Materials Project API - 高级功能与最佳实践

16. 高级搜索技巧

16.1 复合条件搜索

多字段组合筛选

```
def advanced_material_search():
    """
    复合条件搜索示例:
    - 带隙在1.5-2.5 eV之间
    - 稳定相
    - 含有特定元素
    - 晶体结构为立方
    """
    url = f"{BASE_URL}/materials/summary/"
    params = {
        # 带隙范围
        "band_gap_min": 1.5,
        "band_gap_max": 2.5,

        # 元素组成
        "elements": "Ga,N",
        "exclude_elements": "O,F",  # 排除特定元素

        # 稳定性
        "is_stable": True,
        "energy_above_hull_max": 0.05,  # 最大允许能量差

        # 晶体结构
        "crystal_system": "Cubic",
        "spacegroup_number": 216,  # F-43m

        # 返回字段
        "_fields": "material_id,formula_pretty,band_gap," + \
                   "energy_above_hull,density,nsites",

        # 排序和分页
        "_sort_fields": "band_gap",
        "_limit": 50
    }

    response = requests.get(url, headers=headers, params=params)
    results = response.json()["data"]

    print(f"找到 {len(results)} 个符合条件的材料\n")

    for i, mat in enumerate(results[:10], 1):
        print(f"{i}. {mat['formula_pretty']:<15} (ID: {mat['material_id']})")
```

```
            print(f"   带隙: {mat['band_gap']:.3f} eV, "
                  f"密度: {mat.get('density', 'N/A'):.2f} g/cm³")
            print(f"   ΔE_hull: {mat.get('energy_above_hull', 0):.3f} eV/atom\n")

    return results
```

16.2 化学式搜索

精确化学式匹配

```
# 1. 精确化学式（包括化学计量比）
params = {"formula": "TiO2"}

# 2. 化学式匿名匹配（忽略计量比）
params = {"formula": "AB2", "anonymous": True}  # 匹配所有1:2配比材料

# 3. 通配符搜索
params = {"formula": "Ti*O*"}  # 匹配所有Ti-O系统

# 4. 切比雪夫距离匹配
params = {
    "formula": "Ti2O3",
    "chemsys": "Ti-O"  # 限定化学系统
}
```

化学系统搜索

```
# 单元素系统
params = {"chemsys": "Si"}

# 二元系统
params = {"chemsys": "Fe-O"}

# 三元及以上
params = {"chemsys": "Li-Fe-P-O"}
```

16.3 结构相似性搜索

```
def find_similar_structures(reference_material_id, tolerance=0.3):
    """
    基于晶体结构相似性搜索材料

    参数:
        reference_material_id: 参考材料ID
        tolerance: 结构匹配容差
    """
```

```
# 1. 获取参考材料的空间群和晶格参数
url_ref = f"{BASE_URL}/materials/summary/"
ref_params = {
    "material_ids": reference_material_id,
    "_fields": "spacegroup_symbol,spacegroup_number,lattice"
}

ref_response = requests.get(url_ref, headers=headers, params=ref_params)
ref_data = ref_response.json()["data"][0]

sg_number = ref_data["spacegroup_number"]
lattice = ref_data["lattice"]

# 2. 搜索相同空间群的材料
search_params = {
    "spacegroup_number": sg_number,
    "_fields": "material_id,formula_pretty,lattice",
    "_limit": 100
}

similar_response = requests.get(url_ref, headers=headers,
                                params=search_params)
candidates = similar_response.json()["data"]

# 3. 筛选晶格参数相近的材料
results = []
for mat in candidates:
    if mat["material_id"] == reference_material_id:
        continue

    mat_lattice = mat.get("lattice", {})

    # 简单的晶格参数比较
    a_diff = abs(lattice["a"] - mat_lattice.get("a", 0))
    b_diff = abs(lattice["b"] - mat_lattice.get("b", 0))
    c_diff = abs(lattice["c"] - mat_lattice.get("c", 0))

    avg_diff = (a_diff + b_diff + c_diff) / 3

    if avg_diff < tolerance:
        results.append({
            "material_id": mat["material_id"],
            "formula": mat["formula_pretty"],
            "lattice_diff": avg_diff
        })

# 排序
results.sort(key=lambda x: x["lattice_diff"])

print(f"与 {reference_material_id} 结构相似的材料:")
print("-" * 70)

for i, mat in enumerate(results[:10], 1):
    print(f"{i}. {mat['formula']:<15} (ID: {mat['material_id']})")
```

```
            print(f"   晶格参数差异: {mat['lattice_diff']:.3f} Å\n")

    return results
```

---

17. 批量查询和数据导出

17.1 批量材料查询

```
def batch_query_materials(material_ids, batch_size=10):
    """
    批量查询材料信息（避免多次API调用）

    参数:
        material_ids: 材料ID列表
        batch_size: 每批查询的数量
    """
    all_data = []

    for i in range(0, len(material_ids), batch_size):
        batch = material_ids[i:i+batch_size]

        # 使用逗号分隔的ID列表
        params = {
            "material_ids": ",".join(batch),
            "_fields": "material_id,formula_pretty,band_gap," + \
                       "energy_above_hull,density,is_stable",
            "_limit": batch_size
        }

        url = f"{BASE_URL}/materials/summary/"
        response = requests.get(url, headers=headers, params=params)

        if response.status_code == 200:
            batch_data = response.json()["data"]
            all_data.extend(batch_data)
            print(f"已查询 {len(all_data)}/{len(material_ids)} 个材料")
        else:
            print(f"批次 {i//batch_size + 1} 查询失败")

        time.sleep(0.5)  # 避免超过速率限制

    return all_data
```

17.2 分页处理大数据集

```
def paginated_search(search_params, max_results=1000):
    """
    处理超过默认限制的大量搜索结果
```

```
    参数:
        search_params: 搜索参数字典
        max_results: 最大结果数量
    """
    url = f"{BASE_URL}/materials/summary/"
    all_results = []

    page_size = 100  # 每页结果数
    offset = 0

    while len(all_results) < max_results:
        # 添加分页参数
        params = search_params.copy()
        params["_limit"] = page_size
        params["_skip"] = offset

        response = requests.get(url, headers=headers, params=params)

        if response.status_code != 200:
            print(f"请求失败: {response.status_code}")
            break

        page_data = response.json()["data"]

        if not page_data:  # 没有更多数据
            break

        all_results.extend(page_data)
        offset += page_size

        print(f"已获取 {len(all_results)} 条结果...")
        time.sleep(0.5)

        if len(page_data) < page_size:  # 最后一页
            break

    return all_results[:max_results]

# 使用示例
search_params = {
    "elements": "O",
    "nelements": 2,
    "is_stable": True,
    "_fields": "material_id,formula_pretty,band_gap"
}

results = paginated_search(search_params, max_results=500)
print(f"\n总共获取了 {len(results)} 个二元氧化物材料")
```

17.3 数据导出为多种格式

```
import json
import pandas as pd

def export_materials_data(data, output_prefix="materials_export"):
    """
    将材料数据导出为多种格式

    支持格式:
    - Excel (.xlsx)
    - CSV (.csv)
    - JSON (.json)
    - Markdown表格 (.md)
    """
    # 1. Excel格式
    df = pd.DataFrame(data)
    excel_file = f"{output_prefix}.xlsx"
    df.to_excel(excel_file, index=False)
    print(f"✓ Excel文件已保存: {excel_file}")

    # 2. CSV格式
    csv_file = f"{output_prefix}.csv"
    df.to_csv(csv_file, index=False)
    print(f"✓ CSV文件已保存: {csv_file}")

    # 3. JSON格式
    json_file = f"{output_prefix}.json"
    with open(json_file, 'w', encoding='utf-8') as f:
        json.dump(data, f, indent=2, ensure_ascii=False)
    print(f"✓ JSON文件已保存: {json_file}")

    # 4. Markdown表格
    md_file = f"{output_prefix}.md"
    with open(md_file, 'w', encoding='utf-8') as f:
        f.write("# 材料数据导出\n\n")
        f.write(df.to_markdown(index=False))
    print(f"✓ Markdown文件已保存: {md_file}")

    return {
        "excel": excel_file,
        "csv": csv_file,
        "json": json_file,
        "markdown": md_file
    }
```

---

18. 相图分析 (Phase Diagrams)

18.1 相图数据获取

```
url = f"{BASE_URL}/materials/phase_diagram/"

# 二元相图
params = {
    "chemsys": "Fe-O",
    "_fields": "material_id,formula_pretty,energy_per_atom," + \
               "decomposition_energy,is_stable"
}

response = requests.get(url, headers=headers, params=params)
phase_data = response.json()["data"]
```

18.2 凸包分析

```
def analyze_convex_hull(chemsys):
    """
    分析化学系统的凸包
    找出稳定相和亚稳态相
    """
    url = f"{BASE_URL}/materials/thermo/"
    params = {
        "chemsys": chemsys,
        "_fields": "material_id,formula_pretty," + \
                   "energy_above_hull,is_stable,formation_energy_per_atom",
        "_limit": 200
    }

    response = requests.get(url, headers=headers, params=params)
    materials = response.json()["data"]

    # 分类
    stable = [m for m in materials if m.get("is_stable")]
    metastable = [m for m in materials
                  if not m.get("is_stable")
                  and m.get("energy_above_hull", float('inf')) < 0.1]

    print(f"\n化学系统: {chemsys}")
    print("=" * 70)

    print(f"\n稳定相 ({len(stable)} 个):")
    print("-" * 70)
    for mat in stable[:10]:
        print(f"{mat['formula_pretty']:<15} "
              f"形成能: {mat.get('formation_energy_per_atom', 'N/A'):.3f}
eV/atom")

    print(f"\n亚稳态相 ({len(metastable)} 个, ΔE < 0.1 eV/atom):")
    print("-" * 70)
    for mat in metastable[:10]:
        print(f"{mat['formula_pretty']:<15} "
```

```
                f"ΔE_hull: {mat.get('energy_above_hull', 'N/A'):.3f} eV/atom")

    return {"stable": stable, "metastable": metastable}
```

---

19. 材料合成信息

19.1 合成路径查询

```
url = f"{BASE_URL}/materials/synthesis/"
params = {
    "formula": "LiFePO4",
    "_fields": "material_id,synthesis_description,doi,recipe"
}

response = requests.get(url, headers=headers, params=params)
if response.status_code == 200:
    synthesis_data = response.json()["data"]

    if synthesis_data:
        for recipe in synthesis_data:
            print("合成方法:")
            print(recipe.get("synthesis_description", "无描述"))
            print(f"参考文献 DOI: {recipe.get('doi', 'N/A')}")
```

19.2 前驱体分析

```
def analyze_precursors(target_formula):
    """
    分析目标材料的潜在前驱体
    """
    # 获取目标材料的元素组成
    url_summary = f"{BASE_URL}/materials/summary/"
    params = {
        "formula": target_formula,
        "_fields": "material_id,elements"
    }

    response = requests.get(url_summary, headers=headers, params=params)
    target_data = response.json()["data"][0]
    elements = target_data["elements"]

    print(f"目标材料: {target_formula}")
    print(f"元素组成: {', '.join(elements)}")
    print("\n潜在前驱体:")
    print("-" * 70)

    # 搜索单元素和二元化合物作为前驱体
    precursors = []
```

```
    for elem in elements:
        # 单质
        elem_params = {
            "elements": elem,
            "nelements": 1,
            "_fields": "material_id,formula_pretty,formation_energy_per_atom"
        }
        elem_response = requests.get(url_summary, headers=headers,
                                     params=elem_params)
        elem_data = elem_response.json()["data"]

        if elem_data:
            precursors.append({
                "formula": elem_data[0]["formula_pretty"],
                "type": "单质",
                "formation_energy": elem_data[0].get("formation_energy_per_atom",
0)
            })

    # 常见氧化物/盐作为前驱体
    for elem in [e for e in elements if e != 'O']:
        oxide_params = {
            "elements": f"{elem},O",
            "nelements": 2,
            "is_stable": True,
            "_fields": "material_id,formula_pretty,formation_energy_per_atom",
            "_limit": 3
        }
        oxide_response = requests.get(url_summary, headers=headers,
                                      params=oxide_params)
        oxide_data = oxide_response.json()["data"]

        for compound in oxide_data:
            precursors.append({
                "formula": compound["formula_pretty"],
                "type": "氧化物",
                "formation_energy": compound.get("formation_energy_per_atom", 0)
            })

        time.sleep(0.3)

    # 输出
    for i, prec in enumerate(precursors, 1):
        print(f"{i}. {prec['formula']:<15} ({prec['type']})")
        print(f"   形成能: {prec['formation_energy']:.3f} eV/atom\n")

    return precursors
```

---

20. 实用工具函数集合

20.1 材料性质转换器

```python
class MaterialPropertyConverter:
    """材料性质单位转换工具"""

    @staticmethod
    def eV_to_J(energy_eV):
        """电子伏特转焦耳"""
        return energy_eV * 1.602176634e-19

    @staticmethod
    def J_to_eV(energy_J):
        """焦耳转电子伏特"""
        return energy_J / 1.602176634e-19

    @staticmethod
    def eV_per_A_to_V_per_m(field_eV_A):
        """eV/Å转V/m (电场单位) """
        return field_eV_A * 1e10

    @staticmethod
    def band_gap_to_wavelength(band_gap_eV):
        """
        带隙转对应光波波长 (纳米)
        E(eV) = 1240 / λ(nm)
        """
        if band_gap_eV <= 0:
            return float('inf')
        return 1240 / band_gap_eV

    @staticmethod
    def wavelength_to_band_gap(wavelength_nm):
        """波长转带隙能量"""
        return 1240 / wavelength_nm

    @staticmethod
    def gpascal_to_bar(pressure_GPa):
        """GPa转bar"""
        return pressure_GPa * 1e4

    @staticmethod
    def density_to_atomic_density(density_g_cm3, molar_mass_g_mol):
        """
        质量密度转原子数密度
        density: g/cm³
        molar_mass: g/mol
        返回: atoms/cm³
        """
        avogadro = 6.02214076e23
        return (density_g_cm3 / molar_mass_g_mol) * avogadro

# 使用示例
```

```
converter = MaterialPropertyConverter()

band_gap = 1.5  # eV
wavelength = converter.band_gap_to_wavelength(band_gap)
print(f"带隙 {band_gap} eV 对应波长 {wavelength:.1f} nm")
```

20.2 快速材料评估

```
def quick_material_assessment(material_id):
    """
    快速评估材料的综合性能
    返回包含多个性质的简要报告
    """
    # 获取Summary信息
    url_summary = f"{BASE_URL}/materials/summary/"
    summary_params = {
        "material_ids": material_id,
        "_fields": "material_id,formula_pretty,elements,band_gap," + \
                   "is_stable,energy_above_hull,density,nsites," + \
                   "is_metal,efermi,formation_energy_per_atom"
    }

    summary_response = requests.get(url_summary, headers=headers,
                                    params=summary_params)

    if summary_response.status_code != 200:
        return {"error": "Material not found"}

    summary = summary_response.json()["data"][0]

    # 获取对称性信息
    symmetry_params = {
        "material_ids": material_id,
        "_fields": "spacegroup_symbol,spacegroup_number,crystal_system," + \
                   "point_group"
    }

    symmetry_response = requests.get(url_summary, headers=headers,
                                     params=symmetry_params)
    symmetry = symmetry_response.json()["data"][0] if
symmetry_response.status_code == 200 else {}

    # 生成报告
    report = f"""
╔═══════════════════════════════════════════════════════════╗
║           材料快速评估报告                                ║
╠═══════════════════════════════════════════════════════════╣
║ Material ID: {summary['material_id']:<45} ║
║ 化学式: {summary['formula_pretty']:<50} ║
╠═══════════════════════════════════════════════════════════╣
║ 基础性质                                                  ║
```

```
    ╠══════════════════════════════════════════════════════════════╣
    ║ 元素组成: {', '.join(summary.get('elements', [])):<48} ║
    ║ 原子数: {summary.get('nsites', 'N/A'):<52} ║
    ║ 密度: {summary.get('density', 'N/A'):.2f} g/cm³{' '*41} ║
    ╠══════════════════════════════════════════════════════════════╣
    ║ 电子性质                                                     ║
    ╠══════════════════════════════════════════════════════════════╣
    ║ 带隙: {summary.get('band_gap', 'N/A')} eV{' '*48} ║
    ║ 费米能级: {summary.get('efermi', 'N/A')} eV{' '*44} ║
    ║ 金属性: {'是' if summary.get('is_metal') else '否':<51} ║
    ╠══════════════════════════════════════════════════════════════╣
    ║ 稳定性                                                       ║
    ╠══════════════════════════════════════════════════════════════╣
    ║ 热力学稳定: {'是' if summary.get('is_stable') else '否':<43} ║
    ║ 能量高于凸包: {summary.get('energy_above_hull', 'N/A'):.3f} eV/atom{' '*32} ║
    ║ 形成能: {summary.get('formation_energy_per_atom', 'N/A'):.3f} eV/atom{' '*37} ║
    ╠══════════════════════════════════════════════════════════════╣
    ║ 晶体结构                                                     ║
    ╠══════════════════════════════════════════════════════════════╣
    ║ 空间群: {symmetry.get('spacegroup_symbol', 'N/A')} (No.
{symmetry.get('spacegroup_number', 'N/A')}){' '*31} ║
    ║ 晶系: {symmetry.get('crystal_system', 'N/A'):<50} ║
    ║ 点群: {symmetry.get('point_group', 'N/A'):<50} ║
    ╚══════════════════════════════════════════════════════════════╝
    """

    print(report)

    # 应用评估
    print("\n应用潜力评估:")
    print("-" * 60)

    if 0.5 <= summary.get('band_gap', 0) <= 3.0:
        print("✓ 光电材料: 带隙适中，适合光伏/光催化应用")

    if summary.get('is_stable') and summary.get('energy_above_hull', 1) < 0.05:
        print("✓ 合成可行性: 热力学稳定，容易合成")

    if summary.get('is_metal'):
        print("✓ 导体: 适合电极、互连材料")

    if summary.get('density', 0) < 3.0:
        print("✓ 轻质材料: 低密度，适合航空航天")

    return summary
```

20.3 批量比较工具

```
def compare_materials(material_ids, properties):
    """
    批量比较多个材料的指定性质
```

```
    参数:
        material_ids: 材料ID列表
        properties: 要比较的性质列表
    """
    url = f"{BASE_URL}/materials/summary/"
    params = {
        "material_ids": ",".join(material_ids),
        "_fields": "material_id,formula_pretty," + ",".join(properties)
    }

    response = requests.get(url, headers=headers, params=params)
    materials = response.json()["data"]

    # 创建比较表
    df = pd.DataFrame(materials)

    print("\n材料性质比较表:")
    print("=" * 80)
    print(df.to_string(index=False))

    # 生成比较图
    if len(properties) > 0:
        fig, axes = plt.subplots(1, len(properties),
                                 figsize=(6*len(properties), 5))

        if len(properties) == 1:
            axes = [axes]

        for ax, prop in zip(axes, properties):
            formulas = df["formula_pretty"]
            values = df[prop]

            ax.bar(range(len(formulas)), values)
            ax.set_xticks(range(len(formulas)))
            ax.set_xticklabels(formulas, rotation=45, ha='right')
            ax.set_ylabel(prop)
            ax.set_title(f"{prop} 比较")
            ax.grid(axis='y', alpha=0.3)

        plt.tight_layout()
        plt.savefig("material_comparison.pdf")
        print("\n比较图已保存为 material_comparison.pdf")

    return df
```

---

21. 性能优化最佳实践

21.1 请求优化

```
# ❌ 不推荐：多次单独请求
for material_id in material_ids:
    response = requests.get(url, params={"material_ids": material_id})
    # 处理数据...

# ☑ 推荐：批量请求
response = requests.get(url, params={"material_ids": ",".join(material_ids)})
```

21.2 数据缓存

```
import pickle
from datetime import datetime, timedelta

class MaterialDataCache:
    """材料数据缓存管理器"""

    def __init__(self, cache_file="mp_cache.pkl", expire_days=7):
        self.cache_file = cache_file
        self.expire_days = expire_days
        self.cache = self._load_cache()

    def _load_cache(self):
        try:
            with open(self.cache_file, 'rb') as f:
                return pickle.load(f)
        except FileNotFoundError:
            return {}

    def _save_cache(self):
        with open(self.cache_file, 'wb') as f:
            pickle.dump(self.cache, f)

    def get(self, key):
        """从缓存获取数据"""
        if key in self.cache:
            data, timestamp = self.cache[key]
            if datetime.now() - timestamp < timedelta(days=self.expire_days):
                return data
            else:
                del self.cache[key]  # 过期数据清除
        return None

    def set(self, key, data):
        """保存数据到缓存"""
        self.cache[key] = (data, datetime.now())
        self._save_cache()

# 使用示例
cache = MaterialDataCache()
```

```python
def get_material_with_cache(material_id):
    # 尝试从缓存获取
    cached_data = cache.get(material_id)
    if cached_data:
        print(f"从缓存读取 {material_id}")
        return cached_data

    # 缓存未命中，请求API
    print(f"从API获取 {material_id}")
    url = f"{BASE_URL}/materials/summary/"
    response = requests.get(url, headers=headers,
                            params={"material_ids": material_id})
    data = response.json()["data"][0]

    # 保存到缓存
    cache.set(material_id, data)

    return data
```

21.3 并发请求（谨慎使用）

```python
from concurrent.futures import ThreadPoolExecutor, as_completed

def fetch_material_data(material_id):
    """单个材料数据获取函数"""
    url = f"{BASE_URL}/materials/summary/"
    params = {
        "material_ids": material_id,
        "_fields": "material_id,formula_pretty,band_gap"
    }

    response = requests.get(url, headers=headers, params=params)
    time.sleep(0.2)  # 尊重速率限制

    if response.status_code == 200:
        return response.json()["data"][0]
    return None

def concurrent_fetch(material_ids, max_workers=5):
    """
    并发获取多个材料数据
    注意：务必遵守API速率限制！
    """
    results = []

    with ThreadPoolExecutor(max_workers=max_workers) as executor:
        # 提交任务
        future_to_id = {
            executor.submit(fetch_material_data, mid): mid
            for mid in material_ids
        }
```

```
        # 收集结果
        for future in as_completed(future_to_id):
            material_id = future_to_id[future]
            try:
                data = future.result()
                if data:
                    results.append(data)
                    print(f"✓ 已获取 {material_id}")
            except Exception as e:
                print(f"✗ {material_id} 失败: {e}")

    return results
```

---

22. 错误处理和调试

22.1 完善的错误处理

```
def robust_api_request(url, params, max_retries=3):
    """
    带重试机制的API请求
    """
    for attempt in range(max_retries):
        try:
            response = requests.get(url, headers=headers, params=params,
                                    timeout=30)

            # 状态码检查
            if response.status_code == 200:
                return response.json()

            elif response.status_code == 429:  # 速率限制
                wait_time = 2 ** attempt  # 指数退避
                print(f"速率限制，等待 {wait_time} 秒...")
                time.sleep(wait_time)
                continue

            elif response.status_code == 404:
                print("资源未找到")
                return None

            elif response.status_code >= 500:
                print(f"服务器错误 {response.status_code}，重试中...")
                time.sleep(2)
                continue

            else:
                print(f"请求失败: {response.status_code}")
                print(response.text)
                return None
```

```
        except requests.exceptions.Timeout:
            print(f"请求超时，尝试 {attempt + 1}/{max_retries}")
            continue

        except requests.exceptions.ConnectionError:
            print(f"连接失败，尝试 {attempt + 1}/{max_retries}")
            time.sleep(3)
            continue

        except Exception as e:
            print(f"未知错误: {e}")
            return None

    print("达到最大重试次数")
    return None
```

22.2 日志记录

```
import logging

# 配置日志
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('mp_api.log', encoding='utf-8'),
        logging.StreamHandler()
    ]
)

logger = logging.getLogger(__name__)

def logged_api_request(url, params):
    """带日志记录的API请求"""
    logger.info(f"请求URL: {url}")
    logger.info(f"参数: {params}")

    try:
        response = requests.get(url, headers=headers, params=params)
        logger.info(f"响应状态码: {response.status_code}")

        if response.status_code == 200:
            data = response.json()
            logger.info(f"获取到 {len(data.get('data', []))} 条数据")
            return data
        else:
            logger.error(f"请求失败: {response.text}")
            return None

    except Exception as e:
```

```
        logger.exception(f"请求异常: {e}")
        return None
```

---

23. 完整工作流示例

23.1 光伏材料筛选完整流程

```
def complete_photovoltaic_screening():
    """
    光伏材料筛选的完整工作流
    """
    print("=" * 70)
    print("光伏材料筛选系统")
    print("=" * 70)

    # 第一步：初步筛选
    print("\n[1/5] 初步筛选：带隙范围...")
    url_summary = f"{BASE_URL}/materials/summary/"
    params = {
        "band_gap_min": 1.0,
        "band_gap_max": 2.2,
        "is_stable": True,
        "_fields": "material_id,formula_pretty,band_gap,nelements",
        "_limit": 100
    }

    response = requests.get(url_summary, headers=headers, params=params)
    candidates = response.json()["data"]
    print(f"找到 {len(candidates)} 个初步候选材料")

    # 第二步：获取电子结构
    print("\n[2/5] 获取电子结构信息...")
    url_elec = f"{BASE_URL}/materials/electronic_structure/"

    with_dos = []
    for mat in candidates[:20]:  # 限制数量
        elec_params = {
            "material_ids": mat["material_id"],
            "_fields": "material_id,band_gap,efermi,cbm,vbm"
        }

        elec_response = requests.get(url_elec, headers=headers,
                                     params=elec_params)

        if elec_response.status_code == 200:
            elec_data = elec_response.json()["data"]
            if elec_data:
                combined = {**mat, **elec_data[0]}
                with_dos.append(combined)
```

```
        time.sleep(0.3)

    print(f"获得 {len(with_dos)} 个材料的电子结构数据")

    # 第三步：获取光学性质
    print("\n[3/5] 获取光学性质...")
    url_optical = f"{BASE_URL}/materials/dielectric/"

    with_optical = []
    for mat in with_dos:
        opt_params = {
            "material_ids": mat["material_id"],
            "_fields": "material_id,n,eps_electronic"
        }

        opt_response = requests.get(url_optical, headers=headers,
                                    params=opt_params)

        if opt_response.status_code == 200:
            opt_data = opt_response.json()["data"]
            if opt_data:
                combined = {**mat, **opt_data[0]}
                with_optical.append(combined)

        time.sleep(0.3)

    print(f"获得 {len(with_optical)} 个材料的光学数据")

    # 第四步：评分和排序
    print("\n[4/5] 材料评分...")
    scored = []

    for mat in with_optical:
        # 简化评分模型
        score = 0

        # 带隙评分（1.1-1.7 eV最理想）
        band_gap = mat.get("band_gap", 0)
        if 1.1 <= band_gap <= 1.7:
            gap_score = 10
        else:
            gap_score = 10 * np.exp(-((band_gap - 1.4)**2) / 0.5)
        score += gap_score

        # 复合元素数评分（简单为好）
        nelements = mat.get("nelements", 5)
        elem_score = 5 if nelements <= 2 else 5 / nelements
        score += elem_score

        # 折射率评分（高吸收）
        n = mat.get("n", [1.0])[0] if mat.get("n") else 1.0
        n_score = min(5, n / 3 * 5)
        score += n_score
```

```
            mat["score"] = score
            scored.append(mat)

        # 排序
        scored.sort(key=lambda x: x["score"], reverse=True)

        # 第五步：输出报告
        print("\n[5/5] 生成报告...")
        print("\n" + "=" * 70)
        print("TOP 10 光伏候选材料")
        print("=" * 70)

        for i, mat in enumerate(scored[:10], 1):
            print(f"\n{i}. {mat['formula_pretty']:<15} (ID: {mat['material_id']})")
            print(f"   带隙: {mat.get('band_gap', 'N/A'):.3f} eV")
            print(f"   CBM: {mat.get('cbm', 'N/A')} eV, VBM: {mat.get('vbm', 'N/A')}
eV")
            print(f"   折射率: {mat.get('n', ['N/A'])[0]}")
            print(f"   综合评分: {mat['score']:.2f}/20")

        # 保存结果
        df = pd.DataFrame(scored)
        df.to_excel("photovoltaic_screening_results.xlsx", index=False)
        print("\n完整结果已保存到 photovoltaic_screening_results.xlsx")

        return scored
```

---

24. 总结与资源

24.1 API端点总览

端点	用途	数据量	重要性
/materials/summary/	基础信息	~140,000	★★★★★
/materials/electronic_structure/	电子结构	~86,000	★★★★★
/materials/dielectric/	光学性质	~8,000	★★★★
/materials/elasticity/	力学性质	~15,000	★★★★
/materials/magnetism/	磁性性质	~50,000	★★★★
/materials/surface_properties/	表面性质	~100,000	★★★★★
/materials/piezoelectric/	压电性质	~900	★★★
/materials/phonon/	声子性质	~1,500	★★★
/materials/thermo/	热力学	~140,000	★★★★★

24.2 常见应用场景

1. **半导体材料筛选**: Summary + Electronic Structure + Optical
2. **催化剂设计**: Summary + Surface Properties + Electronic Structure
3. **结构材料**: Summary + Elasticity + Thermo
4. **能源材料**: Summary + Electronic Structure + Thermo
5. **磁性材料**: Summary + Magnetism + Electronic Structure

24.3 学习资源

- 官方文档: https://docs.materialsproject.org/
- API交互式文档: https://api.materialsproject.org/docs
- GitHub仓库: https://github.com/materialsproject
- 论文: https://doi.org/10.1063/1.4812323

24.4 联系方式

- 邮件支持: feedback@materialsproject.org
- 论坛: https://matsci.org/materials-project
- 问题追踪: GitHub Issues

文档版本: v1.0
最后更新: 2024
作者: Materials Project API使用指南