是的损失不断减少
        for i in range(self.times):
            y_hat = np.dot(X,self.w_[1:]) + self.w_[0]# y_hat预测值
            # 计算真实值和预测值之间的差距
            error = y - y_hat
            # 将损失值加入到损失列表中
            self.loss_.append(np.sum(error ** 2)/2)
            # 根据差距调整权证w_,根据公式，调整为 权重（j）= 权重（j）+学习率（al
            self.w_[0] += self.alpha * np.sum(error)
            self.w_[1:] += self.alpha * np.dot(X.T, error)
    def predict(self,X):
         """根据参数传递的样本，对样本进行预测
         Parameters
         ------
         X:类数组类型，形状[样本数量，特征数量]
           待测试的样本
         Return
         ------
         result:数组类型
                预测的结果
         """
         X = np.asarray(X)
         result = np.dot(X,self.w_[1:]) + self.w_[0]
         return result
```

In [17]:

```
lr = LinearRegression(alpha=0.001,times=20)
t = data.sample(len(data),random_state=0)
train_X = t.iloc[:400,:-1]
train_y = t.iloc[:400,-1]
test_X = t.iloc[400:,:-1]
test_y = t.iloc[400:,-1]
```

```
lr.fit(train_X, train_y)
result = lr.predict(test_X)
display(np.mean((result - test_y) ** 2)) # 真实值与预测值之差
```

```
4.486053614479303e+205
```

In [27]:
```
# 对每个特征列进行标准化
class StandardScaler:
    """该类对数据进行标准化处理"""
    def fit(self,X):
        """根据传递的样本，计算每个特征的均值与标准差
        Parameters
        ------
        X:类数组类型
          训练数据，用来计算均值与标准差
        """
        X = np.asarray(X)
        self.std_ = np.std(X,axis=0) # 按列计算标准差
        self.mean_ = np.mean(X,axis=0) # 按列计算均值
    def transform(self,X):
        """对给定的数据X，进行标准化处理（将X的每一列都变成标准正态分布的数据）
        Parameters
        ------
        X:类数组类型
          待转换的数据
        Returns
        ------
        result:类数组类型
               参数X转换成标准正态分布后的结果
        """

        return (X-self.mean_) / self.std_
    def fit_transform(self,X):
        """对数据进行训练，并转换，返回转换之后的结果
         Parameters
        ------
        X:类数组类型
          待转换的数据
        Returns
        ------
        result:类数组类型
               参数X转换成标准正态分布后的结果
        """
        self.fit(X)
        return self.transform(X)
```

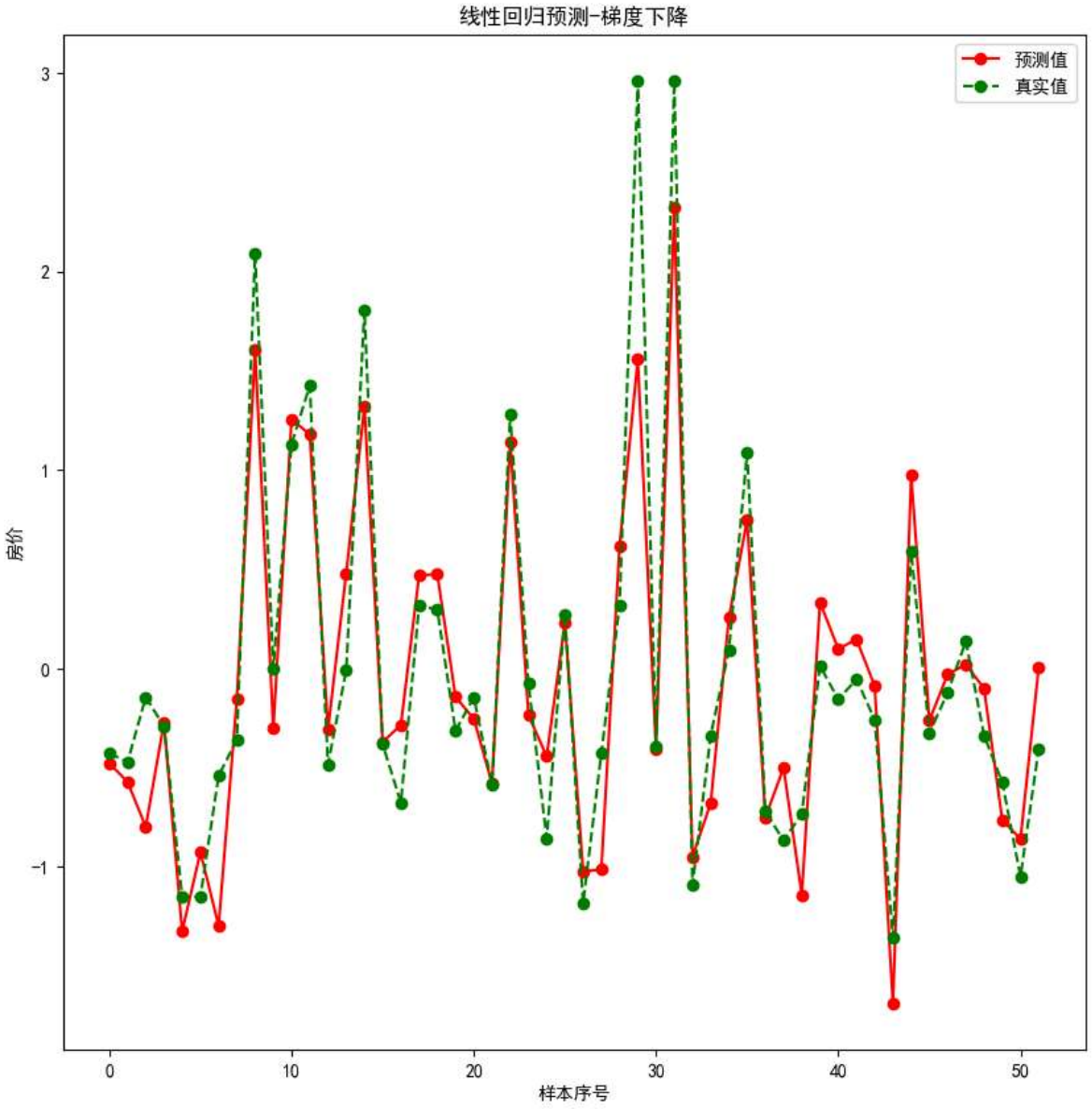
In [31]:
```
# 为了避免每个特征数量级的不同对梯度下降过程的影响
# 我们对每个特征进行标准化处理
lr = LinearRegression(alpha=0.0005,times=20)
t = data.sample(len(data),random_state=0)
train_X = t.iloc[:400,:-1]
train_y = t.iloc[:400,-1]
test_X = t.iloc[400:,:-1]
test_y = t.iloc[400:,-1]

# 对数据进行标准化处理
s = StandardScaler()
train_X = s.fit_transform(train_X)
test_X = s.transform(test_X)
```

```
s2 = StandardScaler()
train_y = s2.fit_transform(train_y)
test_y = s2.transform(test_y)

lr.fit(train_X,train_y)
result = lr.predict(test_X)
display(np.mean((result - test_y) ** 2))
display(lr.w_)
display(lr.loss_)
```

```
0.12963239612865735
array([ 9.57012247e-16, -1.64146479e-03,  5.75033736e-02, -3.30653336e-02,
        8.34510665e-02, -6.92271290e-02,  4.07120084e-01, -3.47680849e-02,
       -2.23087300e-01,  9.30249675e-02, -6.91307515e-02, -1.81919529e-01,
        1.04314459e-01, -3.56539623e-01])
[200.00000000000006,
 109.11551296640118,
 85.03691083589374,
 72.8923012237413,
 66.27988117117292,
 62.57736852280709,
 60.438045635573346,
 59.15297733527496,
 58.344340494344735,
 57.80800322683591,
 57.43188249039254,
 57.15326526483341,
 56.936292455393186,
 56.75995060767148,
 56.61158586998235,
 56.48334903712724,
 56.37021454814138,
 56.26885612463899,
 56.176995822572145,
 56.093019190568945]
```

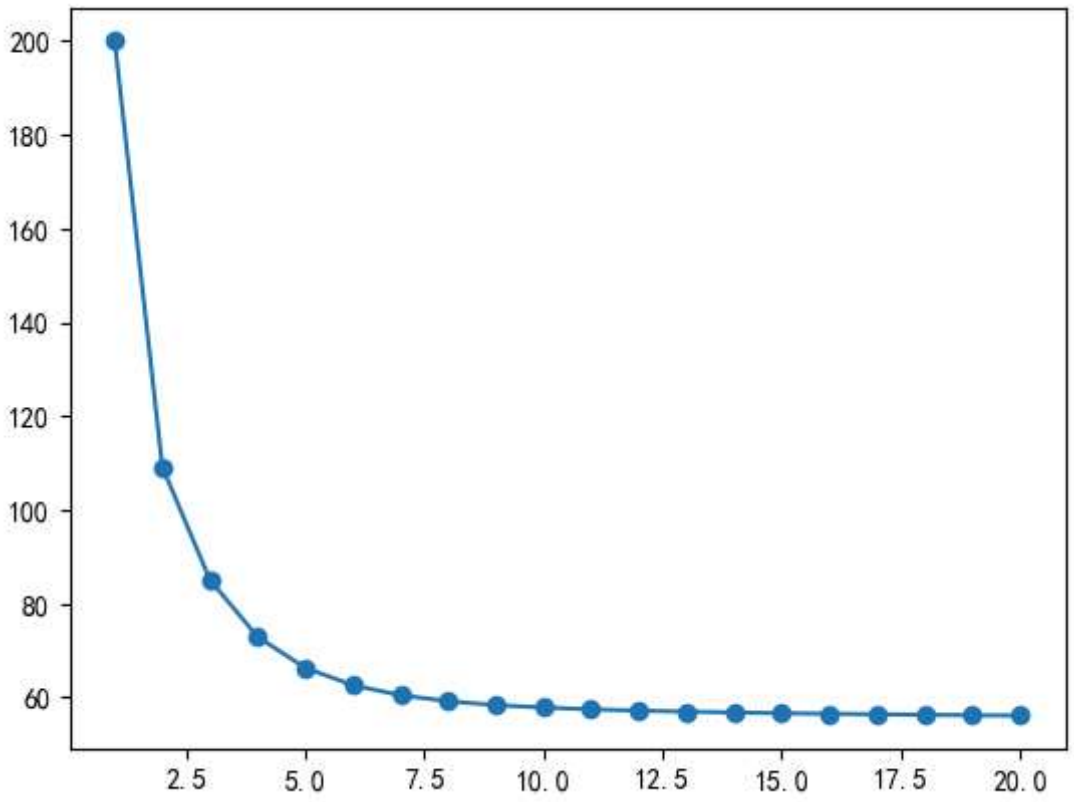
In [42]:
```
# 直线拟合可视化
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rcParams["font.family"] = "SimHei"
mpl.rcParams["axes.unicode_minus"] = False
```

In [35]:
```
plt.figure(figsize=(10,10))
# 绘制预测值
plt.plot(result,"ro-",label="预测值")
# 绘制真实值
plt.plot(test_y.values,"go--",label="真实值")
plt.title("线性回归预测-梯度下降")
plt.xlabel("样本序号")
plt.ylabel("房价")
plt.legend()
plt.show()
```



In [37]:
```
# 绘制累积误差值
plt.plot(range(1,lr.times + 1), lr.loss_, "o-")
```

Out[37]:
```
[<matplotlib.lines.Line2D at 0x1ee8feae9c0>]
```



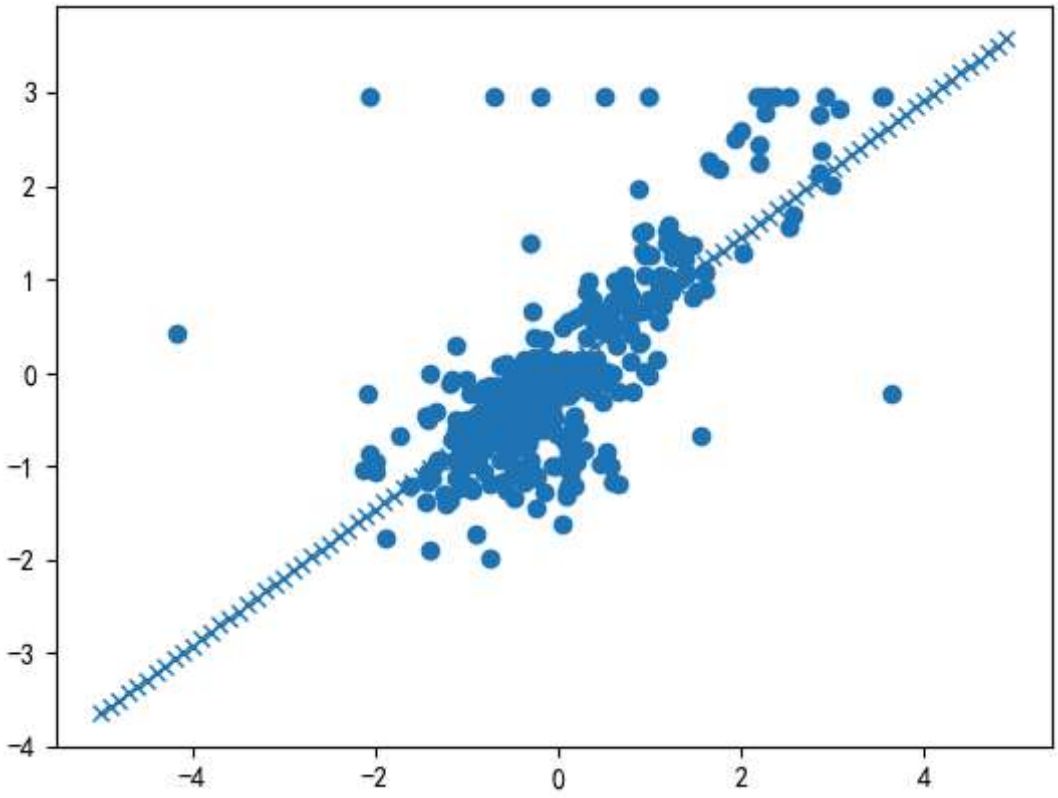
In [39]:
```
# 因为房价分析涉及多个维度，不方便进行可视化显示
# 我们只选取其中的一个维度（RM）作直线拟合可视化
lr = LinearRegression(alpha=0.0005,times=50)
t = data.sample(len(data),random_state=0)
train_X = t.iloc[:400,5:6] # 切片取出第5列的RM，返回二维的dataframe类型，实际只有
train_y = t.iloc[:400,-1]
test_X = t.iloc[400:,5:6]
test_y = t.iloc[400:,-1]

#对数据进行标准化处理
s = StandardScaler()
train_X = s.fit_transform(train_X)
test_X = s.transform(test_X)
s2 = StandardScaler()
train_y = s2.fit_transform(train_y)
test_y = s2.transform(test_y)

lr.fit(train_X, train_y)
result = lr.predict(test_X)
display(np.mean((result-test_y) ** 2))
```

```
0.2799390147433054
```

In [46]:
```
plt.scatter(train_X["RM"],train_y)
# 查看方程系数(根据方程系数绘制直线)
display(lr.w_) # 方程系数即模型的权重，即w
# 构建方程 y = 1.55542246e-15 + 7.29309581e-01 * X
x = np.arange(-5,5,0.1) # 区间
y = 1.55542246e-15 + 7.29309581e-01 * x
# plt.plot(x,y,"x")
# 或者这样画图
plt.plot(x,lr.predict(x.reshape(-1,1)),"x")
```

```
array([1.55542246e-15, 7.29309581e-01])
```

Out[46]:
```
[<matplotlib.lines.Line2D at 0x1ee8fde5340>]
```



In [ ]: