Advanced Python Sample: Async Web Scraper Framework with Plugin System

This advanced Python code demonstrates:
- Abstract Base Classes and Plugin System
- Metaprogramming with Metaclasses
- Async/Await and aiohttp for concurrency
- Custom async context managers
- Caching with async decorators
- Logging and Error Handling
- Dynamic plugin loading
- Type Hints and Clean Architecture

```
import asyncio
import aiohttp
import logging
import time
from typing import Type, Dict, List, Any, Callable, Optional, Union
from abc import ABC, abstractmethod
from contextlib import asynccontextmanager
from functools import wraps, lru_cache
from threading import Lock

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(name)s - %(message)s"
)
logger = logging.getLogger("AsyncScraper")

class ScraperPlugin(ABC):
    name: str

    @abstractmethod
```

```
    async def parse(self, html: str) -> Any:
        pass

class PluginMeta(type):
    _registry: Dict[str, Type[ScraperPlugin]] = {}

    def __new__(cls, name, bases, attrs):
        new_cls = super().__new__(cls, name, bases, attrs)
        if not attrs.get("name") or not issubclass(new_cls, ScraperPlugin):
            return new_cls

        PluginMeta._registry[new_cls.name] = new_cls
        logger.info(f"Registered plugin: {new_cls.name}")
        return new_cls

    @classmethod
    def get_plugin(cls, name: str) -> Optional[Type[ScraperPlugin]]:
        return cls._registry.get(name)

class TitleParser(ScraperPlugin, metaclass=PluginMeta):
    name = "title"

    async def parse(self, html: str) -> str:
        start = html.find("<title>")
        end = html.find("</title>")
        return html[start + 7:end].strip() if start != -1 and end != -1 else "N/A"

def async_cache(ttl: int = 60):
    cache = {}
    lock = Lock()

    def decorator(func: Callable):
        @wraps(func)
        async def wrapper(*args, **kwargs):
            key = (args, tuple(sorted(kwargs.items())))
            now = time.time()

            with lock:
                if key in cache:
                    value, expiry = cache[key]
```

```
                if now < expiry:
                    logger.info("Cache hit.")
                    return value

            result = await func(*args, **kwargs)
            with lock:
                cache[key] = (result, now + ttl)
            return result
        return wrapper
    return decorator

@asynccontextmanager
async def aiohttp_session():
    async with aiohttp.ClientSession() as session:
        yield session

class AsyncScraper:
    def __init__(self, urls: List[str], plugin: Union[str, Type[ScraperPlugin]]):
        self.urls = urls
        self.plugin_cls = self._resolve_plugin(plugin)
        self.plugin = self.plugin_cls()

    def _resolve_plugin(self, plugin: Union[str, Type[ScraperPlugin]]) ->
Type[ScraperPlugin]:
        if isinstance(plugin, str):
            plugin_cls = PluginMeta.get_plugin(plugin)
            if not plugin_cls:
                raise ValueError(f"Plugin '{plugin}' not found.")
            return plugin_cls
        elif issubclass(plugin, ScraperPlugin):
            return plugin
        raise TypeError("Invalid plugin type")

    @async_cache(ttl=120)
    async def fetch(self, url: str, session: aiohttp.ClientSession) -> str:
        async with session.get(url) as response:
            logger.info(f"Fetched: {url}")
            return await response.text()

    async def scrape(self) -> Dict[str, Any]:
```

```
        results = {}
        async with aiohttp_session() as session:
            tasks = [self._scrape_one(url, session) for url in self.urls]
            data = await asyncio.gather(*tasks)
            for url, parsed in data:
                results[url] = parsed
        return results

    async def _scrape_one(self, url: str, session: aiohttp.ClientSession):
        try:
            html = await self.fetch(url, session)
            parsed = await self.plugin.parse(html)
            return url, parsed
        except Exception as e:
            logger.error(f"Error scraping {url}: {e}")
            return url, None

async def main():
    urls = [
        "https://www.python.org",
        "https://www.wikipedia.org",
        "https://www.github.com"
    ]
    scraper = AsyncScraper(urls, plugin="title")
    results = await scraper.scrape()

    for url, title in results.items():
        print(f"[{url}] => {title}")

if __name__ == "__main__":
    asyncio.run(main())
```