筑波大学大学院博士課程

システム情報工学研究科特定課題研究報告書

海底コア CT スキャンイメージ可視化のための クラウドサービスの開発

－3 次元レンダリング処理の負荷分散と コア試料検索のためのミドルウェア およびWeb アプリケーション－

岡本昂也

修士（工学）

(コンピュータサイエンス専攻)

指導教員　田中二郎

２０１３年　３月

概要

本プロジェクトは，筑波大学の高度 IT 人材育成のための実践的ソフトウェア開発専修プログラムにおける特定課題研究として，独立行政法人海洋研究開発機構（JAMSTEC）高知コア研究所との共同研究のもと，「海底コア CT スキャンイメージ可視化のためのクラウドサービス」の開発を行うプロジェクトである．本研究で用いるコア試料の X 線 CT スキャンイメージは，JAMSTEC が保有するものである．コア試料は，JAMSTEC 所有の地球深部探査船「ちきゅう」によって掘削されている．「ちきゅう」によって掘削されたコア試料は，地質学者によって地球科学分野における研究のために活用されている．「ちきゅう」は X 線 CT スキャナが搭載された掘削船である．「ちきゅう」によって掘削されたコア試料は，X 線 CT スキャンにかけられ，コア試料の X 線 CT スキャンイメージとして保存される．コア試料の X 線 CT スキャンイメージが持つ利点として，実物のコア試料を破壊せずに，コア試料の内部構造を閲覧できることが挙げられる．しかし，コア試料の X 線 CT イメージの利用には，いくつかの問題点が挙げられる．1 つ目の問題点は，コア試料の X 線 CT イメージは，データ量が大きく，閲覧に高い端末性能が要求されることである．2 つ目の問題点は，既存の X 線 CT スキャンイメージを閲覧するためのソフトウェアは，コア試料を観察するのに十分な機能を有していないことである．以上の問題点を解決するために，本プロジェクトでは X 線 CT スキャンイメージの画像処理を行うサーバと，コア試料の観察に適した閲覧用ソフトウェアを開発する．開発したシステムの内，著者は，安定したサービスを実現するためのサーバにおける負荷分散処理と，閲覧したいコア試料を効率的に検索することのできる検索機能，および，一般的なブラウザから本サービスを利用できるようにするための Web アプリケーションの開発を担当した．本プロジェクトで開発するクラウドサービスにより，データ量や端末性能を考慮することなく，コア試料の観察に特化したインタフェースを用いて，海底コア CT スキャンイメージを閲覧することが可能となる．本プロジェクトで開発したサービスが普及されることで，X 線 CT イメージの利用促進と，それに伴う地球科学分野における研究の発展が期待できる．

目次

付録

図目次

1. はじめに

本プロジェクトでは，筑波大学の高度 IT 人材育成のための実践的ソフトウェア開発専修プログラムにおける特定課題研究として，独立行政法人海洋研究開発機構（JAMSTEC）高知コア研究所との共同研究のもと，「海底コア CT スキャンイメージ可視化のためのクラウドサービス」の開発を行った．

JAMSTEC は統合国際深海掘削計画(IODP) の一機関として，JAMSTEC 所有の地球深部探査船「ちきゅう」で海底のコア試料を掘削している．IODP は，日本と米国が主導する地球環境変動，地球内部構造及び地殻内生物圏の解明を目的とした国際的な海洋科学掘削計画であり，地球システム変動の解析を目的として，海洋掘削を中心とした数々のプロジェクトを実行している．「ちきゅう」は，IODP の一主力船であり，人類史上初めてマントルや巨大地震発生域への大深度掘削を可能とした世界初のライザー式科学掘削船である．コア試料とは，地中や海底から採取された柱状の地層サンプルのことである．コア試料は，世界中の研究者によって活用され，巨大地震発生のしくみ，地球規模の環境変動，地球内部エネルギーに支えられた地下生命圏，新しい海底資源の解明に役立っている．

「ちきゅう」によって掘削されたコア試料は，「ちきゅう」船上に搭載されている X 線 CT スキャナによって撮影され，デジタルイメージとして保存される．コア試料のデジタルイメージを利用することで，実物のコア試料を手元に置かずにコア試料の観察を行うことが可能となる．

しかし，一部の研究者やユーザしかデジタルイメージを活用できていない現状がある．これは，ローカル端末でのコア試料のデジタルイメージの扱いが難しいことが原因である．従来，コア試料のデジタルイメージを扱うユーザは，自身のローカル端末にデジタルイメージをダウンロードし，専用ソフトウェアを用いることで，コアデータを利用していた．しかし，ローカル端末でデジタルイメージを閲覧するには，デジタルイメージを処理できるだけの高い端末性能が要求される．また，既存の専用ソフトウェアは，コア試料の観察に適した機能が十分に提供されていない．

以上の問題点を解決するために，「海底コア CT スキャンイメージ可視化のためのクラウドサービス」を本プロジェクトで開発する．このサービスでは，従来ユーザがローカル端末で行っていたコア試料のデジタルイメージを表示するための一連の作業を，リモートの専用サーバで行う．リモートの専用サーバでは，ユーザのリクエストに応じて，デジタルイメージを処理し，コア試料の画像を生成する．ユーザは，本サービスが提供するクライアントアプリケーションを介してリモートのサーバにアクセスし，コア試料のデジタルイメージを閲覧することが可能となる．本サービスでは，クライアントアプリケーションとして，コア試料の観察に特化した機能を持つ *Virtual Core Viewer* を提供する．

本サービスを提供することによって，コア試料のデジタルイメージの活用を促進し，地球科学分野における研究の更なる発展が期待できる．様々な地球科学分野における研究の手助けに加えて，博物館での展示や学校教育などの地球科学に携わる様々な方々の活動を活発なものとし，地球科学分野全体の発展の手助けとなることが期待できる．

本プロジェクトの成果として，Virtual Core Viewer を一般公開した．Virtual Core Viewer は，Web アプリケーションと Android アプリケーションで開発し，それぞれ一般公開している．Android アプリケーションは，2012 年 12 月 5 日に Google Play 上で公開している．また，Web アプリケーションを高知コア研究所所有の Web ページである Virtual Core Library

上で公開している．

本プロジェクトのプロジェクト実施体制を表 1 に示す．共同研究者，課題担当教員の指導の下，著者を含む開発メンバ 3 名で開発を行った．

表 1　プロジェクト実施体制

役割	所属	名前(担当)
共同研究者	海洋研究開発機構(JAMSTEC) 高知コア研究所	久光 敏夫 技術主任
課題担当 教員	筑波大学大学院 コンピュータサイエンス専攻	和田耕一　教授 山際伸一　准教授
開発メンバ	筑波大学大学院 コンピュータサイエンス専攻 システム情報工学研究科	坂本侑一郎(レンダリングサーバ開発) 佐々木慎(アプリケーション開発) 岡本昂也(ミドルウェア開発)

本プロジェクトは，2012 年 5 月～2013 年 1 月に遂行した．本プロジェクトの詳細なスケジュールを付録 A に示す．本プロジェクトでは，開発内容を 3 つの開発フェーズに分類し，段階的な開発を行った．1 段階の開発フェーズが終了するごとに，随時，共同研究者の方を初めとする地質学者の方々に公開し，本システムのインタフェースや機能に関するフィードバックを得た．得られたフィードバックに基づいて，本システムに改良を加えた．なお，久光技術主任の協力により，開発を進めるにあたって必要不可欠なコア試料のデジタルイメージを早期に入手することができたため，円滑に開発を進めることができた．

著者は，本プロジェクトにおいて 3 つの役割を担う．1 つ目は，大量のコアデータの中から閲覧したいコアデータを見つけ出すための検索手法を提案し，検索機能として実装することである．2 つ目は，安定したサービスの提供を実現するために，サービスにかかる負荷を軽減・分散する手法を設計し実現することである．3 つ目は，一般的な Web ブラウザからコアデータを閲覧できるように，Web アプリケーションとしてコア試料の観察に特化した機能を有するインタフェースを実装することである．

本論文は，全 7 章で構成されている．まず，第 2 章で本研究の背景について述べる．本研究の背景について述べた後，現状の問題点，それに対する解決策，提案するシステムについて議論する．第 3 章では，第 2 章の議論にて述べた提案システムにもとづいて設計したシステムの全体構成について述べる．第 4 章から第 6 章は，著者が設計・実装を担当した開発内容について述べる．第 4 章では，検索機能の実現に伴う検索 API,検索モジュール，検索用 DB の設計・実装について述べる．第 5 章では，リクエストの負荷の分散・軽減を行うレンダラモジュールの設計・実装について述べる．第 6 章では，実装した Web アプリケーションについて述べる．第 7 章では，まとめと今後の発展をそれぞれ述べる．

2. 本研究の背景

第 2 章では，本研究で用いるコア試料の X 線 CT スキャンイメージと，X 線 CT スキャンイメージの保存形式である DICOM フォーマットについて述べる．その後，コアのデジタルイメージ活用にあたっての現状の問題点とその解決策について議論する．

2.1. 掘削コア試料の X 線 CT データについて

コア試料とは，地中や海底から採取された柱状の地層サンプルのことである．コア試料の掘削は，JAMSTEC 所有の地球深部探査船「ちきゅう」をはじめとする掘削船により行われている．「ちきゅう」船上には，X 線 CT スキャナが搭載されており，掘削されたすべてのコア試料は，筒状の容器に入れられたまま X 線 CT スキャナにかけられ，デジタルデータとして保存される．X 線 CT スキャナで撮影されたコア試料のデジタルデータは,「ちきゅう」で掘削された時のそのままの状態で内部の構造が保存されている．そのため，デジタルデータを用いることで，実物のコア試料を破壊せずに，内部構造を観察することが可能となる．また, コアの X 線 CT スキャンデータを用いて, 事前にコアの内部構造を観察しておくことで, 実物のコアで分析を行う際の分析効率を高めることができる．万が一，実物のコア試料が欠損してしまった場合でも, デジタルデータを用いることでコア試料を観察することができる．デジタルデータそのものが壊れない限り，何度でもコア試料を観察することができる．コア試料の X 線 CT スキャンイメージを用いた研究は，実物のコア試料を用いた分析同様，地質学者の地球システム変動解析に関する研究を行う上で非常に有用な研究手法である．

図 1 は，掘削されたコア試料が，X 線 CT スキャンデータに変換され，保管されるまでの様子である．まず,「ちきゅう」によって掘削されたコア試料は，X 線 CT スキャナにかけられ，X 線 CT スキャンデータとして保存される．実物のコア試料は，高知コア研究所に搬送され，専用の冷蔵庫で保存される．X 線 CT スキャンデータは，JAMSTEC 横浜研究所で保存され，Virtual Core Library を介して公開される．Virtual Core Library は高知コア研究所が公開しているホームページである[1]．このホームページ上では，これまでに掘削されたコア試料の X 線 CT スキャンデータへのリンクが公開されており，自由にダウンロードすることができる．研究で使用するコア試料の X 線 CT スキャンデータは，この Virtual Core Library を介してダウンロードされている．

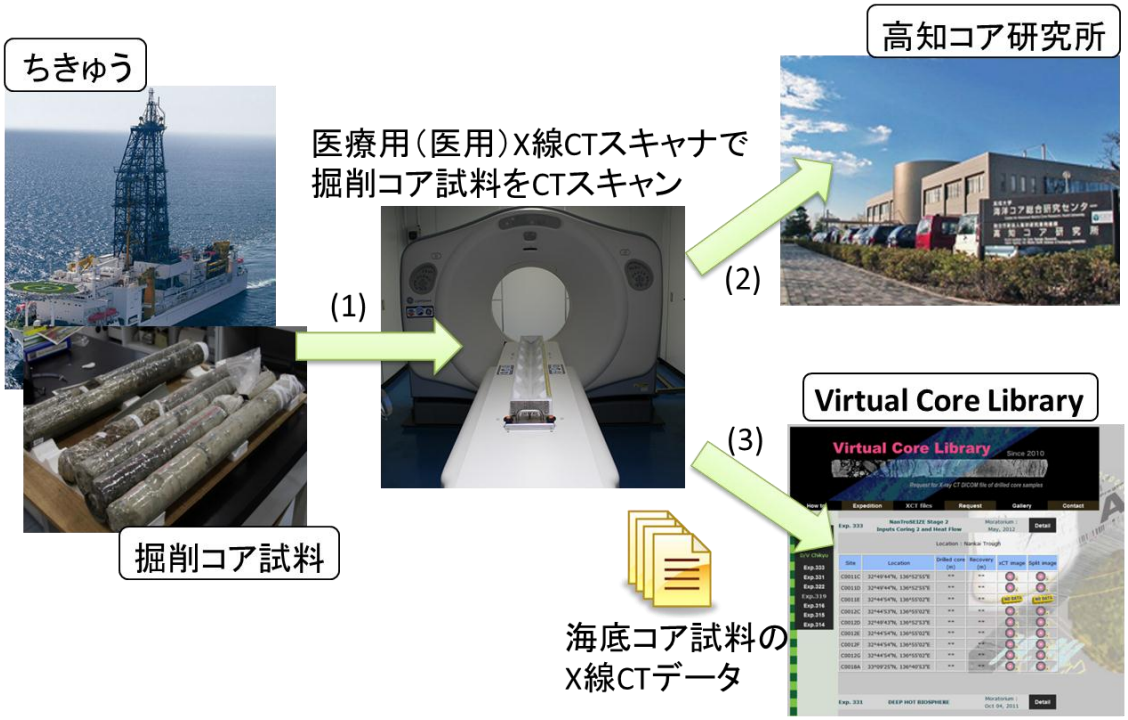


図 1　コア試料が X 線 CT データとして保存される様子

2.2. DICOM について

DICOM (Digital Imaging and COmmunications in Medicine)は，CT や MRI などで撮影された医用画像の世界標準規格である．本研究で取り扱うコアデータは，すべて X 線 CT スキャナによって，DICOM フォーマットで保存されている．X 線 CT スキャンとは，物体へ X 線を照射してその吸収度合いを測定することにより，物体の断面画像を得る技術のことである．物体の断面画像を得ることで，物体を破壊せずに内部構造を調べることができる．

X 線 CT スキャンによって得られた 1 枚の断面画像のことをスライスと呼ぶ．X 線 CT スキャンにより得られる 1 枚のスライスが，1 つの DICOM フォーマットの画像ファイルとして保存される．1 枚のスライスによる断面の表示に加え，複数のスライスを重ねあわせることで，3 次元画像を構成することができる．スライスから 2 次元画像，または，3 次元画像を構成する際は，DICOM ビューワと呼ばれる専用のソフトウェアが用いられる．

DICOM フォーマットの画像ファイルは，メタ情報と画像情報から構成されている．メタ情報には，撮影対象に関する情報が含まれている．医用の画像であれば，患者の名前や年齢，検査時刻などといった情報がメタ情報として保存されている．画像情報には，撮影された物体を画像として生成するのに必要な情報が含まれている．

DICOM フォーマットの画像ファイルは，画像のピクセル値が CT 値に対応している．CT 値とは，X 線の吸収度合を表現したものである．CT 値は密度と元素番号に相関した値を取る．具体的には，密度と元素番号が高いほど CT 値は高くなり，密度と元素番号が低いほど CT 値は低くなる．画像のピクセル値が，CT 値に対応している性質を利用して，CT 値の異なるピクセルに異なる色を振り分けることで，ある CT 値に対応する物体を強調して表示することができる．

DICOM フォーマットの画像ファイルから 3 次元画像を構成し，それに対して色づけを行う様子を図 2 に示す．掘削コア試料を CT スキャンすることで，①のようなコア試料の X 線 CT スキャンデータを得ることができる．X 線 CT スキャンにより得られる 1 枚 1 枚の断面画像がスライスであり，それぞれのスライスが DICOM フォーマットで保存された画像ファイルとなっている．複数のスライスを積み重ねることで，②のような 3 次元画像を構成することができる．構成された 3 次元画像に対して CT 値と色の組み合わせを決めることで，ピクセルと色の組み合わせが決まり，③のような色付きの 3 次元画像を構成することができる．

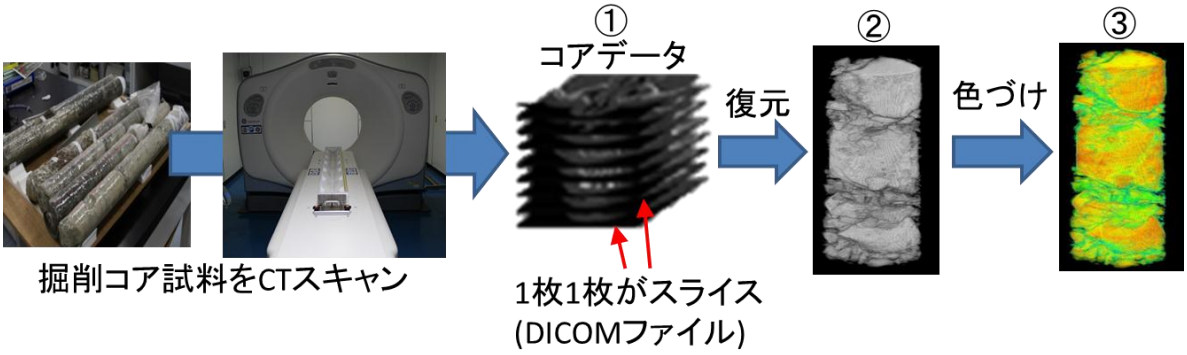


図 2　コアの X 線 CT データを復元する様子

2.3. クラウドサービスについて

クラウドコンピューティングとは，ネットワーク上に存在するハードウェア・ソフトウェア・データといった資源をサービスという形で利用することのできるコンピュータ利用形態のことである．クラウドコンピューティングの技術を用いて，提供するサービスのことをクラウドサービスと呼ぶ．現在稼働中のクラウドサービスとして，コンピュータ資源をサービスとして提供する Amazon Web Services[2]や開発環境をサービスとして提供する Google App Engine[3]などがある．

クラウドサービスでは，サービス提供者によってコンピュータ資源の管理が行われる．従来，コンピュータ資源はユーザが自身で管理しなければいけないものであったが，クラウドサービスを利用することで，ユーザがハードウェア・ソフトウェア・データなどの資源管理を行う必要がなくなる．クラウドサービスは，ネットワークに接続するだけで利用できるので，ネットワークに接続できる機器であれば，PC 端末だけでなく，携帯端末などでもサービスを利用できる．

2.4. 議論

コア試料のデジタルイメージは，地球科学分野の研究において非常に有用である．しかし一方で，コア試料のデジタルイメージを扱うにはいくつかの問題点があり，多くの地質学者は，コア試料のデジタルイメージを活用できていないのが現状である．DICOM 形式で保存されているコア試料のデジタルイメージを専用ソフトウェアで閲覧する際の 3 つの問題点を以下に示す.

(1)データサイズ

コアデータは 1.5m ずつ保存されており，これを 1 セクションとしている．コアデータのデータサイズは，1 セクションあたり 300M～400Mbyte である．コアデータを扱うためには，事前に自身の端末に DICOM ファイルをダウンロードしておく必要があるため，複数のコアデータを扱いたい場合は，ローカル端末に十分な空き容量を確保しておかなければならない.

(2)端末に要求される処理能力

コアデータを閲覧するためには，専用のソフトウェアで 3 次元描画処理を行わなければならない．コアデータの 3 次元描画を行うためには，高速なプロセッサと大容量のメモリが要求される．これらが不足している端末上で，コアデータの 3 次元描画を行うのは困難である．特に携帯端末は，3 次元描画に必要なスペックを満たしていないものがほとんどであり，携帯端末上で，既存のコアデータを 3 次元描画し，閲覧することはほぼ不可能である.

(3)専用ソフトフェアが存在しない

DICOM は，医用画像のファイル形式として使われてきた規格である．そのため，OsiriX[4]などの既存の DICOM ビューワは医用画像の分析に特化した機能を提供している. OsiriX は, 筋肉や骨といった医用の CT スキャンデータの分析には適しているが, コア試料の X 線 CT スキャンデータの観察には適していない．つまり，地質学者の視点で掘削コア試料の X 線 CT データを観察するための十分な機能を持つ DICOM ビューワが存在していない.

以上，3つの問題点より，ローカル端末上でコアデータの3次元描画処理を行い，閲覧するのは困難である．ローカル端末上における問題点を解決するために，リモートサーバを用いたDICOM ファイルの提供・閲覧に関する研究が行われている.

大容量の DICOM ファイルをリモートサーバに保存する方法として，ネットワークに連結されたストレージを利用する研究がある[5]．しかし，このサービスは DICOM ファイルを提供するだけのサービスであり，DICOM ファイルを直接閲覧する環境(ソフトウェアや表示用機材) が存在しないため，問題点(2)(3)が解決されていない.

リモートサーバでDICOMファイルの3次元描画処理を行い，ローカル端末で閲覧する方法として，MPEG ビデオストリーミングを利用した研究がある[6]．この研究ではリモートサーバで視覚化した3 次元モデルを，MPEG ビデオストリーミングを通してクライアントのモバイル端末に表示している．ストリーミングは，通信するデータサイズが大きく，ネットワーク帯域を多く利用するため，サービスの質が，ネットワークの帯域やレイテンシ等のネットワークの品質に大きく左右されてしまう．この研究では，(1)データサイズにおける問題点が

解決できない．

いずれの研究も，先述したコアデータの利用における3つの問題点を解決するには至っていない．本研究では，上記の問題点を解決するために，DICOMファイルを保存・処理するサーバを構築し，コアデータの閲覧が可能となるクラウドサービスを提供する．本サービスの導入前と導入後のイメージを図3に示す．本サービスは，DICOMファイルであるコアデータをリモートのサーバに保存し，そのコアデータに対して3次元描画処理を行う．本サービスにより，ユーザは，ネットワークに接続可能な環境であれば，本サービスが提供するインタフェースを利用し，コアデータの閲覧が可能となる

これにより，ユーザは従来のようにローカルの端末にコアデータをダウンロードする必要がなくなるため，1つ目の問題点である(1)データサイズに関する問題を解消することができる．加えて，コアデータの3次元描画処理がリモートのサーバで行われるため，ローカル端末でのDICOMビューワを使った3次元描画処理が不要となり，端末性能に依存せずにDICOMファイルであるコアデータを閲覧できる．これにより，2つ目の問題点である(2)端末に要求される処理能力を解消できる．端末性能に依存しないため，従来困難であった携帯端末によるDICOMファイルの閲覧が可能となり，タブレット端末などでコアデータを閲覧できるようになる．(3)専用ソフトフェアが存在しない問題に関しては，地質学者からコア試料の観察に関する方法をヒアリングし，コア試料の観察に特化したインタフェースを提供することで問題解決をはかる．

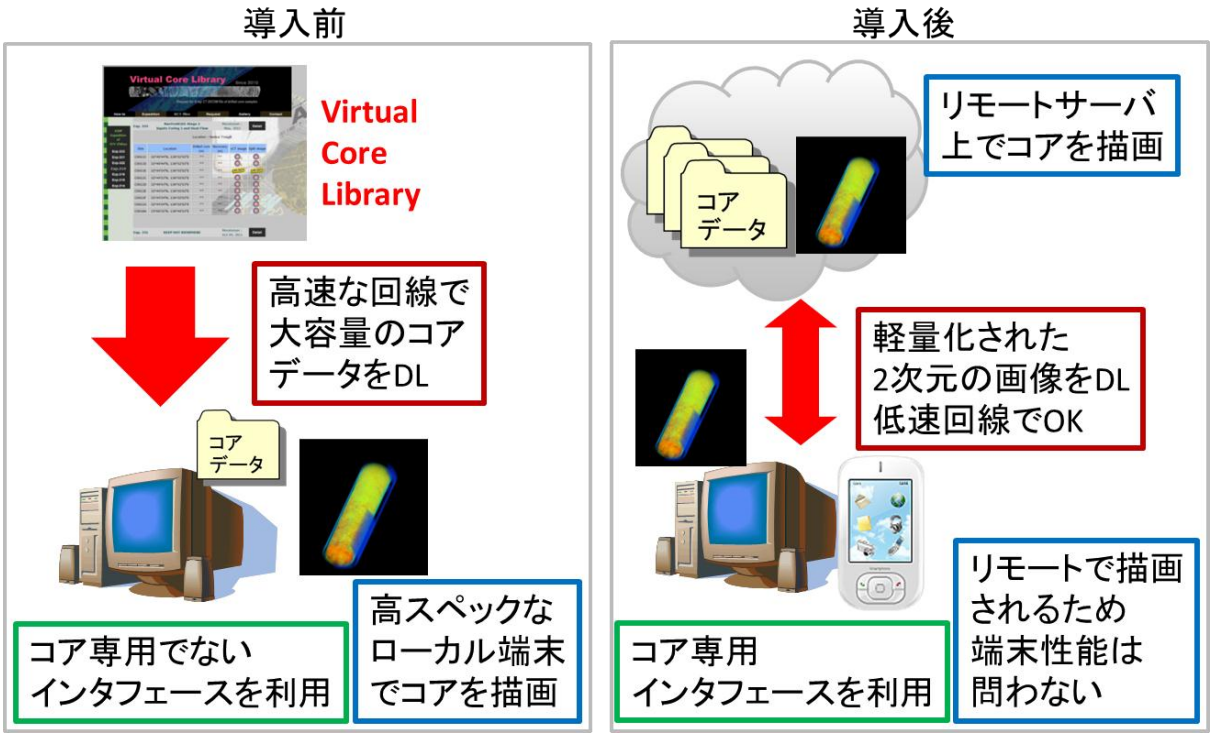


図 3　サービスの導入前と導入後のコアデータ利用方法の変化

3. 海底コア CT スキャンイメージ可視化のためのクラウドサービス

3.1. システム構成

前章の議論の項目で示した DICOM ファイル利用の際の問題点とその解決策を踏まえ，以下に示すシステム要件を定義した．

1. 端末性能に依存せず，掘削コア試料の閲覧が可能であること．
2. コアに対してインタラクティブな操作が可能であること．
3. 掘削コア試料の観察に適したインタフェースであること．

上記3項目の要件から，タブレット端末にて，コアデータの閲覧が可能となるクライアントアプリケーション，及び，コア試料のデジタルデータを処理するサーバから成るクラウドサービスを開発する．そのサービスの全体像を図4に示す．

クライアントが，コア試料のデジタルイメージを取得するまでの流れを以下に示す．まず，クライアントから画像描画リクエストをサーバに対して送信する．画像描画リクエストを受信したサーバは，まず DICOM ファイルから 3 次元モデルを生成する．サーバでは，生成された 3 次元モデルを，データサイズが軽量な 2 次元画像に変換してからクライアントに送信する．3 次元モデルを 2 次元画像に変換することで，データサイズを軽量にし，送信の際にネットワークにかかる負荷を軽減する．2 次元画像に変換することによって，通信回線が貧弱な環境でも本サービスを利用可能できる．

本サービスは，ネットワーク環境さえあれば利用可能であるため，Web ブラウザを介して本サービスにアクセスし，コアデータを閲覧することができる．加えて，タブレット端末による直観的な操作を可能とするため，タブレット端末から本サービスにアクセスできる専用のクライアントアプリケーションを提供する．いずれの方法においても，コア試料の観察を可能とする機能を持つインタフェースをクライアントアプリケーションとして提供する．

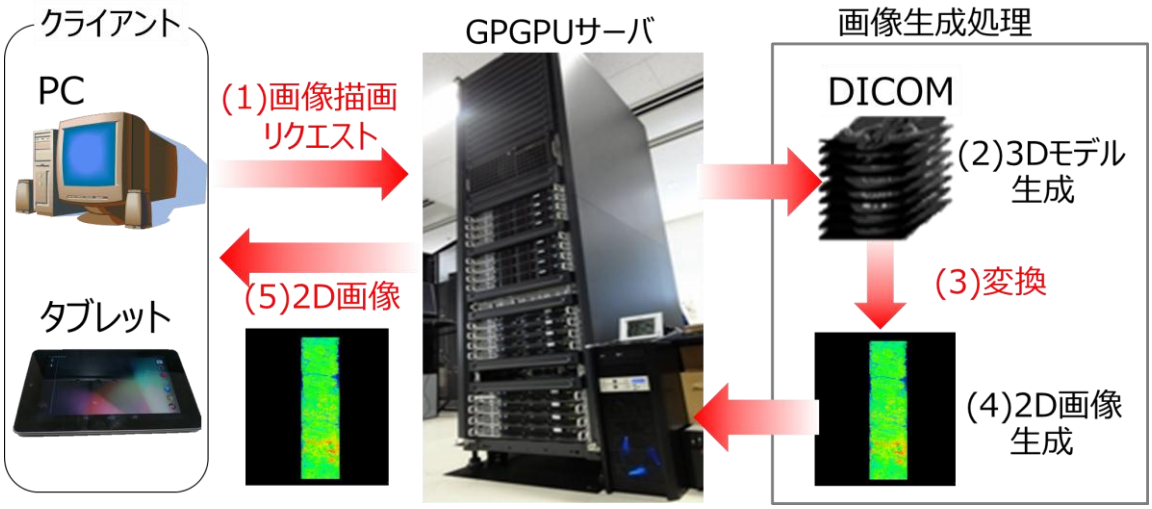


図 4　構築するシステムの構成

本サービスの提供により，モバイル機器でのコアデータの閲覧が可能となるため，出張や学会などの外出先でも，インターネットに接続するだけで，コアデータを閲覧できるようになる．従来，外出先でコアデータを閲覧したい場合は，自身の PC 端末にコアデータをダウンロードしておかなければならなかったが，あらかじめコアデータをサービスに登録しておくことで，自身の端末にコアデータをダウンロードせずに，気軽にデータを閲覧できるようになる．本サービスでは，数十 Kbyte の画像データを読み込むだけでコアデータを閲覧できるため，「ちきゅう」船上などの通信回線が貧弱な環境下でも，コアデータを閲覧することが可能となる．

3.2. 提供する機能

本サービスは，コア試料の観察を可能とする機能を持つクライアントアプリケーションを提供する．提供する機能の一覧を以下に示す．

1．コア試料の検索機能

すべてのコア試料には，掘削が行われた航海プロジェクト，掘削された場所などを識別できる番号にもとづいて，名前がつけられている．例えば，南海トラフへの航海プロジェクトにおいて，北緯 32 度，東経 136 度の場所で掘削されたコア試料は，航海プロジェクトを表す 333 番という番号と，掘削場所を表す C0011C という番号が名前に含まれる．

地質学者は，これらの識別番号にもとづいて，閲覧したいコア試料を探し出す．しかし，現在閲覧可能なコア試料は 4000 セクション以上あるため，自身でひとつひとつ確認しながらコア試料を探し出すのは手間がかかる．そこで，本サービスを提供するにあたって，簡単にコア試料を選択できるような検索機能を提供する．検索機能では，航海プロジェクト，掘削された場所などの情報を選択することで，ユーザが閲覧したいコア試料を探し出す．

2．コア試料の分析機能

地質学者は，掘削コア試料の断面を観察することで研究を行う．掘削コア試料は，地層であり，鉛直方向に重なった形状を示す．地層の縦方向で切断した断面を見ることで，地質学者は，コア試料から様々な情報を得ることができる．本サービスにおいてもコア試料の断面の観察を可能とするために，縦方向の切断を機能として提供する．回転と拡大・縮小機能は，3 次元化されたコアを閲覧するための基本的な操作である．コアの全体像や細部表示，任意の視点からの閲覧を可能とする．拡大により，細部を表示している際に，他の部分が見えなくなってしまう．その場合に拡大したままコアの表示を変更できるように，表示領域を変更する機能を提供する．

3. CT 値に対する色づけ機能

コアデータには，水や筒といったコア試料の観察に不要な物体が多く写りこんでいる．表示する CT 値幅を設定することで，観察に関係のない物体を取り除くことができる．表示する CT 値幅を設定しただけでは，コアを観察するには不十分である．設定した CT 値に対して色付けがされていなければ，物体の違いや構造が画像として表現されない．そこで，コア試料の CT 値に対して，ユーザが色付け・透過設定ができるインタフェースを提供する．

コアデータの観察に適した CT 値幅を推測で設定するのは難しい．CT 値とその発生頻度を CT 値ヒストグラムとしてユーザに表示することで，ユーザの CT 値幅の設定作業を容易にする．

3.3. システム設計

本サービスを実現するために, 図 5 に示すクライアントアプリケーション, ゲートウェイ, レンダリングエンジンから構成されるシステムを設計した.

クライアントアプリケーションは, 本サービスを利用するためのインタフェースを提供し, ユーザの操作に応じてゲートウェイにリクエストを送信する. ゲートウェイは, 受信したリクエストに基づいて, コアデータの 2 次元画像や情報を構造化したデータとして, クライアントアプリケーションに送信する. 2 次元画像の生成は, ゲートウェイに接続されたレンダリングエンジンによって行われる. それぞれの詳細を次の節より示す.

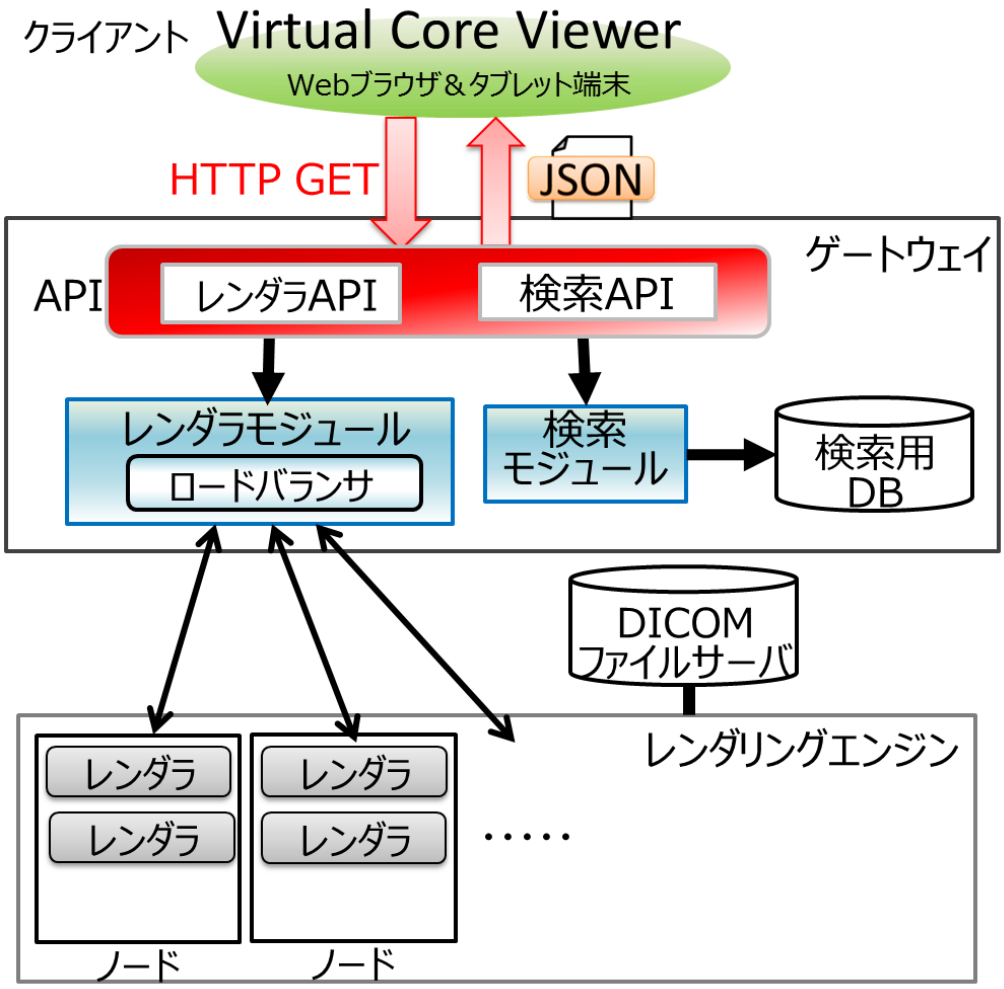


図 5　本システムの全体設計

3.3.1. クライアントアプリケーション

クライアントアプリケーションは, コアの観察に必要な機能を搭載したユーザインタフェースを提供する. タブレット端末からの利用を想定した Android アプリケーションと, PC からの利用を想定した Web アプリケーションの 2 つのクライアントアプリケーションを Virtual Core Viewer として提供する. ユーザからの操作に応じて, ゲートウェイが有する各 API を利用することで, コア画像や各種情報を取得する.

3.3.2. ゲートウェイ

ゲートウェイは，レンダラモジュールと検索モジュールの 2 種類のモジュール，そして，それぞれのモジュールを利用するためのレンダラ API と検索 API を有する．クライアントアプリケーションからリクエストを受けとった API が，対応するモジュールに処理を依頼する．レンダラモジュールとレンダラ API，検索モジュールと検索 API がそれぞれ対応している．

検索モジュールは，掘削コア試料を検索する際に利用するモジュールである．検索モジュールは，クライアントから受け取ったパラメータに基づいて，検索用 DB にアクセスするためのクエリを生成する．生成したクエリで検索用 DB にアクセスすることで，利用可能なコアデータの情報を取得し，その情報をクライアントアプリケーションに返す．検索用 DB には，利用可能なコアデータが登録されている．

レンダラモジュールは，コアデータの画像を取得する際に利用するモジュールである．レンダラモジュールは,閲覧したいコアデータの番号と操作情報をクライアントから受け取り，GPU クラスタのノードに画像描画処理を依頼する．画像描画処理の依頼は，ロードバランサを介して行われる．画像描画処理終了後，コア画像とその情報をクライアントアプリケーションに返す．ロードバランサは，負荷の少ないノードを選択して，コア画像の描画処理を依頼する．あるノードにリクエストが集中した場合，ユーザへのレスポンスが遅くなり，サービスの質の低下を招く可能性がある．複数のノードに対してリクエストを送信し，負荷を分散することで，サービスの質を保つ．

ゲートウェイは，CPU(Core i7 930 2.8GHZ)，メモリ 12GB で構成されている．

3.3.3. レンダリングエンジン

レンダリングエンジンでは，コアデータの画像描画処理を行う．レンダリングエンジンは 16 台のノードで構成されている．各ノードは，CPU(Intel Xeon E5645 2.40GHz (6 cores) ×2，メモリ 12GB で構成されている．

レンダリングエンジンの各ノードにレンダラを起動することで，コアデータの画像が生成される．レンダラは，受信したリクエストにもとづいてコアデータの画像を生成するプロセスである．画像生成に用いるコアデータは，すべて DICOM ファイルサーバに格納されている．コアデータの画像生成処理を行う際は，適宜，DICOM ファイルサーバからコアデータを読み込むことで画像生成処理を行う．なお，ゲートウェイ－GPU クラスタ間は，Infiniband QDR で接続されている．

設計したサービスにおけるコアデータ閲覧までの流れは次のとおりである．まず，コアデータを閲覧するために，コアデータの情報を取得する．コアデータの情報は，クライアントアプリケーションから,検索APIを介して検索モジュールにリクエストを送信することで取得できる．検索モジュールでは，受信したリクエストをもとに検索用DBにアクセスし，閲覧したいコアデータの情報を特定し，クライアントアプリケーションに返す．

次に，取得したコアデータの情報を用いて，レンダラAPIを介してレンダラモジュールにリクエストを送信する．レンダラモジュールは，受信したリクエストに基づいて，レンダリングエンジンのノードで起動しているレンダラに画像生成処理を依頼する．画像生成処理が終了した後に，クライアントアプリケーションにコアデータの画像が表示される．クライアントアプリケーションからコア試料を観察のための操作を実行した場合，コアデータの情報とコアに対する操作情報をリクエストとして送信することで，操作情報が反映されたコア画像

が生成・表示される．レンダラへの画像生成処理の依頼はロードバランサを介して行い，画像生成処理によりかかる負荷を可能な限り分散する．

3.3.4. 著者の担当領域について

ゲートウェイが有する機能の内，レンダラAPIを除く，検索API，検索モジュール，検索用DB，レンダラモジュール・ロードバランサ，そして，クライアントアプリケーションにおけるWebアプリケーションの設計・実装を担当した．

検索API，検索モジュール，検索用DBについては，第4章で述べる．検索API，検索モジュール，検索用DBの設計・実装をすることで，絞り込みによる検索機能を実現する．絞り込みによる検索機能を実現することで，4000セクション以上ある大量のコア試料の中から，ユーザが望む1つのコア試料を取得できるようになる．検索機能は，コア試料を識別する番号にもとづいた検索を行うことで，効率的にコア試料を探し出すことができる．

レンダラモジュールの設計・実装について第5章で述べる．レンダラモジュールは，レンダラの起動と削除，レンダラへのリクエストの送信を行う．レンダラはコアデータの2次元画像を生成するプロセスであり，レンダラの起動は，レンダラモジュールで行う．レンダラで実行されるコアデータの2次元画像生成処理は負荷が高く，レンダリングサーバに負荷がかかる．レンダラリングサーバにかかる負荷を効率的に分散・軽減することで，レンダリングサーバを最大限活用できるようなレンダラの管理方法を設計・実装する．

Webアプリケーションについて第6章で述べる．著者は，一般的なWebブラウザを用いたコアデータの閲覧を可能とするために，クライアントアプリケーションの内，Webアプリケーションの設計・実装を担当する．Webアプリケーションは，操作性を考慮したAjaxによる設計・実装を行う．Webアプリケーションの公開は，Virtual Core Libraryで行う．Webアプリケーションの公開方法を考慮し，それに応じたWebアプリケーションのUIを実装する．

4. コア試料検索のためのミドルウェアの設計・実装

本章ではまず，検索機能を実現するにあたって必要となる，クライアントアプリケーションーコア試料検索機能間の通信プロトコルと，コア試料を効率的に探し出す方法について述べる．次に，効率的にコア試料を探し出す方法を，検索機能として実現するために設計・実装した検索 API，検索モジュール，検索用 DB の 3 項目について述べる．

4.1. コア試料検索のためのミドルウェアの設計

コア試料検索のためのミドルウェアを実装するにあたって，クライアントアプリケーションとコア試料検索機能間での通信プロトコル，効率的にコア試料を検索するための方法を設計した．

4.1.1. クライアントアプリケーションーコア試料検索機能間の通信プロトコル

クライアントアプリケーションから本サービスで提供する検索機能の利用を可能とするために，双方向の通信プロトコルを設計した．Web アプリケーションと Android アプリケーションの 2 種類のクライアントアプリケーションを実装することを考慮に入れ，通信プロトコルの設計を行う．

クライアントアプリケーションからコア試料検索機能への通信方法は，HTTP GET メソッドを用いて実現する．HTTP プロトコルは，プログラミング言語や動作するプラットフォームに依存しない汎用的な通信方法であるため，2 種類のクライアントアプリケーションで開発する本サービスにとって都合が良い．HTTP GET メソッドでは，*http://URL?パラメータ=値* の形式でサーバに対して情報を送ることができる．*http://URL?パラメータ=値&パラメータ=値&パラメータ=値*のようにパラメータと値を「&」でつなぐことで，サーバに対して複数の情報を送信することができる．

コア試料検索機能からクライアントアプリケーションへの通信は，JSON(JavaScript Object Notation)を用いて実現する．JSON は，軽量なデータ記述言語である[7]．JSON は，数字や文字列，配列などの様々なデータを，キーと値の組み合わせを使ったデータ構造で表現できる．同様のデータ記述言語として XML がある[8]．それぞれの記述言語で同じデータ構造を示した例を図 6 に示す．

JSON

```
{
  {
    “id” : 1
    “type” : ”JSON”
  },
  {
    “id” : 2
    “type” : “XML”
  }
}
```

XML

```
<formats>
  <format>
    <id>1</id>
    <type>JSON</type>
  </format>
  <format>
    <id>2</id>
    <type>XML</type>
  </format>
</formats>
```

図 6　JSON と XML における記述の例

XML では，<id></id>や<type></type>といった型を示すタグで値を囲むことでデータの型とその値を記述する．データ構造は，<formats></formats>などのタグを記述しなければならない．一方，JSON では，“データの型”：値の形式で，データの型とその値を簡単に記述することができる．データ構造は，{ }で囲むだけで表現できる．JSON は，XML のようなマークアップ言語と比較して，簡潔な記述でデータ構造を表現できるため，軽量なデータでメッセージをやり取りすることができることがわかる．

JSON は，非常に簡潔なデータ構造であるため，JavaScript に加え，PHP や Java などの様々な言語で，簡単にデータを取り出すことができる．本サービスで提供するクライアントアプリケーションは，Web アプリケーションと Android アプリケーションの 2 種類であるため，JSON の持つ開発言語に依存しないという性質と相性が良い．

Web アプリケーションでは，インタラクティブな操作を可能とするために，JavaScript を用いた動的なインタフェースを検討している．JavaScript は JSON との親和性が高く，容易にデータを解析・取り出すことができる．

データ構造が簡潔であること．データサイズが軽量であること．複数の言語で開発されたクライアントアプリケーションでデータを受け取り，容易に解析が行えること．JavaScript との親和性が高いこと．以上の理由から，クライアントアプリケーションへの処理結果の返信方法として JSON を選択した．

以上より，クライアントアプリケーションーコア試料検索のためのミドルウェア間の通信プロトコルとして，HTTP GET と JSON を用いる．HTTP GET を用いることでコア試料検索機能に対してリクエストを送信し，そのリクエストの結果を JSON 形式で取得する．

4.1.2. コア試料検索のためのミドルウェアの処理方針

コアデータのファイル名は, 表 2 に規定される 6 つの項目の組み合わせで決められている．航海番号と掘削サイト番号，ホール番号は，IODP によって規定されている番号であり，航海番号は航海プロジェクトの番号，掘削サイト番号，ホール番号は位置情報に基づいて決定される．コア番号は，掘削されたコア試料ごとに付けられる連番の番号である．ビットタイプは, コア試料の掘削を行ったドリルの識別番号である. セクション番号は, コア試料の 1.5m 間隔に割り振られた連番の番号のことを指す. セクション番号は, CC という値を取り得る．CC は，コア試料の掘削時にドリルの先端部にて回収されたコア試料のことである．基本的にセクション番号は，昇順の整数になっており，最後の値は CC となる．CC は，多くのコア試料で回収されるが，稀に回収されない場合がある．その場合は例外として，セクション番号 CC の存在しないコア試料となる．

表 2　コア試料の命名規則

項目名	規則
航海番号	3 桁の数字
掘削サイト番号	英字 1 文字+5 桁の数字
ホール番号	英字 1 文字
コア番号	正の整数
ビットタイプ	英字 1 文字
セクション番号	正の整数　または　文字列 CC

コアデータは，掘削が行われた航海プロジェクトや掘削が行われた場所を示す航海番号や掘削サイト番号をもとに探し出すことができる．しかし，2013 年 1 月現時点でコアデータの数は 4000 以上と膨大である．膨大なコアデータの中から，観察したいコアデータを見つけ出すのは難しい．

航海番号，掘削サイト番号，ホール番号，コア番号，ビットタイプ，セクション番号を順番に選択することで，コアデータの候補を徐々に絞り込み，大量のコアデータの中から効率的にコアデータを探し出すことができる．コアデータの候補を徐々に絞り込み，1 つに特定する様子を図 7 に示す．

複数ある航海番号の中から航海番号を 1 つ選択することで，その航海番号に関連づけられた掘削サイト番号の一覧がわかる．図 7 の例では，航海番号として 315 番を選択することで，C0001 と C0002 の 2 つの掘削サイト番号がわかる．続いて，掘削サイト番号を 1 つ選択することで，ホール番号の一覧がわかる．航海番号 315 番，掘削サイト番号 C0001 番を選択すると，E,F,H の 3 つの航海番号がわかる．同様にして，1 項目ずつ順番に番号を選択していくことで，最終的に 315-C0001F-3H-4 というコアデータを探し出すことができる．

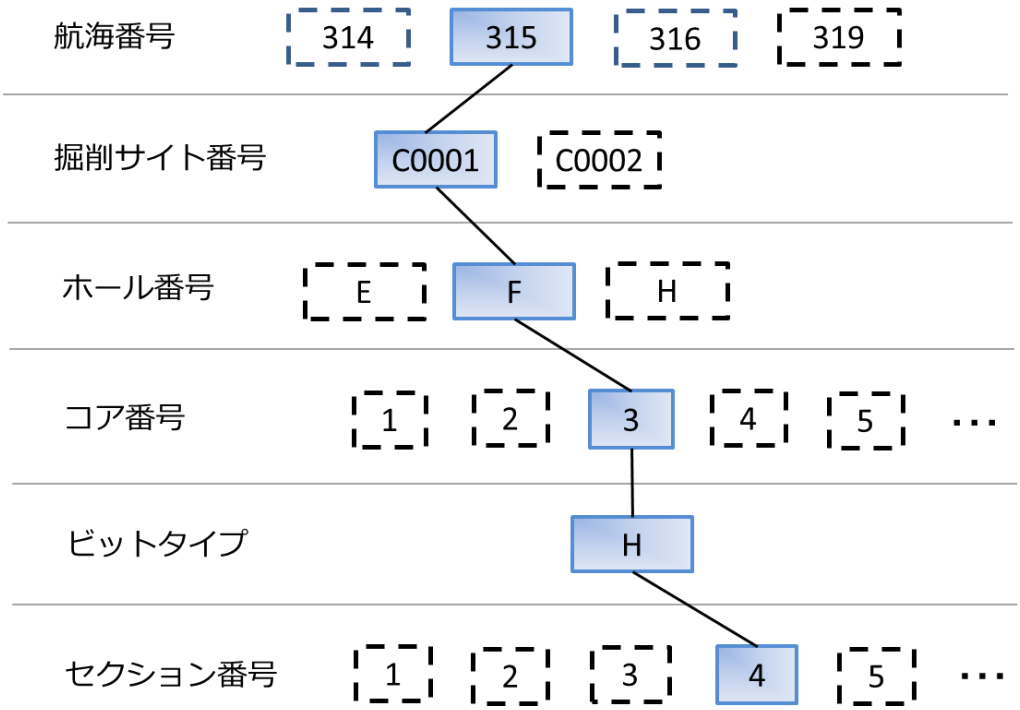


図 7　コア試料検索の様子

図 7 に示す手順で，コアデータの候補を徐々に絞り込み，1 セクションのコア試料を探し出す機能を検索機能として実現する．検索機能の構成を図 8 に示す．検索機能は，検索 API，検索モジュール，検索用 DB で実現する．検索 API，検索モジュールは PHP で実装し，検索用 DB は MySQL で構築する．

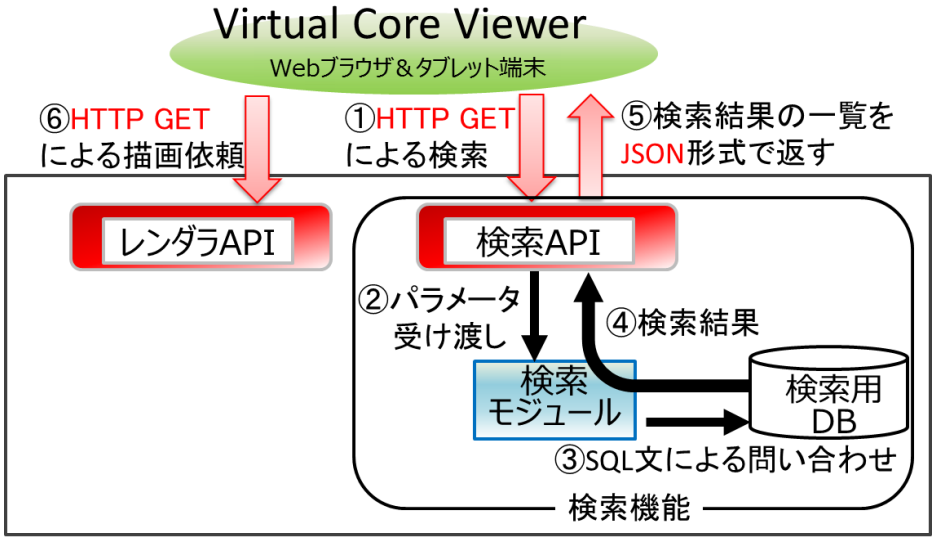


図 8　コア試料検索のためのミドルウェアの設計・実装

絞り込みによる検索を行う際は，はじめに，クライアントアプリケーションから HTTP GET で検索 API にコア試料の 6 つの識別番号を引き渡す．検索 API でコア試料の識別番号を受信したら，受信したパラメータを検索モジュールにそのまま受け渡す．検索用モジュールでは，受け取ったパラメータにもとづいたクエリを発行し，検索用 DB に対して問い合わせを行う．問い合わせにより得られた検索結果は，検索モジュールを介して検索 API に渡される．検索 API では，検索結果を JSON 形式に加工した後に，返り値としてクライアントアプリケーションに返す．

クライアントアプリケーションへの返り値は，検索 API に送信したパラメータにより変化する．航海番号をパラメータとして送信した場合は，選択された航海番号を持つ掘削サイト番号の集合が返り値となる．返り値の中から 1 つ掘削サイト番号を選択し，航海番号と掘削サイト番号をパラメータとして送信することで，選択された航海番号と掘削サイト番号を持つホール番号の集合が返り値となる．同様に，返り値の集合の中から，1 つずつ番号を選択することで，航海番号，掘削サイト番号，ホール番号，コア番号，ビットタイプ，セクション番号を選択し，コア試料を一意に特定することが可能となる．コア試料を一意に特定することができた場合，そのコア試料の 6 つの番号を HTTP GET メソッドで検索 API に引き渡すことで，コア試料に固有に割り振られた ID を返り値として受け取ることができる．検索 API から最終的に受け取る返り値は，コア試料に割り振られた ID となる．ID を，レンダラ API に HTTP GET で引き渡すことによって，探し出したコア試料を閲覧することが可能となる．

4.2. コア試料検索のためのミドルウェアの実装

本項では，絞り込みによるコア試料の選択を可能とするために，実装した検索 API，検索モジュール，検索用 DB について述べる．まず，絞り込みによる検索を可能とするために構築した検索用 DB について述べる．その後，検索用 DB にアクセスするための検索 API,検索モジュールについて述べる．

検索用 DB は，本サービスで利用可能なすべてのコア試料を検索しやすい形で登録してあるデータベースである. 検索用 DB へのコア試料の登録は, 専用のスクリプトを用いて行う．

スクリプトによるデータベースへの登録方法の詳細は，付録 B に示す．

構築したデータモデルを図 9 に示す．coresampleinfo テーブルは，航海番号，掘削サイト番号，ホール番号，コア番号，ビットタイプ，セクション番号の 6 つの番号をカラムとしたテーブルである．コア試料を構成する 6 つの番号をカラムとすることで，それぞれの番号を用いたコア試料の検索が可能となる．coresampleinfo テーブルの主キーである coreid は，検索用 DB に登録されているコアデータに割り振られている固有の ID のことである．システム内でコア試料の番号をやりとりする際は，この ID を使って行われる．検索 API の最終的な返り値となる ID は，この coreid の値である．

fileinfo テーブルは，コアデータのファイルパスを格納したテーブルである．本サービスで利用可能なコアデータはすべて DICOM ファイルサーバに格納されている．fileinfo テーブルの filepath の項目は，それぞれのコア試料が保存されているファイルパスを示している．fileinfo テーブルには，外部キーとして coresampleinfo の主キーである coreid が格納されている．必要に応じて，coreid を用いてファイルパスの検索を行うことで，その coreid に対応するコア試料のファイルパスを取得することができる．

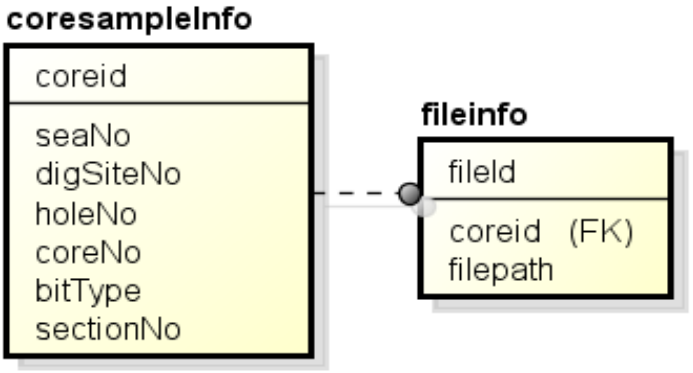


図 9　検索用 DB のデータモデル

検索用 DB には，検索 API と検索モジュールを介してアクセスする．検索 API に HTTP GET を用いてなんらかのパラメータを引き渡すことで，その値が検索モジュールに引き渡され，検索用 DB に問い合わせるためのクエリが発行される．

検索 API に引き渡せるパラメータを表 3 のように定義した．検索 API に送信可能な 6 つのパラメータは，コア試料の 6 つの識別番号にそれぞれ対応している．表 4 は，検索 API に送信したパラメータと，そのときの返り値の対応である．検索 API では，受信したパラメータに応じて，異なる返り値をクライアントアプリケーションに返す．返り値の中から番号を 1 つ選択し，検索 API に送信するパラメータを 1 つずつ追加していくことで，絞り込みによる検索を実現する．検索 API に送信するパラメータを，航海番号，掘削サイト番号，ホール番号，コア番号，ビットタイプ，セクション番号の順に追加していくことで，コア試料を探し出すことができ，最終的に該当するコア試料の ID である coreid を取得できる．

表 3　API のパラメータとその意味

パラメータ	意味
seano	航海番号.
digsite	採掘サイト番号.
hole	ホール番号.
coreno	コア番号.
bittype	ビットタイプ
sectionno	セクション番号.

表 4　検索 API の受信パラメータと返り値

検索 API の受信パラメータ	返り値
なし	seano
seano	digsite
seano, digsite	Hole
seano, digsite, hole	coreno
seano, digsite, hole, coreno	bittype
seano, digsite, hole, coreno, bittype	Sectionno
seano, digsite, hole, coreno, bittype, sectionno	coreid

クライアントアプリケーションでは，以下のように HTTP GET メソッドで検索 API にパラメータを引き渡すことで，送信したパラメータにもとづいて返り値を取得できる．

http://検索 API ?パラメータ=値&パラメータ=値&…

315-C0001F-3H-4 というコア試料を絞り込み機能で探し出す場合は，表 5 のような手順で検索 API に対してリクエストを送信する．検索 API を利用することで得られる返り値の中から 1 つ値を選択して, 検索 API の末尾にパラメータとして付随し, 絞り込み検索を行う．なお，表中の赤文字の値が，返り値の中から選択された番号である．

表 5　検索 API に引き渡すパラメータと返り値の具体例

クライアントから送信するパラメータ	返り値
http://検索 API	314, 315, 316, 319
http://検索 API?seano=315	C0001, C0002
http://検索 API?seano=315&digsite=C0001	E, F, H
http://検索 API?seano=315&digsite=C0001&hole=F	1, 2, 3, 4, 5, …
http://検索 API?seano=315&digsite=C0001&hole=F&coreno=3	H
http://検索 API?seano=315&digsite=C0001&hole=F&coreno=3&bittype=H	1, 2, 3, 4, 5, …
http://検索 API?seano=315&digsite=C0001&hole=F&coreno=3&bittype=H&sectionno=4	Coreid

検索モジュールは, 検索 API が受信したパラメータにもとづいてコア試料を検索するためのクエリを発行する．検索モジュールの発行したクエリによる検索用 DB への問い合わせ結果が, 検索 API の返り値となる．検索モジュールの詳細な設計と処理内容は付録 C に示す．

検索 API から受け取ったパラメータと発行するクエリの対応を表 6 に示す. 受け取ったパラメータを WHERE 句の条件として追加していくことで，コア試料を徐々に絞り込む．整数値であるコア番号，セクション番号を取得する時は，ORDER BY 句を付けることにより，昇順で番号を取得している．航海番号，掘削サイト番号，ホール番号，コア番号，

表 6　受信パラメータと発行するクエリ

API から受信したパラメータ	発行するクエリ
なし	SELECT seano FROM coresampleinfo;
Seano	SELECT digsite FROM coresampleinfo WHERE seaNo = seano;
seano, digsite	SELECT hole FROM coresampleinfo WHERE seaNo = seano AND digsiteNo = digsite;
seano, digsite, hole	SELECT coreno FROM coresampleinfo WHERE seaNo = seano AND digsiteNo = digsite AND holeNo = hole order by sectionno;
seano, digsite, hole, coreno	SELECT coreno FROM coresampleinfo WHERE seaNo = seano AND digsiteNo = digsite AND holeNo = hole AND coreNo = coreno;
seano, digsite, hole, coreno, bittype	SELECT sectionno FROM coresampleinfo WHERE seaNo = seano AND digsiteNo = digsite AND holeNo = hole AND coreNo = coreno AND bittype = bittype ORDER BY sectionno;
seano, digsite, hole, coreno,bittype, sectionno	SELECT coreid FROM coresampleinfo WHERE seaNo = seano AND digsiteNo = digsite AND holeNo = hole AND coreNo = coreno AND bittype = bittype AND sectionno = sectionno;

ビットタイプ，セクション番号の 6 つの番号が受け渡されたときは，コア試料の固有 ID を取得するためのクエリを発行する．これらのクエリによって得られた検索結果を検索 API に渡し，検索 API では検索モジュールから受け取った値を JSON 形式に加工し，クライアントアプリケーションに返す．

4.3. まとめ

本サービスでは，4000 セクション以上あるコア試料から 1 セクションのコア試料を選択するための機能として，検索機能を実現した．

2013 年 1 月時点で 4000 セクションを越えるコアセクションが存在する．本サービスを利用する際に，膨大なコア試料の中からユーザが望むコア試料を 1 セクションのみ探し出すのは困難である．

コア試料は，絞り込みによる検索で効率的に探し出すことが可能であるため，この検索方法をコア試料検索のためのミドルウェアとして実現するために, 検索 API, 検索モジュール, 検索用 DB を設計・実装した．

利用可能なコア試料の情報がすべて登録された検索用 DB,クライアントアプリケーションからリクエストを受け付け，検索結果を返す検索 API，検索 API が受信したパラメータにもとづいて検索用 DB に問い合わせを行う検索モジュールを実装することで，コア試料の絞り込みによる検索を実現した．

コア試料検索のためのミドルウェアを実現したことで, 検索 API との対話的なリクエストのやりとりを通じて，4000 セクションを越えるコア試料の中から，1 セクションのコア試料を容易に探し出すことが可能となった．

5. レンダラモジュールの設計と実装

レンダラモジュールは，画像生成に欠かせないレンダラの起動と削除の役割を担う．本章では，まず，レンダラモジュールの処理方針について述べる．レンダラを起動・削除する際の問題点について述べ，それらの問題点を踏まえた上で設計したレンダラモジュールの処理方針を述べる．次に，その処理方針を踏まえた上で実装したレンダラの起動・削除処理について述べる．

5.1. レンダラモジュールの処理方針

レンダラは，コア試料の画像を生成するためのプログラムであり，レンダリングエンジンのノードに起動される．レンダリングエンジンのノードは 16 台あり，各ノードは，CPU(Intel Xeon E5645 2.40GHz (6 cores) ×2，メモリ 12GB で構成されている．レンダラの起動は，この 1 台のノードに対して行われる．

ユーザからコア試料のリクエストを受けたときに，GPU クラスタの 1 台のノードに，該当するコア試料の画像生成を行うレンダラを 1 つ起動する．ユーザから画像リクエストを受け取ったゲートウェイが，ノードに対して，専用のコマンドを実行することでレンダラを起動する．起動されたレンダラは，ノードに常駐し，ユーザからのリクエストに応じてコア試料の画像を生成する．レンダラは，1 種類のコア試料につき，1 つ起動する．例えば，3 種類のコア試料の画像が要求された場合，コア試料の種類ごとに計 3 つのレンダラが起動する．レンダラの仕様上，様々な種類のコア試料を閲覧した場合，複数のレンダラがノードに起動されることになる．

レンダラは，DICOM ファイルからボリュームデータを読み込んで，3 次元モデルを生成してから 2 次元画像を生成する．これら一連の画像生成処理ではノードに高い負荷がかかる．複数のレンダラが 1 つのノードに集中して起動した場合，そのノードに高い負荷がかかり，画像生成の遅延や失敗を招く可能性がある．16 台のノードに均等にレンダラを起動し，レンダラのプロセスを分散することで，1 つのノードに集中してレンダラが起動するのを防ぐ．また，起動されたレンダラは各ノードに常駐するため，起動したレンダラをそのままにしておくと，大量のプロセスによってレンダリングサーバの CPU やメモリが占有されてしまい，同様に画像生成の遅延や画像生成の失敗を招く可能性がある．そこで，不必要なレンダラのプロセスを定期的に削除し，ノードにかかる負荷を軽減しなければならない．

レンダリングに伴う負荷を可能な限り軽減するために，レンダリングサーバにおける負荷分散方法と負荷軽減方法をそれぞれ設計・実装した．負荷分散と負荷軽減により，レンダリングサーバの性能を最大限に活かし，安定，かつ，高速な画像生成処理を実現する．

5.1.1. レンダラリングサーバにおける負荷分散

ノードへのリクエストを分散させることで，各ノードにかかる負荷を分散し，レンダラの画像生成処理を確実なものとする．GPU クラスタには，計 16 台のノードが存在するため，これらのノードにリクエストを分散させる．DNS ラウンドロビン方式[9]や，各ノードの作業量・負荷のモニタリング[10]を考慮に入れて，GPU クラスタ環境におけるノードの負荷分散手法について検討を進めた．

本サービスでは，1 つのノードにリクエストが集中するのを防ぐため，ラウンドロビン方式の考え方を基本とした負荷分散手法を用いる．ラウンドロビン方式は主に Web サービスで用いられている負荷分散手法である．一般的なラウンドロビン方式では，同様の処理が可能なサーバを複数用意し，利用可能なサーバ群の中から，サーバを順番に選択しリクエストを割り振る．複数のサーバに順番にリクエストを振り分けることで，1 つのサーバに処理が集中することを防ぐ．

ラウンドロビン方式でノードを選択してレンダラを起動することで，1 台のノードにレンダラが集中して起動しないようにする．図 10 のように，16 台のノードを順番に 1 つずつ選択し，レンダラを起動していく．16 台目のノードでレンダラを起動したら，再び 1 番目のノードに戻り，16 番目まで順番にレンダラを起動する．

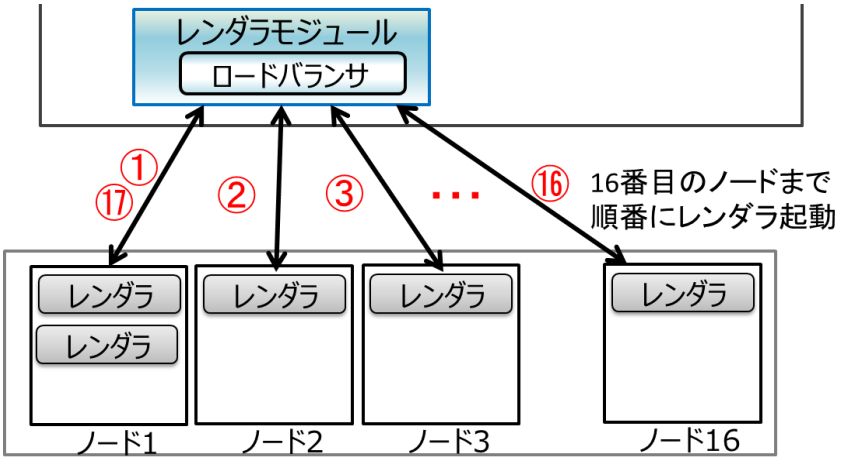


図 10　レンダラー起動の様子

レンダラの起動は，基本的にラウンドロビンに基づいた負荷分散処理によって行う．しかし，ラウンドロビン手法を用いただけでは，均等な負荷分散を実現できない．

例えば，ノード 3 のレンダラに対して削除処理が実行され，図 11 のような状態になったとき，負荷が均等に分散されなくなる．この状態で，レンダラを順番に起動し続けた場合，3 番目のノードのみレンダラが 1 つ少なくなり，負荷が均等に分散されない．この場合は，レンダラの起動数が少ないノードを選択し，優先的にレンダラを起動する．

各ノードにおけるレンダラの起動数を考慮したラウンドロビン方式により，均等な負荷分散を実現する．この方法では，レンダラ起動数が均等である場合は，ラウンドロビン方式で順番にレンダラを起動し，レンダラ起動数の少ないノードがある場合は，そのノードでレンダラを起動する．

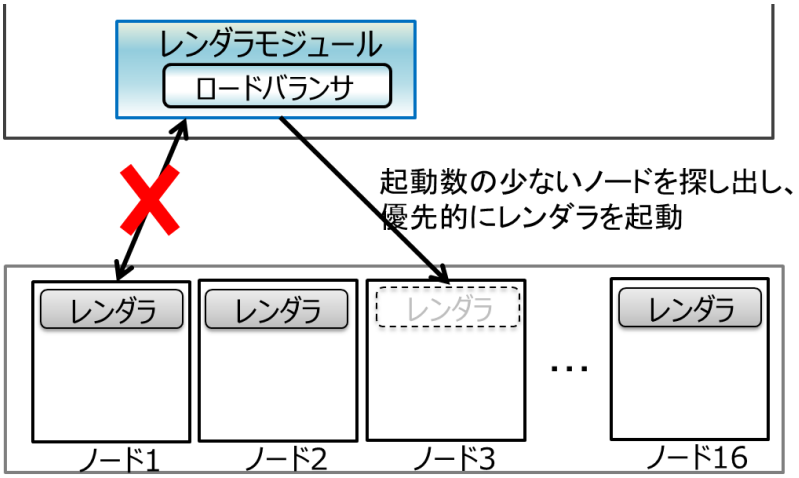


図 11　起動数の少ないノードの選択

5.1.2. レンダラの削除による負荷軽減方法

レンダラモジュールによって起動されたレンダラは，リクエスト待ち受け状態となりノードに常駐する．レンダラの画像生成処理はメモリや CPU を使用し，ノードに負荷をかける．リクエスト待ち受け状態のレンダラも，ノードのメモリを大幅に占有し，CPU もわずかながら占有している．そのため，大量のレンダラが起動していると，ノードに負荷がかかり，画像生成処理の遅延や失敗を招く．レンダラモジュールでは，起動しているレンダラを定期的に削除することで負荷軽減をはかる．レンダラを定期的に削除することで，起動しているレンダラの総数を一定数以下に保つ．

レンダラの起動には数秒の時間を要する．頻繁にアクセスされているレンダラを削除してしまった場合，もう一度レンダラを起動させなければならず，待ち時間が発生してしまう．一方，レンダラは一度起動してしまえば，画像生成処理にはそれほど時間がかからない．不要な待ち時間を発生させないために，レンダラを可能な限り長時間起動させたままにしておくことが望ましい．レンダラを削除する際は，レンダラ起動による待ち時間の発生を防止した上で，資源を有効活用できるように考慮しなければならない．

レンダラモジュールでは，レンダラが一定時間リクエストを受信していない場合と，常駐しているレンダラの総数が一定以上になった場合にレンダラを削除する．一定時間以上リクエストが受信されていないレンダラを削除することで，ユーザが頻繁にアクセスしているレンダラは削除されないため，待ち時間の発生を防ぐことができる．以下の式①，式②を用いてレンダラの削除判定を行う．

$$(\text{最後にアクセスされた日時}) - (\text{スクリプト実行時の日時}) > T \quad \cdots\text{式①}$$
$$(\text{起動しているレンダラの総数}) > N \quad \cdots\text{式②}$$

式①は，一定時間リクエストを受信していないレンダラを削除するための判定式である．定期的に専用のスクリプトを実行することで，一定時間リクエストを受信していないレンダラの存在を確認する．式①の T は，スクリプト実行日時とレンダラが最後にリクエストを受信した日時の差分である．最後にリクエストを受信した日時から T 分以上経過していたレンダラが存在していた場合，該当するレンダラを削除する．T は 30 と設定し，30 分以上アクセスのないすべてのレンダラを削除する．スクリプトの実行内容については付録 D に示す．

式②は，レンダラの総数を一定以下に保つための判定式である．式②の N は，レンダラの

起動上限数である．レンダリングサーバで起動しているレンダラの総数が N を越えたときにレンダラの削除処理が行われる．レンダラを起動する際に起動しているレンダラの総数をカウントし，もし式②を満たしているようであれば，レンダラを 1 つ削除する．削除の際は，最もアクセス日時の古いレンダラを削除する．アクセスされた日時が最も古いレンダラを削除することで，不要な待ち時間を発生させない．

5.1.3. レンダラへのリクエストの送信

レンダラを起動しただけでは，コア試料の画像は生成されない．レンダラは，画像生成リクエストを受信した時点で初めて画像描画処理を行う．レンダラに対する画像生成リクエストの送信は，ソケット通信で行う．また，レンダラを削除する際の終了メッセージも，このソケット通信を介して行われる．

ソケット通信は，IP アドレスとポート番号の組みを決めることで通信相手を選択する．レンダラとソケット通信する際は，ノードの IP アドレスと専用のポート番号を用いて行う．レンダリングサーバの 16 個のノードには異なる IP アドレスが振られているため，レンダラに異なるポート番号を割り振ることで，ある 1 つのレンダラに対して正確にソケット通信でリクエストを送信することができる．ソケット通信で使用きるポート番号は 1～65535 番まで確保されている．しかし，すべてのポート番号が使用可能なわけではない[11]．特定のポート番号とそのポート番号を用いるアプリケーションの組み合わせを表 7 に示す．本システムでは，プライベートなポート番号 50000～65535 番までのポート番号をレンダラに順次振り分けることで，ソケット通信を行う．

表 7　ポート番号

ポート番号	割り当てられているサービス
1～1023	Well known ポート
1024～49151	予約済みポート番号. 既にアプリケーションによる使用が決まっているポート番号.
49151～65535	プライベートなポート番号. ユーザが自由に使用できるポート番号. 本サービスではこのポート番号を使用する.

レンダラへのポート番号の割り振りは，レンダラモジュールで指定し，管理する．レンダラにポート番号を割り振る様子を図 12 に示す．ノードにすでに別のレンダラが起動している場合，起動しているレンダラと異なるポート番号を割り振る．重複しないポート番号をレンダラに割り振ることで，1 つのノードに複数のレンダラを起動している場合においても，ポート番号で通信相手を識別できる．

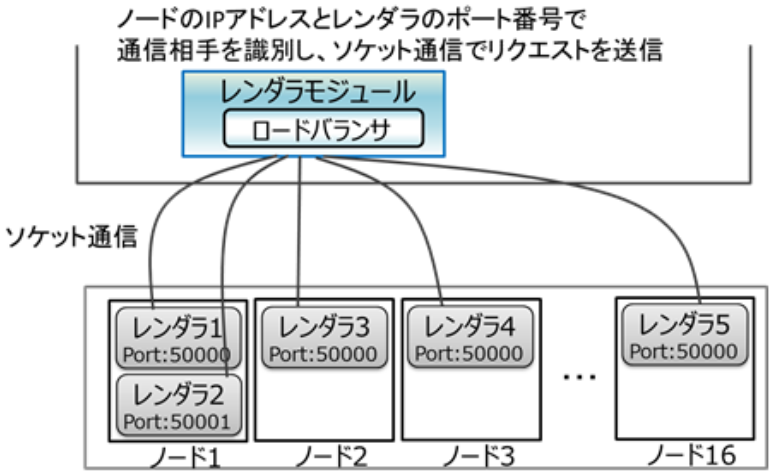


図 12　複数のレンダラとのソケット通信の様子

通信方式には，コネクション型とコネクションレス型が存在する．コネクション型では，データが正しく送信されているかを通信相手と常に確認し合いながら通信を行う．常に確認し合いながらデータ通信を行うことで，データの欠損を防ぎ，送信相手に確実に正しいデータを届ける．コネクションレス型の通信では，通信相手にデータが届いたことを確認せずに一方的に送る．そのため，データの欠損や，正しいデータが届いたかどうかを確認できない．コネクション型では，通信の信頼性を確保できるものの大量のデータの送信には向かない．コネクションレス型では，信頼性は落ちるものの大量のデータを送信することができる．

コネクションレス型では，ソケット通信の失敗を検知できないため，データの送信に失敗した場合，ユーザが再度リクエストを送信しなければならない．ソケット通信が失敗する度に，ユーザ自身でリクエストを再度送信するのは非常に手間である．レンダラモジュールとレンダラ間でやりとりするデータ量は少ない．以上の理由から，コネクションレス型は適していないことがわかる．レンダラモジュールとレンダラ間のソケット通信は，コネクション型の通信プロトコルである TCP による通信で行う．

5.2. 方針に基づいたレンダラモジュールの実装

節 5.1 で述べた処理方針をもとにレンダラモジュールを PHP で実装した．本節では，どのようにレンダラの起動状況の記憶し，5.1 節で述べた方法に基づいてどのようにレンダラを起動・削除しているかを述べる．まずは，レンダラの起動状況を記憶する方法として実装した「起動ログ」と，その起動ログを用いた，負荷分散と負荷軽減を考慮したレンダラの起動・削除，レンダラへのリクエスト送信方法について述べる．次に SSH を用いたレンダラの起動と，ソケット通信を用いたレンダラへのリクエスト送信方法について述べる．レンダラモジュールの設計書と詳細な処理内容については付録 E で示す．

5.2.1. レンダラの起動状況の記憶

レンダラの起動状況を記憶は「起動ログ」によって行う．起動ログは，起動したレンダラの各種情報を記述したファイルである．レンダラ起動時には，起動ログにレンダラの情報を追記し，レンダラ削除時には，該当するレンダラ情報を削除する．レンダラが起動していない場合は，起動ログは何も記載されていないファイルとなる．起動ログには，起動中のレンダラに関する情報を 1 行ずつ記述する．起動ログの例を図 13 に示す．レンダラに関する情

報として，コア試料番号，ノード番号，ポート番号，アクセス日時の 4 つの項目をコンマ区切りで記述する. この例は, レンダラが 3 つ起動しているときのログファイルの状態である.

コア試料のファイルパス,ノード番号,ポート番号,アクセス日時

起動ログの例

```
/home/ccore/coredata/1/314/C0001B/314-C0001B-1H-1.20071118T080602,
0,40001,2013/01/07 11:48
/home/ccore/coredata/1/314/C0001B/314-C0001B-1H-CC.20071118T082724,
1,40001,2013/01/07 11:48
/home/ccore/coredata/1/316/C0006C/316-C0006C-1H-1.20071228T200026,
2,40001,2013/01/07 11:49
```

図 13　起動ログ

コア試料番号には，レンダラ起動の際に指定したコア試料のファイルパスを記述する．ノード番号は，16 個あるノードの内，どのノードでレンダラが起動しているかを 0～15 の番号で記述する．ポート番号は，レンダラ起動の際に指定したポート番号を記述する．ポート番号は，レンダラにソケット通信でリクエストを送信する際に用いる．アクセス日時には，レンダラにリクエストを送信した最新の日時を記述される．アクセス日時は，レンダラを削除する際の指標として用いる．

起動ログは，負荷分散と負荷軽減を考慮したレンダラの起動・削除，レンダラへのリクエストの送信を実現するのに十分な情報を持っている．レンダラモジュールは，起動ログからレンダラの各種情報を読み込むことで，負荷分散と負荷軽減を考慮したレンダラの起動・削除，レンダラへのリクエストの送信を実現できる．

レンダラを起動する直前に，起動ログに記述されたコア試料番号を確認することでレンダラが起動しているかを判定する．起動ログに該当するコア試料が記述されていない場合は，レンダラが起動していないものと判断し，ノードにレンダラを起動する．レンダラを起動する際は，起動ログのノード番号を読み込み，レンダラの起動数を考慮したラウンドロビン方式にもとづいて起動する．

レンダラが起動したら，起動ログに記載されているノード番号とポート番号を用いてソケット通信でレンダラに画像生成リクエストを送信する．レンダラに対してソケット通信でリクエストを送信する度に，レンダラのアクセス日時を更新する．リクエストの度に，アクセス日時を更新することで，頻繁にアクセスされているレンダラかどうかがわかる．

アクセス日時は，主に，レンダラ削除による負荷軽減を行う際に用いる．一定時間リクエストを受信していないレンダラかどうかを判定する際には，起動ログに記されたアクセス日時を用いて判定する．起動ログには，コア試料の番号とノード番号が記載されているため，起動ログをすべて読み込むことで，各ノードで起動しているレンダラの総数を算出することができる．起動しているレンダラの総数が，一定以上である場合，負荷軽減のためにレンダラを削除する．その際，それぞれのレンダラのアクセス日時を比較して，アクセス日時の最も古いレンダラを 1 つ選択して削除する．

5.2.2. レンダラの起動処理

レンダラモジュールから，レンダリングサーバのノードに対して専用のコマンドを実行することで，レンダラを起動できる．以下の 2 つのコマンドを実行することで，レンダラを起動することができる．2 つのコマンドはいずれも常駐プログラムであるため，バックグラウンド動作を指定する．

- Xvfb：*ディスプレイ番号* –screen 0 512x512x24
- ./renderer –display：*ディスプレイ番号* –sendport *ポート番号* –recvport *ポート番号* –dir *コア試料のファイルパス*

レンダラを起動する際には，Xvfb というソフトウェアを起動する．Xvfb は，仮想的なディスプレイを作るためのソフトウェアである．Xvfb で仮想的なディスプレイを作り出し，そのディスプレイにレンダラの描画結果を出力する．そして，その画像をキャプチャリングすることで，コアデータの画像を生成している．そこで，レンダラを起動する際には，まず Xvfb を起動してから，レンダラを起動する．

レンダラを 1 つ起動するごとに Xvfb を 1 つ起動する．複数のレンダラに対して Xvfb を 1 つだけ起動した場合，複数の描画結果が 1 つの仮想的なディスプレイに出力されてしまう．その結果，複数の出力が衝突してしまい，画像生成処理が失敗してしまう可能性がある．画像生成処理を確実なものとするために，複数のレンダラが起動されたときは，複数の Xvfb を起動する．

レンダラを起動する際には，ディスプレイ番号，ソケット通信で用いる送信用のポート番号と受信用のポート番号，画像を生成したいコア試料のファイルパスの 4 つのオプションを指定する．4 つのオプションをレンダラモジュールで設定した上でコマンドを実行する．レンダラを起動するノードが決定した後に 2 つのコマンドを実行することで，あるノードに対してレンダラを起動することができる．

ポート番号は，起動ログに基づいて設定する．ポート番号は 50000～65535 の番号を順次割り振る．しかし，このとき同一ノードで起動するレンダラに同じポート番号を割り振らないようにしなければならない．項 5.1.3 で述べたように，ノードに起動している複数のレンダラに異なるポート番号を割り振ることで，正しいレンダラに対して確実にリクエストを送信することができる．レンダラに異なるポート番号を割り振るために，コマンドを実行する前に起動ログを読み込み，すでに使われているポート番号を調べる．すでに使われているポート番号がわかったら，その番号を除く 50000～65535 の番号を順次割り振る．

ディスプレイ番号も，ポート番号と同様に異なるディスプレイ番号を割り振らなければならない．今回の実装では，ディスプレイ番号は以下の式から導出する．

$$(\text{ディスプレイ番号}) = (\text{ポート番号の下3桁}) + 500 \quad \cdots\text{式③}$$

レンダラごとに異なる番号を割り振られたポート番号を利用することで，レンダラごとに異なるディスプレイ番号を割り振る．

オプションを設定した 2 つのコマンドを SSH でリモートに実行することで，レンダリングサーバのノードに Xvfb とレンダラを起動する．レンダラ起動処理は，Xvfb とレンダラを起動する 2 つのコマンドの実行が成功した時点で完了とみなす．その時点で，起動ログにオプションに指定したポート番号とコア試料のファイルパスとレンダラを起動したノードの番号を新たに追記する．PHP では，SSH によるリモートのコマンドの実行を SSH2 モジュールで行う．SSH2 モジュールの詳細については付録 F で示す．

5.2.3. ソケット通信によるレンダラへのリクエスト送信

起動したレンダラへのメッセージの送信は，すべてソケット通信で行われる．レンダラに

送信されるメッセージには，画像生成を依頼するメッセージとレンダラの終了処理を依頼すするメッセージの 2 種類のメッセージがある．ソケット通信を行う際には，起動ログに記載されているノード番号とポート番号を用いる．ソケット通信で送信するパケットはいずれも¥¥を区切り文字としたものを送信される．

画像生成を依頼するメッセージは，レンダラ API で受け取った値をもとに生成される．レンダラモジュールでは，生成したメッセージを送信した後，パケット待ち受け状態へと遷移する．レンダラでは受け取ったメッセージに基づいてコア試料の画像を生成する．レンダラは, コア試料の生成が終了した時点で, “OK¥¥”の文字列をレンダラモジュールに送信する．レンダラモジュールは“OK¥¥”の文字列を受け取った時点で画像生成処理が成功したものとみなし，クライアントアプリケーションへと画像情報を返す．

レンダラの終了処理を依頼するメッセージは，専用の終了メッセ―ジがパケットとして生成される．”exit¥¥”の文字列を送信することでレンダラの終了依頼をする．”exit¥¥”の文字列が含まれたパケットそ受信したレンダラは終了処理を実行し，プロセスを終了する．レンダラモジュールは，”exit”の文字列を送信した時点でレンダラの削除処理が完了したものとみなし，起動ログから該当するレンダラの行を削除する．

5.3. まとめ

負荷が均等になるようにユーザからのリクエストを分散し，レンダリングサーバを最大限利用するために，レンダラモジュールを実装した．

コア試料の 2 次元画像を生成することのできるレンダラは，1 つのコア試料につき，1 つ起動する．レンダラの仕様上，大量のユーザによって，複数のコア試料が閲覧されている場合, 大量のレンダラが起動することとなる. 大量のレンダラが同時に画像生成処理を行うと，レンダラが起動しているノードに大きな負荷がかかる．その結果，レンダリングエンジンの能力を最大限活用することができなくなる．

レンダラモジュールでは，ノードにかかる負荷を考慮し，レンダラの起動と削除を行う．レンダラの起動数が少ないノードにレンダラを起動することで，1 つのノードにリクエストが集中するのを防ぐ．加えて，レンダラの起動可能数を設定することで，大量のレンダラで同時に画像生成処理が行われるのを防ぎ, サービス全体にかかる負荷を可能な限り軽減した．また，一定時間画像生成処理を行っていないレンダラを削除することで，不必要なプロセスによるノードへの負荷をなくした．

レンダラモジュールの実装により，ノードにかかる負荷を可能な限り軽減したレンダラの管理を可能とし，本サービスを安定して提供することに成功した．

6. Web アプリケーション

本章では，Web アプリケーションの設計と，実装した Web アプリケーションについて述べる．設計では，実装する際に考慮した点と，本 Web アプリケーションがサポートするブラウザについて述べる．その後，実際に実装した Web アプリケーション画面にもとづいて実装した内容について述べる

6.1. Web アプリケーションの設計

ユーザの操作性を考慮して Ajax を用いて UI を実装する．Ajax は，Asynchronous JavaScript ＋ XML の略であり，JavaScript とデータ構造記述言語である XML を用いることで，動的な Web アプリケーションを構築する手法のことである．本 Web アプリケーションでは，データ構造言語として JSON を用いることで，データ構造を単純なものとし，通信するデータ量を削減する．Ajax により非同期にレンダラ API や検索 API にリクエストを送信することで，操作の度に発生する画面遷移やページ全体の更新を最小限に抑え，操作性の向上をはかる．

しかし，Ajax で用いられる JavaScript には問題点がある．JavaScript は，ブラウザによって異なる解釈をされてしまう可能性があり，ブラウザによって異なる挙動を示すことがある．JavaScript を用いた Web アプリケーションを複数のブラウザに対応させたい場合には，各ブラウザに対応した JavaScript を複数記述しなければならず，開発に余計な時間を費やすこととなる．ブラウザ間の違いを解消することのできる JavaScript のフレームワークとして jQuery が存在する[12]．jQuery は，JavaScript のブラウザ間の解釈の違いに対応しており，開発者はブラウザの違いを意識せずに JavaScript による実装ができる．jQuery では，Ajax を実現するライブラリも含まれている．可能な限り jQuery で JavaScript を記述することで，開発に費やす時間を削減する．

jQuery だけでは，Ajax による動的な操作コンポーネントの実現は難しい．動的な操作コンポーネントを提供する jQuery のライブラリとして，jQuery UI が存在する[13]．jQuery UI を用いることで，ボタンやスライダーなどの UI を簡単に配置し，動的な操作コンポーネントを実現できる．jQuery と jQuery UI を用いることで，Web アプリケーションの開発に費やす時間を可能な限り削減する．

本サービスでは，複数のブラウザをサポートする．本サービスでサポートする Web ブラウザは，IE，FireFox，Google Chrome，Safari の 4 つのブラウザとする．NET MARKET SHARE の調査によると，2012 年 12 月時点で，Opera を初めとするその他のブラウザのシェアが 1%以下であるため，これらの Web ブラウザはサポート対象外とする[14]．

Web アプリケーションは，X 線 CT スキャンデータを配布しているページである Virtual Core Library で公開する．Virtual Core Library は，JAMSTEC が公開する Web ページであるため，公開の際には，可能な限り JAMSTEC 側の作業量を減らすことができるように Web アプリケーションを実装する．

6.2. 実装した Web アプリケーションと公開方法

開発した Web アプリケーションを図 14 に示す．なお，Web アプリケーションの詳細なファイル構成などは，付録 G.1 に示す．Web アプリケーションでは，3.2 節で示した①検索機能，②分析機能，③CT 値に対する色づけ機能の 3 つの機能を，1 つの画面に集約している．UI による操作を行うと，非同期に各種 API にリクエストが送信され，コア試料の検索結果やコア試料の画像を取得できる．API からのレスポンス待ち状態のときは，図 15 のようなローディング画面を表示し，一切の操作を不可能とする．ボタンの連打などで大量にリクエストが送信された場合に予期せぬ動作をする場合がある．一切の操作を不可能とすることで，リクエストの大量送信による発生する予期せぬ動作を防ぐ

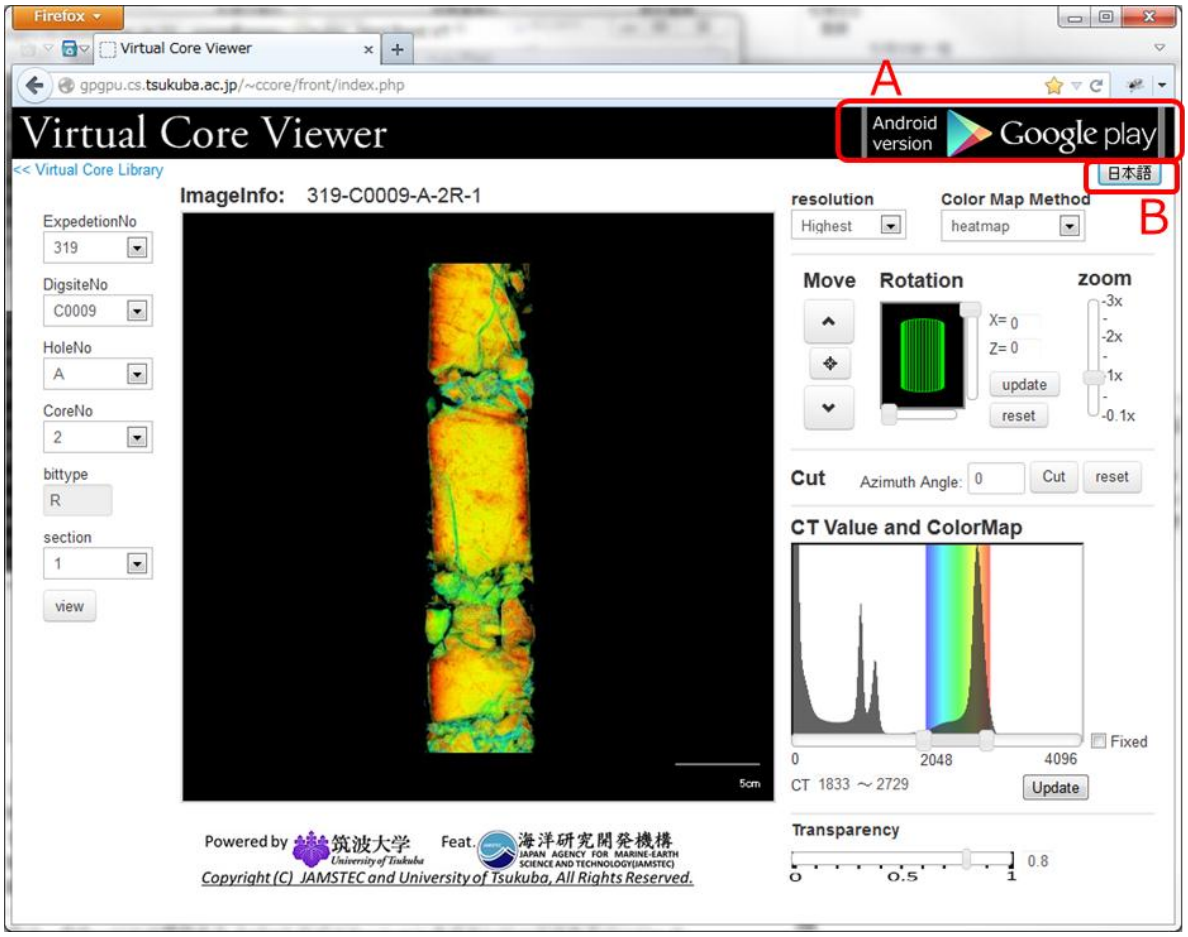


図 14　Web アプリケーション向け Virtual Core Viewer

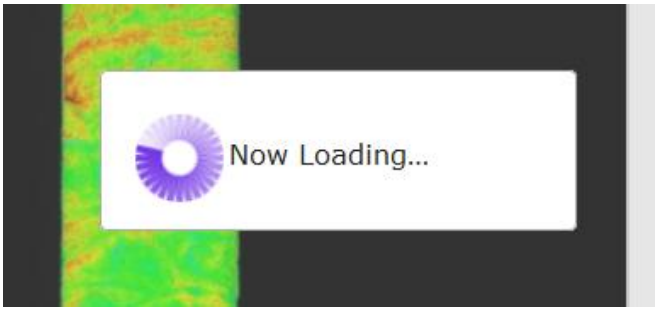


図 15　ローディング画面

①検索機能，②分析機能，③CT 値に対する色づけ機能の 3 つの機能に加えて，言語選択機能，Android 版 Virtual Core Viewer へのリンクを実装した．

図 14 中の A に示すバナーは，本サービスで提供する Andoid アプリケーションをダウンロードできるページへのリンクとなっている．リンク先は，Google play の本サービスの Android アプリケーションの紹介ページとなっている．Google Play では，アプリケーションのダウンロードが可能となっている．

図 14 中の B に示す言語選択ボタンは，Virtual Core Viewer の表示言語を切り替えるためのものである．選択可能な言語は英語と日本語の 2 つの言語のみである．Virtual Core Viewer の主なユーザは，英語を主言語としているため，デフォルトの表示言語は英語となっている．

開発した Virtual Core Viewer は，JAMSTEC 所有の Web ページである Virutual Core Library を介して 2 種類の方法で公開する．2 種類の公開方法に合わせて，図 14 に示した Web アプリケーションを含む 2 種類の Virtual Core Viewer を開発した．

1 つ目の方法は，Virtual Core Library のトップページから公開する方法である．図 16 のページは Virtual Core Viewer のトップページである．このページに本アプリケーションへのリンクを設置し，そのリンクをクリックすることで，図 17 に示した Virtual Core Viewer の画面に遷移することができる．

図 16 Virtual Core Library トップページ

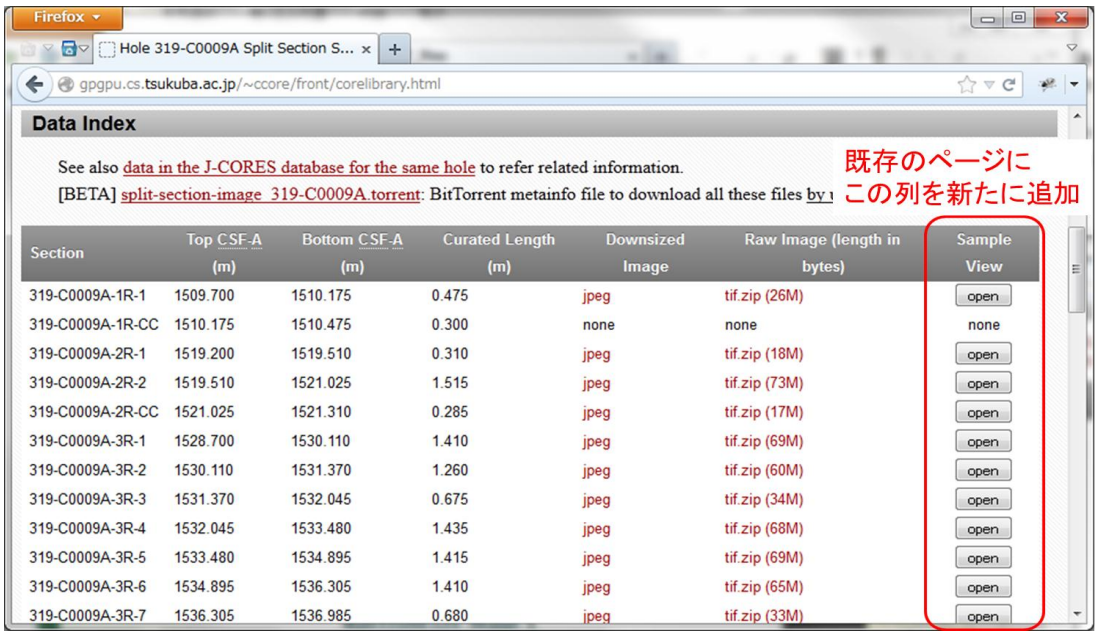


図 17　X 線 CT データの配布ページ

図 17 は，Virtual Core Library の X 線 CT データを配布しているページである．先ほどの Virtual Core Library のトップページにある「XCT files」のリンクからこのページに遷移することができる．このページには，ダウンロード可能なコアデータの一覧表が表示されている．Raw Image の列にある赤文字のリンクをクリックすることで，X 線 CT データを自身の端末にダウンロードすることができる．

2 つ目の公開方法は，コアデータの一覧表に Virtual Core Viewer へのリンクを設置することで公開する．コアデータの一覧表の各行に，図 17 の赤枠で示すようにリンクをボタンとして設置する．

赤枠で囲まれた Sample View の行は，スクリプトを用いることで自動生成する．Virtual Core Library を公開しているサーバに，作成したスクリプトを設置し，該当ページでスクリプトを読み込ませることで Sample View の行が追加される．スクリプト内では，一覧表の Section の行と Raw Image の行を読み込んでいる．Section の行に記されたコア試料のセクション番号に基づいて，Virtual Core Viewer で該当するコア試料を表示する．Raw Image の項目が none になっている場合，そのコア試料に関してリンクを設置しない．スクリプトでリンクを自動的に追加することにより，ページの公開者がリンク先のページの仕様を気にすることなく，リンクを配置することが可能となり，Web アプリケーション公開の際に JAMSTEC 側が負担する作業量を削減した．

このボタンを押下することで，図 18 のような画面が新しいウィンドウで表示される．図 18 は，319-C0009A-2R-2 の行のリンクをクリックした場合の画面である．このページから Virtual Core Viewer にアクセスする場合は，図 19 のように複数のコアデータを同時に表示することができる．しかし，同時に表示できるのは異なるコアデータだけであり，同じコアデータを複数のウィンドウで表示することはできない. この方法で公開される Web アプリケーションは検索機能を提供しない．2 つ目の方法では．あらかじめコアデータが選択された状態で Virtual Core Viewer にアクセスするため, 検索機能は不要であると判断した．また，検索機能を提供しないことで画面サイズを縮小することができるため，複数のコア試料を並べて観察しやすくなる．

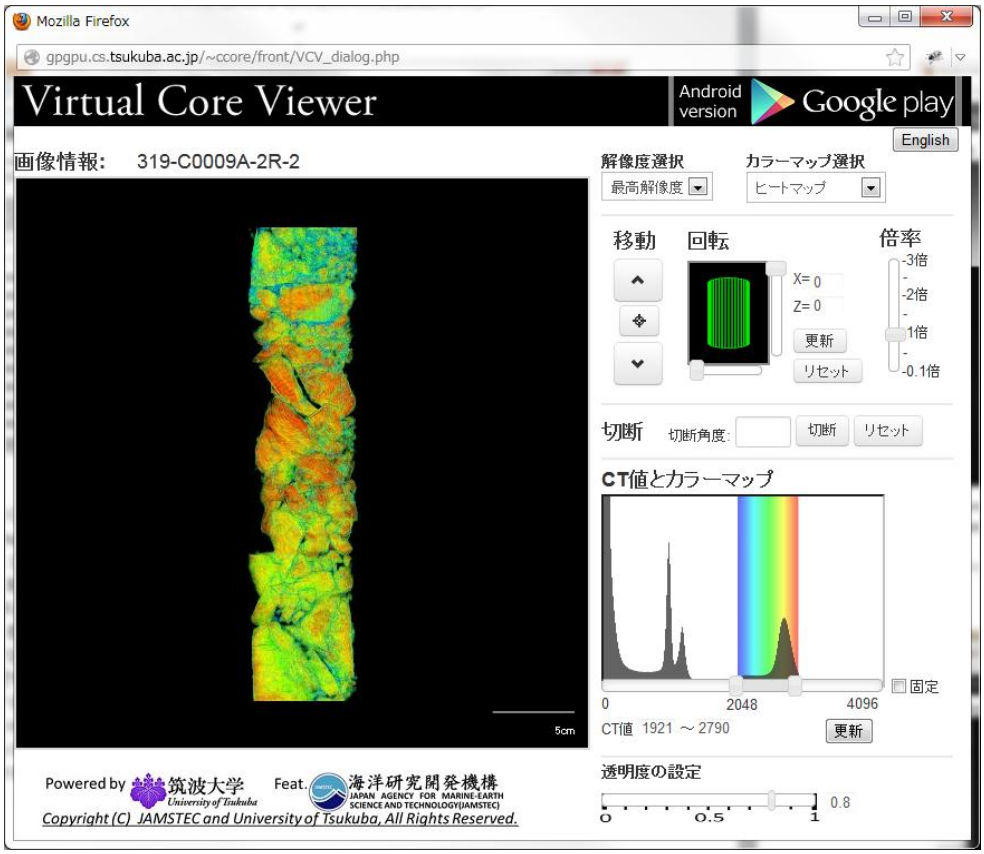


図 18　コア試料一覧から選択した場合のインタフェース

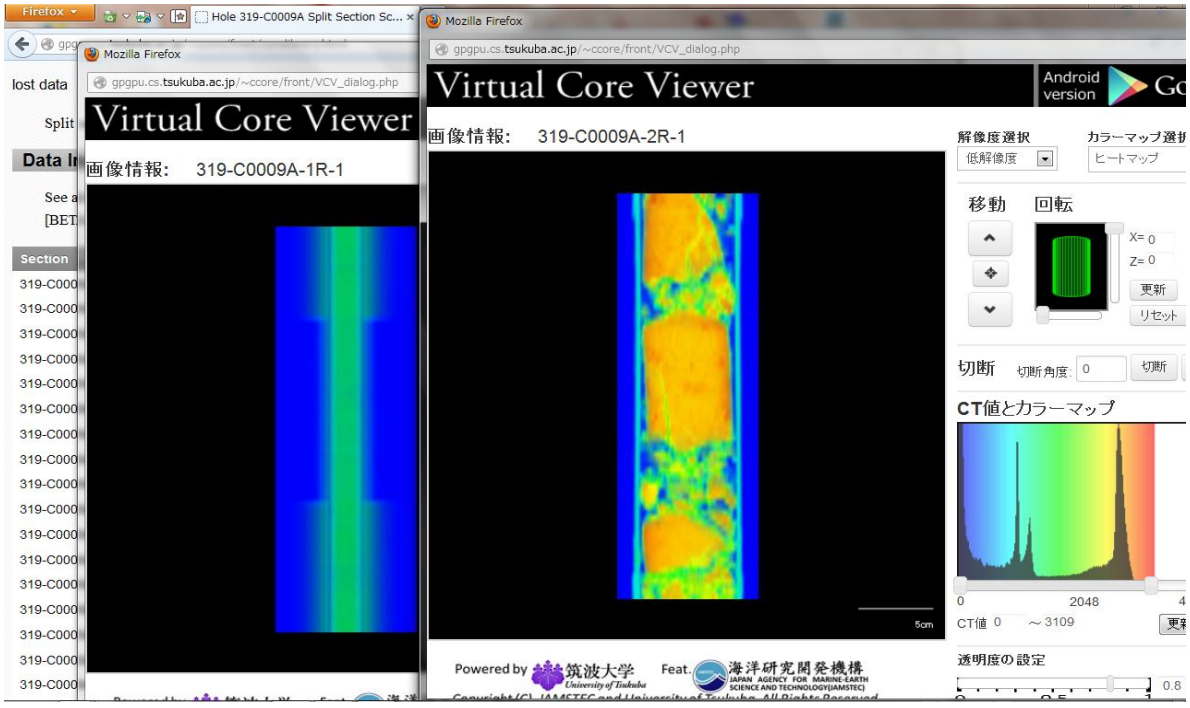


図 19　Virtual Core Viewer の複数起動したときの様子

6.3. まとめ

本サービスでは，コア試料を観察するための機能を有するインタフェースを提供するために，Web アプリケーションと Android アプリケーションの 2 つの方法でクライアントアプリケーションを提供する．著者は，2 種類あるクライアントアプリケーションの内，Web アプリケーションを実装した．

Web アプリケーションでは，コア試料を観察するために必要な機能を提供する．Web アプリケーションの UI は，顧客のフィードバックにもとづき，コア試料の観察が可能となる UI を実装した．ユーザの操作性を向上させるために，Ajax による動的な UI を実装する．画面遷移やページの更新を発生させずにコア試料を観察させることが可能となり，円滑にコア試料を観察することができる．Web アプリケーションは利用率の高い IE, FireFox, Google Chrome, Safari の 4 つのブラウザを対応ブラウザとする．

開発した Web アプリケーションは，JAMSTEC 所有の Web ページである Virtual Core Library を介して，2 種類の方法で公開した．それぞれの公開方法に対応した 2 種類の Web アプリケーションを実装した．

7. まとめ・今後の発展

本プロジェクトでは，「海底コアCT スキャンイメージ可視化のためのクラウドサービス」を開発した．

著者は，本プロジェクトにおいて，検索機能を提供するミドルウェアと，レンダラモジュールの実装，Web アプリケーションの実装を担当した．

Virtual Core Viewer は, 本サービスで提供するコアデータの観察に特化したクライアントアプリケーションである. Web ブラウザと Android 端末向けに同名のクライアントアプリケーションをリリースした．著者は，Web ブラウザ向け Virtual Core Viewer の設計・実装を担当した．Web ブラウザ向け Virtual Core Viewer では，コア試料を閲覧するための機能を 1 つの画面で提供することで, 操作性の向上を図った．Web アプリケーションとして Virtual Core Viewer を実装することで, 一般的な Web ブラウザから本サービスを利用することが可能となる．Android 端末向けの Virtual Core Viewer の設計・実装は，佐々木氏が担当した．Android 端末向けの Virtual Core Viewer を図 20 に示す．

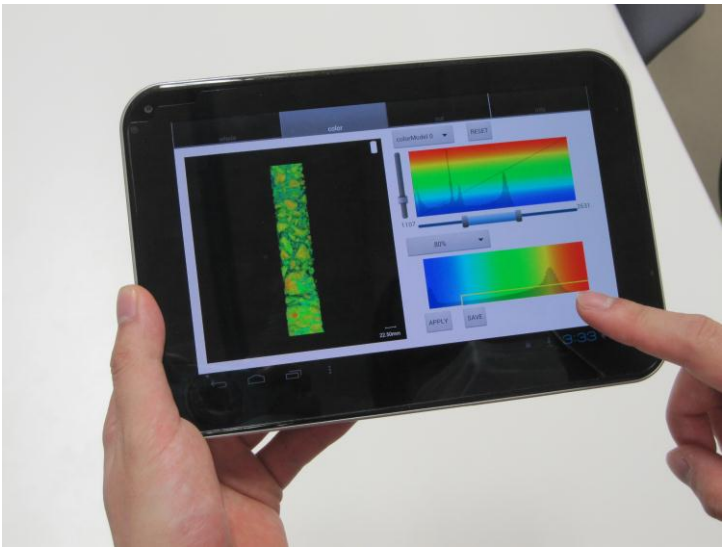


図 20　Android 端末向け“Virtual Core Viewer”

著者が開発を担当した検索機能を提供するミドルウェアでは，4 千セクション以上存在するコアデータの中から，1 セクションのコアデータを探し出す機能を提供する．効率的に閲覧したいコア試料を探し出せるよう絞り込み検索機能を実現した．絞り込み検索機能では，航海番号や掘削サイト番号などの情報を選択することで, コア試料を探し出すことができる．

コア試料の 2 次元画像はレンダラによって生成される. レンダラは, コア試料の DICOM スライス画像から 2 次元画像を生成するソフトウェアであり，ユーザからのリクエストにもとづいてコア試料の 2 次元画像を生成する．クライアントアプリケーションは，レンダラ API を介してレンダラに画像生成を依頼する．レンダラとレンダラ API はいずれも坂本氏が開発を担当した．

著者はレンダラの起動・削除を行うレンダラモジュールの設計・実装を担当した．レンダラの画像生成処理は，レンダリングサーバに大きな負荷をかける．そこで，レンダリングサーバにかかる負荷を考慮したレンダラモジュールを設計・実装した．レンダリングサーバの負荷分散と負荷軽減の2つの方法を用いることによって，レンダリングサーバの性能を最大限

に引き出し，安定したサービスを提供することに成功した．

本サービスの提供によって，これまで利用が難しかった掘削コア試料のCTスキャンイメージの利用が容易となる．CTスキャンイメージはDICOM形式で保存されており，DICOM形式のデータを処理するには端末に大きな負荷がかかる．CTスキャンイメージの処理をすべてリモートのサーバで行うことで，ローカル端末の性能を気にすることなく掘削コア試料のCTスキャンイメージを利用できるようになった．本サービスの普及により，CTスキャンイメージの活用を促進し，地球科学分野における研究の更なる発展が期待できる．

本サービスは，医用画像への応用が可能である．レンダラは，DICOM 形式のファイルであれば，掘削コアデータ以外のデータも 2 次元画像として生成することができる．MRI スキャン画像や CT スキャン画像などの医用画像は DICOM 形式であるため，レンダラで 2 次元画像に変換することができる．図 21 は実際に医用画像をレンダリングした画像である．左図がる脳の MRI スキャン画像であり，右図が腕の CT スキャン画像である．

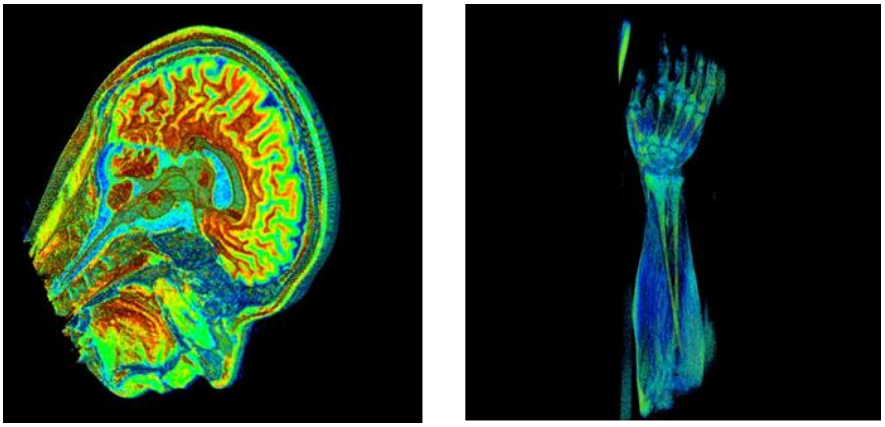


図 21　本サービスでレンダリングした医用画像

本サービスでは，コア試料を観察するためのクライアントアプリケーションのみを実装したが, 医用画像を閲覧するためのクライアントアプリケーションと専用の Web API を追加実装することで，端末性能に依存せずに医用画像を閲覧できる．

従来は，DICOM ファイルを個人端末にダウンロードすることで閲覧していた．これに対して，本サービスでは同じ DICOM ファイルを複数のユーザで共有して閲覧している．つまり，本サービスによって DICOM ファイルの利用形態は，個人使用から共有使用へと変化する．サーバ上の DICOM ファイルに対して，注釈や記号などのアノテーションを付与することで，本サービスを利用するすべてのユーザ間でアノテーションによる情報交換を行うことができるようになる．アノテーション機能に加えて，DICOM ファイルを扱う地球科学分野や医療分野の関係者で SNS を構築することによって，インターネット上で DICOM ファイルを閲覧しながら議論することが可能となる．これらの機能を構築することで，DICOM ファイルの活用をより促進し，各分野における研究価値が高まることが期待できる．

謝辞

本プロジェクトの担当教員である山際伸一准教授，和田耕一教授には，大変お世話になりました．平日休日問わず，熱心な御指導，御助言を頂きました．また，様々な成果報告の機会を与えていただいたことをここに深く感謝致します．

共同研究者である高知コア研究所の久光敏夫先生には，ご多忙の中，システム開発から論文の執筆まで，本プロジェクトに関して幅広く御協力頂きました．また，高知コア研究所への出張や地質学会への参加を通して，大変貴重な体験をすることができました．ここに深く感謝致します．

インタラクティブプログラミング研究室の，田中二郎教授，三末和男准教授，高橋伸准教授，志築文太郎准教授，Simona Vasilache 助教には大変お世話になりました．特に，指導教官である田中二郎教授には，2 年間にわたる大学生活やプロジェクト活動について大変貴重な御指導をいただきました．ここに深く感謝致します．

高度 IT 人材育成のための実践的ソフトウェア開発専修プログラムの担当である中沢研也教授，山戸昭三教授には，授業に加えて，円滑なプロジェクト活動のための御助言をいただきました．2 年間，様々な御助言をいただき，支えとなっていただいたことに深く感謝致します．

並列分散処理研究室の櫻井美知代さんには，出張の手配や書類等の作成など大変お世話になりました．

プロジェクトメンバーである坂本侑一郎氏，佐々木慎氏には，プロジェクト活動から日々の生活まで広くお世話になりました．おかげさまで，1 年間，プロジェクト活動を楽しく進めることができました．

最後に，大学院で学ぶ機会を与えてくれた両親や，大学院で共に学んだ友人，そして，大学生活でお世話になったすべての方々に感謝致します．

参考文献

[1] Virtual Core Library
http://www.kochi-core.jp/VCL/

[2]Amazon Web Services
http://aws.amazon.com/jp/

[3]Google App Engine
https://developers.google.com/appengine/

[4] OsiriX Imaging software
http://www.osirix-viewer.com/

[5] M. Yakami, K. Ishizu, T. Kubo. T. Okada. and K. Togashi, ”Development and Evaluation of a Low-Cost and High-Capacity DICOM Image Data Storage System for Research,” Journal of Digital Imaging, Springer, vol.24, no.2, pp. 190-195, April 2011.

[6] F. Lamberti and A. Sanna. A streaming-based solution for remote visualization of 3D graphics on mobile devices. IEEE Transactions on Visualization and Computer Graphics, 13(2):247.260, 2007.

[7] JSON
http://www.json.org/

[8] XML
http://www.w3.org/XML/

[9] Y.S.Hong, J.H.No and S.Y.Kim, DNS-Based Load Balancing in Distributed Web-server Systems (WCCIA 2006) (2006), p. 4

[10] A.Nigan, T.Singh, A.Tiwari and A.Singhal, Adaptive Load Balacing for cluster using content awareness with taraffic monitoring,International Journal of Advanced Research in Computer Engineering & Technology, Volume 1, Issue 1, March 2012

[11] Service Name and Transport Protocol Port Number Registry
http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml

[12] jQuery
http://jquery.com/

[13] jQuery ui
http://jqueryui.com/

[14] NETMARKETSHARE -Market Share Statistics for InternetTechnologies-
http://marketshare.hitslink.com/browser-market-share.aspx?qprid=1&qpcustomb=0

付録 A：プロジェクトの進行計画

本プロジェクトで提供する機能とその開発段階の対応を A.1 に示す．スコープ S0 が 1 段階目の開発内容，スコープ S1 が 2 段階目の開発内容，スコープ S2 が 3 段階目の開発内容を示している．1 段階の開発が終わるごとに共同研究者にサービスを提供しフィードバックを得る．得られたフィードバックをもとに次段階の開発でシステムの改善を行う．最終段階であるスコープ S2 の開発が終了した時点で，開発システムを一般公開する．

A.1　開発スコープ一覧

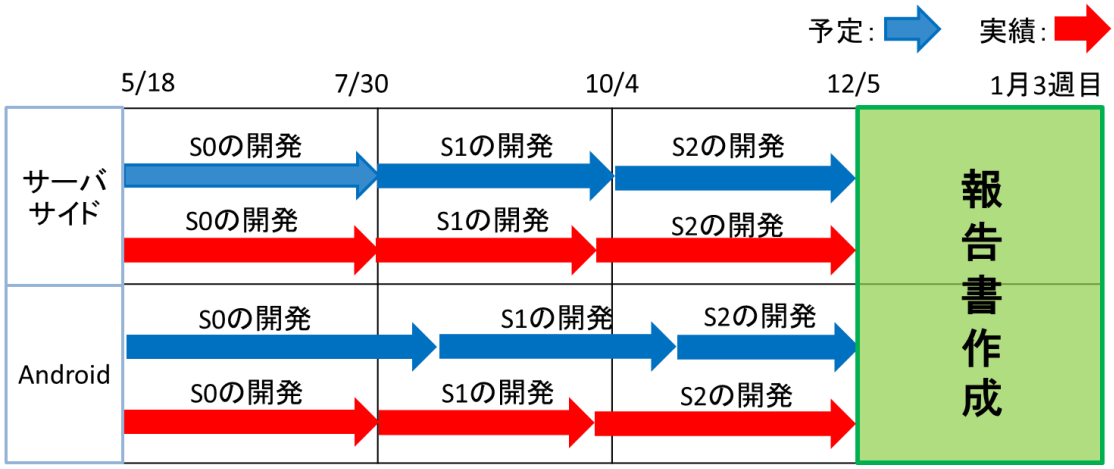
スコープ S0	スコープ S1	スコープ S2
拡大・縮小	検索機能	CT 値と色の対応の保存
コアの選択	スケールの表示	CT 値と色の対応の選択
色づけ機能	CT 値のヒストグラムの表示	UI の改善
CT 値と色の対応の表示	UI の改善	
縦方向の切断		
透過		

✓　スコープ S0：解析に必要な基本機能の実装(β版)
✓　スコープ S1　：コア試料の検索，新機能の実装，UI の改善
✓　スコープ S2：基本機能の拡張，インタフェースの改善
(※スコープ S2 の上記 2 項目は Android 端末向け Virtual Core Viewer でのみ実装)

各スコープにおける開発スケジュールの予定と実績を A.2 に示す．本プロジェクトでは，以下のような目標を立て開発を進めた．

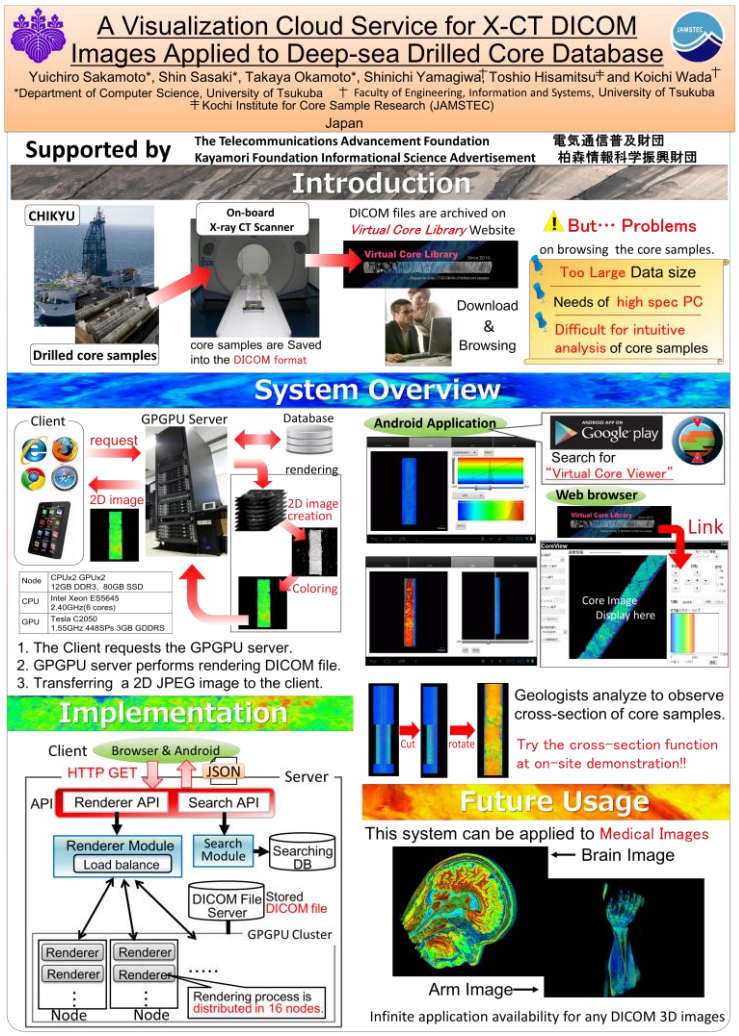
- 7 月末：第 1 回リリース　⇒　スコープ S0 の機能の実装完了
- 10 月 4 日：第 2 回リリース　⇒　スコープ S1 の機能の実装完了，中間報告でデモ発表
- 12 月 5 日：最終リリース　⇒　スコープ S2 の機能の実装完了，沖縄の学会で成果報告

本プロジェクトは当初の予定通りに順調に開発を行い，12 月 1 日には 3 段階の開発をすべて終え，Android 端末向け Virtual Core Viewer を Google Play で公開した．A.3 は，Virtual Core Viewer を公開している様子である．なお，12 月 5 日の学会で用いたポスターを成果報告として A.4 に示す．



A.2　開発スケジュールの予定と実績

A.3 Google PlayでのAndroid端末向けVirtual Core Viewerの公開

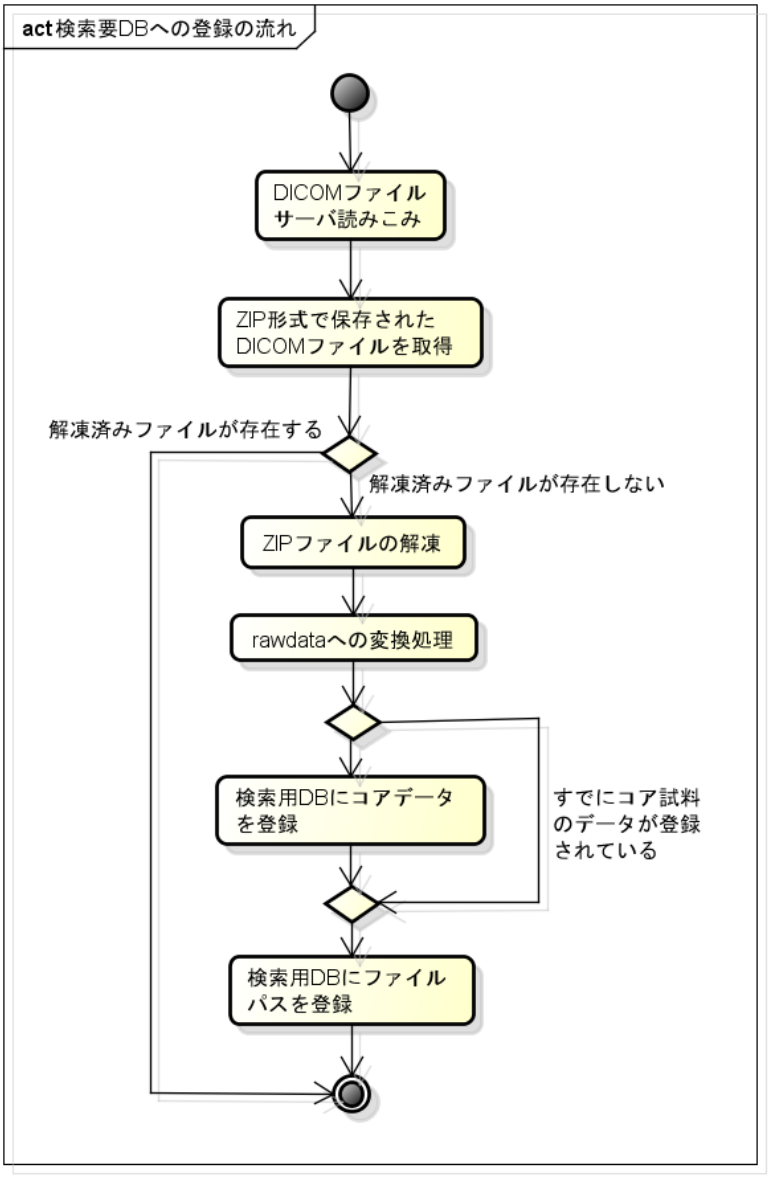


A.4　学会で用いたポスター

付録B：検索用DBへのデータ登録

本項では，検索用 DB へのデータの登録方法について示す．専用のスクリプトを実行することで，検索用 DB へのデータの登録が行われる．

コアデータはすべて DICOM ファイルサーバに登録されている．スクリプトでは，検索用 DB に登録されていないコア試料が新たに DICOM ファイルサーバに格納されたときに，そのコア試料のファイルを本サービスで利用できる形に変換する処理を施し，新たに検索用 DB にデータを登録する．スクリプトの処理の流れを B.1 に示す．



B.1　データベース登録までの流れ

データ登録スクリプトは，まずDICOMファイルサーバに保存されているファイルを取得する．新たに取得されたDICOMファイルは，ZIP形式で圧縮されているので，ZIP形式で圧縮されたDICOMファイルのみを取得する．次に，取得したファイルの解凍処理を行う．この時，すでに解凍済みのファイルが存在する場合は，すでに登録処理が終わっているものとみなし，そのファイルに対する登録処理を終了する．解凍済みのファイルが存在しない場合

は，DICOMファイルを解凍し，DICOMファイルサーバに展開する．DICOMファイルの展開が終了したら，そのファイルに対してrawdata作成処理を行う．解凍されたDICOMファイルをrawdataに変換しておくことで，高速な描画が可能となる．rawdataの作成が終了したら，検索用DBへの登録処理に移る．検索用DBへの登録は，coresampleテーブルへの登録処理と，fileinfoテーブルへの登録処理の2つの登録処理を行う．まずは，coresampleテーブルに問い合わせ，コアデータがすでに登録されていないかを確認する．コアデータが登録されていない場合は，新規に登録処理を行い，登録されている場合はコアデータの登録処理を行わない．次にfileinfoテーブルにコアデータが保存されているファイルパスを登録する．新規でコアデータを登録する場合は，ファイルパスを新規登録する．すでにコアデータが登録されている場合は，登録済みのファイルパスに対して更新処理を行う．この一連の処理を，DICOMファイルサーバに保存されているすべてのZIP形式で圧縮されたDICOMファイルに対して行うことで，検索用DBにデータを登録する．

定期的に検索用DBのそれぞれのテーブルについてバックアップを取得し，保存している．

バックアップファイルの保存は，cron によるバックアップ保存スクリプトの定期実行により行う．cron は，UNIX 系 OS において，コマンドやシェルスクリプトを指定した日時に自動的に実行するためのコマンドである．cron の設定ファイルである crontab を編集し，デーモンプロセスである crond を起動させることで，設定ファイルにもとづいて定期的にコマンドが実行される．crontab による設定ファイルは，*分 時 日 月 曜日 コマンド*のようにスペース区切りで各項目の設定を記述する．crontab でバックアップ取得スクリプトの実行スケジュールを以下のように設定した．

```
0 2 * * * DBbackup.sh >> /home/ccore/log/backup.log 2>&1
```

この設定では，バックアップを取得するスクリプトが毎日午前2時に取得される．バックアップを取得した際のログを，DBbackup.logというファイルに保存する．バックアップのファイルは，最大5つまで保存され，5つを越えると，最も古いバックアップファイルが削除される．

◆Q&A

・検索用DBに登録されているデータが消えてしまった場合

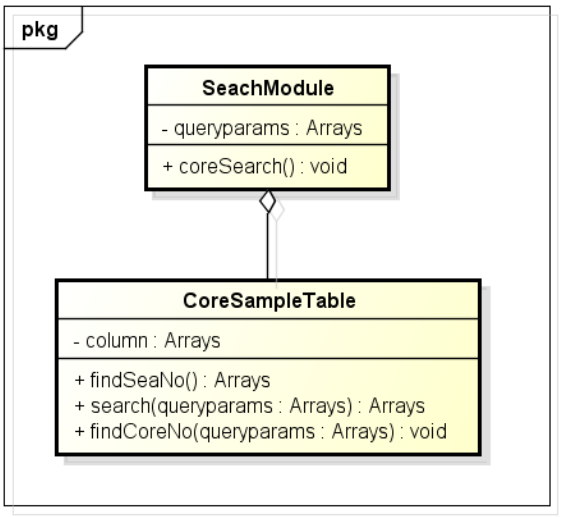
バックアップからデータを復元したい場合は以下のコマンドを実行することで検索用DBのデータを復元することができる．

```
%mysqldump　–user=ユーザ名　–password=パスワード　データベース名
　テーブル名　<　復元に用いるバックアップファイル
```

テーブル名は，coresampleとfileinfoのいずれかのテーブル名を選択する．復元に用いるバックアップファイルは，coresampleとfileinfoに対応したバックアップファイルを指定する．

付録C：検索モジュールの設計書

レンダラモジュールの設計をC.1に示す．C.1のクラス図では，コンストラクタとデストラクタ，変数の値を設定・取得するためのsetメソッドとgetメソッドの表記は省略する．



C.1　検索モジュールのクラス構成

SearchModuleクラス

検索APIから受け取ったパラメータにもとづいて，CoreSampleTableの検索処理を呼び出し，検索用DBに問い合わせを行う．

◆変数説明

・queryparams　検索APIが受信した$_GETの値を格納する配列．

◆メソッド説明

・coreSearch()

引数：なし

返り値：検索結果の配列

処理内容：検索APIが受信したパラメータにもとづいてCoreSampleTableのメソッドを呼ぶ．検索APIが受信したパラメータが空の場合は，findSeaNoを呼び出す．検索APIが受信したパラメータが6つ未満の場合は，searchを呼び出す．検索APIが受信したパラメータが航海番号，掘削サイト番号，ホール番号，コア番号，ビットタイプ，セクション番号の6つで合った場合，findCoreNoを呼び出す．

CoreSampleTableクラス

coresampleテーブルに問い合わせ文を発行するためのクラス．取得したいデータに応じた検を行える．

◆変数説明

・$columns　航海番号，掘削サイト番号，ホール番号，コア番号，ビットタイプ，セクション番号を格納する配列

◆メソッド説明

・findSeaNo()

引数：なし

返り値：検索結果の引数

処理内容：航海番号を取得するためのクエリを発行する．発行したクエリで検索用DBにアクセスし，航海番号を検索する．

・search($queryparams)

引数：Array $queryparams　HTTP GETで検索APIが受け取ったパラメータ

返り値：検索結果の配列

処理内容：検索用DBにアクセスし，検索APIが受け取ったパラメータの数に応じて異なるクエリを生成する．生成したクエリで検索用DBにアクセスし，掘削サイト番号，コア番号，ビットタイプ，セクション番号のいずれかの番号を取得する．

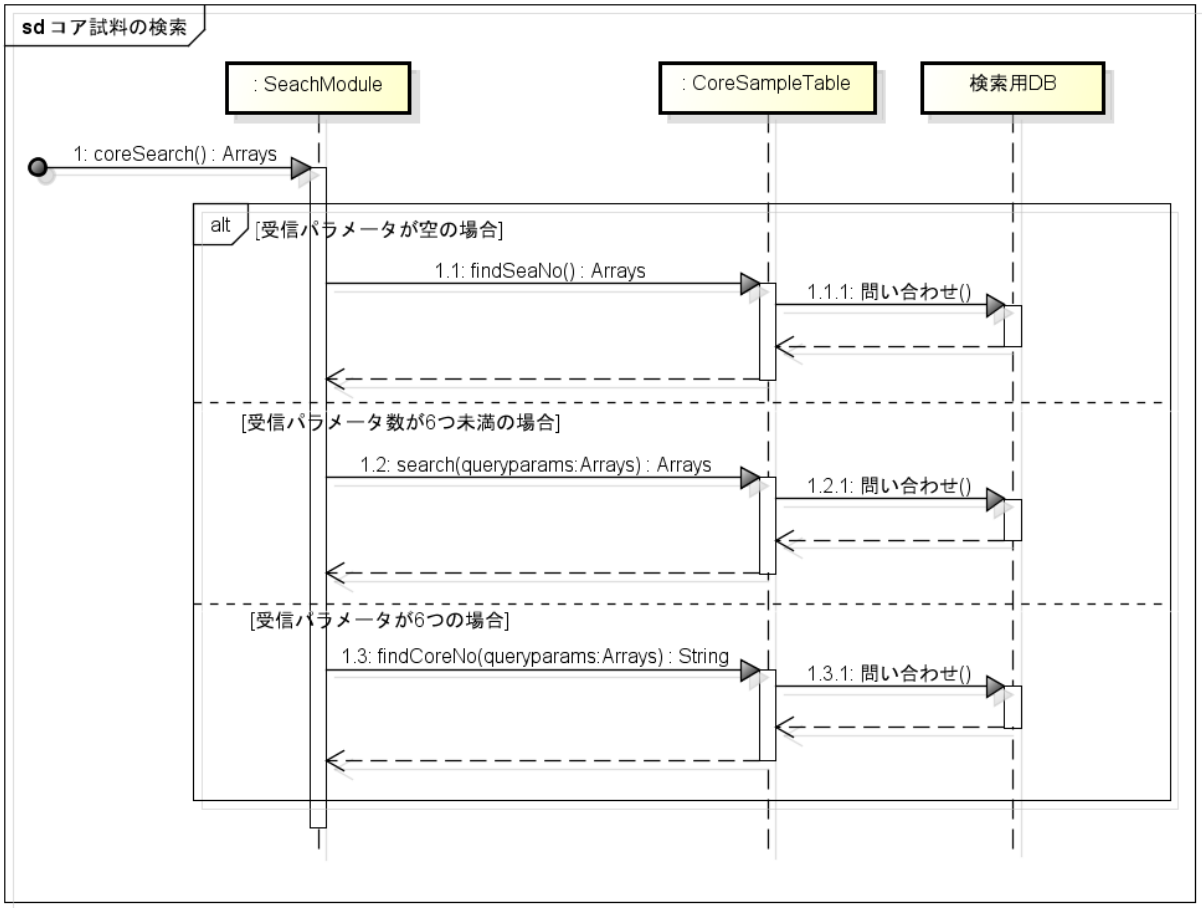
・findCoreID(Array $queryparams)

引数：Array $queryparams　HTTP GETで検索APIが受け取ったパラメータ

返り値：コア試料の固有ID

処理内容：検索用DBにアクセスし，航海番号，掘削サイト番号，コア番号，ビットタイプ, セクション番号の6つの番号からコア試料の固有IDを取得するためのクエリを生成し，検索DBに問い合わせる．

コア試料の検索を行う様子をC.2にシーケンス図で示す．まず検索APIを介して送信されたパラメータをSearchModuleクラスが受け取り，coreSearchメソッドを実行する．coreSearchメソッドでは，受け取ったパラメータの両に応じて，CoreSampleTableの異なるメソッドを呼び出す．検索APIが受信したパラメータが空の場合は，findSeaNoを呼び出す．検索APIが受信したパラメータが6つ未満の場合は，searchを呼び出す．検索APIが受信したパラメータが航海番号，掘削サイト番号，ホール番号，コア番号，ビットタイプ，セクション番号の6つで合った場合，findCoreNoを呼び出す．いずれのメソッドも検索用DBに対して問い合わせを行うメソッドとなっており，メソッド実行による検索結果を返り値として返す．



C.2　検索モジュールにおける処理の流れ

付録D：レンダラ削除スクリプト

スクリプトの定期実行は，cron で行う．レンダラ削除スクリプトは，スクリプト実行時の日時分と起動ログに記述されたアクセス日時分を比較し，その差が 30 分以上あるレンダラをすべて削除する．crontab でレンダラ削除スクリプトの実行スケジュールを設定以下のように設定した．

```
10, 30, 50 * * * * rendererkiller >> kill.log 2>&1
```

この設定では, 毎時 10 分, 20 分, 50 分にレンダラを削除するスクリプトを実行している．ワイルドカードは，その項目に対して何も条件を設定しない場合に用いる．レンダラ削除スクリプトの実行による出力はすべて kill.log にリダイレクトしている．

crontab には，以下の zombiekiller というコマンドの設定も記述されている．

```
0 * * * * zombiekiller >> kill.log 2>&1
```

このコマンドは，rendererkiller と同様にレンダラを削除するスクリプトであるが，負荷分散や軽減を目的としたものではない．基本的に，レンダラのプロセスは，起動ログが削除されたときに削除される．しかし，起動ログ，または，レンダラになんらかの障害が発生したときに，起動ログに記載されていないレンダラが起動され状態のままになることがある．起動ログに記載されていないレンダラのプロセスが存在していると，システムに対して障害を起こす可能性がある．zombiekiller コマンドは，起動ログに記載されていないレンダラの存在を見つけだし，削除するコマンドである．このコマンドは毎時 0 分に実行される．

◆Q&A

・crond の起動確認

以下のコマンドを実行し，以下のような表示がされたら crond が起動している．

```
# /etc/rc.d/init.d/crond status
   crond (pid   2833) is running...
```

crond が起動していない場合は以下のコマンドを実行し，crond を起動する．

```
# /etc/rc.d/init.d/crond start
```

・crontab の設定確認

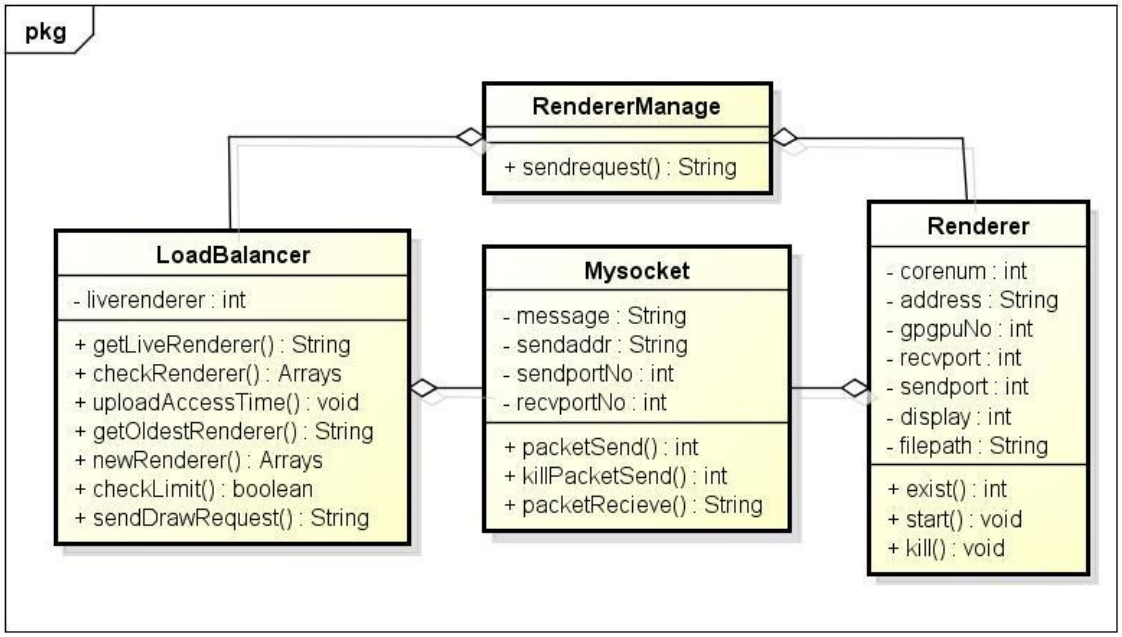
crontab –e コマンドを実行することで crontab の設定ファイルを確認できる．

・crontab の設定ファイルが消えてしまった場合

crontab_backup.txt に同様の設定が記述されているので，そのファイルの中身をコピーし，crontab の設定ファイルにペーストすることで復旧できる．

付録 E：レンダラモジュールの設計書

レンダラモジュールの設計をE.1に示す．E.1のクラス図では，コンストラクタとデストラクタ，変数の値を設定・取得するためのsetメソッドとgetメソッドを省略している．



E.1　レンダラモジュールのクラス構成

LoadBalancerクラス

起動ログを読み込むことでレンダラの起動状況を取得するクラスである．レンダラの起動状況から，アクセス頻度の少ないレンダラの特定，新しくレンダラを起動する際のノードの取得などを行う．

◆変数説明

・$liverenderer：起動しているレンダラの情報を二次元配列で格納する．コア試料のファイルパス，レンダラが起動しているポート番号，レンダラに割り振っているポート番号，アクセス日時を，レンダラの数だけ格納する．

◆メソッド説明

・getLiveRenderer()

引数：なし

返り値：配列

処理内容：起動ログを1行ずつ読み込み，起動しているすべてのレンダラの情報を配列に格納する．すべての起動ログの読み込みが完了した時点で，レンダラの情報を格納した配列を返す．

・checkRenderer()

引数：なし

返り値：true, または, false

処理内容：取得したレンダラの情報を参照し，ユーザのリクエストのコア試料がすでに起動しているかどうかを確認する．すでにレンダラが起動している場合は，trueを，レンダラが起動していない場合は，falseを返す．

・getOldestRenderer()

引数：なし

返り値：String $oldestrenderer　最もアクセス日時の古いレンダラのコア試料名

処理内容：起動ログに記されたアクセス日時を比較し，もっともアクセス日時の古いレンダラを見つけだし，そのレンダラのコア試料名を返す.

・newRenderer()

引数：なし

返り値：Array($nodeNo, $portNo) レンダリングサーバのノード番号，ポート番号

処理内容：起動ログに記されたレンダラの情報にもとづいて，レンダラの起動数が均等になるようなノードとポートを見つけだす．ノード番号とポート番号をそれぞれ配列に格納し，返す.

・checkLimit()

引数：なし

返り値：true, または, false

処理内容：現在起動しているレンダラの総数をカウントし，レンダラの起動数が上限を越えていないかをチェックする．レンダラの総起動数が，上限を越えている場合，trueを返し，上限を越えていない場合は，falseを返す.

・sendDrawRequest($nodeIP, $portNo, $params, $savefilepath)

引数：$nodeIP　ノードのIPアドレス

$portNo　ポート番号

$params　描画に用いる各種パラメータ

$savefilepath　生成した画像を保存するパス

返り値：$result, または, “NG”

処理内容：起動ログから，レンダラが起動しているノード番号と，レンダラのポート番号を取得し, 2つの情報を用いてMySocketオブジェクトを用いてソケット通信を行う．レンダラとのソケット通信の結果を返す．レンダラでの描画処理に成功した場合は, $resultにレンダラの成功メッセージを格納し，返り値として返す．失敗した場合は，NGの文字列を返す.

Rendererクラス

RendererManageクラスの要求に応じて，レンダラの起動・削除を実行するクラスである.

◆変数説明

・$filepath：レンダラと対応しているコア試料のファイルパス

・$nodeNo：レンダラが起動しているノード

・$address：レンダラが起動しているノードのアドレス

・$sendport：レンダラの送信用ポート番号

・$recvport：レンダラの受信用ポート番号

・$display：Xvfbで用いるディスプレイ番号

◆メソッド説明

・exist()

引数：なし

返り値：true，または, false

処理内容：引数で受け取ったコア試料のファイルパスにもとづいて，該当するレンダラが起動しているかどうかを確認する．該当するレンダラが起動している場合は，trueを返し，該当するレンダラが起動していない場合は，falseを返す．

・start()

引数：なし

返り値：なし

処理内容：Xvfb起動コマンドとレンダラ起動コマンドを生成し，SSHでそれぞれのコマンドを実行する．レンダラの実行に必要な各パラメータはsetメソッドであらかじめ設定しておく．必要なパラメータは，$filepath, $nodeNo, $address, $sendport, $recvport, $displayの6つすべての変数である．

・kill($renderer)

引数：$renderer 配列(コア試料のファイルパス，レンダラの起動しているノード，レンダラが使用しているポート番号)

返り値：なし

処理内容：受け取った引数にもとづいてレンダラを削除するためのメッセージを作成する．その後，ソケット通信の接続を確立し，レンダラ削除用のメッセージを送信する．

MySocketクラス

レンダラに対してソケット通信を行うためのクラスである．与えられたIPアドレス，ポート番号にもとづいてレンダラに対して通信を行う．

◆変数説明

・$message：ソケット通信で送信するメッセージ．描画用メッセ―ジか，削除用メッセージのいずれかを格納する．

・$sendaddr : 送信先のIPアドレス．レンダラが起動しているノードのIPアドレスとなる．

・$sendportno：ソケット通信の送信で使用するポート番号

・$recvportno：ソケット通信の受信で使用するポート番号

◆メソッド説明

・packetSend()

引数：なし

返り値：1, または, -1

処理内容：レンダラとのコネクションを確立し，レンダラに描画用のメッセージを送信する．描画用のメッセージはsetメソッドを用いてあらかじめmessageに格納しておく．MySocket生成時にコンストラクタで指定したポート番号とノードのIPアドレスにもとづいてソケット通信を行う．ソケット通

信プロトコルはTCPを指定する．コネクションの確率に失敗した場合にエラー番号を示す”-1”を返す．通信に成功した場合は，1を返す．

・killPacketSend()
引数：なし
返り値：1, または, -1
処理内容：レンダラにレンダラ削除用のメッセージである”exit¥¥”をソケット通信で送信する． メッセージの送信に成功した場合は1を返し，失敗した場合は-1を返す．

・packetRecieve()
引数：なし
返り値：$reply または, -1
処理内容：ポートを解放し，レンダラから処理結果を受信する．レンダラから処理結果を受信できた場合は，受け取ったメッセージを$replyに格納し，返り値とする．一定時間レンダラから処理が返ってこない場合は，エラー番号を示す“-1”を返す．

RendererManageクラス

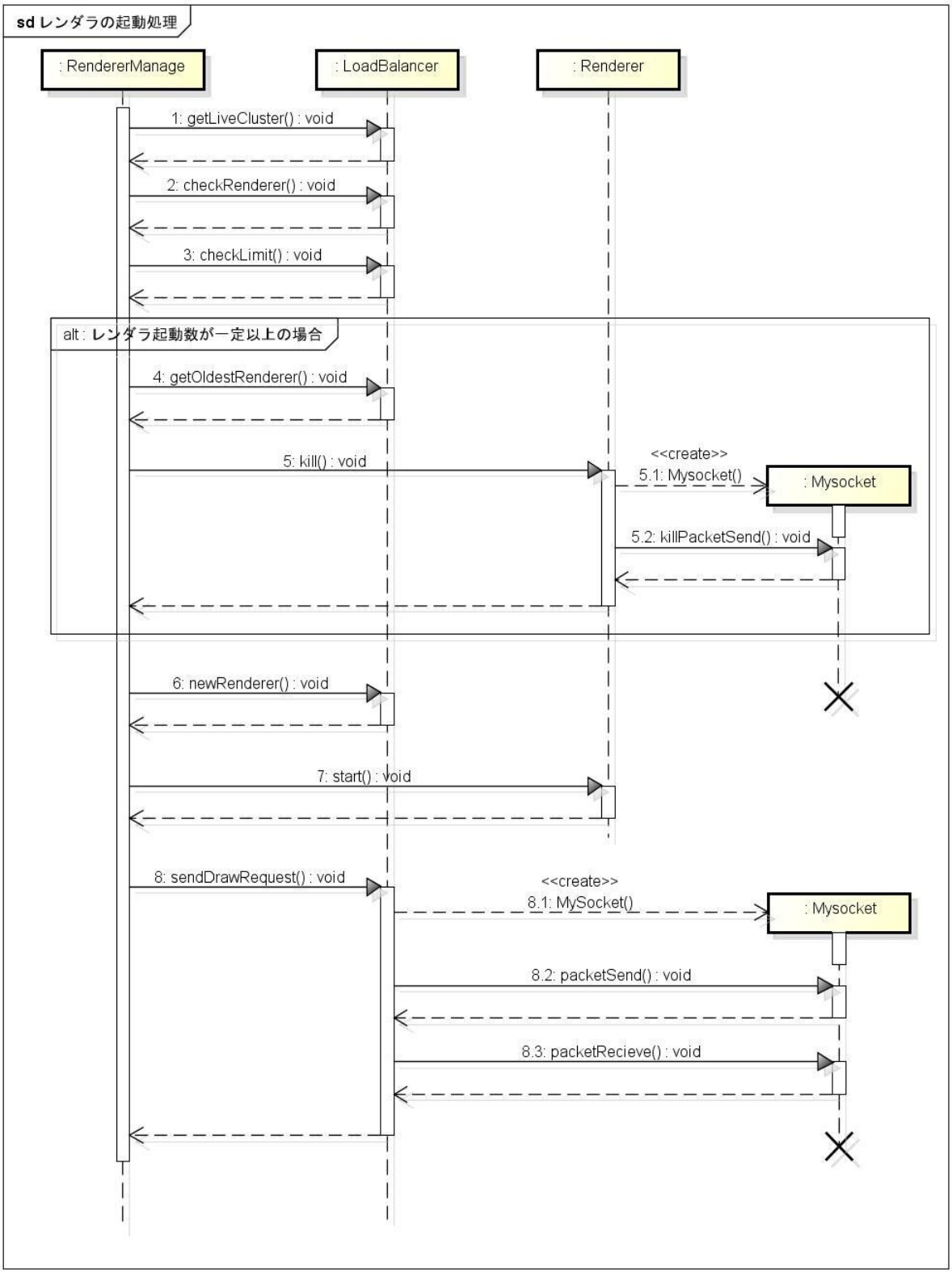
LoadBalancerクラス，Rendererクラスを用いて，負荷分散・軽減を考慮したレンダラの起動・削除を行うクラスである．

◆メソッド説明

・sendrequest($params, $savefilepath)
引数：$params　レンダラAPIが受信した描画用メッセージ
$savefilepath　レンダラで生成する画像のファイルパス
返り値：$result, または, “NG”
処理内容：LoadBalancerクラスからレンダラに関する情報を取得し，Rendererクラスに対して起動・削除の命令を出す．Sendrequestメソッドを実行することで，一連の作業を実行する．レンダラにおける画像生成処理に成功した場合は，それを表すメッセージを格納した$resultを返り値とし，失敗した場合は”NG”を返す．

レンダラモジュールの全体の処理の流れをE.2に示す．レンダラモジュールの処理は，RendererManageクラスのsendrequestメソッドがレンダラAPIから起動されたときに始まる．まず，ロードバランサクラスのgetLiveRendererメソッドにより，現在起動しているレンダラの一覧を取得する．次に，checkRendererメソッドを用いてすでにレンダラが起動しているかを確認する．すでにレンダラが起動している場合は，この時点で該当するレンダラにソケット通信を行い，処理を完了する．レンダラを新たに起動する場合は，checkLimitメソッドでレンダラの起動数を確認する．すでにレンダラ起動数が閾値を越えている場合は，getOldestRendererメソッドにより，もっともアクセス日時の古いレンダラを取得する．次にkillメソッドで該当するレンダラを削除する．レンダラの削除は，Rendererクラスにソケット通信の処理を依頼し，レンダラを削除するパケットを送信する．次に，newRendererメソッドで新たにレンダラを起動するノードとポート番号の組み合わせを取得する．取得した情報

に基づいてRendererクラスのstartメソッドを呼び出し，新たにレンダラを起動する処理を実行する．レンダラの起動処理が終了した時点で，LoadBalancerクラスのsendRequestメソッドでソケット通信を行い，レンダラに対してコア試料の画像生成を依頼する．コア試料の生成が完了したら，その結果を返り値として返す．



E.2　レンダラモジュールにおける処理の流れ

付録 F：SSH2 によるコマンドの実行

開発言語：PHP
使用モジュール：SSH2

PHP での ssh によるコマンドの実行には，SSH2 モジュールを用いる．SSH2 モジュールでは，レンダラが起動しているノードの IP アドレス，ユーザ名，パスワードを用いてノードにログインする．SSH2 によるリモートでのコマンド実行は以下のような手順で行う．

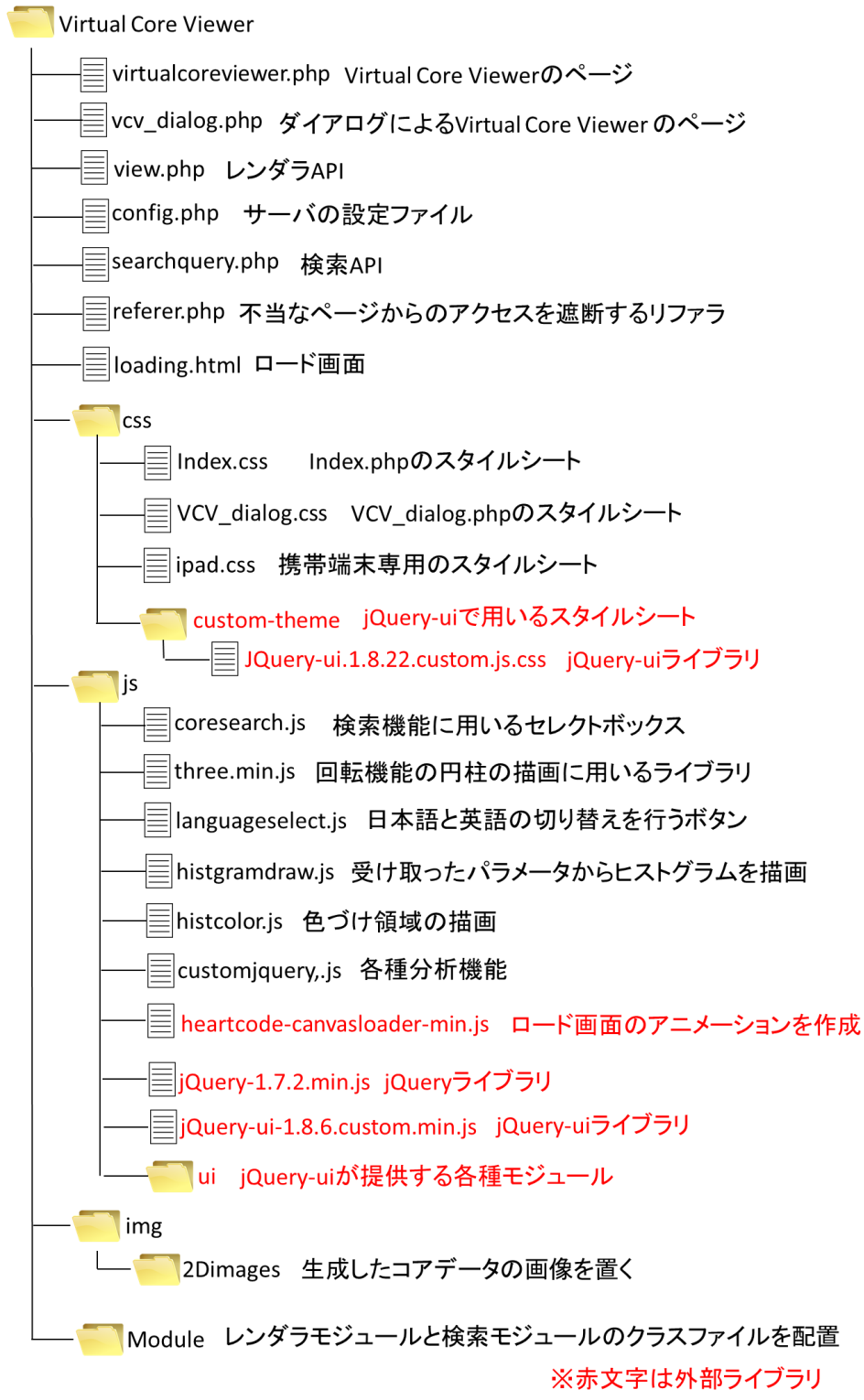
```
セッション =  SSH2_connect(“ノードの IP アドレス”, 22);
SSH2_auth_password(“セッション”, “ユーザ名”, “パスワード”);
SSH2_exec(“コマンド”);
```

SSH2_connect でノードに接続するためのセッションを生成する．セッションを作成したら，ユーザ名，パスワードを指定して SSH2_auth_password でノードにログインする．ログイン後，SSH2_exec でコマンドを実行できる．

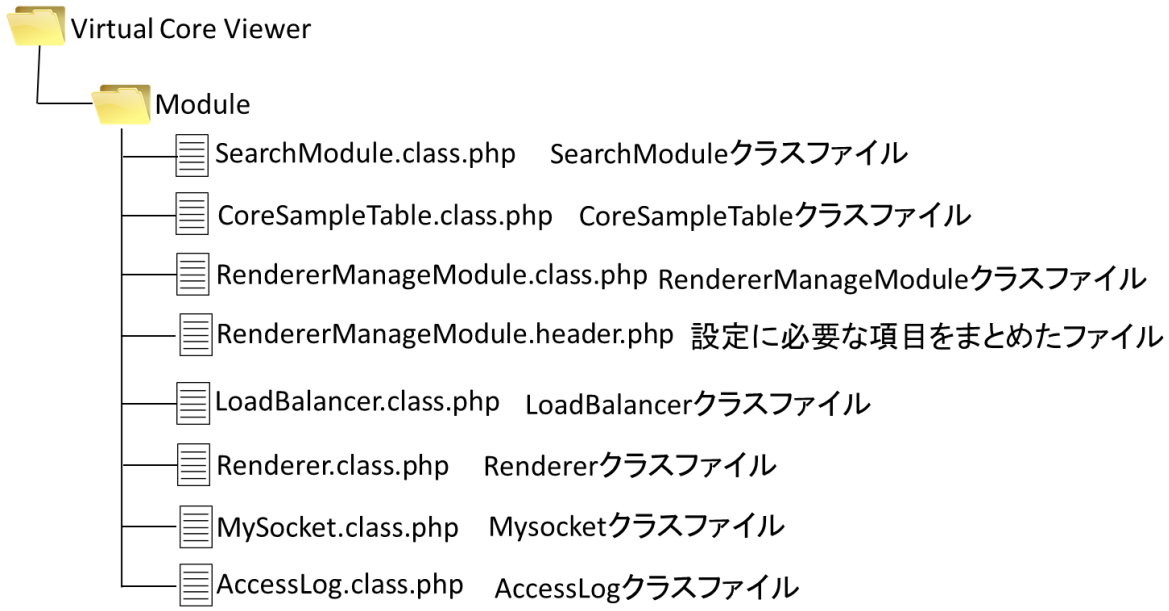
付録 G：本サービスのファイル構成と運用における各種設定

本サービスを構成するファイル構成は，以下の4つに分類することができる．

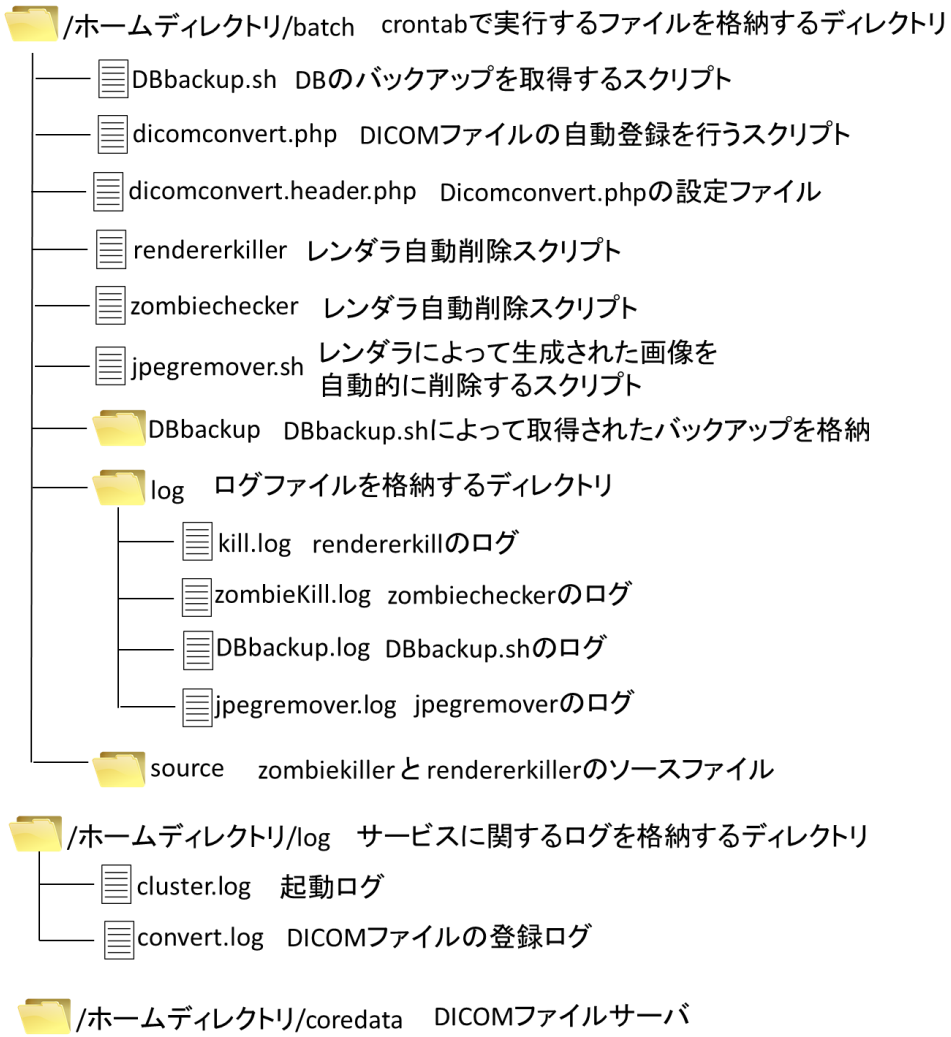
- G.1　Webアプリケーションを構成するファイル



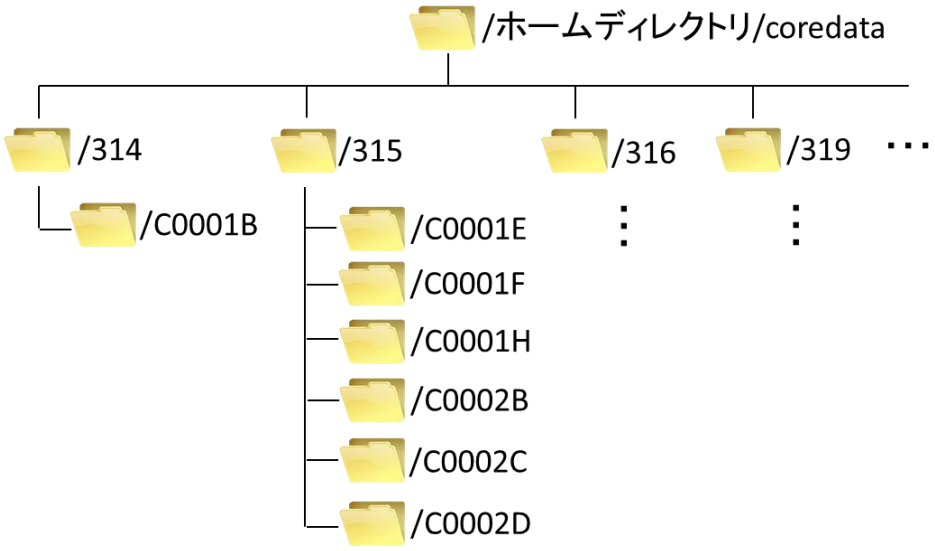
- G.2　レンダラモジュールと検索モジュールを構成するファイル



- G.3　定期的に実行するスクリプトやログを格納するファイル



- G.4　DICOMファイルサーバ



G.5　運用に関する設定事項

サーバを新調したときなど，ディレクトリ構造になんらかの変化が発生した際は，以下に示す設定項目に基づき，いくつかのモジュールのインストールや，ファイルの設定を変更しなければならない．

- インストールが必要なモジュール

  本Webアプリケーションを実現するのに必要なソフトウェアや外部モジュールを以下に示す．サーバを新調して新たに設定をする必要がある場合は，各ソフトウェア・モジュールのインストールをサーバ管理者に依頼する．

  - ✓ PHP
  - ✓ MySQL
  - ✓ Apache
  - ✓ php-mysql(PHPのMySQL拡張モジュール)
  - ✓ SSH2(PHPのssh拡張モジュール)

- 設定の変更が必要なファイル

  サーバを新調したときなど，ディレクトリ構造になんらかの変化が発生した際は，以下の表に示す設定事項に基づき，ファイルの設定を変更する．

設定事項	設定ファイル(参照する付録番号)
サーバのホスト名と画像の保存パスの変更	config.php(G.1)
レンダラに関する設定の変更	RendererManageModule.header.php(G.2)
データベースのユーザ名などの変更	RendererManageModule.header.php(G.2) dicomconvert.header.php(G.3)
ログファイルのパスの変更	crontab

✓ サーバのホスト名と画像の保存パスの変更

config.phpの設定項目はそれぞれ以下のとおりである．サーバ移転時に必要に応じて，下記項目を設定する．

・SAVEFILEPATH：画像を保存するファイルパス

・SERVICEURL：サーバのURL．デフォルト値はhttp://gpgpu.cs.tsukuba.ac.jp/である．

・READFILEPATH：生成されたコア試料の画像をクライアントアプリケーションが読み込むためのパスを設定する．SERVICEURLに続く，ファイルのパスを指定する．デフォルト値は，/img/2Dimages/coreNoとなっている．

✓ レンダラに関する設定の変更

RendererManageModule.header.php pの設定項目はそれぞれ以下のとおりである．レンダリングサーバにおけるレンダラ起動数の上限や，負荷分散の際に選択するノード数を設定したい場合は，下記項目を設定する．

・NODEMAX：レンダリングサーバの内，ロードバランサに選択を許可するノード数．0～16の値で設定すること．

・RENDERERMAX：1つのサーバに起動可能なレンダラの上限数を設定する．10以上の値を指定するとノードに負荷がかかりすぎるため，注意して設定すること．

・READFILEPATH：生成されたコア試料の画像をクライアントアプリケーションが読み込むためのパスを設定する．SERVICEURLに続く，ファイル　のパスを指定する．デフォルト値は，/img/2Dimages/coreNoとなっている．

✓ DBに関する設定の変更

DBのユーザ名，パスワードなどを変更する際には，RendererManageModule.header.phpとdicomconvert.header.phpにおいて，以下の項目を設定する．

・DBNAME：データベース名を設定する．

・USER：データベースを使用する際のユーザ名を設定する．

・PASSWORD：データベースを使用する際のパスワードを設定する．

✓ ログファイルのパスの変更

crontabにより実行されるスクリプトのログファイルを変更したい場合は，crontab –eコマンドを実行し，該当するスクリプトのリダイレクト先を変更する．

G.6　DCIOMファイルサーバへの新規データの追加

DICOMファイルサーバは，付録G.4に示すようなファイル構成となっている．コア試料の航海番号と掘削サイト番号がディレクトリの階層となっている．たとえば，314-C0001-1H-1といったコア試料は，/ホームディレクトリ/coredata/314/C0001/下のディレクトリに新規に保存される．

新たにコア試料のデータを追加したい場合は，該当する航海番号と掘削サイト番号ディレクトリ下にデータを置くことでデータを追加することができる．たとえば，315-C00001E-1H-1というコア試料を新たに登録したい場合は，/ホームディレクトリ/coredata/315/C0001/下のディレクトリにデータを配置する．もし，該当する航海番号と掘削サイト番号が存在しない場合は，自身でコアデータを保存するディレクトリを生成し，そのディレクトリ下にコアデータを置く．なお，新規登録するコアデータはZIP形式のもののみが可能である．

新たに追加したいコアデータの配置が完了したら，付録G.3で示したdicomconvert.phpを実行することで，付録Bで示したコアデータの登録処理が行われ，自動的にデータの登録が行われる．dicomconvert.phpを実行する際には，dicomconvert.header.phpの設定をよく確認した上で実行すること．